Methods and Applications of Uncertainty Quantification for Object Recognition

BERGISCHE UNIVERSITÄT WUPPERTAL

Dissertation

University of Wuppertal
Faculty 4 — Mathematics and Computer Science

submitted by **Tobias Riedlinger, M. Sc.**
for the degree of Doctor of Natural Sciences (Dr. rer. nat.)

Supervisor	Prof. Dr. Hanno Gottschalk
Co-Supervisor	PD Dr. Matthias Rottmann

Wuppertal, September 4, 2023

Acknowledgements

While working on the projects presented in this thesis and in the course of compiling all the contents into the form it has today, I have received invaluable support which I would like to express my deep gratitude for. First and foremost, I would like to thank both of my supervisors Prof. Dr. Hanno Gottschalk and Dr. Matthias Rottmann for giving me the opportunity of working with them on the projects which form the basis of this thesis. Their frequent support and guidance have taught me much over the last couple of years, all while making work and scientific discourse highly enjoyable.

Further, I would like to especially thank my co-authors Marius Schubert, Karsten Kahl, Kira Maag, Siniša Šegvić, Sarina Penquitt and Pascal Colling for the pleasant and close collaboration on some of the projects. This gratitude extends to the entire "BUW-KI" team under Hanno Gottschalk and Matthias Rottmann which was always a pleasure to be a part of. I am looking back to the great time and fun moments we shared and will keep fond memories of my time as a doctoral candidate. Special thanks go to Annika Mütze, Pascal Colling and Patrick Krüger for proofreading vital parts of this manuscript and for unceasing mental support throughout the process.

Finally, I thank my parents and close relatives as well as my close friends for their enduring support and belief in me.

Foreword

In the present thesis, the plural first person writing style has been chosen as the standard of formulation as it is the usual style in mathematics and computer science research. The contents of chapters 3 to 7 are in large parts taken word-by-word from the publications listed below. Redundancies have been replaced by references to the first appearance in the text which clarify notation or deliver the necessary background information. Notation has been adjusted in an effort for consistency from the theoretical foundation described in chapter 2 through to chapter 7. The following list of publications includes a short description of the contributions made to each work.

(I) T. Riedlinger, M. Rottmann, M. Schubert, and H. Gottschalk, *Gradient-Based Quantification of Epistemic Uncertainty for Deep Object Detectors*, in 2023 IEEE/ CVF Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, Jan. 2023, IEEE, pp. 3910–3920

(Independently carried out with the exception of the implementation of the MetaDetect baseline which we gratefully thank Marius Schubert for)

(II) K. Maag and T. Riedlinger, *Pixel-wise gradient uncertainty for convolutional neural networks applied to out-of-distribution segmentation*, arXiv preprint arXiv: 2303.06920, (2023)

(Theoretical consideration and motivation, development and implementation of the gradient computation methodology resulting in efficiency)

(III) T. Riedlinger, M. Schubert, K. Kahl, H. Gottschalk, and M. Rottmann, *Towards rapid prototyping and comparability in active learning for deep object detection*, arXiv preprint arXiv:2212.10836, (2022)

(Design and generation of datasets, experiment logistics and management, statistical evaluation of generalization results and runtime experiments)

(IV) M. Schubert, T. Riedlinger, K. Kahl, D. Kröll, S. Schoenen, S. Šegvić, and M. Rottmann, *Identifying label errors in object detection datasets by loss inspection*, arXiv preprint arXiv:2303.06999, (2023)

(Formulation of theoretical considerations on the method's viability, proofs for statistical loss separation in classification case)

(V) T. Riedlinger, M. Schubert, S. Penquitt, J. Kezmann, P. Colling, K. Kahl, L. Roese-Koerner, M. Arnold, U. Zimmermann, and M. Rottmann, *LMD: Light-weight Prediction Quality Estimation for Object Detection in Lidar Point Clouds*, June 2023

(joint co-supervision of the original Master's thesis, introduction, related work, methodlogy description, coordination and review)

An additional article which is closely related to and in parts redundant with (I) has been published over the course of the project "KI-Absicherung — Safe AI for Automated Driving":

T. Riedlinger, M. Schubert, K. Kahl, and M. Rottmann, *Uncertainty quantification for object detection: Output-and gradient-based approaches*, in Deep Neural Networks and Data for Automated Driving, Springer, Cham, 2022, pp. 251–275

The publication-based chapters 3 to 7 have published supplementary material merged with the main parts of the respective papers and rearranged with the aim to read more like a book chapter than a research paper. To this aim, results from the supplementary materials which extend main results are often moved to subsections or paragraphs marked by the symbol "⑂" indicating text that contains additional or deeper insights that are not vital parts of the contributions.

Contents

1 Introduction

Deep learning applications and research have experienced overwhelming growth over the last few decades. During the earliest considerations of computational models resembling biological neural connections [81, 117, 150], hardware computations were generally limited. Over time, when hardware slowly caught up to the mathematical models [91, 216], until recently, when acceleration via graphics processing units have made statistical research possible on a larger scale, machine learning research has come a long way. Nowadays, applications of machine learning and deep learning already permeate everyday life in countless areas [71, 127, 133, 135, 184] and applications become increasingly common. Whenever tasks can be automated or interpolation from recorded data is possible, deep learning approaches are not far such as in diagnosis, monitoring and surveillance tasks. Such applications are often closely related to perception and visual recognition tasks.

Deep learning models designed for a specific application frequently surpass human performance due to being able to recognize patterns in data which are hard to perceive for humans [193]. Deep neural networks (DNNs) are oftentimes called "black boxes" due to their obscure decision-making process which is difficult to explain and interpret for humans. The measured performance of DNNs is, however, hard to deny. Given their use, particularly in safety-critical applications [41, 42, 200], creates the need for understanding possible failure modes and developing methods to prevent errors. Making perception algorithms ready for medical and transportation applications as well as robotics requires focusing on perception errors to prevent potentially fatal consequences [140, 197]. Misclassifications, false detections and overlooked instances are crucial scenarios which need to be under control. One approach to identify possible shortcomings of DNNs is the integration of prediction uncertainty estimation components. This way, credibility measures can be assigned to each prediction. Forms of such measures can be variance measures, confidence estimation measures or estimates of the prediction accuracy or quality, e.g., for regression tasks. So-called Bayesian neural networks (BNNs [115], see also section 2.2.4.2) give a theoretical foundation for treating model uncertainty in DNNs by regarding parameters as random variables. Their application to state-of-the-art DNN models is presently, however, unfeasible due to computation and runtime concerns. The fitting of BNNs is computationally highly costly. Moreover, in order to compute prediction uncertainty measures, weights in the network need to be sampled, and several forward passes performed. This procedure is unfeasible for applications where runtime is of importance such as automotive applications. Approximations to BNNs have been developed for feed-forward neural networks in the form of Monte-Carlo dropout [45, 169] and deep ensembles [88, 179] which alleviate some burden of BNNs, particularly the training aspect (see section 2.2.4.2). However, these methods still suffer from the shortcoming of having high inference times albeit that their application can be regarded as somewhat universal for feed-forward DNNs.

Uncertainty Quantification in Object Detection. In contrast, uncertainty quantification (UQ) methods which are strictly tailored to a task at hand, like object detection of semantic segmentation, might be able to circumvent the runtime concerns. Moreover, similar or even superior prevention of prediction errors may be achieved by such methods. In object detection, foreground instances from a particular set of semantic categories are supposed to be found and localized on the input. The input for such an algorithm might be a camera image or a point cloud with three-dimensional localization. Perhaps the simplest practically used UQ method for object detectors is the intrinsic confidence rating standard object detectors [101, 106, 142, 144] produce for each potential detection. Possible error modes of object detectors are false positives, i.e., hallucinated objects which are not there in reality, and false negatives, i.e., overlooked instances. A correctly detected instance with incorrectly assigned category may be regarded as either of the previous errors. All these failures can have detrimental consequences in a safety-critical environment such as traffic and could be in principle prevented. Conditioned on the input, each image region is assigned a quantity indicating the estimated probability of the existence of an object. This quantity is often referred-to as “confidence score”. Thresholding based on the confidence score at some fixed value determines foreground instances and what will remain as a prediction. Regions falling below this threshold will be ignored and regarded as background which does not contain objects of interest. By assigning improved confidence scores to the right regions prevents false positives by suppressing the respective predictions and false negatives by moving regions to the foreground. Adjustment of the confidence assignment to different image regions can, therefore, make predictions statistically more reliable. The first more sophisticated methods for object detection were adaptations of Monte-Carlo dropout [120, 121] and slightly later of deep ensembles [109], focusing on sampling-based methods approximating BNNs. Another branch of UQ methods is based on estimating prediction uncertainty as additional output variables [80] of the DNN [41, 57, 85, 90] by altering the loss function. Yet other approaches to UQ estimate the instance-wise localization accuracy, either by also modifying the loss function [76], or via post-processing of the object detector output [161].

A novel approach to UQ in object detection is investigated in the work reported in chapter 3 based on [145]. The method is inspired by the way object detectors are trained, where parameters receive iterative updates which are larger the less accurate the prediction is. In utilizing this idea, ground truth feedback used to compute gradients is replaced by the network’s own prediction. Assuming a pre-trained model, the resulting instance-wise self-learning gradients contain uncertainty information. The computed gradient quantities are used in a post-processing manner to assign alternative confidence scores. The results suggest that the alternative confidence assignment is statistically able to better separate true from false predictions while being less susceptible to the choice of decision threshold. Moreover, the alternative confidence assignments show improved statistical reliability, i.e., calibration, and the confidence estimation yields better object detection performance, i.e., a reduction in prediction errors.

Uncertainty Quantification in Semantic Segmentation. In semantic segmentation, concepts like Monte-Carlo dropout have already been investigated [64, 94, 188] on the pixel-level in addition to the maximum softmax probability and softmax entropy [4]. Post-processing models using output information to assess prediction quality estimation on the segment level have been previously developed and applied to considerable success [16, 113, 152, 153] Contrasted with object detection, semantic segmentation usually does not have a background option. Instead, models determine a discrete probability distribution over the semantic classes for each pixel. Being forced to make one class prediction, the most likely class under the predicted distribution is chosen. This forces the model to decide on one of the predefined semantic classes. In the example of street scenes this can lead to errors whenever a previously not encountered object is present, e.g., a giraffe in a European street scene. Such scenarios are unusual for European streets, meaning that they are outside the data distribution used for training, hence, such scenes or objects are called out-of-distribution (OoD). Such objects are often mis-classified at high predicted pixel-wise confidence making them hard to detect by simple pixel-wise uncertainty methods like softmax entropy. It is, therefore, imperative that semantic segmentation models employed in complex and safety-critical environments must involve some mechanism warning about OoD objects. Such a mechanism can act as a reject-option indicating when a segmentation model is unable to confidently determine one of the given semantic classes. Simple models based on Monte-Carlo dropout [1, 4, 124] have been investigated in early stages of the development of OoD segmentation. Other simple methods such as softmax uncertainty [4] have also been under consideration as canonical UQ methods. More recent work oftentimes relies on the usage of additional training data [3, 5, 15, 54], alterations of the training scheme [15, 53, 105] of a segmentation model, significant increases in inference time [5, 53, 99] or combinations thereof.

In order to achieve a less constrained OoD segmentation method, we propose an alternative method in chapter 4 based on [112]. Similarly to the method described in chapter 3, it is based on pixel-wise self-learning gradients. For semantic segmentation, however, these can be computed with minimal computational overhead without alteration of the given segmentation model. The only requirement is that pixel-wise segmentation is based on convolutional layers. The latter requirement is a light restriction since, at the time of writing, segmentation models are primarily convolution-based with few exceptions. We show that anomaly segmentation based on the proposed gradient scores achieves performance close to that of state-of-the-art methods which demand additional training data, influence the segmentation quality or employ models of significantly increased complexity.

Uncertainty Quantification Applied in Active Learning for Object Detection. In addition to correcting the prediction of a neural network or abstaining from predictions on the basis of uncertainty, there are other applications of UQ in deep learning. One such application is active learning [164] (AL). Here, the predictive uncertainty of a model for samples of unlabeled data are utilized in order to decide which to include in training data. Industrially, this scheme is interesting due to the high cost of annotating data [9, 212]. Restricting annotation costs only to the most important data or prioritizing data can accelerate the performance evolution of deep learning algorithms. Deciding on the importance of data can be done on the basis of uncertainty [30, 96, 98, 159, 165], although this is not strictly necessary [25, 163, 168]. The effectiveness of "querying" data for annotation by a human following a specific selection strategy is compared with other methods, e.g., random selection of data, by repeatedly fitting models with increasing amounts of data. Each time, the DNN needs to be trained to convergence in order for the selection strategy to be based on a converged model. In order to account for stochastic fluctuations, experiments are repeated under different random seeds resulting in severe computational cost in research and development of AL strategies. Likely due to this and related circumstances, development of AL methods for object detection has not received a lot of attention in the literature over the years. Early approaches utilized predictive UQ based on class probability distributions [8, 156] or committee-like strategies [156]. A task agnostic method based on loss estimation [206] has been applied to object detection as well. More recently, uncertainty measures derived from Monte-Carlo dropout [58] have been further investigated. Such measures are comparable with committee selection strategies. Learned uncertainty measures [23, 210] generate uncertainty estimation by modifying the object detection architecture.

Reaching meaningful results in AL for object detection frequently involves manual hyperparameter tuning. Training for different hyperparameter configurations slows down research considerably and leads to inconsistent use of AL hyperparameters between different methods. In chapter 5, a quickly converging and controllable development environment for AL strategies of deep object detectors is introduced based on the work [146]. The sandbox environment involves two non-trivial object detection datasets and down-scaled versions of standard object detection architectures. Experiments of common uncertainty-based AL strategies suggest that results in terms of which method performs better than another generalize to some capacity between datasets. More precisely, we regard the similarity of performance rankings of different methods between two datasets. We find that the similarity between sandbox results and results on public benchmark datasets are numerically comparable to the similarity of results on two different benchmark datasets. Quantitative results on the sandbox environment, however, can be generated at a fraction of the time consumed for development and experiment execution.

Uncertainty Quantification Applied in Label Error Detection. Another application of uncertainty is the detection faulty annotations in public benchmark datasets [130, 154]. While being time-consuming endeavors, annotating datasets is also an exhausting occupation, so human workers are naturally error-prone leading to incorrect annotations. The latter oftentimes go unnoticed in training and testing data due to automation in the implementation. Testing results on faulty annotations are taken at face value since performance metrics of models are computed utilizing the "ground truth" in the data, not further questioning the "truth" contained in the data. Regarding false predictions of high-performing models sometimes reveals that the model actually generalized correctly for some samples. Disagreements with the ground truth occur due to the latter being incorrect. While automated label error detection is not a completely new idea [35], it has only recently started receiving attention in the context of image data [130,154]. Work in image classification [130,131] using prediction uncertainty has shown that even the MNIST classification dataset [92] contains label errors. Object detection label errors were first detected based on box-wise classification uncertainty [67]. Recently, label error detection methods based on post-processing UQ have been developed in semantic segmentation [154].

Label error detection for lidar data has not been addressed before in the literature. In chapter 7 (based on [148]), an UQ method similar to the one described in chapter 3, however, based only on the DNN output, is introduced for three-dimensional object detection in Lidar point clouds. This method, producing advanced confidence assignments, is then utilized to detect annotation errors in Lidar point cloud datasets. Such datasets are especially complex to review and annotate due to the rich three-dimensional scene geometry.

Another short-coming of previous label error detection methods for object detection and semantic segmentation is their use of the ground truth. Ground truth information has previously only been used to identify faulty predictions, leaving a large amount of information unused. An alternative approach for camera images is taken in chapter 6 which is based on [162]. Here, the instance-wise learning loss using the given, potentially faulty, ground truth is employed to generate bounding box proposals of high interest. Regarding the error decomposition studied in empirical risk minimization, such regions with high loss are intimately connected with sources of uncertainty. However, the computed loss using the ground truth is not exactly an expression of uncertainty concerning the prediction of the DNN. It can, instead, be regarded as an uncertainty in the combination of network prediction and annotation given the respective DNN input.

Structure. The remainder of the present thesis is structured as follows. We introduce the theoretical foundations and necessary notation in chapter 2. Foundations for the publication-based chapters 3 to 7 involve probabilistic foundations of machine learning (section 2.1), neural network building blocks and associated theory (section 2.2) and specialized machine learning tasks from computer vision (section 2.3). The following chapters 3 to 7 are structured in the classical format

Introduction — Related Work — Method — Experiments — Conclusion.

Starting with chapter 3, we introduce a novel method to quantify uncertainty in deep object detectors based on gradient quantities. In chapter 4, a related method to compute uncertainty gradient scores per pixel in semantic segmentation is presented and applied to OoD detection. Chapter 5 introduces an environment for rapidly producing and testing prototype implementations of AL strategies in deep object detection. We mainly focus on uncertainty-based selection mechanisms. Another application of UQ is investigated in chapter 6 where annotation errors in object detection datasets are automatically identified. The novel method introduced is based on instance-wise loss values. The last publication treated in chapter 7 introduces a post-processing UQ method for three-dimensional object detection based on lidar point clouds. Annotation error identification is also treated as a direct application of the presented method. Finally, we close in chapter 8 by putting the different publications into a common context, comparing them and defining unanswered questions and research areas for future work.

2 Foundations

Chapters 3 to 7 are based on work in applied deep learning. Here in chapter 2, we lay the theoretical foundations and notation which underlies all the following, starting with the mathematical basis of statistical learning theory in section 2.1. Dealing with learning models in a general sense, statistical learning theory makes probabilistic statements about when learning from data succeeds and what that means. Afterwards in section 2.2, we explain deep neural network architectures, the center stones of deep leaning and which components they contain. Specificities needed for the particular computer vision applications of object detection and semantic segmentation are explained in section 2.3, the final section of this chapter.

2.1 Statistical Learning Theory: Probabilistic Foundation

The field of statistical learning is the mathematical formulation of the tasks of machine learning and generally investigates the task of understanding structure in data and recognizing patterns on the basis of data. This section draws from [51,166] and has a survey-like character.

The setting of statistical learning theory is concerned with a set or space of data $\mathcal{X}$. Oftentimes, we will have $\mathcal{X} \subseteq \mathbb{R}^d$ like, e.g., in the case of RGB images[1]. Data points are further modeled as random variables $\boldsymbol{X} : (\Omega, \mathscr{A}, \mathrm{Pr}) \to \mathcal{X}$ over the probability space $(\Omega, \mathscr{A}, \mathrm{Pr})$ following some distribution[2] $\mu_{\boldsymbol{X}} := \boldsymbol{X}_* \mathrm{Pr}$ over $\mathcal{X}$. We write $\boldsymbol{X} \sim \mu_{\boldsymbol{X}}$. Based on the task at hand, we usually aim at one of two things. In supervised learning such as classification or regression tasks, we want to *assign a target* quantity y to each realized

[1]RGB images can be encoded as data points $\boldsymbol{x} \in \mathcal{X} = [0,1]^{3\times H\times W}$ represented by 3 color channels per pixel, where $H \times W$ is the spatial resolution of the image. We elaborate on data and target spaces for computer vision in section 2.3.

[2]Strictly speaking, the push-forward measure $\mu_{\boldsymbol{X}}$ of Pr along the measurable function $\boldsymbol{X}$ is a probability measure over $\mathcal{X}$ equipped with the Borel-σ-algebra respective to some topology given on $\mathcal{X}$.

data point $\boldsymbol{x} \in \mathcal{X}$. This mapping is often accomplished by learning or fitting some model[3] to an available sample of coupled data-target pairs. This amounts to the search of some predictor function $f : \boldsymbol{x} \mapsto y$. In unsupervised learning such as density estimation or clustering, we are usually interested in learning or *estimating the distribution* $\mu_{\boldsymbol{X}}$ directly from a sample drawn from $\mu_{\boldsymbol{X}}$. We will briefly describe both settings in some more detail, introducing the necessary notation that is used in the language of statistical learning theory.

2.1.1 Supervised vs. Unsupervised Learning

Learning tasks can be roughly separated into three categories: supervised learning, unsupervised learning and reinforcement learning. The first two are related to each other and can be mostly distinguished by the structure of the data which is obtainable. Reinforcement learning deals with decision-making of an "agent" in some environment to some modelled aim. We shall not deal with reinforcement learning in this thesis.

2.1.1.1 Supervised Learning: Recognizing Targets

Supervised learning deals with the assignment of target values $y \in \mathcal{Y}$ in some target space $\mathcal{Y}$ to data points $\boldsymbol{x} \in \mathcal{X}$. Targets can live in a continuous space (*regression*) like $\mathcal{Y} = \mathbb{R}^s$ or in a discrete space (*classification*) $\mathcal{Y} = \{1, 2, \dots, C\} =: [C]$ for some $C \in \mathbb{N}$. Available data then comes in coupled pairs $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$ where $\boldsymbol{x}$ is a realization of the random variable $\boldsymbol{X}$ which follows some distribution $\mu_{\boldsymbol{X}}$. In a slightly simplified setting, one assumes the existence of a labeling function $f : \mathcal{X} \to \mathcal{Y}$ assigning exactly one value $y := f(\boldsymbol{x}) \in \mathcal{Y}$ to each $\boldsymbol{x} \in \mathcal{X}$. A more general case will be treated in section 2.1.1.3. Given a sample

$$\chi_n := ((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)) \tag{2.1}$$

of realizations[4] the aim is to *find a predictive function* $\widehat{f} : \mathcal{X} \to \mathcal{Y}$ which is close to the labeling function f in a suitable sense. To this end, $\widehat{f}$ oftentimes is assumed as a parametric model which undergoes some fitting procedure given χ_n. In our applications, $\widehat{f}$ tends to be a linear model, a tree model or a neural network (see section 2.2) which is optimized by least squares regression, some related optimization procedure or by some flavor of stochastic gradient descent (see section 2.2.2.1).

Given some metric[5] $d(\cdot\|\cdot) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ on $\mathcal{Y}$, the theoretical or distributional error of a model $\widehat{f}$ compared with f can be expressed as

$$\mathcal{L}_{\mu_{\boldsymbol{X}}}(\widehat{f}, f) := \mathbb{E}_{\boldsymbol{X} \sim \mu_{\boldsymbol{X}}} \left[d\left(\widehat{f}(\boldsymbol{X}) \middle\| f(\boldsymbol{X}) \right) \right]. \tag{2.2}$$

[3]Often, the models used in practice are parametric models. In sections 2.2 and 2.3 we focus on deep neural networks as models.

[4]The individual realizations $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ are usually assumed to be drawn from the same distribution. Later, $\chi_n = \{\boldsymbol{X}_1, \dots, \boldsymbol{X}_n\} \sim \mu_{\boldsymbol{X}}^n$ will be modelled as a set of identically and independently distributed (short i.i.d.) random variables.

[5]This metric can originate from a norm in case $\mathcal{Y} = \mathbb{R}^s$ or be discrete, e.g., $d(a\|b) := \delta_{ab}$ with the Kronecker symbol δ_{ab} if $\mathcal{Y}$ is a discrete space. Strictly speaking, d need not be symmetric or fulfill a triangle inequality as long as $d(a\|b) \geq 0$, $d(a\|a) \leq d(a\|b)$ for any $a, b \in \mathcal{Y}$ and $d(a\|b) = 0$ if and only if $a = b$. We will continue to assume a metric structure, however.

However, this quantity is in practice not accessible since the entire distribution $\mu_{\boldsymbol{X}}$ is not known. In order to assess the quality of a model, one therefore approximates $\mathcal{L}_{\mu_{\boldsymbol{X}}}$ on a finite test sample $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{|\mathcal{D}|}, y_{|\mathcal{D}|})\} \subset \mathcal{X} \times \mathcal{Y}$. This test sample is assumed to originate from the same distribution as χ_n and one computes the empirical error

$$\mathcal{L}_{\mathcal{D}}(\widehat{f}, f) := \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} d\left(\widehat{f}(\boldsymbol{x}) \middle\| \, y\right). \tag{2.3}$$

Note that $y = f(\boldsymbol{x})$. Under additional assumptions[6], the law of large numbers then guarantees that $\mathcal{L}_{\mathcal{D}}(\widehat{f}, f) \to \mathcal{L}_{\mu_{\boldsymbol{X}}}(\widehat{f}, f)$ as the sample size grows ($|\mathcal{D}| \to \infty$).

2.1.1.2 Unsupervised Learning: Finding Structure

In unsupervised learning, given data $\chi_n := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, the goal is to find an estimate $\widehat{\mu}_n = \widehat{\mu}_n(\chi_n)$ of $\mu_{\boldsymbol{X}}$ based on χ_n. For example, a parametric model can be used to estimate the data density of $\mu_{\boldsymbol{X}}$, that is if $\mu_{\boldsymbol{X}} = p_{\boldsymbol{X}} \cdot \mathrm{d}x$ where $\mathrm{d}x$ denotes the Lebesgue measure over $(\mathcal{X}, \mathscr{B}_{\mathcal{X}})$ (assuming $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathscr{B}_{\mathcal{X}}$ the Borel-σ-algebra over $\mathcal{X}$), we may aim at estimating a probability density $\widehat{p}_n : \mathcal{X} \to \mathbb{R}_+$ such that

$$\mathcal{L}_{\mu_{\boldsymbol{X}}}(\widehat{\mu}_n, \mu_{\boldsymbol{X}}) := \mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}\left[d\left(\widehat{p}_n(\boldsymbol{X}) \| \, p(\boldsymbol{X})\right)\right] \tag{2.4}$$

measures the true theoretical or distributional quality of the estimation $\widehat{\mu}_n$ given some metric d on $\mathbb{R}_+$. However, since the true value of $p(\boldsymbol{x})$ is again not known, we cannot test the quality of $\widehat{\mu}_n := \widehat{p}_n \cdot \mathrm{d}x$. Instead of the expectation in eq. (2.4), more general distance measures d can be defined directly on the space of probability measures $\mathscr{M}_1(\mathcal{X}, \mathscr{B}_{\mathcal{X}}) =: \mathscr{M}_1(\mathcal{X})$:

$$\mathsf{d}(\cdot\|\cdot) : \mathscr{M}_1(\mathcal{X}) \times \mathscr{M}_1(\mathcal{X}) \to \mathbb{R}_+. \tag{2.5}$$

Such distance measures are called *risk or loss functionals* and in some cases these can be approximated on a finite set of test data $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|\mathcal{D}|}\}$. For estimates $\widehat{\mu}_n$ that are absolutely continuous w.r.t. $\mu_{\boldsymbol{X}}$, one important example of such a distance measure is the Kullback-Leibler (KL) divergence $\mathsf{d} = D_{\mathrm{KL}}$

$$\begin{aligned} D_{\mathrm{KL}}(\mu_{\boldsymbol{X}} \| \widehat{\mu}_n) &= \mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}\left[-\log\left(\frac{\mathrm{d}\widehat{\mu}_n}{\mathrm{d}\mu_{\boldsymbol{X}}}\right)\right] = -\int_{\mathcal{X}} \log\left(\frac{\mathrm{d}\widehat{\mu}_n}{\mathrm{d}\mu_{\boldsymbol{X}}}\right) \mathrm{d}\mu_{\boldsymbol{X}} \\ &= -\int_{\mathcal{X}} \log\left(\frac{\widehat{p}_n(\boldsymbol{x})}{p_{\boldsymbol{X}}(\boldsymbol{x})}\right) p_{\boldsymbol{X}}(\boldsymbol{x}) \, \mathrm{d}x. \end{aligned} \tag{2.6}$$

Here, $\frac{\mathrm{d}\widehat{\mu}_n}{\mathrm{d}\mu_{\boldsymbol{X}}}$ denotes the Radon-Nikodym derivative. In the case of the KL divergence, an approximation of $D_{\mathrm{KL}}(\mu_{\boldsymbol{X}} \| \widehat{\mu}_n)$ can be computed which we will re-visit in the context of empirical risk minimization in section 2.1.3.

[6] In the language of probably approximately correct learning (see section 2.1.2) this is captured by the notion of ε-representativity [166, Def. 4.1] of the sample $\mathcal{D}$. This convergence property plays a large role in empirical risk minimization (see section 2.1.3) where probably approximately correct learning is guaranteed by ε-representativity. The latter can then be reframed as a requirement for the hypothesis space $\mathscr{H}$ which must satisfy the probabilistic requirement of having the "uniform convergence" property [166, Def. 4.3].

2.1.1.3 Probabilistic Supervised Learning: Dealing with Aleatoric Uncertainty

In supervised learning, the existence of the labeling function f oftentimes is not given in practice. This is due to data $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$ being noisy in nature, which is naturally the case in regression problems, but also often the case in classification tasks. We then assume that the data point $z := (\boldsymbol{x}, y)$ was drawn from a probability distribution μ_Z over the space $\mathcal{X} \times \mathcal{Y}$. The fact that for fixed $\boldsymbol{x} \in \mathcal{X}$, the associated label y follows a conditional probability distribution $\mu_{|\boldsymbol{X}=\boldsymbol{x}} = Y_* \Pr|_{\boldsymbol{X}=\boldsymbol{x}} \in \mathscr{M}_1(\mathcal{Y})$ is called *aleatoric uncertainty*, i.e., a kind of uncertainty in the data which is inherent to the data generating distribution or process. Since both components of $(\boldsymbol{x}, y)$ are probabilistic in nature, it is useful to understand both of them in terms of random variables $Z = (\boldsymbol{X}, Y) : (\Omega, \mathscr{A}, \Pr) \to \mathcal{X} \times \mathcal{Y}$. Here, $\boldsymbol{X}$ follows the marginal distribution $\mu_{\boldsymbol{X}}$ of the initial μ_Z and Y follows the conditional distribution $\mu_{|\boldsymbol{X}=(\cdot)}$. Statements in statistical learning theory will be probabilistic over the choice of χ_n.

Given a sample $\chi_n = (Z_1, \dots, Z_n) \sim \mu_Z^n$, supervised learning can then be framed as unsupervised learning that aims to estimate the conditional distribution $\mu_{\boldsymbol{X}=(\cdot)}$. The latter is formally captured by the notion of a Markov kernel[7]. A predictor on a test sample $\boldsymbol{x} \in \mathcal{X}$ as in section 2.1.1.1 can be recovered from an estimation $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}$ by computing[8]

$$\widehat{f}(\boldsymbol{x}) := \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\ \widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}(\{y\}). \tag{2.7}$$

That is, the value $y \in \mathcal{Y}$ which maximizes the predictive probability distribution $\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}$ of $\boldsymbol{x}$ over $\mathcal{Y}$. Measuring the accuracy of the estimated $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}$ from the true $\mu_{\boldsymbol{X}=(\cdot)}$ can then be accomplished in the same way as in unsupervised learning by means of distance measures

$$\mathsf{D}(\cdot\|\cdot) : \mathscr{K}(\mathcal{X}; (\mathcal{Y}, \mathscr{A}_{\mathcal{Y}})) \times \mathscr{K}(\mathcal{X}; (\mathcal{Y}, \mathscr{A}_{\mathcal{Y}})) \to \mathbb{R}_+. \tag{2.8}$$

Distance measures for Markov kernels can also be generated from distance measures d on $\mathscr{M}_1(\mathcal{Y}, \mathscr{A}_{\mathcal{Y}})$ by integrating over $\mathcal{X}$. In particular, let $\mu_{|\boldsymbol{X}=(\cdot)}, \nu_{|\boldsymbol{X}=(\cdot)} \in \mathscr{K}(\mathcal{X}; (\mathcal{Y}, \mathscr{A}_{\mathcal{Y}}))$, then

$$\mathsf{D}_{\mathsf{d}}(\mu_{|\boldsymbol{X}=(\cdot)} \| \nu_{|\boldsymbol{X}=(\cdot)}) := \mathbb{E}_{\boldsymbol{\Xi} \sim \boldsymbol{X}_* \Pr}\left[\mathsf{d}\left(\mu_{|\boldsymbol{X}=\boldsymbol{\Xi}} \middle\| \nu_{|\boldsymbol{X}=\boldsymbol{\Xi}}\right)\right]. \tag{2.9}$$

In the following, we will often use the notation of unsupervised learning for brevity. Analogous definitions or notions can usually be given in the language of Markov kernels and integrated distance measures D_{d} by expansion of notation which we will omit. Statements on when supervised learning w.r.t. D or unsupervised learning w.r.t. d succeeds can be framed in the language of probably approximately correct (PAC) learning.

[7] Given a σ-algebra $\mathscr{A}_{\mathcal{Y}}$ over $\mathcal{Y}$, a Markov kernel K is a map $K_{(\cdot)} : (\mathcal{X}, \mathscr{B}_{\mathcal{X}}) \mapsto \mathscr{M}_1(\mathcal{Y}, \mathscr{A}_{\mathcal{Y}})$ such that $K_{(\cdot)}(A)$ is measurable for any $A \in \mathscr{A}_{\mathcal{Y}}$. For fixed $\boldsymbol{x} \in \mathcal{X}$, K_x is, therefore, a probability measure over $(\mathcal{Y}, \mathscr{A}_{\mathcal{Y}})$. Markov kernels build on the notion of conditional expectation, see e.g., [82, Chapter 8.3]. We denote the set of Markov kernels in this setting by $\mathscr{K}(\mathcal{X}; (\mathcal{Y}, \mathscr{A}_{\mathcal{Y}}))$.

[8] Note, that we assume uniqueness of $\operatorname{argmax}_{y \in \mathcal{Y}} \widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}(\{y\})$ here. While this is numerically often true, uniqueness is not guaranteed in general.

2.1.2 PAC Learning: Stochastic Guarantees on Learning

In practice, we do not know the true data-generating distribution μ, so knowing how close any estimated distribution $\widehat{\mu}_n$ is to μ is not possible. However, probabilistic statements about the deviation of $\widehat{\mu}_n$ from μ with respect to some distance measure d can be formulated in the language of PAC learning. Such statements relate the sample size n and the amount of error we allow with the kind of model we use for learning. If we understand the estimated model $\widehat{\mu}_n$ as depending on the sample χ_n of training data, we require $\widehat{\mu}_n$ to be a measurable map

$$\widehat{\mu}_n : (\mathcal{X}^n, \mathscr{B}_{\mathcal{X}^n}) \to (\mathscr{M}_1(\mathcal{X}), \mathscr{B}_{\mathsf{d}}) \tag{2.10}$$

where $\mathscr{B}_{\mathsf{d}}$ is the Borel-σ-algebra generated by the distance measure d on $\mathscr{M}_1(\mathcal{X})$. If there is such a map for any $n \in \mathbb{N}$, we call $\{\widehat{\mu}_n\}_{n\in\mathbb{N}}$ a *learning algorithm*[9] and $\mathscr{H}_n := \operatorname{Img}(\widehat{\mu}_n)$ the *hypothesis space* of the learning algorithm for sample size n. In case $\operatorname{Img}(\widehat{\mu}_n)$ is not dependent on n, we call $\mathscr{H} := \mathscr{H}_n$ simply the hypothesis space of the learning algorithm which is assumed in the following.

Given a sample $\chi_n = (\boldsymbol{X}_1, \dots, \boldsymbol{X}_n) \sim \mu^n_{\boldsymbol{X}}$, the composition $\widehat{\mu}_n \circ \chi_n$ can be regarded as a $\mathscr{M}_1(\mathcal{X})$-valued random variable for which probabilistic statements about learning can be formulated. Particularly, a measure $\mu_{\boldsymbol{X}} \in \mathscr{M}_1(\mathcal{X})$ is called d-learnable by $\widehat{\mu}_n$ for $\chi_n \sim \mu^n_{\boldsymbol{X}}$ if

$$\mathsf{d}\left(\mu_{\boldsymbol{X}} \,\|\widehat{\mu}_n \circ \chi_n\right) \to 0 \quad (n \to \infty) \tag{2.11}$$

in probability w.r.t. the fundamental probability measure Pr. Some subset $\mathscr{T} \subseteq \mathscr{M}_1(\mathcal{X})$ is called d-*learnable* by $\widehat{\mu}_n$ if each element $\mu \in \mathscr{T}$ is d-learnable. $\mathscr{T}$ is called d-*PAC-learnable* if there is $n = n(\varepsilon, \delta) \in \mathbb{N}$ for any $\varepsilon > 0$ and $\delta \in (0, 1)$ such that for all $\mu \in \mathscr{T}$

$$\Pr\left(\mathsf{d}\left(\mu_{\boldsymbol{X}} \,\|\widehat{\mu}_n \circ \chi_n\right) > \varepsilon\right) \leq \delta \qquad \forall n \geq n(\varepsilon, \delta). \tag{2.12}$$

This means that we fix ε, the precision which the learning algorithm $\widehat{\mu}_n$ is required to have with respect to the measure $\mu_{\boldsymbol{X}}$ and $1 - \delta$, the confidence which the probabilistic statement needs to hold with. Then we can give a minimum required number $n(\varepsilon, \delta)$ of sampled data points χ_n the learning algorithm must obtain.

In case a given measure $\mu_{\boldsymbol{X}} \notin \mathscr{H}$, there is a minimal model error made in (PAC-) learning due to mis-specification of the hypothesis class of learning models

$$\varepsilon_{\text{model}}(\mu_{\boldsymbol{X}}) := \inf_{\nu \in \mathscr{H}} \mathsf{d}(\mu_{\boldsymbol{X}} \| \nu). \tag{2.13}$$

The concepts of learnability and PAC learnability can be relaxed to so-called agnostic (PAC-) learnability which requires $\mathsf{d}(\mu_{\boldsymbol{X}} \| \widehat{\mu}_n \circ \chi_n) \to \varepsilon_{\text{model}}$ or

$$\Pr\left(\mathsf{d}\left(\mu_{\boldsymbol{X}} \,\|\widehat{\mu}_n \circ \chi_n\right) > \varepsilon_{\text{model}} + \varepsilon\right) \leq \delta \qquad \forall n \geq n(\varepsilon, \delta), \tag{2.14}$$

respectively. Since the quality of estimation of $\mu_{\boldsymbol{X}}$ depends explicitly on the distance measure d, learning can be understood as an optimization problem. In this optimization,

[9]We shall abbreviate notation of the family $\{\widehat{\mu}_n\}_{n\in\mathbb{N}}$ to simply $\widehat{\mu}_n$

we want to minimize $\mathsf{d}(\mu_{\boldsymbol{X}}\|\widehat{\mu}_n)$ with respect to $\widehat{\mu}_n$. However, since $\mu_{\boldsymbol{X}}$ is in practice unknown, we can only utilize the samples $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$ drawn for the learning algorithm. This can be accomplished by means of empirical risk functions and empirical risk minimization (ERM).

2.1.3 ERM Learning: Implementing Optimization Objectives

Empirical risk minimization allows for the definition of an optimization problem without knowing the true distribution $\mu_{\boldsymbol{X}}$ which generated the training sample χ_n. Let $\mathscr{H}$ be the hypothesis space of some learning algorithm $\widehat{\mu}_n$, an *empirical risk function* $R = \{R_n\}_{n\in\mathbb{N}}$ for a distance measure d over a target space $\mathscr{T}$ is a sequence of measurable functions $R_n : (\mathscr{H} \times \mathcal{X}^n, \mathscr{B}_{\mathsf{d}}|_{\mathscr{H}} \otimes \mathscr{B}_{\mathcal{X}^n}) \to (\mathbb{R}, \mathscr{B}(\mathbb{R}))$ such that there is a sequence $\{c_n\}_{n\in\mathbb{N}}$, $c_n > 0$ and a sequence of functions $\{h_n\}_{n\in\mathbb{N}}$ with $h_n : \mathscr{T} \times \mathcal{X}^n \to \mathbb{R}$ such that for all $\nu \in \mathscr{H}$ and all $\mu_X \in \mathscr{T}$

$$c_n \cdot R_n(\nu, \chi_n) + h_n(\mu_{\boldsymbol{X}}, \chi_n) \to \mathsf{d}(\mu_{\boldsymbol{X}}\|\nu), \qquad (n \to \infty) \tag{2.15}$$

in probability. This property allows for the definition of an optimization problem by separating the hypothesis ν from the unknown distribution $\mu_{\boldsymbol{X}}$. In particular, a learning algorithm is an *ERM-learner* if it returns

$$\widehat{\mu}_n(\chi_n) \in \underset{\nu\in\mathscr{H}}{\operatorname{argmin}}\ R_n(\nu, \chi_n). \tag{2.16}$$

Practically, this optimization task is still difficult to solve and only approximate solutions ν to local minima of the function $R_n(\nu, \chi_n)$ can be found. Oftentimes, this takes the shape of optimizing the parameters of a parametric model $\widehat{\mu}_n^{\boldsymbol{\theta}}$ (see section 2.2) where $\boldsymbol{\theta} \in \Theta$ over some parameter space Θ. A class of parametric models also defines a parametric hypothesis space in which minima can sometimes be determined analytically, such as in simple least squares regression. However, when it comes to deep learning, it is usually not feasible to find analytic minima, so one also has to resort to approximate solutions. Such an approximate solution can be accomplished, for instance, by means of stochastic gradient descent or related methods (see section 2.2.2.1).

2.1.3.1 Error Decomposition: Trade-Offs in Learning

Given an empirical risk function R for d over $\mathscr{T}$, the distance $\mathsf{d}(\mu_{\boldsymbol{X}}\|\widehat{\mu}_n)$ is bounded by the following *error decomposition*[10]:

$$\mathsf{d}(\mu_{\boldsymbol{X}}\|\widehat{\mu}_n) \leq \varepsilon_{\text{model}} + \varepsilon_{\text{learn},n} + 2 \cdot \varepsilon_{\text{sample},n}. \tag{2.17}$$

Here, the three terms have the following meaning

- The *model error* (or model mis-specification error) is the same as mentioned in section 2.1.2, i.e.,

$$\varepsilon_{\text{model}} = \inf_{\nu\in\mathscr{H}} \mathsf{d}(\mu_{\boldsymbol{X}}\|\nu). \tag{2.18}$$

[10]In statistics, this is also sometimes called oracle inequality.

This is the minimal error accomplishable by the learning algorithm which has $\mathscr{H}$ as its hypothesis space. In the case of deep learning, the question of the size of $\mathscr{H}$ is linked to the concept of the universal approximation property of certain classes of statistical models (see section 2.2.3).

- The *optimization error* is given by

$$\varepsilon_{\mathrm{learn},n} = c_n \left(R_n(\widehat{\mu}_n(\chi_n), \chi_n) - \inf_{\nu \in \mathscr{H}} R_n(\nu, \chi_n) \right). \tag{2.19}$$

  This term is due to approximation errors (by only finding local instead of global minima) in the learning algorithm like in stochastic gradient descent. It is the difference to the value that a true ERM $\nu \in \mathscr{H}$ will obtain. In principle, this term can be made arbitrarily small by performing some exhaustive search on $\mathscr{H}$ given some approximation accuracy of the ERM.
- The *statistical error* (or sampling error)

$$\varepsilon_{\mathrm{sample},n} = \sup_{\nu \in \mathscr{H}} |\mathsf{d}(\mu_{\boldsymbol{X}} \| \nu) - (c_n R_n(\nu, \chi_n) + h_n(\mu_{\boldsymbol{X}}))| \tag{2.20}$$

  describes the approximation accuracy of the empirical risk function to the true distance measure d. If $\mathscr{H}$ satisfies the uniform convergence property (see [166, Def. 4.3]) this error can be made arbitrarily small in probability by increasing the amount of training data (increasing n). When the empirical risk becomes smaller even though the term $\mathsf{d}(\mu_{\boldsymbol{X}} \| \widehat{\mu}_n) - (c_n R_n(\widehat{\mu}_n) + h_n(\mu_{\boldsymbol{X}}))$ increases, the model over-adapts to the sample of training data χ_n and starts diverging from $\mu_{\boldsymbol{X}}$. This phenomenon is called "overfitting" and tends to play a large role in many areas of machine learning and in particular deep learning.

When increasing the size of $\mathscr{H}$, i.e., the number of possible models or capacity of the model class to represent certain classes of functions, we also increase $\varepsilon_{\mathrm{sample,n}}$. This is because a larger amount of data is necessary to rule out some additional hypotheses. This demonstrates the interaction between $\varepsilon_{\mathrm{model}}$ and $\varepsilon_{\mathrm{sample},n}$ which is also sometimes called the bias-variance trade-off[11].

2.1.3.2 Maximum-Likelihood Estimation: Frequentist Risk Functions

Maximum likelihood estimation can be regarded as one special case of empirical risk minimization, namely when we choose the KL divergence as the distance measure. Given the sample $\chi_n = (\boldsymbol{X}_1, \dots, \boldsymbol{X}_n)$ drawn i.i.d. from $\mu_{\boldsymbol{X}}$ and given a measure ν over $\mathcal{X}$, the *likelihood* $\mathscr{L}(\chi_n|\nu)$ is given by

$$\mathscr{L}(\chi_n|\nu) = \prod_{j=1}^{n} \nu(\{\boldsymbol{X}_j\}) \tag{2.21}$$

[11]The initial choice of $\mathscr{H}$ involves assumptions on the distribution $\mu_{\boldsymbol{X}}$ we aim to learn. This assumption and the corresponding choice of $\mathscr{H}$ is called inductive bias. There is a set of mathematical statements and theorems which assert that there is no hypothesis space $\mathscr{H}$ by which arbitrary distributions $\mu_{\boldsymbol{X}}$ can be learned. These statements are oftentimes referred-to as No-Free-Lunch theorems.

in the case that ν is a discrete distribution. Respectively, if ν is continuous, and it has a Lebesgue density $\nu = \rho_\nu \cdot \mathrm{d}x$, we define the likelihood by

$$\mathscr{L}(\chi_n|\nu) := \prod_{j=1}^{n} \rho_\nu(\boldsymbol{X}_j). \tag{2.22}$$

By choosing the $\nu \in \mathscr{H}$ which maximizes the likelihood of observing the i.i.d. chosen $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$, we obtain some

$$\widehat{\mu}_n(\chi_n) \in \operatorname*{argmax}_{\nu \in \mathscr{H}} \mathscr{L}(\chi_n|\nu) \tag{2.23}$$

which is called a *maximum likelihood learner* or in the parametric case a *maximum likelihood estimator*.

Assuming integrability of $\log(\rho_\nu)$ in the continuous case, the negative log-likelihood given by $R_n(\nu, \chi_n) = -\log \mathscr{L}(\chi_n|\nu)$ is an unbiased empirical risk function for the KL divergence with $c_n := \frac{1}{n}$ and the entropy $h_n(\mu) := \mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}[\log(\mathscr{L}(\boldsymbol{X}|\mu_{\boldsymbol{X}}))]$. This fact is particularly interesting for the case of supervised learning where we are learning a model $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}$ and our sample consists of $(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)$. If $\mathcal{Y}$ is a discrete space as for a classification task over, say, C classes, the negative log-likelihood

$$-\log \mathscr{L}\left(\chi_n|\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}\right) = -\sum_{j=1}^{n} \log \widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{X}_j}(\{Y_j\}) = -\sum_{j=1}^{n}\sum_{c=1}^{C} \delta_{cY_j} \cdot \log\left(\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{X}_j}(\{c\})\right) \tag{2.24}$$

is also called *cross entropy* and is frequently used as a risk or loss function in deep learning tasks, see section 2.3.

In a regression task, if we similarly assume that our model learns normally distributed residuals, the distribution of residuals can be described by two predictive functions $\widehat{\varsigma}_n : \mathcal{X} \to \mathbb{R}_+$ and $\widehat{m}_n : \mathcal{X} \to \mathbb{R}$ which represent $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}$ by

$$\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}} = \frac{1}{\sqrt{2\pi\widehat{\varsigma}_n^2(\boldsymbol{x})}} \exp\left(-\frac{1}{2\widehat{\varsigma}_n^2(\boldsymbol{x})}(\widehat{m}_n(\boldsymbol{x}) - y)^2\right) \cdot \mathrm{d}y. \tag{2.25}$$

Then, the negative log-likelihood

$$\begin{aligned} -\log \mathscr{L}\left(\chi_n|\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}\right) &= -\sum_{j=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\widehat{\varsigma}_n^2(\boldsymbol{X}_j)}} \mathrm{e}^{-\frac{1}{2\widehat{\varsigma}_n^2(\boldsymbol{X}_j)}(\widehat{m}_n(\boldsymbol{X}_j)-Y_j)^2}\right) \\ &= \sum_{j=1}^{n} \log\left(\sqrt{2\pi\widehat{\varsigma}_n^2(\boldsymbol{X}_j)}\right) + \frac{1}{2\widehat{\varsigma}_n^2(\boldsymbol{X}_j)}(\widehat{m}_n(\boldsymbol{X}_j) - Y_j)^2 \end{aligned} \tag{2.26}$$

consists of two terms. If our model is non-probabilistic, i.e., the variance parameter $\widehat{\varsigma}_n$ is not estimated jointly with the mean parameter $\widehat{m}_n$, the first term does not affect the optimization goal posed by the negative log-likelihood. A model whose variance is not dependent on the input is also called homoscedastic. In contrast to models which return input-dependent uncertainty which are called heteroscedastic. Similarly, the pre-factor of the second term only involves $\widehat{\varsigma}_n$ and yields a constant re-scaling of the squared difference $(\widehat{m}_n(\boldsymbol{X}_j) - Y_j)^2$ which is known as the *mean squared error*, one of the standard choices of loss functions for regression tasks, see section 2.3.2.

2.2 Deep Learning and Deep Neural Networks

2.2.1 Neural Network Architectures: Building Blocks

In various areas of science and engineering, machine learning algorithms, particularly deep neural networks, define the current state of the art when it comes to performance. While they have countless areas of applications, here we focus only on aspects that are relevant to image recognition and computer vision tasks. We, therefore, focus on feed-forward neural networks involving fully connected (section 2.2.1.1), convolutional (section 2.2.1.2) and self-attention layers (section 2.2.1.3) which have become standard components in deep learning. Parts of this section and some notation is inspired by [125].

2.2.1.1 Perceptrons and Fully Connected Layers: Origins and Basic Concepts

Multilayer Perceptron. Artifical neural networks (in the following just neural networks for brevity) are parametric statistical models taking on a schematic form loosely inspired by biological neural networks [117, 150]. Typically, the functional form is visualized by a graph structure arranged in layers which resembles biological neural connections. One layer maps inputs or "*activations*" $\boldsymbol{x} = (x_1, \dots, x_{d_0}) \in \mathbb{R}^{d_0}$ to the pre-activation[12]

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{b} \in \mathbb{R}^{d_1}, \qquad y_k = \sum_{j=1}^{d_1} w_{kj} x_j + b_k, \quad k \in [d_1]. \tag{2.27}$$

Here, $W \in \mathbb{R}^{d_1 \times d_0}$ and $\boldsymbol{b} \in \mathbb{R}^{d_1}$ are parameters of the layer. This basic affine map is the transformation taking place in a so-called *fully connected* layer, since the relation of input $\boldsymbol{x}$ and output $\boldsymbol{y}$ can be visualized by a graph in which each in-going node x_j is connected to each out-going node y_k by an edge carrying the weight w_{jk}, see fig. 2.1 on the left. A multilayer perceptron is a sequence of $L \in \mathbb{N}$ fully connected layers, however, since the composition of affine linear maps stays affine linear, the expressivity of such a model is highly limited. Therefore, non-linearities $\alpha : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1}$ are introduced, called activation functions. Activation functions are typically application of some function $\mathbb{R} \to \mathbb{R}$ to each element of y. The entire feed-forward layer $f_\ell = f_\ell(\cdot|\boldsymbol{\theta}_\ell, \alpha_\ell)$ then has learnable parameters $\boldsymbol{\theta}_\ell = (W_\ell, \boldsymbol{b}_\ell)$ and an activation function α_ℓ and maps

$$\mathbb{R}^{d_{\ell-1}} \ni \boldsymbol{x} \mapsto f_\ell(\boldsymbol{x}|\boldsymbol{\theta}_\ell, \alpha_\ell) = \alpha_\ell(W_\ell \boldsymbol{x} + \boldsymbol{b}_\ell) \in \mathbb{R}^{d_\ell} \tag{2.28}$$

for all $\ell \in [L]$. The multilayer perceptron function $\mathrm{MLP} : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$ is a composition

$$\mathrm{MLP}(\boldsymbol{x}|\boldsymbol{\theta}) = f_L \circ f_{L-1} \circ \cdots \circ f_2 \circ f_1(\boldsymbol{x}) \tag{2.29}$$

of feed-forward layers where we have suppressed the dependence of f_ℓ on $\boldsymbol{\theta}_\ell$ and α_ℓ. The multilayer perceptron has learnable parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_L)$. Moreover, the target space dimension of each so-called *intermediate or hidden* layer $\ell = 2, \dots, L-1$ depends on the parameter dimensionality and can be adjusted to each new problem alongside the

[12]Throughout this thesis we will use the notation $[n] := \{1, \dots, n\}$.

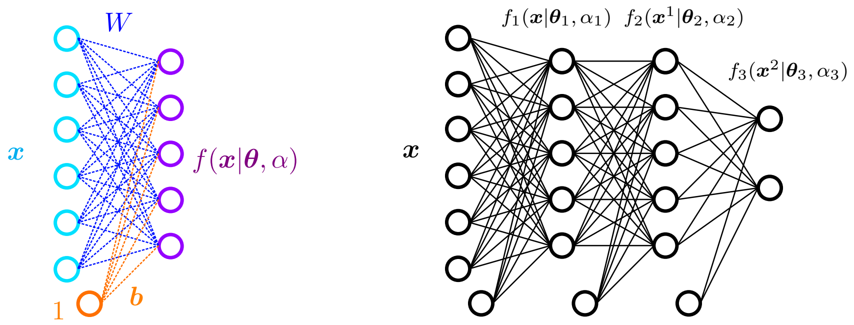

W
$\boldsymbol{x}$
$f(\boldsymbol{x}|\boldsymbol{\theta},\alpha)$
1
$\boldsymbol{b}$
$f_1(\boldsymbol{x}|\boldsymbol{\theta}_1,\alpha_1)$
$f_2(\boldsymbol{x}^1|\boldsymbol{\theta}_2,\alpha_2)$
$f_3(\boldsymbol{x}^2|\boldsymbol{\theta}_3,\alpha_3)$
$\boldsymbol{x}$

Figure 2.1: *Left*: Schematic illustration of a fully connected layer mapping an input (light blue) $\mathbb{R}^d \ni \boldsymbol{x} \mapsto f(\boldsymbol{x}|\boldsymbol{\theta},\alpha) \in \mathbb{R}^n$ to the layer output (purple) involving a matrix $W \in \mathbb{R}^{n\times d}$ of weights along the blue edges and a vector $\boldsymbol{b} \in \mathbb{R}^d$ along the orange edges. Here, $d = 6$ and $n = 5$. Each input component x_i, $i \in [d]$ can be visualized as a node in a graph, and so can the n components of $f(\boldsymbol{x}|\boldsymbol{\theta},\alpha)$. The bias variables are visualized as edges connecting to a bias node carrying the constant activation 1. *Right*: Schematic illustration of the architecture of an MLP with two hidden layers of width 5 and two output neurons.

number of layers. See fig. 2.1 to the right for a graphical depiction of an MLP with two hidden layers and $d_0 = 6$, $d_1 = 5 = d_2$, $d_3 = 2$ and $L = 3$. We denote $f_\ell(\boldsymbol{x}^{\ell-1}|\boldsymbol{\theta}_\ell,\alpha_\ell) =: \boldsymbol{x}^\ell$ and $\boldsymbol{x}^0 := \boldsymbol{x}$ as the input. The spaces in which activations of hidden layers live are called latent spaces and the corresponding activation of some fixed input $\boldsymbol{x}$ is called a latent representation of $\boldsymbol{x}$.

Activation Functions. There is a set of popular choices for activation functions which are frequently considered and implemented in neural networks. Of typical interest are functions that are particularly simple or functions which have desirable properties.

- *Identity*: The identity mapping $x \mapsto x$ is the simplest possible activation and leads to an affine linear model. Depending on the concrete problem at hand, the utilization of this activation boils down to logistic or linear regression. For its lack of expressivity, this activation function is usually not used in deep learning.
- *Heaviside or step function*: The Heaviside function

$$\Theta : \mathbb{R} \to \mathbb{R}, \quad x \mapsto \Theta(x) = \chi_{\mathbb{R}_+}(x) \tag{2.30}$$

  with the indicator function χ was one of the early activation functions considered and solved the so-called XOR problem wherein the logical XOR operation was originally not representable with one-layer perceptrons. The Heaviside function is not differentiable at zero and yields zero derivative everywhere else which makes it unpractical for gradient-based parameter optimization, such as SGD.
- *Sigmoid or logistic function*: The sigmoid or logistic function

$$\sigma : \mathbb{R} \to \mathbb{R}, \quad x \mapsto \sigma(x) = \frac{1}{1+\mathrm{e}^{-x}} \tag{2.31}$$

  constrains activations to $(0,1)$ and has a linear regime around 0. While being everywhere differentiable, the saturation property of the sigmoid activation function lead to the so-called vanishing gradient problem in the saturation regime where the derivatives are close to zero. Although this is no hard obstruction to the use of gradient optimization methods, this fact can lead to practical difficulties in the fitting procedure. The same rationale applies to the *hyperbolic tangent* activation function

$$\tanh : \mathbb{R} \to \mathbb{R}, \quad x \mapsto \tanh(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}} \tag{2.32}$$

which shows a similar saturation behavior as the sigmoid function. However, it constrains activations to $(-1, 1)$. Note, that in contrast to the previous activations or the following ReLU activation function, here the evaluation of exponential functions is required. This leads in increased computational cost, e.g., by evaluating a truncated series.

- *Rectified Linear Unity* (ReLU) function: The ReLU function

$$\mathrm{ReLU} : \mathbb{R} \to \mathbb{R}, \quad x \mapsto \mathrm{ReLU}(x) = \max\{0, x\} \tag{2.33}$$

  is a simple activation function which disposes with the vanishing gradient problem by having a non-saturating positive regime while still having non-zero gradients for the positive half axis. Neural networks with ReLU activations have a universal approximation property (see section 2.2.3.1) which makes them suitable candidates for neural network activation functions. However, whenever considering second derivatives, the same argument applies to ReLU which already applied to the Heaviside function. ReLU has no meaningful second derivative and, therefore, does not give rise to a meaningful concept of curvature.
- *Rectified Quadratic Unity* (ReQU) function: The ReQU activation function

$$\mathrm{ReQU} : \mathbb{R} \to \mathbb{R}, \quad x \mapsto \mathrm{ReQU}(x) = (\max\{0, x\})^2 \tag{2.34}$$

  is an adaptation of the ReLU activation which preserves universal approximation while allowing for curvature in the approximated function.
- *Gaussian Error Linear Unity* (GELU) function: The GELU activation function

$$\mathrm{GELU} : \mathbb{R} \to \mathbb{R}, \quad x \mapsto \mathrm{GELU}(x) = x \cdot \Phi(x), \tag{2.35}$$

  where Φ is the cumulative distribution function of the standard normal distribution is a smooth activation function which is frequently used in vision transformer models (see section 2.2.1.3).

Another activation which is usually used to convert non-normalized feature activations $\boldsymbol{y} = (y_1, \ldots, y_d) \in \mathbb{R}^d$ to a probability distribution is the so-called *softmax function* $\boldsymbol{\Sigma} : \mathbb{R}^d \to (0,1)^d$ defined by

$$\Sigma_j(\boldsymbol{y}) := \frac{\mathrm{e}^{y_j}}{\sum_{i\in[d]} \mathrm{e}^{y_j}}. \tag{2.36}$$

The softmax function clearly returns a vector which encodes a probability distribution[13].

[13]This means that $\sum_{j=1}^{d} \Sigma_j(\boldsymbol{y}) = 1$ and all entries are non-negative. Here, in particular, all entries are strictly positive.

Apart from fully connected layers in feed-forward networks, additional structures have emerged from deep learning applications in computer vision and in natural language processing. In some sense, these represent constraints or special cases of fully connected layers which have their merits in specific applications.

Normalization Layers. During training and testing it is desirable for deep neural networks to control the distribution of activations. Otherwise, extreme depth of neural networks can lead to instable training and either vanishing or diverging gradients. Common approaches to control the distribution is to insert normalization layers in the network which adds parameters to the model. These parameters are responsible for ensuring that the resulting distribution will be standardized, having zero mean and unit standard deviation. We will see that this can also be understood as a constrained fully connected layer.

Batch normalization (BN [73]) is oftentimes utilized in discriminative neural networks, e.g., in classification. The batched activations $y \in \mathbb{R}^{B \times n}$ in some layer serves as the input of the BN layer which introduces new parameters $\gamma, \beta \in \mathbb{R}^n$. The forward pass through the BN layer takes on the form

$$\tilde{y} = \mathsf{BN}(y|\gamma, \beta) = \frac{y - \mu_y}{\sqrt{\sigma_y^2 + \varepsilon}} \odot \gamma + \beta. \tag{2.37}$$

Here, $\odot$ denotes entry-wise (Hadamard) multiplication, ε is some small regularization parameter (e.g., 10^{-5}) and

$$\mu_y = \frac{1}{B} \sum_{i=1}^{B} y_i \in \mathbb{R}^n, \qquad \sigma_y^2 = \frac{1}{B} \sum_{i=1}^{B} (y_i - \mu_y)^2 \in \mathbb{R}^n \tag{2.38}$$

are the expectation and (biased) variance estimators of the features across the batch $[B]$. In case the batch is already standardized, we would obtain (up to the regularization parameter ε)

$$\gamma = \sigma_y, \qquad \beta = \mu_y. \tag{2.39}$$

In the case of convolutional neural networks (see section 2.2.1.2), the parameters γ and β are applied channel-wise over the entire feature map and carry as many parameters as there are channels in the input y. Since (2.37) defines an affine linear transformation of y, it can be represented as a fully connected layer under constraints.

Batch normalization works well in many applications and seems to smoothen the loss landscape allowing for more stable optimization [158]. However, parameter estimation in BN runs into complications when the utilized batches are small. Whenever this is the case, other normalization techniques may be utilized, e.g., in the case of multidimensional inputs such as feature maps in computer vision. Then, a batched input $\psi \in \mathbb{R}^{B \times C \times H_\psi \times W_\psi}$ has a channel (C), height (H_ψ) and width (W_ψ) dimension in addition to the batch dimension (B) which may be small. The *layer norm* (LN, [2]) does not standardize along the batch dimension. Instead, across all other dimensions are standardized, i.e., LN introduces parameters $\gamma, \beta \in \mathbb{R}^B$ which are applied by

$$\tilde{\psi} = \mathsf{LN}(\psi|\gamma,\beta) = \frac{\psi - \mu_\psi^{\mathsf{LN}}}{\sqrt{\left(\sigma_\psi^{\mathsf{LN}}\right)^2 + \varepsilon}} \cdot \gamma + \beta \tag{2.40}$$

where

$$\begin{aligned} \mu_\psi^{\mathsf{LN}} &= \frac{1}{CH_\psi W_\psi} \sum_{c=0}^{C-1} \sum_{i=0}^{H_\psi - 1} \sum_{j=0}^{W_\psi - 1} \psi_{ij}^c \in \mathbb{R}^B, \\ \left(\sigma_\psi^{\mathsf{LN}}\right)^2 &= \frac{1}{CH_\psi W_\psi} \sum_{c=0}^{C-1} \sum_{i=0}^{H_\psi - 1} \sum_{j=0}^{W_\psi - 1} \left(\psi_{ij}^c - \mu_\psi^{\mathsf{LN}}\right)^2 \in \mathbb{R}^B. \end{aligned} \tag{2.41}$$

Addition and Multiplication in (2.40) is applied along the batch dimension and constant across other dimensions of ψ. In principle, normalization can also be applied only for certain sub-dimensions of ψ, e.g., standardizing across width and height but not the channel dimension leading to parameters $\gamma, \beta \in \mathbb{R}^{B \times C}$, so-called *instance normalization* [178].

Dropout Layers. In early investigations into neural networks, Srivastava et al. [169] found that the activations which certain neurons receive leads to a fixation of the entire model. Only a small amount of neuronal passages are used in the architecture. Meanwhile, other neurons barely play a role in the computations. It was found that this is connected to overfitting phenomena. There are a couple of methods which can be used in order to regularize this kind of behavior such that training becomes more stable and less overfitting occurs. One such method is the inclusion of so-called *dropout layers* which non-deterministically “turn off” neurons by setting their weights to zero during training. Thereby, a training sample which gives a large activation to one certain neuron will with a certain probability $p_{\text{drop}} \in (0,1)$ not be able to do so which leads to a “spreading” of the training signal during backpropagation (see section 2.2.2.2). Alternatively, the same regularizing effect can be accomplished by not masking the activation but instead masking columns of the weight matrix W_ℓ of a fully connected layer. Therefore, dropout layers can also be regarded as a special case of a fully connected layer. Already in [169] it was proposed to utilize dropout during the forward pass to obtain stochastic sampling of weights, an idea which was later refined and called Monte-Carlo dropout, see section 2.2.4.2.

2.2.1.2 Convolutions: Parameter-Efficient Pattern Recognition

Recognition tasks in sequential, one-dimensional data or in computer vision (e.g., on RGB camera images which is data in $\mathbb{R}^{3 \times W \times H}$) are oftentimes invariant with respect to the concrete pixel position. For example, the optical features of a cat will be the same whether it is represented on the left half of an RGB image or on the right half. This invariance property can be exploited in order to reduce the number of necessary parameters in a neural network, as well as to specialize the functional form of layers to the task at hand. This exploit resulted in the development of convolutional neural networks which incorporate the discrete convolution operation in their forward pass.

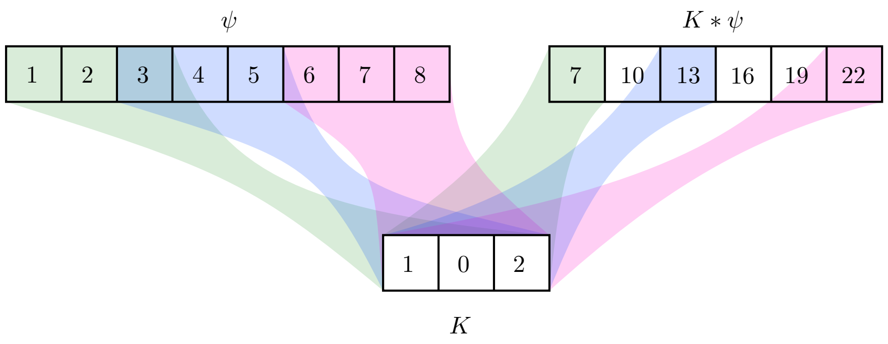


Figure 2.2: Illustration of the one-dimensional cross-correlation operation of a filter K on a sequence ψ.

Convolution and Cross Correlation. The convolution operation between two functions $\phi, \psi : \mathbb{R}^d \to \mathbb{R}$ is defined as the function[14]

$$[\phi \star \psi](\boldsymbol{x}) = \int_{\mathbb{R}^d} \phi(\boldsymbol{y})\psi(\boldsymbol{x} - \boldsymbol{y}) \,\mathrm{d}y. \tag{2.42}$$

The discretization of this operation has found heavy use in signal processing in the past where the functions can be represented by finite-dimensional vectors

$$\phi = (\phi_0, \dots, \phi_{n-1}) \in \mathbb{R}^n, \quad \psi = (\psi_0, \dots, \psi_{m-1}) \in \mathbb{R}^m. \tag{2.43}$$

In principle, infinitely many function evaluations are possible leading to the notion of convolution between sequences. However, the convolutions in deep learning practice are always finite. The integral then becomes the finite sum

$$[\phi \star \psi]_i = \sum_{k=0}^{n-1} \phi_k \psi_{i-k}, \qquad i \in [n-1 : m-n] \tag{2.44}$$

representing the components of the vector $[\phi \star \psi] \in \mathbb{R}^{m-2n+1}$. In deep learning, ψ is considered the *input signal to the convolutional layer* while ϕ contains the learnable weights denoted by K in the following. K is also called the kernel or filter of the convolution in reference to the integral kernel above. Implementations of convolutional layers oftentimes utilize the equivalent *cross-correlation operation* (see fig. 2.2) instead of the convolution:

$$[K * \psi]_i = \sum_{k=0}^{n-1} K_k \psi_{i+k} = \sum_{j=1-n}^{0} K_{-j} \psi_{i-j}, \qquad i \in [0 : m-n+1]. \tag{2.45}$$

This is equivalent to the convolution operation for deep learning purposes since it is convolution with a flipped kernel up to re-indexing. Cross-correlation is more convenient with respect to the index arithmetic than the convolution. For simplicity, we shall write convolution instead of cross-correlation in the following whenever this equivalence holds.

[14]Here, we assume convergence of the integral in some suitable sense.

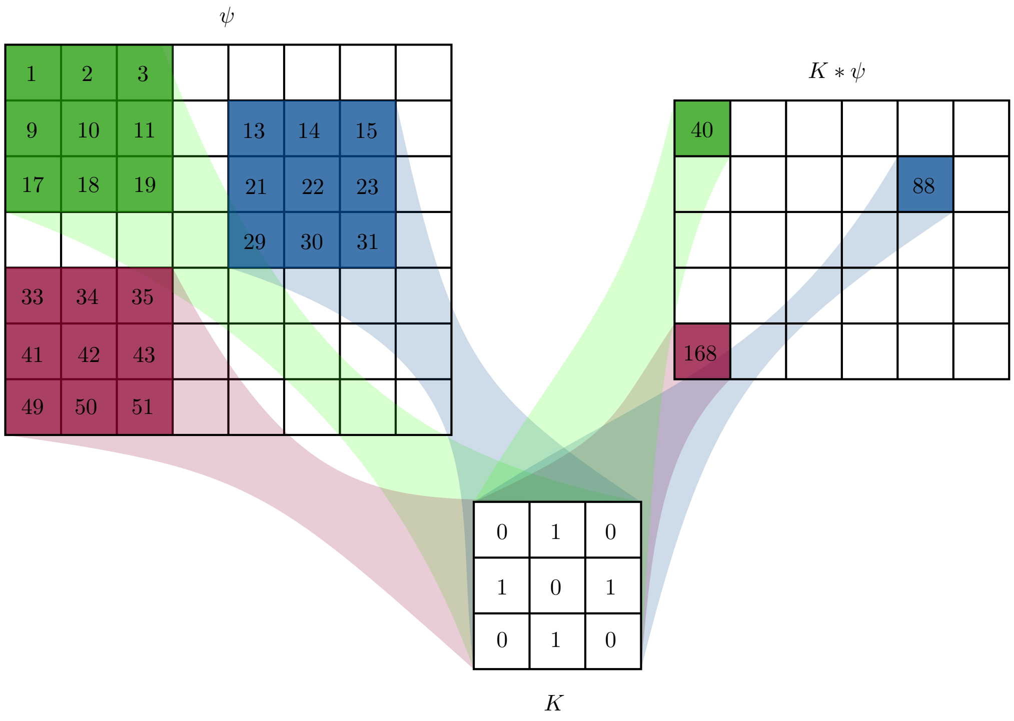


Figure 2.3: Illustration of the two-dimensional cross-correlation of a filter K over a two-dimensional feature map ψ.

Two-Dimensional Convolutions. The straight-forward two-dimensional extension of the one-dimensional convolution uses a kernel $K \in \mathbb{R}^{H_K \times W_K}$ and an activation signal $\psi \in \mathbb{R}^{H_\psi \times W_\psi}$ both of which are two-dimensional, see fig. 2.3. The resulting so-called *feature map* is of the form

$$[K * \psi]_{ij} = \sum_{k=0}^{H_K-1} \sum_{l=0}^{W_K-1} K_{kl} \psi_{i+k,j+l}, \tag{2.46}$$

where[15] $i \in [0 : H_\psi - H_K + 1]$ and $j \in [0 : W_\psi - W_K + 1]$. Since the entries of this map are linear in the input ψ, convolution with K can be represented as matrix multiplication. The matrix representation acts on the flattened vector $\underline{\boldsymbol{\psi}} \in \mathbb{R}^{H_\psi \cdot W_\psi}$ of the form

$$\underline{\boldsymbol{\psi}} = \begin{bmatrix} \psi_{:,0} \\ \psi_{:,1} \\ \vdots \\ \psi_{:,W_\psi - 1} \end{bmatrix} \tag{2.47}$$

corresponding to the two-dimensional ψ under the correspondence[16] $\underline{\boldsymbol{\psi}}_{i \cdot H_\psi + j} = \psi_{i,j}$ for $i \in [0 : H_\psi - 1]$ and $j \in [0 : W_\psi - 1]$. Here, $\psi_{:i} = [\psi_{1,i}, \ldots, \psi_{W_\psi - 1,i}]^\top \in \mathbb{R}^{H_\psi}$ denotes the

[15]Here, we employ the notation $[n : m] := \{n, n+1, \ldots, m-1, m\}$.

[16]This correspondence is ambiguous, as components of ψ could also first be gathered row-wise and afterwards column-wise. We will keep with the "flattening rule" defined here.

i-th column of ψ. We denote the matrix representation of the convolution with the kernel K by $\mathrm{Mat}[K] \in \mathbb{R}^{(H_\psi - H_K+1)(W_\psi - W_K+1)\times H_\psi W_\psi}$, that is

$$(\mathrm{Mat}[K] \cdot \underline{\psi})_{i\cdot(H_\psi - H_K+1)+j} = [K * \psi]_{ij} \tag{2.48}$$

with $i \in [0 : H_\psi - H_K + 1]$ and $j \in [0 : W_\psi - W_K + 1]$. This leads to the representation

$$\mathrm{Mat}[K] = \begin{bmatrix} \mathfrak{C}(K_{0,:}) & \cdots & \mathfrak{C}(K_{H_K-1,:}) & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \mathfrak{C}(K_{0,:}) & \cdots & \mathfrak{C}(K_{H_K-1,:}) & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \mathfrak{C}(K_{0,:}) & \cdots & \mathfrak{C}(K_{H_K-1,:}) & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & \mathfrak{C}(K_{0,:}) & \cdots & \mathfrak{C}(K_{H_K-1,:}) \end{bmatrix} \tag{2.49}$$

where we have used the one-dimensional convolution representation

$$\mathfrak{C}(K_{j,:}) = \begin{bmatrix} K_{:,j}^\top & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & K_{:,j}^\top & 0 & \cdots & \cdots & 0 \\ 0 & 0 & K_{:,j}^\top & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & K_{:,j}^\top & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & K_{:,j}^\top \end{bmatrix} \in \mathbb{R}^{(H_\psi - H_K+1)\times H_\psi} \tag{2.50}$$

of the j-th column of K over a vector of length H_ψ (i.e., the column length of ψ). Note, that in the entry (i, j) of the convolution $[K * \psi]$, the $H_K \cdot W_K$ filter weights only couple to as many (i.e., $H_K \cdot W_K$) values of the feature map ψ and, therefore, $\underline{\psi}$. The "fully connected" matrix representation $\mathrm{Mat}[K]$ is, therefore, a sparse matrix since all other couplings per row are zero. In this way, convolutions can be regarded as a constrained fully connected layer drastically reducing the number of parameters used.

Moreover, we have $\mathrm{Mat}[K] \in \mathbb{R}^{(H_\psi - H_K+1)(W_\psi - W_K+1)\times H_\psi W_\psi}$ which is a reflection of the fact, that convolutions typically reduce the size of the feature map. This can be circumvented by introducing an artificial padding with zeros around the input boundary, so-called *zero-padding*. There are also other ways of padding the input boundary which are linked to other boundary conditions for the convolution. When adding H_{pad} rows of zero padding symmetrically to both top and bottom and W_{pad} left and right we obtain an input feature map $\mathrm{pad}[\psi; H_{\text{pad}}, W_{\text{pad}}] \in \mathbb{R}^{(H_\psi + 2H_{\text{pad}})\times(W_\psi + 2W_{\text{pad}})}$. We obtain an output of the same shape as the input by setting $2H_{\text{pad}} = H_K - 1$ and $2W_{\text{pad}} = W_K - 1$ which is also called "same-padding".

Strided Two-Dimensional Convolutions. For each fixed output pixel $[K * \psi]_{ij}$ we call the patch[17] $\{\psi_{kl}\}_{k\in[i:i+H_K]}^{l\in[j:j+W_K]}$, which K couples to, the *receptive field* of K at the position (i, j). Neighboring output pixels can have a large overlap in input space and might, therefore, be similar in value introducing redundancy in the result. This redundancy can be reduced and computation accelerated by introducing a skip in the receptive field

[17] We denote $[a : b] := \{a, a + 1, \ldots, b\}$.

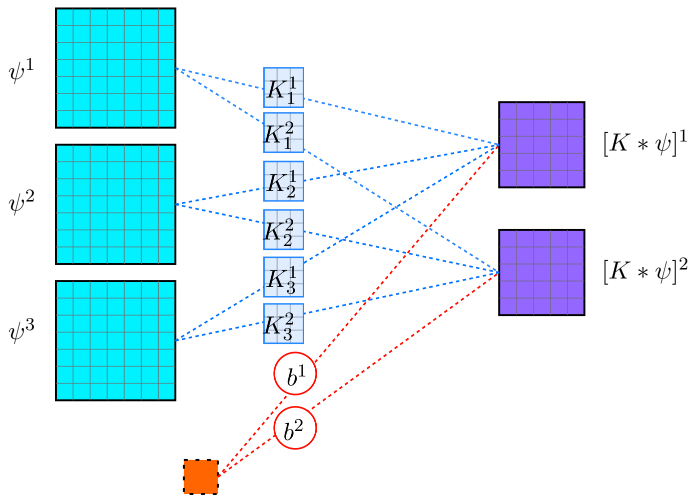


Figure 2.4: Illustration of the multichannel convolution layer mapping $C = 3$ in-going feature maps (ψ^1, ψ^2, ψ^3) to $D = 2$ out-going feature maps $([K * \psi]^1, [K * \psi]^2)$. The 6 filters have dimensions $H_K = W_K = 3$ is this case. In reminiscence of fully connected layers, we indicate a bias neuron (orange) adding output-wise biases that are constant over the spatial extent of each out-going feature map.

between neighboring outputs. This is called a *strided convolution* where the strides H_{str} and W_{str} determine how many input pixels the kernel is moved over in vertical (H_{str}) and horizontal (W_{str}) direction. A convolution with $H_{\text{str}} > 1$ and/or $W_{\text{str}} > 1$ can be regarded as a learned form of pooling since it will reduce the output size by a fixed factor and performs as weighted linear transformation to each input patch. The resulting feature map size after a strided convolution which also involves padding is

$$\left\lfloor \frac{H_\psi + 2H_{\text{pad}} - H_K + H_{\text{str}}}{H_{\text{str}}} \right\rfloor \times \left\lfloor \frac{W_\psi + 2W_{\text{pad}} - W_K + W_{\text{str}}}{W_{\text{str}}} \right\rfloor . \tag{2.51}$$

Multi-Channel Convolutions. The above description of the convolution map applies to single-channel, i.e., gray scale inputs $\psi \in \mathbb{R}^{H_\psi \times W_\psi}$. In many tasks of computer vision, the input image is represented in terms of three color channels corresponding to RGB values in which case ψ lives in $\mathbb{R}^{3 \times H_\psi \times W_\psi}$. Drawing from the neural network structure again, the convolution map can be generalized to inputs having C channels and D output channels. In that case, the result can be given with regard to a third channel index

$$[\text{Conv}_{K,\boldsymbol{b}}(\psi)]_{ij}^d = [K * \psi]_{ij}^d + b^d = \sum_{c=0}^{C-1} \left(\sum_{k=0}^{H_K-1} \sum_{l=0}^{W_K-1} (K_c^d)_{kl} \psi_{i+k,j+l}^c \right) + b^d, \tag{2.52}$$

where $d \in [0 : D-1]$, so we obtain the convolution result as the sum of single-channel convolutions. This utilizes one filter per combination of input and output channel. Per output, we add a bias variable b^d which is constant across the spatial extent of the output feature map $[K * \psi]^d$. The introduction of multiple output channels is beneficial to the goal of detecting multiple kinds of features in the input and also introduces additional parameters in the kernel $K \in \mathbb{R}^{C\times D\times H_K\times W_K}$ to the model leading to increased capacity.

Most convolutional layers in neural networks for computer vision tasks are multichannel. One special case which is especially important in semantic segmentation (see section 2.3.3) are (1×1)-convolutions, i.e., convolutions against $K \in \mathbb{R}^{C\times D\times 1\times 1}$. From eq. (2.52) we see that this reduces to a weighted combination of features at each pixel while changing the feature map dimension from C to D. Formally, this is equivalent to the application of a single layer MLP to the features of each pixel.

In addition to strided convolutions, the dimensionality of feature maps can be reduced (and information thereby compressed) by aggregating feature map values over small rectangular patches within channels. This operation is called *pooling*, where the entries within patches of each feature map are mapped to one single scalar value in the respective position in the output feature map. Usually the applied maps are either taking the maximum value over each patch (*max pooling*) or averaging the obtained values (*average pooling*). In classification or extraction of latent variable representations, one often utilizes so-called global average pooling where the average pooling patch extends to the entire feature map. This maps $\mathbb{R}^{D\times H_\psi\times W_\psi} \to \mathbb{R}^{D\times 1\times 1}$ which can then either be fed into an MLP for image classification or used for determining lower-dimensional distances between samples. The latter is often infeasible for the full dimension $D \times H_\psi \times W_\psi$.

ResNet Blocks and Skip Connections. Modern convolutional or transformer (see section 2.2.1.3) neural networks utilize so-called *skip connections* around blocks $F^\ell : \mathbb{R}^d \to \mathbb{R}^d$ within the network [61]. Such a block F^ℓ may be a combination of weight layers like convolutions, normalization layers or activations. The skip connection then adds the input $\boldsymbol{x}^\ell \in \mathbb{R}^d$ to the output value of F_ℓ, i.e.,

$$\mathrm{Res}[F_\ell](\boldsymbol{x}^\ell) := \boldsymbol{x}^\ell + F_\ell(\boldsymbol{x}^\ell). \tag{2.53}$$

The motivation for this kind of structure is that the parameters within F_ℓ need to only be used to learn the residuals between a desired output y^ℓ and the respective input $\boldsymbol{x}^\ell$. This is a simpler task, given that the neural network is sufficiently deep. The standard *ResNet block* has the specific structure involving two convolutional layers given in fig. 2.5. ResNet blocks in particular are often part of backbone architectures for computer vision tasks, see sections 2.3.1.3, 2.3.2.3 and 2.3.3.3.

2.2.1.3 Transformer Neural Networks: Attention as a Deep Learning Concept

Originally designed in the context of natural language processing [182], *transformer models* have since reached outstanding performance also in computer vision tasks [108]. In this section, we describe the functional principles of transformer layers and will narrow down to vision transformers in particular, which play an important role especially in computer vision backbones. The background of natural language processing has lead to the introduction of the term “token” which are originally encodings of text fragments which a given text is divided into by some specific “tokenization rule”. Such tokens undergo an

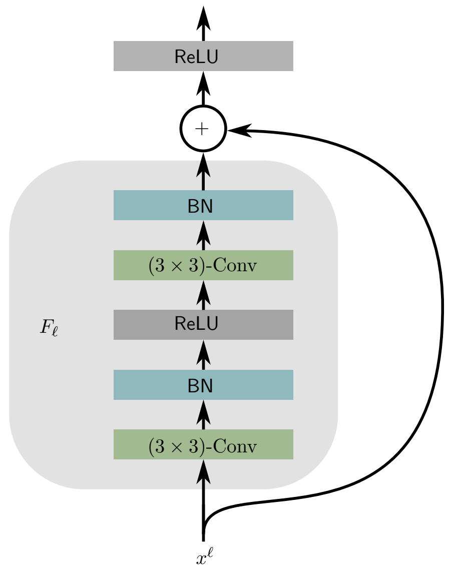


Figure 2.5: Convolutional ResNet block with skip connection as initially introduced by He et al. [61].

embedding into some t-dimensional space. Given embeddings $\mathsf{X} \in \mathbb{R}^{t\times n}$ of n tokens, the so-called *attention mechanism* then learns a $(t \times n)$-dimensional output by weighting different learned outputs by a similarity measure between the embeddings of different tokens. This section draws from the notation of [125] and [211].

Attention Mechanism. Learned attention between t-dimensional embeddings

$$\mathsf{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n) \in \mathbb{R}^{t\times n} \tag{2.54}$$

of n tokens is based on three weight matrices $W_V \in \mathbb{R}^{d\times t}$, $W_Q \in \mathbb{R}^{d_q\times t}$ and $W_K \in \mathbb{R}^{d_q\times t}$. Hereby, convex combinations of the value output

$$V := W_V\mathsf{X} = (W_V\boldsymbol{x}_1, \dots, W_V\boldsymbol{x}_n) \in \mathbb{R}^{d\times n} \tag{2.55}$$

are generated from the attention between different tokens which is defined by some similarity measure between the query ($W_Q\mathsf{X} \in \mathbb{R}^{d_q\times n}$) and key ($W_K\mathsf{X} \in \mathbb{R}^{d_q\times n}$) values of the tokens. The latent dimension $d_q \in \mathbb{N}$ in which query and key live can be viewed as a hyperparameter of the attention mechanism. One computationally efficient way of computing similarity measures is by computing the mutual scalar products between the query and key representations. Frequently, in order to stabilize training, the scalar products are normalized through division by the square root of the latent dimension d_q. In order to obtain normalized weights, the softmax function $\boldsymbol{\Sigma}$ (see eq. (2.36)) may be applied

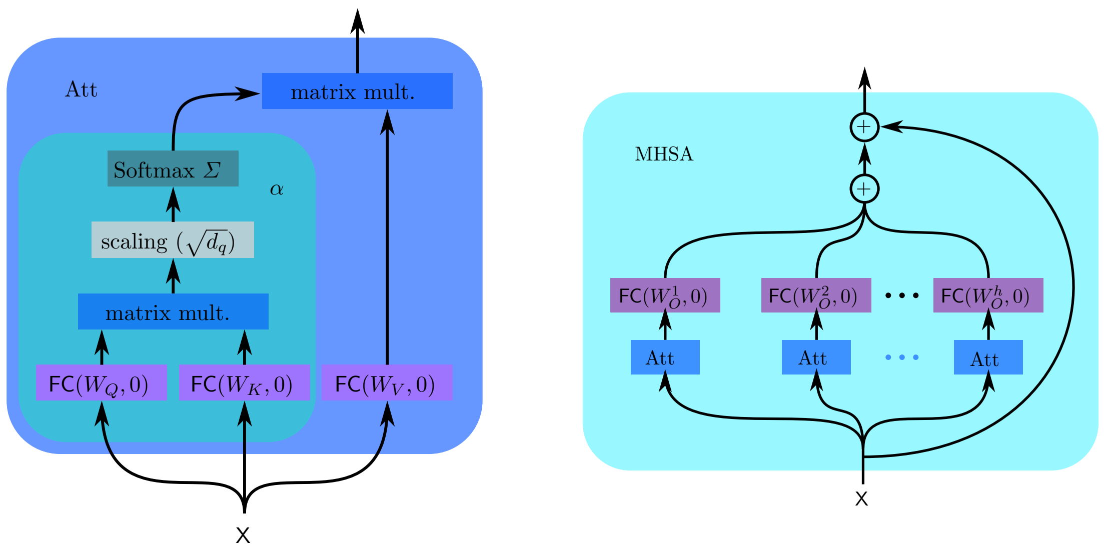


Figure 2.6: *Left*: Schematic depiction of an attention "block" involving the scaled dot-product attention α defined in eq. (2.57). *Right*: Illustration of the multi-head self attention mechanism from eq. (2.58).

column-wise[18],

$$\begin{aligned}\alpha(\mathsf{X}; W_Q, W_k) &= \Sigma\left(\frac{(W_Q\mathsf{X})^\top W_K\mathsf{X}}{\sqrt{d_q}}\right)\\ &= \left[\boldsymbol{\Sigma}\left(\frac{(W_Q\mathsf{X})^\top W_K\boldsymbol{x}_1}{\sqrt{d_q}}\right) \quad \cdots \quad \boldsymbol{\Sigma}\left(\frac{(W_Q\mathsf{X})^\top W_K\boldsymbol{x}_n}{\sqrt{d_q}}\right)\right] \in \mathbb{R}^{n\times n}.\end{aligned} \tag{2.56}$$

Note that by introducing different matrices W_Q and W_K, attention between tokens is *non-symmetric*, i.e., $\alpha(\mathsf{X}; W_Q, W_K)_{i,j} \neq \alpha(\mathsf{X}; W_Q, W_K)_{j,i}$ for $i, \neq j$ in general. The attention output is then given by applying these normalized weights to the value vectors

$$\begin{aligned}&\mathrm{Att}(\mathsf{X}; W_Q, W_K, W_V) := V \cdot \alpha(\mathsf{X}; W_Q, W_K)\\ &= \left[\sum_{j=1}^n W_V\boldsymbol{x}_j \cdot \boldsymbol{\Sigma}_j\left(\frac{(W_Q\mathsf{X})^\top W_K\boldsymbol{x}_1}{\sqrt{d_q}}\right) \quad \cdots \quad \sum_{j=1}^n W_V\boldsymbol{x}_j \cdot \boldsymbol{\Sigma}_j\left(\frac{(W_Q\mathsf{X})^\top W_K\boldsymbol{x}_n}{\sqrt{d_q}}\right)\right] \in \mathbb{R}^{d\times n}\end{aligned} \tag{2.57}$$

yielding *attention-weighted convex combinations* of the learned values. An illustration of the neural network architecture defined by such an attention block can be found in fig. 2.6 on the left.

Multi-Head Self-Attention and Transformer Layer. A straight generalization of the attention layer eq. (2.57) and a more expressive model is given by learning $h \in \mathbb{N}$ query-key-value triples $\{(W_Q^k, W_K^k, W_V^k)\}_{k\in[h]}$ jointly in parallel. This is known as *multi-head attention* where each $k \in [h]$ denotes one of the "attention heads". Then, the value

[18] We denote this by Σ applied to a matrix with slight abuse of notation.

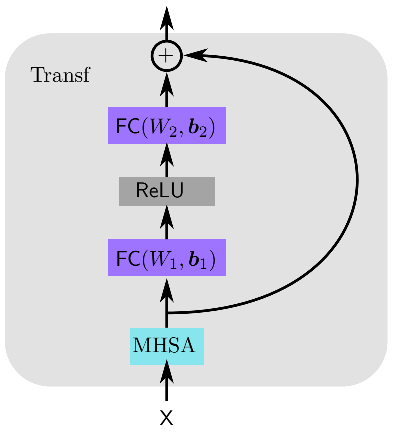


Figure 2.7: Schematic depiction of the transformer block described in eq. (2.59).

matrices $W_V^k \in \mathbb{R}^{d_q \times n}$ are oftentimes defined to map to the same d_q-dimensional latent space as W_Q^k and W_K^k. This way, all matrix multiplications can be computed in parallel. Additionally, an output matrix $W_O^k \in \mathbb{R}^{t \times d_q}$ for each head is defined, mapping to the output dimension t. The output of the multi-head self-attention (MHSA) layer is then given by introduction of an additional skip to the input X by

$$\mathrm{MHSA}(\mathsf{X}) := \mathsf{X} + \sum_{k=1}^{h} W_O^k \mathrm{Att}(\mathsf{X}; W_Q^k, W_K^k, W_V^k) \in \mathbb{R}^{t \times n}. \tag{2.58}$$

Figure 2.6 shows an illustration of the MHSA block on the right. In the transformer layer, an additional skip connection is introduced, after MHSA is computed and two fully connected layers with weight matrices $W_1 \in \mathbb{R}^{r \times t}$ and $W_2 \in \mathbb{R}^{t \times r}$, biases $\boldsymbol{b}_1 \in \mathbb{R}^r$ and $\boldsymbol{b}_2 \in \mathbb{R}^t$ and ReLU activation are applied:

$$\mathrm{T}_{h,d_q,r}(\mathsf{X}) = \mathrm{MHSA}(\mathsf{X}) + W_2 \cdot \mathrm{ReLU}\left(W_1 \cdot \mathrm{MHSA}(\mathsf{X}) + \boldsymbol{b}_1\right) + \boldsymbol{b}_2 \in \mathbb{R}^{d \times n}. \tag{2.59}$$

Here, the bias variables are constant across the token dimension n. An illustration of the transformer block defined by eq. (2.59) can be found in fig. 2.7.

Transformers for Computer Vision: ViT and Swin. One of the first significant steps towards applying transformer architectures to computer vision problems was taken by transforming image patches directly to token embeddings [36]. The image is first divided up into a fixed number (16×16) of square patches, see fig. 2.8 on the left. The two-dimensional image patches are further flattened to n one-dimensional vectors acting as the n tokens and transformed by a linear learnable map to an embedding X. Further, position embeddings help to preserve localization information of each patch. In ViT [36] vision transformers, also layer normalization was introduced before the MHSA and the final MLP block and the MLP was equipped with GELU activation functions instead of ReLU activations.

While good performance for image classification tasks was achieved by the ViT transformer models, scaling to more complex computer vision tasks like object detection and semantic

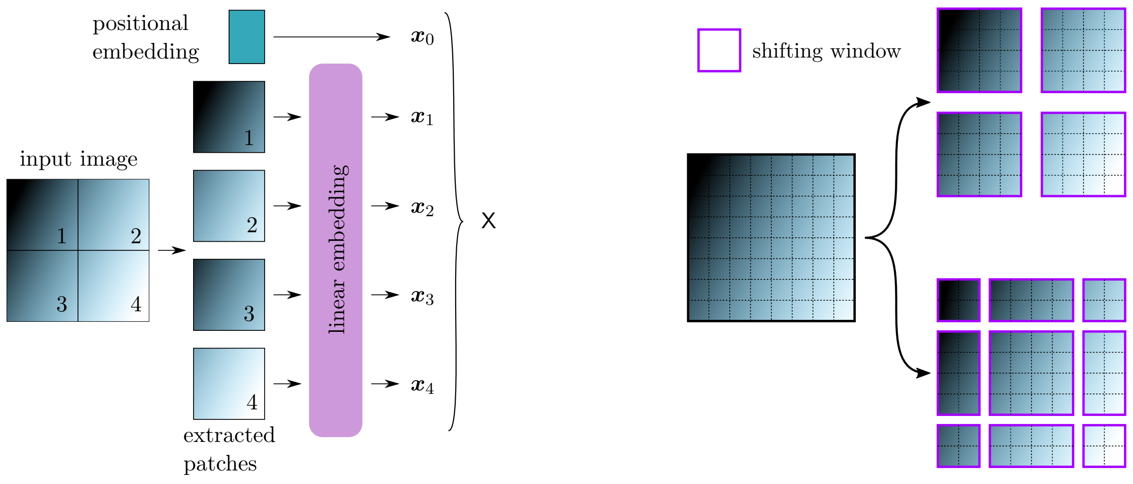


Figure 2.8: *Left*: Illustration of the image token embedding mechanism introduced in [36]. The linear embedding gives rise to token embeddings X. *Right*: Illustration of the shifting window mechanism from [108]. Self-attention is computed between token embeddings within each window. Cross-window interaction is achieved by cyclical window shifting between consecutive transformer layers.

segmentation was still unfeasible. The central reason was that computing self-attention between tokens scales quadratically with the number of tokens, i.e., with the number of patches the image is divided into (this is called "global self-attention"). Therefore, dense predictions like semantic segmentation and large input resolutions were an obstacle for the ViT transformer. A way of resolution was presented with the introduction of the Swin transformer [108] (short for "shifted window") which introduced a tiling of the input image into windows first ("local self-attention"). Within each window, again, a fixed number of patches is extracted and self-attention computed between the token embeddings of all patches, see fig. 2.8 on the right. However, no attention is computed to patches from other windows. This reduces the computational complexity of the self-attention by which it is merely quadratic in the window size. No self-attention is then computed between different windows leading to a loss of local interaction. In order to re-introduce interaction between windows, the windows tiling is cyclically shifted between consecutive transformer layers in the Swin transformer architecture.

2.2.2 Training Neural Networks: Parameter Estimation

In this section we cover *stochastic gradient descent* (SGD), the widely used optimization algorithm utilized in the training of deep neural networks. The setting for utilizing SGD involves a parametric model $\widehat{\mu}_n^{\boldsymbol{\theta}}$ where $\boldsymbol{\theta} \in \Theta$ for some parameter space[19] Θ. From observing a set of training data χ_n, the aim is to minimize some empirical risk function (see

[19]E.g., $\Theta \subseteq \mathbb{R}^q$. In some of our applications, we consider the weights in the layers of neural networks which is a space isomorphic to some $\mathbb{R}^q$.

section 2.1.3) which we call “loss” in the following.

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}\left(\widehat{\mu}_n^{\boldsymbol{\theta}}, \chi_n\right) \tag{2.60}$$

with respect to $\boldsymbol{\theta}$. In the supervised setting, $\chi_n = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n))$ and $\widehat{\mu}_n^{\boldsymbol{\theta}} = \widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}^{\boldsymbol{\theta}}$ is a supervised learning model[20] like a deep neural network. Then, the loss can usually be interpreted as a comparison of the model's prediction $\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}_i}^{\boldsymbol{\theta}}$ and the corresponding target y_i over the dataset χ_n, i.e.,

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}_i}^{\boldsymbol{\theta}} | y_i\right). \tag{2.61}$$

The following section on SGD is loosely based on [166].

2.2.2.1 Stochastic Gradient Descent: Optimizing Loss Functions with Respect to Parameters

Gradient descent optimization is concerned with a parametric model $\widehat{\mu}_n(\boldsymbol{\theta})$ and a loss function $\mathcal{L}(\boldsymbol{\theta})$ which depends on the model parameters. The parameters are optimized by iteratively computing $\mathcal{L}(\boldsymbol{\theta})$ at the current point $\boldsymbol{\theta} \in \Theta$ and taking a step in Θ in the direction of the steepest descent of the function $\mathcal{L}$. For this, the model $\widehat{\mu}_n^{\boldsymbol{\theta}}$ will be initialized[21] at some parameter $\boldsymbol{\theta}_0 \in \Theta$. Gradient steps are then taken by first computing the value of the loss function $\mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}$ at the current point in Θ and “updating” the model weights by the following iterative prescription

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \cdot \left[\nabla_{\boldsymbol{\theta}} \mathcal{L}(\widehat{\mu}_n^{\boldsymbol{\theta}}, \chi_n)\right]\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{t-1}} \qquad \text{for} \quad t = 1, 2, \ldots \tag{2.62}$$

Here, $\eta > 0$ is a scalar parameter called the *learning rate* which controls the scaling of the step size taken. Oftentimes, the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\widehat{\mu}_n^{\boldsymbol{\theta}}, \chi_n)$ is first normalized before scaling with η such that $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}\| = \eta$. While η may be a simple constant, and often is in implementations, it can also be variable $\eta = \eta_t$ over the step time t. This procedure can be shown to converge to a minimizer of $\mathcal{L}(\boldsymbol{\theta})$ in the case of a strictly convex Lipschitz optimization problem [166, Cor. 14.2]. However, more often than not, the loss functions optimized in deep learning are highly non-convex. In such a case guarantees of convergence can only be given for local minima of $\mathcal{L}(\boldsymbol{\theta})$ which may, however, be far from optimal. Not only is the loss surface ($\{(\boldsymbol{\theta}, \mathcal{L}(\boldsymbol{\theta})) \in \Theta \times \mathbb{R} | \boldsymbol{\theta} \in \Theta\}$) non-convex, but it also oftentimes has steep regions and sharp cliffs which further complicate the optimization procedure. Regularization techniques can help remedy this phenomenon, e.g., by introducing additional terms to the loss function. So-called weight decay regularization is essentially proportional to $\|\boldsymbol{\theta}\|_2$. Alternatively, $\widehat{\mu}_n^{\boldsymbol{\theta}}$ can be altered such that the resulting loss $\mathcal{L}(\boldsymbol{\theta})$ is smoother.

[20] In section 2.1.1.1, we defined what a supervised learning algorithm $\{\widehat{\mu}_n\}_{n\in\mathbb{N}}$ is. Here, a “supervised learning model” is the parametric representation of the hypothesis space $\mathscr{H}$. For deep neural networks, this hypothesis space is defined by the network architecture. Together with a specification of how the model processes data χ_n, a model defines a supervised learning algorithm.

[21] Usually, this initialization is done by randomly sampling values, e.g., from a standard normal distribution or from a uniform distribution over some predefined range of values.

This may be accomplished by, for example, including normalization layers which counteract distributional drift between hidden layers or dropout layers which stochastically regularize the dependency of $\mathcal{L}(\boldsymbol{\theta})$ on individual neurons and, therefore, the gradient.

Training data χ_n may be composed of several tens of thousands of data points. In supervised learning, the computation of $\mathcal{L}(\boldsymbol{\theta})$ requires the computation of $\widehat{\mu}^{\boldsymbol{\theta}}_{n|\boldsymbol{X}=\boldsymbol{x}_i}$ for each individual sample once before even one gradient step is performed. In practice, one then instead uses so-called mini batches which are randomly selected subsamples

$$\chi_B\{(\boldsymbol{x}_i, y_i) \in \chi_n | i \in B_t\} \subset \chi_n \tag{2.63}$$

for some index set $B_t \subset [n]$ randomly selected at step time t. Computing the mini batch loss $\mathcal{L}_{B_t}(\boldsymbol{\theta}) := \frac{1}{|B_t|}\sum_{i\in B_t}\mathcal{L}(\widehat{\mu}^{\boldsymbol{\theta}}_{n|\boldsymbol{X}=\boldsymbol{x}_i}|y_i)$ only on this subsample yields an estimate of the "true" gradient which would be computed from all samples.

Technically, stochastic gradient descent is the gradient descent update rule

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \boldsymbol{V}_t \tag{2.64}$$

with any random vector $\boldsymbol{V}_t \in \Theta$ such that $\mathbb{E}[\boldsymbol{V}]$ is a sub-gradient[22] of $\mathcal{L}(\boldsymbol{\theta})$. Taking gradients of mini batch losses $\mathcal{L}_{B_t}(\boldsymbol{\theta})$ is an easily implementable and efficient method to obtain unbiased estimators of the training gradient over the entire dataset. In particular, we obtain an unbiased estimator of a subgradient. Again, this procedure can be shown to converge to a minimizer in case of convex optimization problem.

2.2.2.2 Backpropagation: Efficient Gradient Computation for Feed-Forward Neural Networks

In order to apply the SGD algorithm to deep neural networks, we need to be able to compute the gradient

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\widehat{\mu}^{\boldsymbol{\theta}}_{n|\boldsymbol{X}=\boldsymbol{x}_i}|y_i) \tag{2.65}$$

on a sample $(\boldsymbol{x}_i, y_i)$. This object can first be separated into two contributions by the chain rule, namely

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\widehat{\mu}^{\boldsymbol{\theta}}_{n|\boldsymbol{X}=\boldsymbol{x}_i}|y_i) = D_f\mathcal{L}(f|y_i)|_{f=\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}_i}}(\boldsymbol{\theta}) \cdot D_{\boldsymbol{\theta}}\widehat{\mu}^{\boldsymbol{\theta}}_{n|\boldsymbol{X}=\boldsymbol{x}_i}. \tag{2.66}$$

The first contribution can be explicitly computed from the forward pass through the network and by knowing the explicit functional form of the loss function $\mathcal{L}$. The second term contributes the derivative of the parameter gradients stemming from the model itself. Its computation requires the derivative of the model's prediction with respect to the parameters $\boldsymbol{\theta}$. Since deep neural networks generally have the structure

$$f(\cdot|\boldsymbol{\theta}) = f_L^{\boldsymbol{\theta}_L} \circ f_{L-1}^{\boldsymbol{\theta}_{L-1}} \circ \cdots \circ f_2^{\boldsymbol{\theta}_2} \circ f_1^{\boldsymbol{\theta}_1}(\cdot), \tag{2.67}$$

[22]Here, we follow [166] and call a vector $\boldsymbol{v} \in \Theta$ a "sub-gradient" of the function $\mathcal{L}$ at $\boldsymbol{\theta}$, iff $\forall \boldsymbol{u} \in \Theta$, we have $\mathcal{L}(\boldsymbol{u}) \geq \mathcal{L}(\boldsymbol{\theta}) + \langle \boldsymbol{u} - \boldsymbol{\theta}, \boldsymbol{v}\rangle$. To prevent ambiguity, the set of sub-gradients of $\mathcal{L}$ at θ is called the "differential set" which is also sometimes called *the* "sub-gradient" of $\mathcal{L}$ at $\boldsymbol{\theta}$ in the literature.

where each $f_\ell^{\boldsymbol{\theta}_\ell}(\boldsymbol{x}^{\ell-1}) = \alpha_\ell(W_\ell \boldsymbol{x}^{\ell-1} + \boldsymbol{b}_\ell)$ is a network layer such as a fully connected layer with an activation function α_ℓ and parameters[23] $\boldsymbol{\theta}_\ell = (W_\ell, \boldsymbol{b}_\ell)$. In this notation, $\boldsymbol{x}^\ell = f_\ell^{\boldsymbol{\theta}_\ell}(\boldsymbol{x}^{\ell-1}) \in \mathbb{R}^{d_\ell}$. In a sequential model like this, earlier parameters such as $\boldsymbol{\theta}_1$ influence the activation of later layers like $f_L^{\boldsymbol{\theta}_L}(\boldsymbol{x}^{L-1})$.

Due to the chain rule for such functions at some fixed point $\overline{\boldsymbol{\theta}} \in \Theta$ we have

$$D_{\boldsymbol{\theta}_\ell} f(\cdot|\boldsymbol{\theta})|_{\overline{\boldsymbol{\theta}}} = D_{\boldsymbol{x}^{L-1}} f_L^{\overline{\boldsymbol{\theta}}_L}|_{\boldsymbol{x}^{L-1}(\overline{\boldsymbol{\theta}}_L)} \cdots D_{\boldsymbol{x}^\ell} f_{\ell+1}^{\overline{\boldsymbol{\theta}}_{\ell+1}}|_{\boldsymbol{x}^\ell(\overline{\boldsymbol{\theta}}_{\ell+1})} \cdot D_{\boldsymbol{\theta}_\ell} f_1^{\boldsymbol{\theta}_\ell}|_{\overline{\boldsymbol{\theta}}_\ell}. \tag{2.68}$$

In order to obtain the full gradient $D_{\boldsymbol{\theta}} f(\cdot|\boldsymbol{\theta})$, the components technically need to be collected via the chain rule over all layers ℓ. However, by starting from the last layer L and iteratively moving to preceding layers, parts of the computation can be re-used due to the associativity of matrix multiplication, namely the derivatives with respect to the activations $\boldsymbol{x}^\ell$. This iterative computation of gradients earlier in the network from later activation values is called *backpropagation*. Backpropagation is an effective method of reducing the computational load and memory consumption required to compute the gradient. The derivatives of a fully connected layer are given by

$$\begin{aligned} D_{\boldsymbol{x}^{\ell-1}} f_\ell^{\boldsymbol{\theta}_\ell}(\boldsymbol{x}^{\ell-1}) &= \left.\frac{\partial \alpha_\ell(y)}{\partial y}\right|_{y=W_\ell \boldsymbol{x}^{\ell-1}+\boldsymbol{b}_\ell} \cdot \nabla_{\boldsymbol{x}^{\ell-1}} W_\ell \boldsymbol{x}^{\ell-1} = \alpha_\ell'(W_\ell \boldsymbol{x}^{\ell-1} + \boldsymbol{b}_\ell) \cdot W_\ell^\top \\ D_{W_\ell} f_\ell^{\boldsymbol{\theta}_\ell}(\boldsymbol{x}^{\ell-1}) &= \left.\frac{\partial \alpha_\ell(y)}{\partial y}\right|_{y=W_\ell \boldsymbol{x}^{\ell-1}+\boldsymbol{b}_\ell} \cdot \nabla_{W_\ell} W_\ell \boldsymbol{x}^{\ell-1} = \alpha_\ell'(W_\ell \boldsymbol{x}^{\ell-1} + \boldsymbol{b}_\ell) \cdot (\boldsymbol{x}^{\ell-1})^\top, \\ D_{\boldsymbol{b}_\ell} f_\ell^{\boldsymbol{\theta}_\ell}(\boldsymbol{x}^{\ell-1}) &= \left.\frac{\partial \alpha_\ell(y)}{\partial y}\right|_{y=W_\ell \boldsymbol{x}^{\ell-1}+\boldsymbol{b}_\ell} \cdot \nabla_{\boldsymbol{b}_\ell} \boldsymbol{b}_\ell = \alpha_\ell'(W_\ell \boldsymbol{x}^{\ell-1} + \boldsymbol{b}_\ell). \end{aligned} \tag{2.69}$$

Here, $\alpha_\ell : \mathbb{R} \to \mathbb{R}$ is applied element-wise and $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$, $\boldsymbol{b}_\ell \in \mathbb{R}^{d_{\ell+1}}$. The dyadic product in eq. (2.69) comes about as follows. The Jacobi tensor of the matrix-vector multiplication has components

$$\frac{\partial [W_\ell \boldsymbol{x}^{\ell-1} + \boldsymbol{b}_\ell]_k}{\partial [W_\ell]_{ij}} = \frac{\partial}{\partial [W_\ell]_{ij}} \sum_{m=0}^{d_\ell} [W_\ell]_{km} [\boldsymbol{x}^{\ell-1}]_m = \delta_{ik} [\boldsymbol{x}^{\ell-1}]_j \tag{2.70}$$

with $k \in [d_{\ell+1}]$, $i, j \in [d_\ell]$. By re-assembling vectorized structures, in particular W_ℓ and $\boldsymbol{x}^{\ell-1}$, we find the first form of eq. (2.69). Backpropagation finally yields the full gradient of the neural network by combining eq. (2.66), eq. (2.68) and eq. (2.69).

2.2.3 Universal Approximation: Hypothesis Space Size of DNN Architectures

Whether a statistical learning model $\widehat{\mu}_n$ like a deep neural network can sufficiently approximate a given target set[24] of functions or distributions $\mathscr{T}$ from which training data

[23] We have seen that also convolution layers, batch norm layers, dropout layers identified as fully connected layers under certain constraints.

[24] This could be a function space as in the following sections or $\mathscr{M}_1(\mathcal{X})$ as discussed in section 2.1.

χ_n is sampled is investigated in an area of research called *universal approximation.* This question is often not concerned with how complex it is to find the best or an arbitrarily good model $\widehat{\mu}_n^* \in \mathscr{H}_n = \operatorname{Img}(\widehat{\mu}_n)$ which would be the desired object. Rather, guarantees about the existence of such a model are given under certain circumstances and requirements on model classes derived under which such a best model exists. However, there are also constructive approaches which try to answer exactly that question in a specified setting.

A learning algorithm $\widehat{\mu}_n$ is called a *universal approximator* of a metric space $(\mathscr{T}, \mathsf{d})$, if $\mathscr{H}_n \cap \mathscr{T}$ is d-dense in $\mathscr{T}$. That is, for any $\mu_X \in \mathscr{T}$, let $\varepsilon > 0$. Then, there exists $\widehat{\mu}_n^* \in \mathscr{H}_n$ such that $\mathsf{d}(\mu_X \| \widehat{\mu}_n^*) < \varepsilon$. Therefore, a candidate from the hypothesis space $\mathscr{H}_n$ is always arbitrarily close to a given data-generating distribution $\mu_{\boldsymbol{X}}$.

2.2.3.1 Learning Continuous Functions via Infinitely Wide Single-Layer Perceptrons

The following result by Cybenko [29] asserts that by allowing for shallow (i.e., one-hidden-layer) but infinitely wide neural networks, continuous functions on the d-dimensional unit interval $I_d := [0, 1]^d$ can be approximated in the following sense.

Let $\alpha : \mathbb{R} \to \mathbb{R}$ be an arbitrary continuous function that is "sigmoidal", i.e.,

$$\lim_{t\to-\infty} \alpha(t) = 0, \qquad \lim_{t\to\infty} \alpha(t) = 1. \tag{2.71}$$

Then, the set of single-layer perceptrons

$$\left\{ S : I_d \to \mathbb{R} \,\middle|\, S(\boldsymbol{x}) = \sum_{j=1}^{N} \lambda_j \cdot \alpha(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j), \quad N \in \mathbb{N}, \boldsymbol{\lambda}, \boldsymbol{b} \in \mathbb{R}^N, \boldsymbol{w}_j \in \mathbb{R}^d \right\} \tag{2.72}$$

is $\|\cdot\|_\infty$-dense in the set of continuous functions $C(I_d)$. This means that given a function $g \in C(I_d)$, there is a single layer perceptron of width $N \in \mathbb{N}$ with

- a weight matrix $W = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N) \in \mathbb{R}^{d\times n}$,
- a bias vector $\boldsymbol{b} \in \mathbb{R}^N$ and
- a vector $\boldsymbol{\lambda} \in \mathbb{R}^N$ of output weights connecting the N hidden neurons to the one output neuron

which is ε-close to g in the supremum norm. The proof of this statement makes use of the Riesz representation theorem as well as the Hahn-Banach theorem. Due to the usage of the Riesz representation theorem, this proof is explicitly non-constructive. Actually, a more general statement than the one stated above applies. For obtaining denseness, it is sufficient for α to be "discriminatory", meaning that given a signed regular measure $\mu \in \mathscr{M}(I_d)$,

$$\int_{I_d} \alpha(\boldsymbol{w}^\top \boldsymbol{x} + b)\,\mathrm{d}\mu(\boldsymbol{x}) = 0 \quad \forall \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R} \implies \mu = 0. \tag{2.73}$$

It is shown that any bounded and measurable sigmoidal function is discriminatory, in particular continuous sigmoidal functions.

Further, a result by Pinkus [138] asserts approximation of continuous functions on compact sets by single-layer perceptrons when the activation α is any non-polynomial smooth function.

2.2.3.2 Learning Essentially Bounded Sobolev Functions via Arbitrary-Depth ReLU *Networks*

This recent result by Yarotzky [203] allows for the approximation of functions in the unit ball of the Sobolev space[25] $W^{n,\infty}(I_d)$ with respect to the Sobolev norm

$$\|f\|_{n,\infty} = \max_{\boldsymbol{n}\in\mathbb{N}^d:|\boldsymbol{n}|<n} \operatorname*{ess\,sup}_{\boldsymbol{x}\in I_d} |\partial^{\boldsymbol{n}} f(\boldsymbol{x})|. \tag{2.74}$$

This can be achieved by finite-width ReLU multi-layer perceptrons, however, the approximation is constructive. One consequence of this fact is that explicit bounds on the depth, width and number of parameters can be given for fixed error $\varepsilon \in (0,1)$. ReLU networks are concatenations $f_L^{W_L,\boldsymbol{b}_L} \circ \cdots \circ f_1^{W_1,\boldsymbol{b}_1}$ of fully-connected ReLU layers of the form

$$f_\ell^{W_\ell,\boldsymbol{b}_\ell}(\boldsymbol{x}) = \mathrm{ReLU}\left(W_\ell \boldsymbol{x} + \boldsymbol{b}_\ell\right), \qquad \ell \in [L], \tag{2.75}$$

where $\boldsymbol{x} \in \mathbb{R}^{d_i}$, $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$, $\boldsymbol{b}_\ell \in \mathbb{R}^{d_\ell}$ are the weight matrix and bias vector of layer ℓ and d_ℓ is the number of output neurons of layer ℓ.

The first central result concerns the unit ball

$$F_{n,d} := \{g \in W^{n,\infty}(I_d) : \|g\|_{n,\infty} \le 1\} \tag{2.76}$$

and asserts that given $d, n \in \mathbb{N}$ and $\varepsilon \in (0,1)$, there is a ReLU network architecture capable of expressing any function $g \in F_{n,d}$ with $\|\cdot\|_\infty$-error of at most ε. Moreover, such an architecture has depth of at most $c \cdot (\ln(1/\varepsilon) + 1)$ and at most $c\varepsilon^{-d/n} \cdot (\ln(1/\varepsilon) + 1)$ weights and neurons, where $c = c(d,n)$ is constant.

The approach taken in order to obtain these bounds first constructs an *approximation of the square* function $x \mapsto x^2$ via ReLU networks. This also allows for approximation of the multiplication function via

$$a \cdot b = \frac{1}{2}\left((a+b)^2 - a^2 - b^2\right), \qquad a, b \in \mathbb{R}. \tag{2.77}$$

It is then possible to construct an approximation of d-dimensional piece-wise linear partitions of unity. While this alone is technically enough for universal approximation, bounds can be derived from this constructive approach. Given such a partition of unity over I_d, the local degree-$(n-1)$ Taylor polynomial of $g \in F_{n,d}$ can be approximated also from the multiplication approximation on each individual partition factor. The approximation then takes the form of a sum (over all partition factors) over products (of partition functions times local Taylor polynomials) which can be represented by a deep ReLU network.

[25]That is, Lebesgue-essentially bounded functions $L^\infty(I_d)$ with essentially bounded distributional derivative up to order n. We denote the magnitude of a multi-index $\boldsymbol{n} \in \mathbb{N}^d$ by $|\boldsymbol{n}| := \sum_{j=1}^d n_j$.

2.2.3.3 Learning Group-Equivariant Maps via Convolutional Neural Network

In another work by Yarotzky [204], group-equivariant maps are treated. In the case of classical convolutional neural networks, the symmetry group is the group $\Gamma = \mathbb{Z}_{H_\psi} \otimes \mathbb{Z}_{W_\psi}$ of translations across the input feature map. Under these translations, the convolution map

$$[\mathrm{Conv}_{K,\boldsymbol{b}}(\psi)]^d_{ij} := \sum_{c=1}^{C} \sum_{k=1}^{H_K} \sum_{l=1}^{W_K} (K^d_c)_{kl} \psi^c_{i+k,j+l} + b^d \tag{2.78}$$

from $V := \mathbb{R}^{C \times H_\psi \times W_\psi}$ to $U := \mathbb{R}^{D \times H_\psi \times W_\psi}$ is equivariant with respect to representations R_V and R_U of Γ. Such representations act on V and U, respectively by

$$[R_V(a \otimes b)\psi]^c_{ij} = \psi^c_{i+a \bmod H_\psi, j+b \bmod W_\psi}, \quad [R_U(a \otimes b)\psi]^d_{ij} = \psi^d_{i+a \bmod H_\psi, j+b \bmod H_\psi}. \tag{2.79}$$

That is, by shifting the feature map cyclically by the numbers $(a, b) \in \Gamma$ of pixels. R_V-R_U-equivariance of a map $f : V \to U$ means that

$$f(R_V(\gamma)\psi) = R_U(\gamma) f(\psi), \qquad \forall \gamma \in \Gamma, \tag{2.80}$$

i.e., mapping a γ-shifted feature map is equivalent to first mapping with f and γ-shifting the result $f(\psi)$. That the convolution map (with cyclic padding) is equivariant under the representations defined in eq. (2.79) can easily be seen.

Feature maps $\psi \in \mathbb{R}^{C \times H_\psi \times W_\psi}$ can also be regarded as functions $\psi^c : \Gamma \to \mathbb{R}$, i.e., $\psi^c \in \mathbb{R}^\Gamma$ for any $c = 1, \ldots, C$. Since Γ is finite in this case, we can express ψ^c by a vector $\boldsymbol{\psi}^c$ which has components $\{\boldsymbol{\psi}^c_\gamma | \gamma \in \Gamma\}$. An entire feature map can, therefore, be expressed as a vector

$$\boldsymbol{\psi} = \sum_{c=1}^{C} \sum_{\gamma \in \Gamma} \boldsymbol{\psi}^c_\gamma \cdot \boldsymbol{e}_c \otimes \boldsymbol{e}^\gamma \in \mathbb{R}^C \otimes \mathbb{R}^\Gamma \tag{2.81}$$

in the module $V_{\text{in}} = \mathbb{R}^C \otimes \mathbb{R}^\Gamma$ carrying the representation R of Γ on the second factor. This representation then acts by

$$R(\gamma)\, \boldsymbol{v} \otimes \boldsymbol{e}^\theta = \boldsymbol{v} \otimes \boldsymbol{e}^{\theta+\gamma}, \qquad \forall \boldsymbol{v} \in \mathbb{R}^C, \gamma, \theta \in \Gamma. \tag{2.82}$$

Similarly, there is a representation R_{out} on $V_{\text{out}} := \mathbb{R}^D \otimes \mathbb{R}^\Gamma$. In [204] it is established that with some continuous function $\alpha : \mathbb{R} \to \mathbb{R}$ that is not a polynomial, any continuous R_{in}-R_{out}-equivariant map $g : V_{\text{in}} \to V_{\text{out}}$ can be $\|\cdot\|_\infty$-approximated by one-layer convolutional neural networks. That is, by equivariant maps $S : V_{\text{in}} \to V_{\text{out}}$ of the form

$$S^{D,K,\boldsymbol{b},\boldsymbol{\lambda}}_\gamma(\boldsymbol{\psi}) = \sum_{d=1}^{D} \lambda_d\, \alpha \left(\sum_{c=1}^{C} \sum_{\theta \in \Gamma} \left(K^d_c \right)_{\boldsymbol{\theta}} \boldsymbol{\psi}^c_{\gamma+\theta} + b^d \right). \tag{2.83}$$

Here, D is the number of “out-going channels” which are added in the output layer, K is the $\mathbb{R}^{C \times D} \otimes \mathbb{R}^\Gamma$-valued kernel tensor. This kernel tensor holds the learnable coefficients and $\boldsymbol{b} \in \mathbb{R}^D$ is the bias vector. The vector $\boldsymbol{\lambda} \in \mathbb{R}^D$ holds the weights which connect to the output layer. The argument of α is a convolution layer with bias applied to the feature map representation $\boldsymbol{\psi}$. Note that here, the kernel K is allowed to have the full, same spatial size as ψ. This "non-locality" of the convolution used is necessary for a single-layer convolutional network since otherwise the receptive field of the kernel constrains the functions which can be approximated.

Any compact set $K_{\text{in}} \subset V_{\text{in}}$ is symmetrized under the continuous group action to a Γ-invariant compact set $K_{\text{sym}} = \bigcup_{\gamma\in\Gamma} R_{\text{in}}(\gamma)K_{\text{in}}$. On K_{sym}, the previously stated result by Pinkus guarantees an approximating single-layer perceptron function $f_1 : V_{\text{in}} \to \mathbb{R}$ with $\sup_{x\in K_{\text{sym}}} |g(x) - f_1(x)| < \varepsilon$. This leads to Γ-equivariant ε-approximation by the equivariant function

$$\widehat{f}(x) := \int_\Gamma \sum_{j=1}^{N} R(\gamma)^{-1}\boldsymbol{\lambda}_j\alpha\left(\boldsymbol{w}_j^\top R(\gamma)x + b_j\right) \mathrm{dH}(\gamma). \tag{2.84}$$

Here, H is the Haar measure on Γ and $\boldsymbol{\lambda}_j, \boldsymbol{w}_j \in V_{\text{out}}$ and $\boldsymbol{b}_j \in \mathbb{R}^N$ are parameters. The action of a linear functional $\boldsymbol{w}_j^\top$ on a vector $V_{\text{out}} \ni \boldsymbol{\phi} \mapsto \boldsymbol{w}_j^\top\boldsymbol{\phi}$ can be re-written in components as

$$\boldsymbol{w}_j^\top\boldsymbol{\phi} = \sum_{\gamma\in\Gamma}\sum_{d=1}^{D}(\boldsymbol{w}_j)_d^\gamma\boldsymbol{\phi}_\gamma^d. \tag{2.85}$$

Plugging this and $\boldsymbol{\lambda}_j = \sum_{d=1}^{D}\sum_{\theta\in\Gamma}(\lambda_j)_{\boldsymbol{\theta}}^d\boldsymbol{e}_d \otimes \boldsymbol{e}^\theta$ into eq. (2.84) reveals the form in eq. (2.83).

2.2.3.4 Learning Permutation-Equivariant Sequence-to-Sequence Maps via Transformer Neural Networks

Yun et al. [211] showed that transformer neural networks

$$f_{\text{transf}} = \mathrm{T}^L_{h,d_q,r} \circ \ldots \circ \mathrm{T}^1_{h,d_q,r} \tag{2.86}$$

consisting of consecutive transformer blocks $\mathrm{T}^\ell_{h,d_q,r}$ (see eq. (2.59)) are universal L^p-approximators of continuous permutation-equivariant sequence-to-sequence functions $g : \mathbb{R}^{t\times n} \to \mathbb{R}^{t\times n}$. Permutation equivariance here means equivariance with respect to permutation of the token embeddings. This means that permutation matrices P act from the right and equivariance of a function g holds if and only if $g(\mathsf{X}P) = g(\mathsf{X})P$ for all $\mathsf{X} \in \mathbb{R}^{t\times n}$. This connection arises from the fact that transformer neural networks as defined in eq. (2.59) define permutation-equivariant maps. Note that bias addition is, in fact, permutation-invariant.

Given any $1 \le p < \infty$, $\varepsilon > 0$ and any continuous permutation-equivariant function $g : \mathbb{R}^{t\times n} \to \mathbb{R}^{t\times n}$ with compact support, there is a transformer network f of the form eq. (2.86) with $h = 2$, $m = 1$ and $r = 4$ such that

$$\|g - f_{\text{transf}}\|_{L^p(\mathbb{R}^{t\times n};\mathbb{R}^{t\times n})} < \varepsilon. \tag{2.87}$$

Firstly, continuity of g on its compact support implies uniform continuity, from which g can be approximated to $\varepsilon/3$ by a piece-wise constant function $\overline{g}$ over a grid with stride δ

determined by ε. Since g is permutation-equivariant, $\overline{g}$ inherits permutation-equivariance due to being constant over hypercubes in the determined grid. Further, piece-wise constant functions that are permutation-equivariant can be approximated by "modified transformer" models. Modified transformers have the softmax function replaced by the argmax function and allow for general activation functions that are piece-wise linear with at most three pieces and at least one constant piece. It is shown that $\overline{g}$ can be approximated by a modified transformer $\overline{f}_{\text{mod}}$ with $h = 2$, $m = 1$ and $r = 1$ with $\|\overline{g} - \overline{f}_{\text{mod}}\|_{L^p} = O(\delta^{d/p})$. Finally, since the softmax function Σ approaches the argmax $\Sigma_i(\lambda \boldsymbol{x}) \to \delta_{i,\text{argmax}_{c=1,\ldots,t} x_c}$ for $\lambda \to \infty$ and piece-wise linear functions can be approximated by ReLU functions, $\overline{f}_{\text{mod}}$ can be $\varepsilon/3$-approximated by a transformer f_{transf} with h, m, r as stated above. Finally,

$$\|g - f_{\text{transf}}\|_{L^p} \leq \|g - \overline{g}\|_{L^p} + \left\|\overline{g} - \overline{f}_{\text{mod}}\right\|_{L^p} + \left\|\overline{f}_{\text{mod}} - f_{\text{transf}}\right\|_{L^p} < \frac{2\varepsilon}{3} + O(\delta^{d/p}) \quad (2.88)$$

allows for ε-approximation of g with δ chosen small enough.

Further, Yun et al. show that incorporating learnable positional encodings which are added to the input X removes the restriction of permutation-equivariance. Therefore, positional encodings allow for ε-approximation of arbitrary continuous functions defined on a compactum in $\mathbb{R}^{t \times n}$ by transformer models with h, d_q, r as above in the permutation-equivariant case.

2.2.4 Prediction Uncertainty Quantification for Deep Neural Networks

Learning from i.i.d. data χ_n as described in section 2.1 which leads to a predictive model $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}$ involves a series of approximation steps. These may lead to propagation of errors and initial uncertainty. Approximations during training, lack of data, mis-specification of the hypothesis space $\mathscr{H}$ or theoretical assumptions that do not usually hold in practice[26] may lead to prediction error of the model. If uncertainty in the data distribution or the model fitting process is completely or partially ignored, and a point prediction is enforced given a previously unseen example $\boldsymbol{x} \in \mathcal{X}$, statistical models may fail in their predictions. This can lead to erroneous predictions such as the one shown in fig. 2.9 where an object detection model correctly identified the localization of the "cat" foreground object. However, the predicted class is incorrect. In down-stream tasks of a machine learning algorithm, such as trajectory planning in automated driving, this behavior may lead to further errors. Such errors can then have catastrophic consequences when the model is applied in safety-relevant domains such as automated driving or surgery. In this section, drawing from [70], we describe some sources of uncertainty in statistical models. We are particularly interested in neural networks and methods for estimating predictive uncertainty.

2.2.4.1 Sources of Uncertainty in Trained Deep Neural Networks

First, the data-generating distribution typically carries inherent uncertainty. This is especially noticeable in supervised learning, where the data-generating distribution $\mu_{|\boldsymbol{X}=(\cdot)}$

[26]Such as independent and identical distribution of data samples.

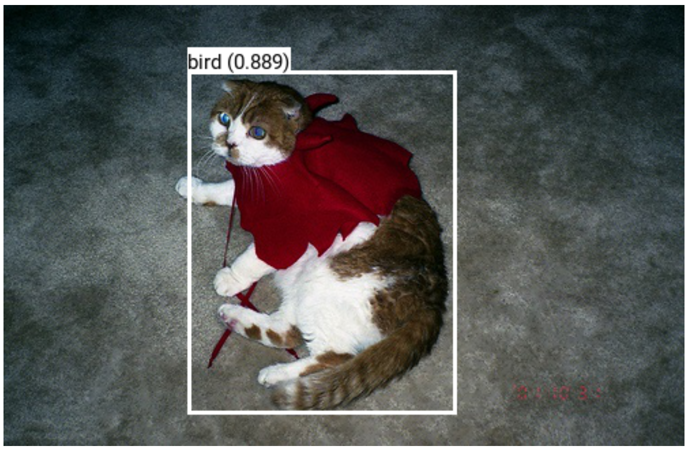


Figure 2.9: Prediction error of an object detection model (YOLOv3 [39]) on a previously unseen test sample of the Pascal VOC [38] dataset. While the localization of the predicted box is only slightly off, the predicted class is “bird” instead of “cat”. The number 0.889 in parentheses indicates the objectness score of the prediction, i.e., the overall confidence of the bounding box containing an object.

is conditional. The distribution does not uniquely identify an outcome y for an input $\boldsymbol{x}$. Therefore, learning from data samples $(\boldsymbol{x}, y)$ drawn from $\mu_{|\boldsymbol{X}=(\cdot)}$ intrinsically leads to so-called aleatoric uncertainty (see section 2.1.1.3) of point estimates. Even under full knowledge of the distribution, the pointwise Bayes predictor f^* with respect to a distance measure D_d given by

$$f^*(\boldsymbol{x}) \in \underset{y^* \in \mathcal{Y}}{\operatorname{argmin}} \int_{\mathcal{Y}} \mathsf{D}_\mathsf{d}\left(\mu_{|\boldsymbol{X}=\boldsymbol{x}} \| \delta_{y=y^*}\right) \, \mathrm{d}\mu_{|\boldsymbol{X}=\boldsymbol{x}}(y) \tag{2.89}$$

may disagree with a test sample $(\boldsymbol{x}, y)$ drawn from $\mu_{|\boldsymbol{X}=(\cdot)}$.

Moreover, due to mis-specification of the hypothesis space $\mathscr{H}$, there may be further a discrepancy between $f^*(\boldsymbol{x})$ and a best possible model

$$\nu^*_{|\boldsymbol{X}=\boldsymbol{x}} \in \underset{\nu_{|\boldsymbol{X}=(\cdot)} \in \mathscr{H}}{\operatorname{argmin}} \mathsf{D}_\mathsf{d}(\mu_{\boldsymbol{X}=\boldsymbol{x}} \| \nu_{|\boldsymbol{X}=\boldsymbol{x}}) \tag{2.90}$$

in $\mathscr{H}$. This discrepancy is called *model uncertainty* and related with the model error term $\varepsilon_{\text{model}}$ in the ERM error decomposition. Similarly, there is *approximation uncertainty* due to the estimation process in ERM learning and sampling of the data[27]. These latter types of uncertainty are oftentimes collected under the umbrella term of *epistemic uncertainty* [70]. In contrast to aleatoric uncertainty which is oftentimes seen as intrinsic to the data-generating distribution, epistemic uncertainty is regarded to be reducible. This can be achieved by increasing model capacity (appealing to universal approximation), such that model uncertainty is decreased. Alternatively, more elaborate optimization and sampling of sufficient amounts of data would be a way of reducing approximation uncertainty.

[27]These types of uncertainty are related to the optimization error $\varepsilon_{\text{learn},n}$ and the statistical error $\varepsilon_{\text{sample},n}$, respectively.

Note, that broadly in the literature this separation of aleatoric and epistemic uncertainty is not always as strictly adopted and in practice both types are also not always as clearly distinguishable.

Aleatoric uncertainty given a previously unseen $\boldsymbol{x} \in \mathcal{X}$ is hard to quantify from the predictive distribution $\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}$ if nothing is known about the epistemic uncertainty of the model on x. Combined uncertainty measures of the predictive distribution such as the *variance or the Shannon entropy* [167]

$$H(\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}) = -\mathbb{E}_{Y \sim \widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}} \left[\log\left(\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}(\{Y\})\right)\right] \tag{2.91}$$

give a joint estimation of epistemic and aleatoric uncertainty. The closer $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}$ is to[28] $\mu_{\boldsymbol{X}=(\cdot)}$, the closer the predictive entropy in eq. (2.91) will be to the entropy of the data-generating distribution and, therefore, aleatoric uncertainty.

2.2.4.2 Methods to Estimate Uncertainty for Deep Neural Networks

The output variables of a neural network can be modeled such that they represent a predictive probability distribution, such as in classification tasks. Here, typically a final softmax activation yields a discrete probability distribution over possible outcomes $y \in \mathcal{Y}$ conditioned on $\boldsymbol{x} \in \mathcal{X}$. For regression problems, a parametric continuous output distribution can be modeled by, e.g., modeling normally distributed outputs. In both cases, it is possible to estimate aleatoric uncertainty via some uncertainty measure of the predictive distribution $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}$.

The architecture and hence, the parametric form of the neural network specifies and fixes the hypothesis space $\mathscr{H}$ explored during training. Therefore, epistemic uncertainty in the context of neural networks is usually understood as uncertainty concerning the parameters $\boldsymbol{\theta} \in \Theta$. Such a consideration addresses approximation uncertainty and can be captured by a Bayesian adaptation of neural networks called Bayesian neural networks.

Bayesian Neural Networks. In *Bayesian neural networks*, the parameters $\boldsymbol{\theta} \in \Theta$ are modeled as random variables themselves. That is, they follow some posterior probability distribution $\boldsymbol{\theta} \sim \nu_{|\chi_n}$ conditional on the training samples χ_n via Bayesian inference. If the posterior $\nu_{|\chi_n}$ is known, a prediction on a new, previously unseen example is obtained from Bayesian model averaging

$$\widehat{p}_{|\boldsymbol{X}=\boldsymbol{x}}(y|\chi_n) = \int_{\Theta} \widehat{\mu}_{n|X=x}(y|\boldsymbol{\theta})\,\mathrm{d}\nu_{|\chi_n}(\boldsymbol{\theta}). \tag{2.92}$$

Again, this prediction is inherently probabilistic over $y \in \mathcal{Y}$ and uncertainty may be quantified via its entropy or variance. However, the posterior is practically intractable, so approximate variational techniques can be applied in some cases. Such approximations can take the form of modeling a parametric distribution $\widehat{\nu}_{\boldsymbol{\zeta}}$ over Θ with parameters in some parameter space $\boldsymbol{\zeta} \in Z \subseteq \mathbb{R}^{d_\zeta}$ with fixed dimensionality $d_\zeta \in \mathbb{N}$. One then seeks to minimize the distance

$$D_{\mathrm{KL}}\left(\widehat{\nu}_{\boldsymbol{\zeta}} \| \, \nu_{|\chi_n}\right) \tag{2.93}$$

[28]That is, the smaller the epistemic error of the model.

with respect to $\boldsymbol{\zeta}$ during training by sampling from the posterior. This is also only tractable for rather small neural networks. In more complex applications of neural networks such as object detection or semantic segmentation of high-resolution images, Bayesian neural networks cannot be utilized in this way. Rather, one seeks *empirical approximations or estimations* of the posterior $\nu_{|\chi_n}$. Monte-Carlo dropout and deep ensembles are two such approaches which have been applied widely in the deep learning literature.

Monte-Carlo Dropout. Srivastava et al. [169] originally introduced dropout as a method for regularizing training of deep neural networks, however, reasoned empirically that dropout can be used during inference for weight averaging. This method is known as *Monte-Carlo dropout*. Gal and Ghahramani [45] showed that neural networks with dropout used in each weight layer is equivalent to Bayesian approximation via a Gaussian process over the parameters. To this end, fully connected neural networks trained with some task objective $\mathcal{L}_{\text{task}}$ is optimized with weight decay, i.e., L^2-regularization over the weights and biases and with dropout on each layer. Application of dropout to the DNN's parameters $\boldsymbol{\theta}$ leads to a projection $\underline{\boldsymbol{\theta}}$ of a random subset of the parameters to the parameter hyperplanes by being set to zero. Performing N_{DO} forward passes under dropout can be viewed as sampling weights from a parameter distribution obtained from $\boldsymbol{\theta}$ by random projections $\{\underline{\boldsymbol{\theta}}_j\}_{j\in[N_{\text{DO}}]}$

$$\widehat{\nu}_{\text{DO}} = \frac{1}{N_{\text{DO}}} \sum_{j=1}^{N_{\text{DO}}} \delta_{\boldsymbol{\zeta}=\underline{\boldsymbol{\theta}}_j}. \tag{2.94}$$

Obtaining a predictive distribution over $y \in \mathcal{Y}$ on a previously unseen sample $\boldsymbol{x} \in \mathcal{X}$ then boils down to averaging the predictions under active dropout

$$\widehat{p}_{|\boldsymbol{X}=\boldsymbol{x}}(y|\chi_n) = \int_{\Theta} \widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}(y|\boldsymbol{\zeta})\,\mathrm{d}\widehat{\nu}_{\text{DO}}(\boldsymbol{\zeta}) = \frac{1}{N_{\text{DO}}} \sum_{j=1}^{N_{\text{DO}}} \widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}(y|\underline{\boldsymbol{\theta}}_j). \tag{2.95}$$

One possible way of quantifying the uncertainty of this prediction is to compute the standard deviation

$$\widehat{\varsigma}_{|\boldsymbol{X}=\boldsymbol{x}}(y|\chi_n) = \sqrt{\frac{1}{N_{\text{DO}}-1} \sum_{j=1}^{N_{\text{DO}}} \left(\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}(y|\underline{\boldsymbol{\theta}}_j) - \widehat{p}_{|\boldsymbol{X}=\boldsymbol{x}}(y|\chi_n)\right)^2} \tag{2.96}$$

or the variance of the predictions. Note, that $\widehat{\nu}_{\text{DO}}$ is dependent on the parameter vector $\boldsymbol{\theta}$ obtained from training under dropout. In practice, it has been found that Monte-Carlo dropout can be effectively used by applying dropout only on part of the neural network. Specifically, in order to reduce inference time, dropout is often used only on the last couple of DNN layers. This way, forward pass result up until then can be re-used for all dropout samples. Major advantages of MC dropout are its simplicity of implementation and universality in application to different prediction tasks such as object detection and semantic segmentation.

Deep Ensembles. Similarly to MC dropout, Lakhshminarayanan et al. [88] propose a different way of sampling from a distribution over the DNN parameters. Instead of taking one pre-trained parameter vector $\boldsymbol{\theta}$ and modifying it in order to obtain samples, an *ensemble of models*, i.e., parameter vectors $\{\boldsymbol{\theta}_j\}_{j=1}^{N_{\mathrm{DE}}}$ is obtained by training from scratch. It is argued that in contrast to classical boosting and bagging models, no subsampling of the training data is necessary. Rather, random initialization of the parameters before training and the stochastic batch sampling during training introduce enough stochasticity into the ensemble. In fact, it is argued that sub-sampling might negatively influence ensemble performance since deep neural networks inherently need large amounts of data due to having comparably many parameters. Moreover, choosing a "proper scoring rule" $\mathcal{L}$ between the model $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}(\cdot|\boldsymbol{\theta})$ and the data-generating distribution $\mu_{\boldsymbol{X}=(\cdot)}$ is used during training. Minimization of the negative log-likelihood is identified as providing such a proper scoring rule. Furthermore, it is proposed to use adversarial training in order to "smoothen" the predictive distributions. Predictions are then obtained, analogously to MC dropout via model averaging over $\widehat{\nu}_{\mathrm{DE}} = 1/N_{\mathrm{DE}} \sum_{j=1}^{N_{\mathrm{DE}}} \delta_{\boldsymbol{\zeta}=\boldsymbol{\theta}_j}$. In practice, adversarial training is not necessarily performed for convenience of application. For more complex computer vision tasks such as object detection and semantic segmentation, adversarial attacks are also far more complicated than in simple classification tasks. Deep ensembles have proven to perform well without adversarial training.

Applications of Predictive Uncertainty. In safety-critical applications of deep learning, estimation of prediction uncertainty is crucial for down-stream tasks. Uncertainty quantification allows for the definition of protocols for how to deal reliably and conservatively with different failure modes of the neural network. Beyond such down-stream tasks, there are more immediate applications of uncertainty quantification, which will be explored and studied in later chapters (see chapters 5 to 7). First, some uncertainty estimation methods allow for the computation of more reliable confidence estimates. Basing predictions on such advanced "confidence scores" can lead to a performance increase, e.g., in object detection (chapter 3). Further, one crucial failure mode of neural networks is the confrontation with so-called out-of-distribution (OoD) objects in their input. Since neural networks for classification, object detection or semantic segmentation are usually trained on a fixed set $[C] = \{1, \ldots, C\}$ of classes, confrontation with something that is not part of this semantic space leads to prediction errors. Classifiers are usually forced to commit to one of the C classes for each prediction. In case of presentation with an OoD object, the model should ideally have a mechanism to refuse the prediction since it cannot give a correct answer. Uncertainty quantification methods can give rise to such mechanisms (chapter 4), particularly, OoD detection is closely related with epistemic uncertainty. On the side of model uncertainty, active learning utilizes information from a trained model in order to select unseen samples based on informativeness to add to the training dataset. Such informativeness measures oftentimes involve uncertainty estimates (chapter 5) which can be computed based on the input without the annotation. The motivation for active learning is that not all available and recorded data points $\boldsymbol{x}$ can be annotated with a target y since this annotation process is expensive and time-consuming. Moreover, it involves human annotators which are fallible. As a consequence, already obtained annotations in training and test data are error-prone. Annotation errors occur not only on industrial datasets which may have an experimental character, but also on large-scale public benchmark datasets. Uncertainty quantification methods for deep learning models can be used to produce proposals for annotation errors in present data in an automated way (chapter 6 and chapter 7).

2.3 Advanced Computer Vision Tasks: The Need for Machine Learning

Perception tasks on different kinds of data can have various forms. While one of the simplest forms is input classification, more complex, multimodal tasks can give richer information about the semantic and geometric content of information present. In this section we describe the central primary computer vision tasks on which the presented work builds upon and what their specificities are.

2.3.1 Image Classification: Recognition of Categories

In image classification, one of a predefined list $\mathcal{Y} = [C]$ of $C \in \mathbb{N}$ categories is assigned to a given input image[29] $\boldsymbol{x} \in \mathcal{X} = [0,1]^{3\times H\times W}$ where $H, W \in \mathbb{N}$. For future reference, we denote the set of image pixel locations

$$\mathcal{I} := \{(i,j) : i \in [H], j \in [W]\}. \tag{2.97}$$

Training data, therefore, consists of tuples $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$. Classification models are oftentimes probabilistic which means that they predict a conditional probability distribution $\widehat{\mu}_{n|\boldsymbol{X}=(\cdot)}$ over $\mathcal{Y}$. This distribution is usually the result of a softmax activation applied to the C-dimensional logit output of the model. We denote this conditional probability distribution by a $[0,1]^C$-valued function

$$\widehat{f}(\cdot|\boldsymbol{\theta}) : \mathcal{X} \to [0,1]^C, \qquad \widehat{f}_c(\boldsymbol{x}|\boldsymbol{\theta}) = \widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}(\{c\}), \quad \forall c \in \mathcal{Y}. \tag{2.98}$$

The argmax of this function yields the maximum a posteriori estimate of the predicted class $\widehat{c}(\boldsymbol{x}|\boldsymbol{\theta})$. The corresponding maximal probability

$$\widehat{s}(\boldsymbol{x}|\boldsymbol{\theta}) = \widehat{f}_{\widehat{c}(\boldsymbol{x}|\theta)}(\boldsymbol{x}|\boldsymbol{\theta}) = \max_{c\in\mathcal{Y}} \widehat{f}_c(\boldsymbol{x}|\boldsymbol{\theta}) \tag{2.99}$$

yields the confidence with which the prediction $\widehat{c}(\boldsymbol{x}|\boldsymbol{\theta})$ is given. This quantity is also sometimes called the *maximum softmax score* or simply score.

2.3.1.1 Loss Functions in Classification: Differences in Probability Distributions

The so-called 0-1-loss which only punishes misclassifications with loss 1 was used very early in the study of neural networks. While it is useful in statistical learning theory for the study of classification function estimators, it is not very useful in modern classification problems. The reason for this is, that it is not even continuous and, therefore, unpractical for gradient optimization algorithms. An early adoption from least-squares regression is the *mean squared error* which can be applied in the classification setting

$$\mathcal{L}_{\mathrm{MSE}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle| y\right) = \frac{1}{C}\sum_{c=1}^{C}\left(\widehat{f}_c(\boldsymbol{x}|\boldsymbol{\theta}) - \delta_{yc}\right)^2. \tag{2.100}$$

[29]We can easily regard grayscale images as the one-dimensional diagonal in $[0,1]^3$.

More often, however, the *cross entropy loss* from section 2.1.3.2

$$\mathcal{L}_{\mathrm{CE}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\, y\right) = -\log\left(\widehat{f}_y(\boldsymbol{x}|\boldsymbol{\theta})\right) = -\sum_{c=1}^{C}\delta_{yc}\log\left(\widehat{f}_c(\boldsymbol{x}|\boldsymbol{\theta})\right) \tag{2.101}$$

is used. This function is identical to the negative log-likelihood loss in classification. The cross entropy loss has the advantage of not having small gradients where $\delta_{yc} \approx \widehat{f}_c(\boldsymbol{x}|\boldsymbol{\theta})$.

2.3.1.2 Evaluation Metrics in Classification: Accuracy and Area Under Curve

Given a test dataset

$$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{|\mathcal{D}|}, y_{|\mathcal{D}|})\} \subset \mathcal{X} \times \mathcal{Y} \tag{2.102}$$

of annotated samples which is disjoint from the training sample χ_n, a classifier $\widehat{f}$ can be assigned several performance metrics. These indicate how accurately it predicts the labels y_i from the input $\boldsymbol{x}_i$. Depending on the type of classification task (binary versus multi-class), different metrics can be computed. While the computed loss is in principle a valid metric for a single model, it does not easily allow for an interpretable value which relates directly to the test samples in $\mathcal{D}$.

Accuracy. The *accuracy* is a performance measure which is applicable to binary as well as to multi-class classification problems. Accuracy denotes simply the fraction of correctly classified samples by the prediction $\widehat{c}(\cdot|\boldsymbol{\theta})$ of $\widehat{f}$:

$$\mathrm{Acc}\left(\widehat{f};\mathcal{D}\right) := \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\delta_{\widehat{c}(\boldsymbol{x}_i|\boldsymbol{\theta}),y_i} \in [0,1]. \tag{2.103}$$

Therefore, higher values mean more precise predictions. In the multi-class setting, this can be generalized to the top-k accuracy, which is the fraction of samples where y_i is among the k classes with the highest predicted probability of $\widehat{f}$. The number $k \in \mathbb{N}$ can be regarded as a hyperparameter of this metric.

Area under Precision-Recall Curve. In binary classification, i.e., $\mathcal{Y} = \{0, 1\}$, outcomes are often denoted as "positive" (1) or "negative" (0)

$$\begin{aligned} \mathrm{P} :=& \{(\boldsymbol{x}, y) \in \mathcal{D} : y = 1\}, \\ \mathrm{N} :=& \{(\boldsymbol{x}, y) \in \mathcal{D} : y = 0\}, \end{aligned} \tag{2.104}$$

and so can the predictions be denoted as *predicted positives* and *predicted negatives*. Oftentimes, what is a positive prediction is determined by a threshold $\tau \in [0,1]$ and $\boldsymbol{x} \in \mathcal{X}$ is predicted as a positive sample, if $\widehat{f}_1(\boldsymbol{x}|\boldsymbol{\theta}) > \tau$ and as a negative sample otherwise. Over $\mathcal{D}$, one then defines the sets of predicted positives (PP) and predicted negatives (PN)

$$\begin{aligned} \mathrm{PP}(\tau) :=& \{(\boldsymbol{x}, y) \in \mathcal{D} : \widehat{f}_1(\boldsymbol{x}|\boldsymbol{\theta}) > \tau\}, \\ \mathrm{PN}(\tau) :=& \{(\boldsymbol{x}, y) \in \mathcal{D} : \widehat{f}_1(\boldsymbol{x}|\boldsymbol{\theta}) \leq \tau\}, \end{aligned} \tag{2.105}$$

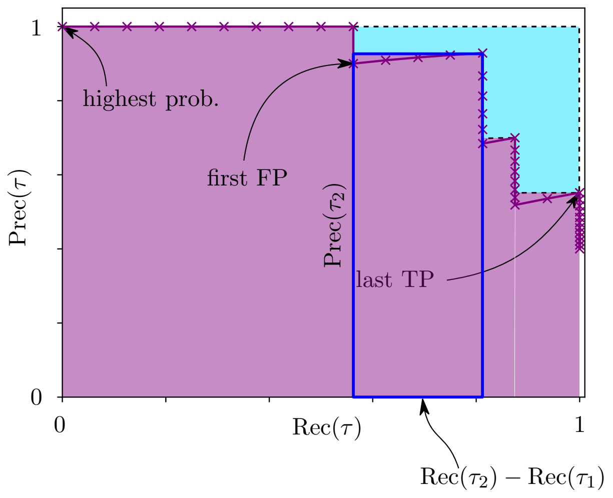


Figure 2.10: Precision-recall curve with prediction samples indicated by crosses along the curve. Upper bound areas of the average-precision area are shown with the remainder area in light blue (top right). This curve corresponds to an average precision of 0.904.

such that $\mathcal{D} = \mathrm{PP}(\tau) \cup \mathrm{PN}(\tau)$. Predictions can then be true or false depending on whether they agree with the labels given by $\mathcal{D}$ or not. The sets of *true positives* (TP) and *true negatives* (TN) and, respectively, *false positives* (FP) and *false negatives* (FN) are given by

$$\mathrm{TP}(\tau) := \mathrm{P} \cap \mathrm{PP}(\tau), \qquad \mathrm{TN}(\tau) := \mathrm{N} \cap \mathrm{PN}(\tau), \tag{2.106}$$

$$\mathrm{FP}(\tau) := \mathrm{N} \cap \mathrm{PP}(\tau), \qquad \mathrm{FN}(\tau) := \mathrm{P} \cap \mathrm{PN}(\tau). \tag{2.107}$$

One then defines the threshold-dependent fraction quantities of *precision* (Prec) and *recall* (Rec)

$$\mathrm{Prec}(\tau) := \frac{|\mathrm{TP}(\tau)|}{|\mathrm{PP}(\tau)|} \in [0,1], \qquad \mathrm{Rec}(\tau) := \frac{|\mathrm{TP}(\tau)|}{|\mathrm{P}|} \in [0,1]. \tag{2.108}$$

That is, the precision (or “positive predictive value”, PPV) is the fraction of correctly classified samples out of all samples that were predicted as positives. This can also be seen as the model’s accuracy conditioned on $\mathrm{PP}(\tau)$. Recall (or “true positive rate”) is the fraction of correctly identified ground truth positive samples.

For $\tau = 1$, precision is not defined, however, a well-performing model will give high class-1-probability $\widehat{f}_1$ to samples that tend to be true. Ordering samples in $\mathcal{D}$ in descending order with respect to $\widehat{f}_1$ will tend to have TP samples high up in the ordering and TN samples far down the ordering. Sweeping τ from high to low values yields a sequence of $(\mathrm{Rec}(\tau), \mathrm{Prec}(\tau))$-tuples which can be drawn graphically as a precision over recall curve, see fig. 2.10. The first point of the curve is the sample obtaining the highest probability $\widehat{f}_1$ starting in the top left of the curve. Up until the first FP is found, recall increases at constant precision. Precision drops with every FP found at constant recall. Another found TP after the first FP increases both, precision (compared with the previous point) and recall (overall) leading to a saw tooth pattern of the curve. Finally, with small enough threshold τ, all positive samples will be predicted as positive, leading to points with recall 1. Further positive predictions will be FPs since only negative samples remain, leading to drops in precision. Based on precision-recall curves, the *area under precision-recall curve* (AuPRC) is given as the Stieltjes integral

$$\text{AuPRC} = \int_0^1 \text{Prec}(\tau)\,\text{dRec}(\tau) \in [0,1]. \tag{2.109}$$

As an approximation, the average precision (AP) with descending thresholds $\tau_1, \ldots, \tau_{|\text{TP}|}$ before each new TP prediction,

$$\text{AP} = \sum_{j=2}^{|\text{TP}|} (\text{Rec}(\tau_j) - \text{Rec}(\tau_j - 1)) \cdot \text{Prec}(\tau_j) \in [0,1] \tag{2.110}$$

is usually computed. Both metrics are equal to 1 for a perfectly separating classifier. This means that in descending probability order, first only correct predictions are made until Rec = 1 and then only FPs follow. *Average precision* is an upper bound of the AuPRC which is often used in practice and given by the area shown in fig. 2.10 consisting of rectangles.

F_1-Score. The *F_1-score* is a combined performance measure of the precision and recall and, therefore, threshold-dependent. In particular, the F_1-score is the harmonic mean

$$F_1(\tau) = \frac{2}{\frac{1}{\text{Prec}(\tau)} + \frac{1}{\text{Rec}(\tau)}} \tag{2.111}$$

of precision and recall. The F_1-score a special case of the F_β-score which is a weighted average

$$F_\beta(\tau) = (1+\beta^2)\frac{\text{Prec}(\tau)\cdot\text{Rec}(\tau)}{\beta^2\cdot\text{Prec}(\tau)+\text{Rec}(\tau)}, \tag{2.112}$$

where $\beta > 0$ determines how much weight precision has relative to recall. As these metrics are weighted means between precision and recall, higher values also indicate better classification.

Area under the ROC Curve. The *receiver operating characteristic (ROC) curve* is, similarly to precision-recall, a curve obtained by sweeping a threshold τ. Here, recall, i.e., true positive rate, is drawn in dependence of the false positive rate

$$\text{FPR}(\tau) = \frac{|\text{FP}(\tau)|}{|\text{N}|}. \tag{2.113}$$

See fig. 2.11 for an illustration. With predictions sorted again with descending positive class probability, the curve starts with a recall of 0 and also an FPR of 0. The recall value increases with the high-probability predictions being TPs until the first FP prediction. This increases the FPR and leaves the recall constant. Therefore, steps of the curve are parallel to the axes. Similarly to precision-recall, the curve remains at constant recall of

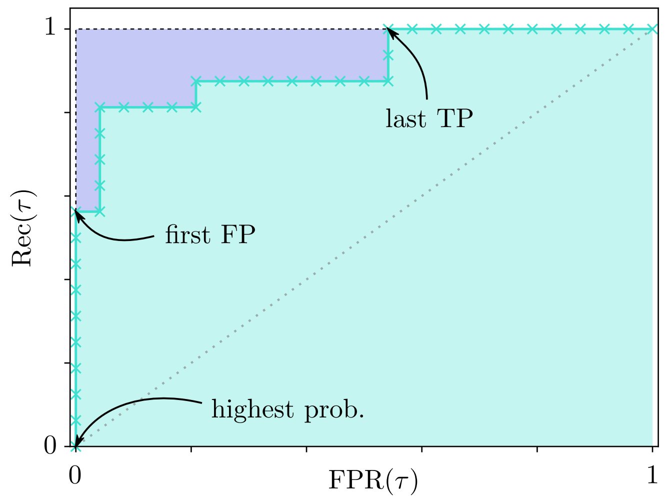


Figure 2.11: ROC curve of a binary classifier with prediction samples indicated as crosses along the curves. The area under the curve is shown with the remainder in dark blue at the top left. This curve corresponds to an AuROC of 0.909.

1 after the last TP prediction, from where the FPR increases to 1. The area under ROC curve (AuROC)

$$\text{AuROC} = \int_0^1 \text{Rec}(\tau)\, \text{dFPR}(\tau) \in [0,1] \tag{2.114}$$

is a common performance measure of binary classifiers[30]. A perfect classifier has AuROC = 1 whereas a classifier always giving the wrong answer will have AuROC = 0. A random decision will show a ROC curve close to the diagonal between $(0,0)$ and $(1,1)$ and, therefore, have an AuROC of around 0.5.

2.3.1.3 Example Architectures for Image Classification

Model architectures for image classification, object detection and semantic segmentation tend to roughly follow typical templates for each individual task. In image classification specifically, architectures usually consist of a backbone and a classification head. The backbone is designed to extract features from the input $\boldsymbol{x} \in [0,1]^{3\times H\times W}$. To this end, convolutional or transformer layers (see section 2.2.1.2, respectively section 2.2.1.3) have proven especially useful. Typically, the spatial resolution of the feature maps is reduced (and, therefore, the overall number of activation values is also reduced) iteratively with the depth of the network. This leads to information compression during the forward pass through the backbone. The output of the backbone consists of a number of feature maps which are either flattened to a one-dimensional vector or some global[31] pooling operation is performed over each channel.

[30]Note, that in the multi-class setting, sorting by probability is ambiguous. Hence, AP and AuROC can only ever be defined in terms of class-wise binary one-versus-all classification.

[31]This means that the entire spatial extent of the feature map is the receptive field. This leads to a single output number per channel.

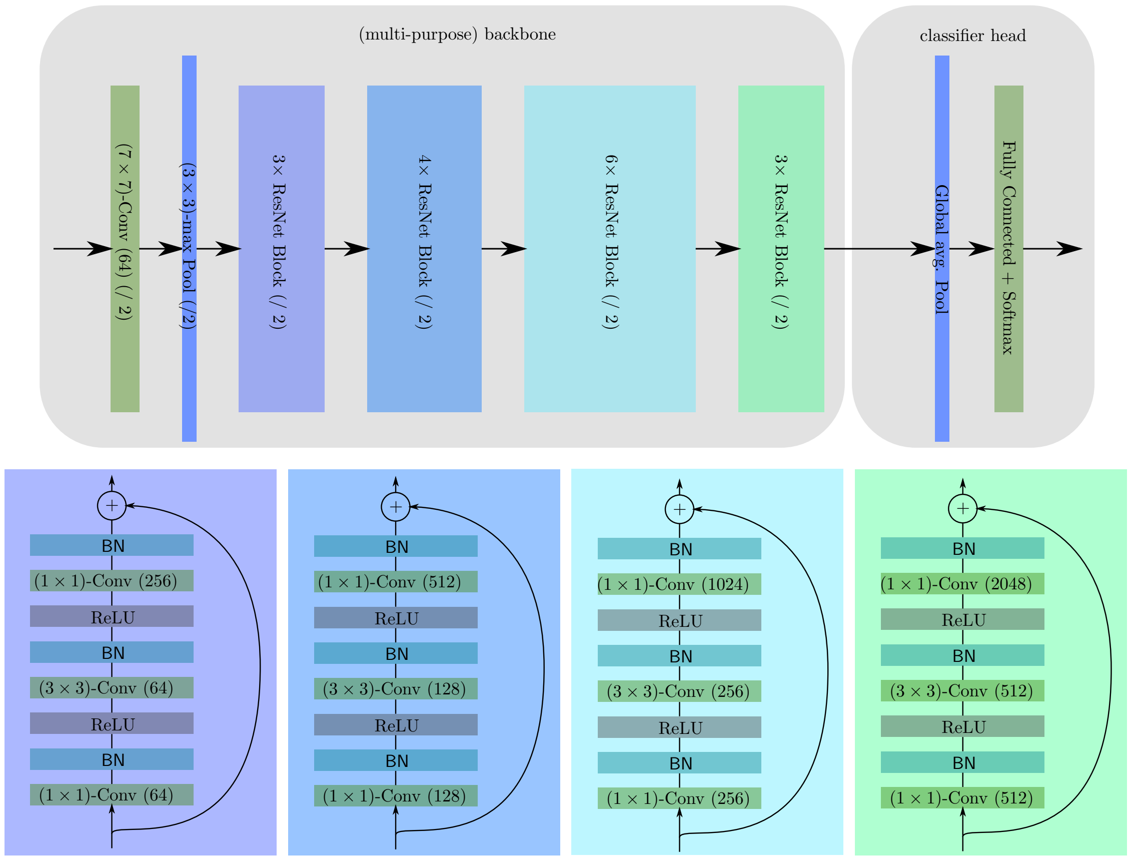


Figure 2.12: Illustration of the ResNet50 architecture [61]. The model is composed of a backbone network which can also serve as a feature extractor for an object detection or semantic segmentation model and a classifier head. In the classifier, feature maps are compressed to a one-dimensional vector on which a fully connected layer acts as a classifier. The different repeating ResNet blocks are depicted below the ResNet architecture and "(/ 2)" indicates a reduction of the feature map size by a factor of 2 in width and height.

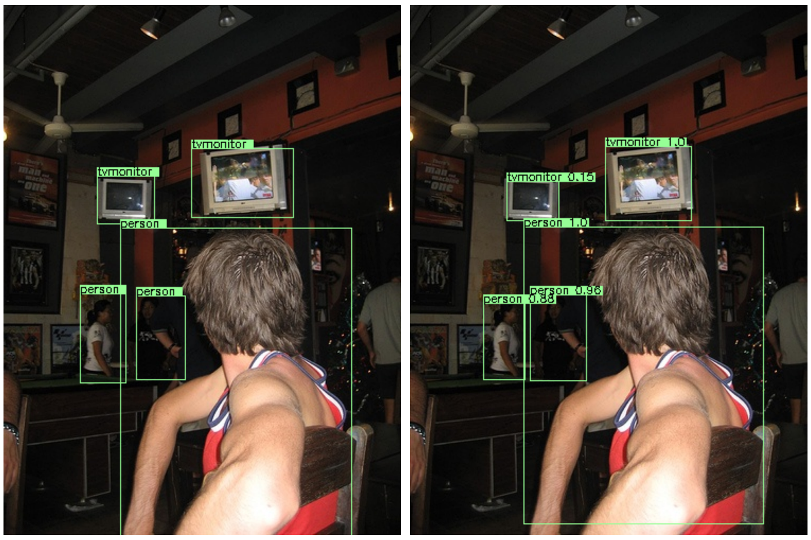


Figure 2.13: Object detection ground truth annotation (*left*) and prediction (*right*). The base image is taken from the Pascal VOC [38] 2007 test dataset. The prediction was produced by the YOLOv3 model with Darknet53 backbone used in the investigations in chapter 3. The ground truth consists of bounding box localization and category. Meanwhile, the final prediction has localization, assigned category and confidence score assigned. Usually, the category is determined from a predicted probability distribution.

Several popular architectures follow this scheme like the LeNet [92] family of architectures, AlexNet [86], ResNet [61] but also Swin transformer [108] models. In fig. 2.12, the ResNet50 model is illustrated. The backbone consists of a convolution layer with a large (7×7)-filter and a max pooling layer. Afterwards, groups of consecutive ResNet blocks (recall section 2.2.1.2) with varying number of out-going channels, denoted in parentheses, are stacked until the global average pooling. The latter is the first layer belonging to the classification head. A single fully connected layer takes care of the input classification and yields a softmax probability distribution $\widehat{f}(\cdot|\boldsymbol{\theta})$. Here, $\boldsymbol{\theta}$ are the fully-connected weights, filter weights, biases and batch norm parameters in the architecture.

2.3.2 Object Detection in Camera Images: Recognizing Foreground Instances

In object detection, the goal is to detect foreground instances of one of a predefined list of categories $[C]$ on a given input image $\boldsymbol{x} \in \mathcal{X} = [0,1]^{3\times H\times W}$. Detection is done by *defining axis-aligned bounding boxes* around each object tightly enclosing all pixels belonging to that object. Throughout the course of previous research, object detection has been a subject which progressed strongly due to deep learning engineering advancements. One of the consequences is that formulations in the literature [39,47,101,142,144] differ substantially. This makes it challenging to treat the object detection task in a manner reflecting how it fits into the theoretical framework presented in section 2.1.1. Another reason for this difficulty may be the task itself which deals with variable numbers of predicted instances. This makes the definition of the target space $\mathcal{Y}$ and of targets in datasets somewhat ambiguous and conceals how they relate with implementations. The following sections are an attempt to unify the different approaches which were presented throughout the years. The goal is to have a common notation while staying close to the concepts in section 2.1.1 as well as actual implementations. Hence, the following sections have a survey-like character.

For simplicity, we assume here, that the center point of each object to be detected must be contained in the image. The target space can be defined to be

$$\mathcal{Y} := \left(\mathbb{R}^2 \times [C] \times \{0,1\}\right)^{H \times W}. \tag{2.115}$$

We allow for each pixel to define the center point of exactly one object. The factor $\mathbb{R}^2$ then defines the spatial extent of the bounding box, i.e., the total width and height where the center is defined by the pixel position. The last factor $\{0,1\}$ defines whether there is a bounding box to be detected with center point at pixel $(i,j) \in \mathcal{I}$ or not. Annotations in object detection are often represented as a list $\overline{y} = \{\mathrm{b}^1, \dots, \mathrm{b}^N\}$ of bounding boxes

$$\mathrm{b}^n = (\mathrm{x}^n, \mathrm{y}^n, \mathrm{w}^n, \mathrm{h}^n, \kappa^n), \qquad n \in [N]. \tag{2.116}$$

Existent foreground instances are encoded by their center pixel $(\mathrm{x}^n, \mathrm{y}^n) \in \mathcal{I}$, spatial extent is given by w^n and h^n and their category given by κ_n. The corresponding target y has

$$y_{\mathrm{y}^n, \mathrm{x}^n} = (\mathrm{w}^n, \mathrm{h}^n, \kappa^n, 1) \tag{2.117}$$

for $n \in [N]$ and $y_{i,j} = (*, *, *, 0)$. Otherwise, it may have arbitrary entries for $\mathbb{R}^2 \times [C]$ since only the information about the absence of any object is relevant to the learning task. We denote the localization component of any ground truth box by $\xi^n := (\mathrm{x}^n, \mathrm{y}^n, \mathrm{w}^n, \mathrm{h}^n)$, i.e., with slight abuse of notation $\mathrm{b}^n = (\xi^n, \kappa^n)$.

An object detection model $\widehat{f}(\cdot|\boldsymbol{\theta})$ usually makes bounding box predictions on a coarsened space $\widehat{\mathcal{Y}}$ where the image resolution is down-scaled by some fixed stride ς

$$\widehat{f}(\cdot|\boldsymbol{\theta}) : \mathcal{X} \to \left(\mathbb{R}^4 \times [0,1]^C \times [0,1]\right)^{N_{\mathrm{anch}} \times \lceil \frac{H}{\varsigma} \rceil \times \lceil \frac{W}{\varsigma} \rceil} =: \widehat{\mathcal{Y}}. \tag{2.118}$$

The stride[32] ς introduces cells arranged in a grid $\mathcal{I}/\varsigma^2 := [\lceil H/\varsigma \rceil] \times [\lceil W/\varsigma \rceil]$ over $\mathcal{I}$ on which *localization coordinates* $\widehat{\xi} = (\widehat{\mathrm{x}}, \widehat{\mathrm{y}}, \widehat{\mathrm{w}}, \widehat{\mathrm{h}}) \in \mathbb{R}^4$ are predicted. See fig. 2.14 for an illustration. Here, $(\widehat{\mathrm{x}}, \widehat{\mathrm{y}})$ is the center point of the bounding box which obtains two additional regression variables to counteract the coarsened resolution of the prediction. A *class probability distribution* $\widehat{\pi} = (\widehat{\pi}_1, \dots, \widehat{\pi}_C) \in [0,1]^C$ is predicted alongside a *confidence or objectness score* $\widehat{s} \in [0,1]$ which indicates the probability of the box defined by $\widehat{\xi}$ containing an object. In order to allow for more than one prediction per cell, $N_{\mathrm{anch}} \in \mathbb{N}$ bounding

[32]This stride is a hyperparameter of the model and, therefore, necessary to define $\widehat{f}$. Similarly, the number N_{Anch} is a hyperparameter of the model.

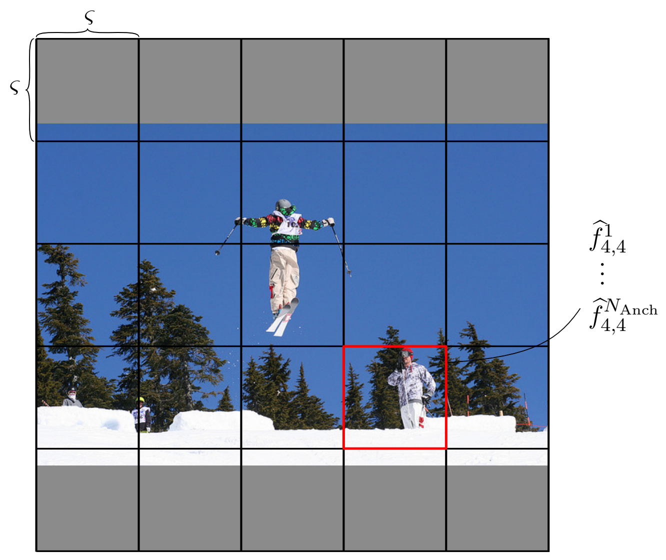


Figure 2.14: Quadratic prediction grid of size (5×5) over a gray-padded input image from the MS COCO [102] 2017 validation dataset.

boxes[33] are predicted for each cell. A model $\widehat{f}(\cdot|\boldsymbol{\theta})$ then defines predictions

$$\begin{aligned}\widehat{f}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) &:= \Big(\widehat{\xi}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}), \widehat{\pi}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}), \widehat{s}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\Big), \\ \widehat{\xi}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) &:= \Big(\widehat{\mathrm{x}}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}), \widehat{\mathrm{y}}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}), \widehat{\mathrm{w}}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}), \widehat{\mathrm{h}}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\Big), \\ \widehat{\pi}^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) &:= \Big((\widehat{\pi}_1)^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}), \ldots, (\widehat{\pi}_C)^{n}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\Big).\end{aligned} \tag{2.119}$$

This leads to $N_{\mathrm{anch}} \times \lceil H/\varsigma \rceil \times \lceil W/\varsigma \rceil$ possible predictions per input $\boldsymbol{x} \in \mathcal{X}$ which we call "*proposal boxes*" from which only a small amount is desired as the *model prediction* in the end in order to assess quality. Therefore, two crucial filter mechanisms are in place, both of which depend on a threshold hyperparameter. Proposal boxes with a confidence score $\widehat{s} < \tau_s$ for a confidence threshold $\tau_s \in [0, 1]$ are not considered for the final prediction. In practice, this leads to leftover clusters of boxes which tend to concentrate around objects that are found by the detector. A widely-established algorithm called non-maximum suppression reduces such clusters to individual bounding boxes based on a measure of localization accuracy. The measure usually employed to this end is called the Jaccard index [74] or "Intersection over Union" which is defined in the following paragraph.

Intersection over Union. The *intersection over union* (IoU, or *Jaccard index*) is a convenient metric to measure how close two bounding boxes ξ_1 and ξ_2 are to each other. We identify a bounding box $\xi = (\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h})$ with the set of points in $\mathbb{R}^2$ contained in the

[33] Predicted bounding boxes are in practice often defined via offsets of so-called "anchor boxes", see sections 2.3.2.1 and 2.3.2.3. Anchor boxes are priors either obtained via pre-processing or by an attempt to cover different possible aspect ratios of bounding boxes.

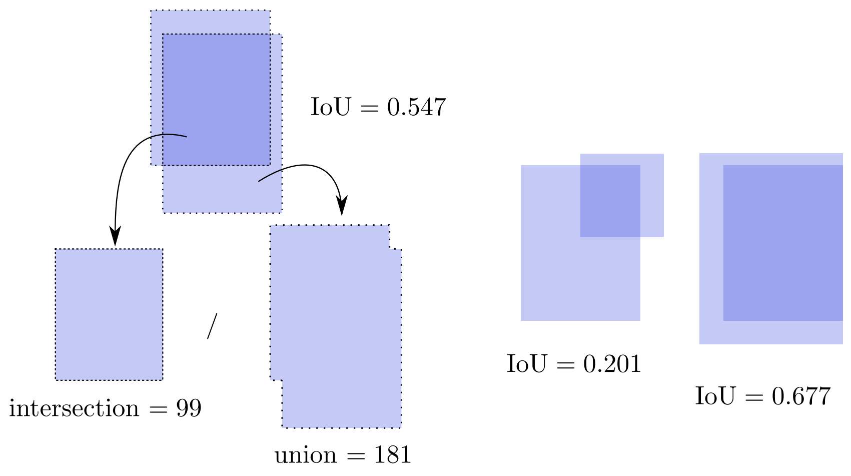


Figure 2.15: Illustration of the intersection over union localization quality estimate.

bounding box

$$\mathrm{set}(\xi) := \left\{(x,y) \in \mathbb{R}^2 : x \in [\mathrm{x} - \tfrac{\mathrm{w}}{2}, \mathrm{x} + \tfrac{\mathrm{w}}{2}], y \in [\mathrm{y} - \tfrac{\mathrm{h}}{2}, \mathrm{y} + \tfrac{\mathrm{h}}{2}]\right\}. \tag{2.120}$$

Then, the IoU is the fraction of area of intersection and the area of union between the two bounding boxes

$$\mathrm{IoU}(\xi_1, \xi_2) := \frac{\mathrm{d}x\,(\mathrm{set}(\xi_1) \cap \mathrm{set}(\xi_2))}{\mathrm{d}x\,(\mathrm{set}(\xi_1) \cup \mathrm{set}(\xi_2))} \in [0,1]. \tag{2.121}$$

See fig. 2.15 for a graphical illustration of the IoU between different combinations of bounding boxes. This definition of the IoU in fact generalizes to arbitrary measurable sets of $\mathbb{R}^2$. In particular, IoU is applicable to segments obtained in semantic segmentation[34]. The IoU is a localization quality metric taking on its maximum 1 for matching (up to sets of measure 0) subsets of $\mathbb{R}^2$. Therefore, the IoU also quantifies *how close* two bounding boxes are to each other and allows to identify clusters of bounding boxes.

A predicted bounding box $\widehat{\xi}$ is counted as a TP prediction, if and only if it has maximal IoU with any ground truth instance above some threshold $\tau_{\mathrm{IoU}}^{\mathrm{TP}}$, i.e., if and only if

$$\max_{\overline{\xi}:(\overline{\xi},\overline{\kappa})\in\overline{y}} \mathrm{IoU}\left(\widehat{\xi}, \overline{\xi}\right) > \tau_{\mathrm{IoU}}^{\mathrm{TP}}. \tag{2.122}$$

Otherwise, $\widehat{\xi}$ is counted as a FP. A typical value of $\tau_{\mathrm{IoU}}^{\mathrm{TP}}$ is 0.5. A ground truth box $(\xi, \kappa) \in \overline{y}$ is called a FN, iff there is no TP $\widehat{\xi}$ with $\mathrm{IoU}(\xi, \widehat{\xi}) > \tau_{\mathrm{IoU}}^{\mathrm{TP}}$. That is, no predicted box is sufficiently closely located to ξ. One does not speak of TNs in object detection since the prediction of the absence of a foreground object is ambiguous.

Non-Maximum Suppression. *Non-maximum suppression* (NMS) is a reduction mechanism for localization predictions. This algorithm has long been used in computer vision before deep learning became the state-of-the-art and is now a common step in

[34]Technically, segments are discretized on pixel-level and are, therefore, made up of rectangles.

object detection post-processing [151]. Given a set $B := \{(\xi_1, s_1), \dots, (\xi_n, s_n)\}$ of tuples consisting of n bounding boxes ξ_i and some scores[35] s_i ordered descendingly by s_i, i.e., the first tuple (ξ_1, s_1) has the highest score. NMS filters the set B iteratively. Namely, the first sample (ξ_1, s_1) with the highest score will "suppress" any other $(\xi_i, s_i) \in B$ with $\mathrm{IoU}(\xi_1, \xi_i) > \tau_{\mathrm{IoU}}^{\mathrm{NMS}}$ for some IoU-threshold $\tau_{\mathrm{IoU}}^{\mathrm{NMS}} \in [0, 1]$. This means that any such box is removed from B. The remaining ordered set

$$B(\xi_1) = \{(\xi_1^{(1)}, s_1^{(1)}), \dots, (\xi_{n_1}^{(1)}, s_{n_1}^{(1)})\} \subset B \tag{2.123}$$

only contains samples $(\xi_i^{(1)}, s_i^{(1)}) \in B$ which have not been suppressed by (ξ_1, s_1) and, therefore, have lower or equal IoU with it. The NMS algorithm applies the same filtering in the next iteration with $B(\xi_1)$ with respect to the bounding box $(\xi_1^{(1)}, s_1^{(1)})$ having the highest score in $B(\xi_1)$. Again, boxes in $B(\xi_1)$ are suppressed which in turn constructs a set $B(\xi_1, \xi_1^{(1)}) \subset B(\xi_1)$. Bounding boxes in $B(\xi_1, \xi_1^{(1)})$ now have low IoU with both, ξ_1 and $\xi_1^{(1)}$. This algorithm continues iteratively like so and stops as soon as

$$B(\xi_1, \xi_1^{(1)}, \dots, \xi_1^{(k)}) = \emptyset, \tag{2.124}$$

i.e., when no further boxes are available. We then define the NMS result on B by

$$\mathrm{NMS}(B) = \{(\xi_1, s_1), (\xi_1^{(1)}, s_1^{(1)}), \dots, (\xi_1^{(k)}, s_1^{(k)})\}. \tag{2.125}$$

In some cases, it is also desirable to regard triples ξ_i, κ_i, s_i where κ_i is a class in $[C]$. One then sets $\mathrm{IoU}(\xi_1, \xi_i) = 0$ if the class $\kappa_1 \neq \kappa_i$ disagree. This is done in order to prevent predicted bounding boxes with similar localization but different predicted class to suppress each other. Typical values of $\tau_{\mathrm{IoU}}^{\mathrm{NMS}}$ are 0.5 and 0.7.

2.3.2.1 Loss Functions in Object Detection: Semantics, Localization and Objectness

In order to efficiently compute losses from the prediction $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ (see eq. (2.118)) and a given ground truth $\overline{y}$, foreground instances in $\overline{y}$ are "*assigned*" to a subset of the N_{anch} anchor boxes. The *anchor boxes* themselves are independent of the neural network weights and also of the input $\boldsymbol{x}$ and define fixed bounding boxes

$$\mathrm{Anch}\left(\widehat{f}\right) = \left\{\widetilde{a}_{i,j}^n = \left(\widetilde{\mathrm{x}}_{i,j}^n, \widetilde{\mathrm{y}}_{i,j}^n, \widetilde{\mathrm{w}}_{i,j}^n, \widetilde{\mathrm{h}}_{i,j}^n\right)\right\}_{(i,j)\in\mathcal{I}/\varsigma^2}^{n\in[N_{\mathrm{Anch}}]} \tag{2.126}$$

distributed over $\mathcal{I}/\varsigma^2$. Anchor boxes can be understood as being part of the model architecture. The *assignment* can then be understood as a function from indices, encoding the anchor position and shape, to an extended ground truth

$$\mathrm{Asgn}_{\widehat{f},\overline{y}}(\cdot) : (\mathcal{I}/\varsigma^2) \times [N_{\mathrm{Anch}}] \to \overline{y} \cup \{\emptyset, \mathrm{BG}\}. \tag{2.127}$$

This assignment specifies how the activations for an anchor receive training feedback from the ground truth $\overline{y}$. Here, $\emptyset$ means that no assignment is made, neither foreground nor

[35]This does not necessarily have to be the objectness score of an object detector, although that is the most important case of application for us. We will, therefore, stick with the notation.

background. Anchors with $\emptyset$ assigned usually receive no training feedback from $\overline{y}$. BG means that the anchor receives "background" feedback during training, i.e., should have low objectness activations. The assignment depends explicitly on the entire ground truth $\overline{y}$ since it is usually constructed algorithmically by the following steps:

1. By default, anchors first point to background, i.e., $\mathrm{Asgn}_{\widehat{f},\overline{y}}(i,j,n) = \mathrm{BG}$ for all $(i,j,n) \in (\mathcal{I}/\varsigma^2) \times [N_{\mathrm{Anch}}]$.
2. Iteratively over $\mathrm{b} \in \overline{y}$, all anchors with sufficient $\mathrm{IoU} > \tau_{\mathrm{IoU}}^{\mathrm{Asgn},\emptyset}$ are assigned neutrally, i.e., $\emptyset$. Here, $\tau_{\mathrm{IoU}}^{\mathrm{Asgn},\emptyset} \in [0,1]$ is a threshold which typically has values 0.5 or 0.7, i.e.,
$$\mathrm{IoU}\left(\widetilde{a}_{i,j}^n, \mathrm{b}\right) > \tau_{\mathrm{IoU}}^{\mathrm{Asgn},\emptyset} \implies \mathrm{Asgn}_{\widehat{f},\overline{y}}(i,j,n) = \emptyset. \tag{2.128}$$
3. Again, iterating over $\mathrm{b} \in \overline{y}$, all anchors with sufficient $\mathrm{IoU} > \tau_{\mathrm{IoU}}^{\mathrm{Asgn},\overline{y}}$ with $b \in \overline{y}$ will be assigned b. That is,
$$\mathrm{IoU}\left(\widetilde{a}_{i,j}^n, \mathrm{b}\right) > \tau_{\mathrm{IoU}}^{\mathrm{Asgn},\overline{y}} \implies \mathrm{Asgn}_{\widehat{f},\overline{y}}(i,j,n) = \mathrm{b} \tag{2.129}$$
for all $(i,j,n) \in (\mathcal{I}/\varsigma^2) \times [N_{\mathrm{Anch}}]$ where assignment is iteratively over some ordering of $\mathrm{b} \in \overline{y}$. Here, $\tau_{\mathrm{IoU}}^{\mathrm{Asgn},\overline{y}} \geq \tau_{\mathrm{IoU}}^{\mathrm{Asgn},\emptyset}$ is again a threshold in $[0,1]$. This enforces closer localization for assigning to b.
4. Lastly, with the goal of obtaining an assignment of at least one anchor to each $\mathrm{b} \in \overline{y}$, an anchor with maximal IoU with b is assigned to each b. That is,
$$(i,j,n) \in \underset{(i,j,n)\in(\mathcal{I}/\varsigma^2)\times[N_{\mathrm{Anch}}]}{\mathrm{argmax}} \mathrm{IoU}(\widetilde{a}_{i,j}^n, \mathrm{b}) \implies \mathrm{Asgn}_{\widehat{f},\overline{y}}(i,j,n) = \mathrm{b}. \tag{2.130}$$

The precise mapping defined by the assignment depends on an ordering imposed on $\overline{y}$ and does not necessarily guarantee that there is at least one anchor assigned to each ground truth box[36] $\mathrm{b} \in \overline{y}$. However, such cases are rare and pose no obstruction in applications.

Loss functions in object detection can be divided into roughly three categories which are responsible for the estimations $\widehat{\pi}$, $\widehat{\xi}$ and $\widehat{s}$, respectively.

Classification Losses. In order to estimate a probability distribution $\widehat{\pi} = (\widehat{\pi}_1, \ldots, \widehat{\pi}_C)$ over all possible classes, usually the standard *cross entropy loss* (cf. eq. (2.101)) is used between ground truth boxes (ξ, κ) and prediction generated from assigned anchors. That is,
$$\mathcal{L}_{\mathrm{CE}}^{\mathrm{Asgn}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := \sum_{(\xi,\kappa)\in\overline{y}} \sum_{(i,j,n)\in\mathrm{Asgn}_{\widehat{f},\overline{y}}^{-1}(\{\xi\})} \mathcal{L}_{\mathrm{CE}}\left(\widehat{\pi}_{i,j}^n(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\kappa\right). \tag{2.131}$$

An adaptation of this loss function involves a $(C+1)$-st class probability $\widehat{\pi}_0$. The latter indicates the probability of a prediction $\widehat{f}_{i,j}^n$ belonging to the background class. This adaptation plays a role when the detector does not have a separate objectness score $\widehat{s}$.

[36]This can happen by having simply two ground truth boxes with equal localization but different categorical label. Moreover, assignments in step 4 can overwrite earlier assignments from steps 3 and 4.

In that case, the assignment of ground truth instances still works the same, however, an additional term is responsible for learning background feedback:

$$\mathcal{L}^{\mathrm{Asgn}}_{\mathrm{CE},C+1}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := \mathcal{L}^{\mathrm{Asgn}}_{\mathrm{CE}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) - \sum_{(i,j,n)\in\mathrm{Asgn}^{-1}_{\widehat{f},\overline{y}}(\{\mathrm{BG}\})} \log\left((\widehat{\pi}_0)^n_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\right). \tag{2.132}$$

The second term is the *negative log-likelihood* of the background class prediction and applies to all BG-assigned anchors. Alternatively, sometimes classification is learned in a one-vs-all fashion with class-wise sigmoid activation instead of $\widehat{\pi}$ being the result of a softmax activation. In such a case, $\widehat{\pi}$ is generally no longer a probability distribution due to lack of normalization. The loss function then involves the class-wise binary cross entropy:

$$\mathcal{L}^{\mathrm{Asgn}}_{\mathrm{BCE}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := \sum_{(\xi,\kappa)\in\overline{y}}\ \sum_{(i,j,n)\in\mathrm{Asgn}^{-1}_{\widehat{f},\overline{y}}(\{\xi\})} \mathcal{L}_{\mathrm{BCE}}\left(\widehat{\pi}^n_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\kappa\right), \tag{2.133}$$

where

$$\mathcal{L}_{\mathrm{BCE}}\left(\widehat{\pi}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\kappa\right) := -\sum_{c=1}^{C}\delta_{\kappa c}\log(\widehat{\pi}_c) + (1-\delta_{yc})\log(1-\widehat{\pi}_c) \tag{2.134}$$

is the sum of class-wise binary cross entropy values.

Bounding Box Regression Losses. For the estimation of bounding box localization $\widehat{\xi}$, classical regression losses such as the *MSE loss* for regression variables in d dimensions can be applied. Such a loss

$$\mathcal{L}_{\mathrm{MSE}}\left(\widehat{\xi}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\xi\right) := \frac{1}{d}\sum_{l=1}^{d}\left(\widehat{\xi}_l(\boldsymbol{x}|\boldsymbol{\theta}) - \xi_l\right)^2 \tag{2.135}$$

is applied to foreground-assigned anchors in which case $d = 4$:

$$\mathcal{L}^{\mathrm{Asgn}}_{\mathrm{MSE}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := \sum_{(\xi,\kappa)\in\overline{y}}\ \sum_{(i,j,n)\in\mathrm{Asgn}^{-1}_{\widehat{f},\overline{y}}(\{\xi\})} \mathcal{L}_{\mathrm{MSE}}\left(\widehat{\xi}^n_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\xi\right). \tag{2.136}$$

For large deviations, the quadratic increase of the L^2-loss leads to linearly increasing gradients. This can lead to overstepping local minima during gradient descent. Instead of an L^2-loss, the L^1-*loss*

$$\mathcal{L}^{\mathrm{Asgn}}_{L^1}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := \sum_{(\xi,\kappa)\in\overline{y}}\ \sum_{(i,j,n)\in\mathrm{Asgn}^{-1}_{\widehat{f},\overline{y}}(\{\xi\})} \frac{1}{d}\sum_{l=1}^{4}\left|\left(\widehat{\xi}^n_{i,j}\right)_l(\boldsymbol{x}|\boldsymbol{\theta}) - \xi_l\right| \tag{2.137}$$

can be applied leading to constant gradient steps. However, this has the downside of not being continuously differentiable everywhere. A compromise between the two which comes at the expense of a hyperparameter can be found in the so-called "*smooth L^1 distance*"

$$L^1_{\mathrm{sm},\beta}\left(\widehat{\xi}\middle|\,\xi\right) := \left\{\begin{array}{l|l} \frac{1}{2\beta}\left(\widehat{\xi}-\xi\right)^2 & |\widehat{\xi}-\xi| < \beta \\ \left|\widehat{\xi}-\xi\right| - \frac{\beta}{2} & |\widehat{\xi}-\xi| \geq \beta \end{array}\right. . \tag{2.138}$$

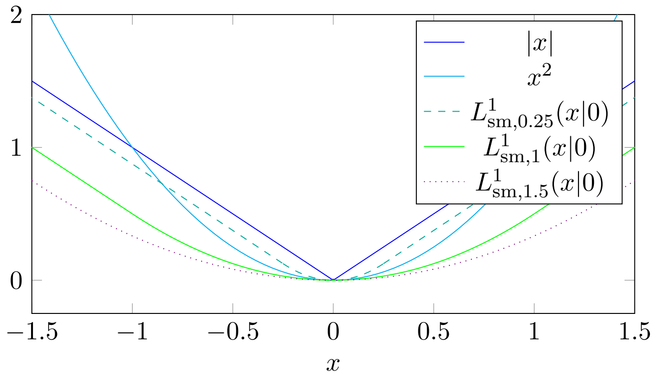


Figure 2.16: Comparison of the L^1 and L^2 distance with the smooth L^1 distance $L^1_{\text{sm},\beta}$ for different values of β.

Figure 2.16 shows plots of the smooth L^1 distance for different values of β. Here, $\beta > 0$ determines the point of transition from the parabola to the linear function. This function is continuously differentiable everywhere and has constant gradients for large differences $\left|\widehat{\xi} - \xi\right|$. The respective object detection loss for foreground instances then is

$$\mathcal{L}^{\text{Asgn}}_{L^1_{\text{sm},\beta}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := \sum_{(\xi,\kappa)\in\overline{y}} \sum_{(i,j,n)\in\text{Asgn}^{-1}_{\widehat{f},\overline{y}}(\{\xi\})} \frac{1}{d}\sum_{l=1}^{4} L^1_{\text{sm},\beta}\left(\left(\widehat{\xi}^n_{i,j}\right)_l(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\xi_l\right). \tag{2.139}$$

Objectness Losses. The distinction between foreground and background instances via the objectness score $\widehat{s}$ is oftentimes learned as a binary classification problem over $\{0,1\}$ for each anchor. Therefore, the binary cross entropy loss is given by

$$\begin{aligned}\mathcal{L}^{\text{Asgn}}_{\text{BCE},s}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := &\sum_{(\xi,\kappa)\in\overline{y}} \sum_{(i,j,n)\in\text{Asgn}^{-1}_{\widehat{f},\overline{y}}(\{\xi\})} \mathcal{L}_{\text{BCE}}\left(\widehat{s}^n_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,1\right)\\ &+ \sum_{(i,j,n)\in\text{Asgn}^{-1}_{\widehat{f},\overline{y}}(\{\text{BG}\})} \mathcal{L}_{\text{BCE}}\left(\widehat{s}^n_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,0\right).\end{aligned} \tag{2.140}$$

This is one of the standard choices of loss functions where the foreground label 1 is learned on all positive assignments and the background label 0 is learned for all background assignments. Since instances in most applications of object detection tend to be *sparsely distributed*, there are magnitudes more anchors assigned to the background class than to foreground instances. This can lead to far stronger background feedback than foreground feedback during training. In order to counteract this phenomenon, different approaches have been taken. In the BCE loss, *random subsampling* can be applied to obtain a fixed number of anchors in $\text{Asgn}^{-1}_{\widehat{f},\overline{y}}(\{\text{BG}\})$ for the loss computation. Another approach is taken in the RetinaNet architecture [101] where the *focal loss* for object detection was introduced. Here, both terms of the BCE loss are dynamically weighted by a power of the objectness score

$$\mathcal{L}_{\mathrm{Foc}}^{\mathrm{Asgn}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := -\sum_{(\xi,\kappa)\in\overline{y}}\ \sum_{(i,j,n)\in\mathrm{Asgn}_{\widehat{f},\overline{y}}^{-1}(\{\xi\})}\left(1-\widehat{s}_{i,j}^{n}\right)^{\gamma_{\mathrm{Foc}}}\log\left(\widehat{s}_{i,j}^{n}\right) \tag{2.141}$$

$$-\sum_{(i,j,n)\in\mathrm{Asgn}_{\widehat{f},\overline{y}}^{-1}(\{\mathrm{BG}\})}\left(\widehat{s}_{i,j}^{n}\right)^{\gamma_{\mathrm{Foc}}}\log\left(1-\widehat{s}_{i,j}^{n}\right). \tag{2.142}$$

The parameter $\gamma_{\mathrm{Foc}} \geq 0$ is a hyperparameter of the loss function and typically takes the value 2 which has worked well in a range of applications. The pre-factors $(1-\widehat{s}_{i,j}^{n})^{\gamma_{\mathrm{Foc}}}$ leads to a dynamic up-weighting if the respective anchor prediction $\widehat{s}_{i,j}^{n}$ was incorrect (close to 0 in the case of foreground assignments). Similarly, for correctly assigned objectness to background-assigned samples, the term $(\widehat{s}_{i,j}^{n})^{\gamma_{\mathrm{Foc}}}$ leads to a dynamic down-weighting of the background loss.

2.3.2.2 Evaluation Metrics in Object Detection: Mean Average Precision

Object detection performance is usually assessed by comparing the prediction of an object detector after objectness score thresholding and NMS. The performance, therefore, inherently depends on two thresholds τ_s and $\tau_{\mathrm{IoU}}^{\mathrm{NMS}}$. Typically, since TP, FP and FN are defined in object detection, one computes the *average precision metric* AP in object detection.[37] Usually, the dependency of the AP on the confidence threshold τ_s is not explicitly stated. Smaller τ_s usually only increases AP, see [39] for a discussion of the influence of additional FPs to the AP metric. Average precision $\mathrm{AP}\left(c;\tau_{\mathrm{IoU}}^{\mathrm{NMS}}\right)$ is computed for each class $c \in [C]$ separately and averaged afterwards. This metric is called the *mean average precision*

$$\mathrm{mAP}\left(\tau_{\mathrm{IoU}}^{\mathrm{NMS}}\right) := \frac{1}{C}\sum_{c=1}^{C}\mathrm{AP}\left(c;\tau_{\mathrm{IoU}}^{\mathrm{NMS}}\right) \tag{2.143}$$

which is dependent on the IoU threshold. As part of the MS COCO evaluation benchmark [102], the metrics

$$\mathrm{AP}_{50} := \mathrm{mAP}\left(0.5\right), \quad \mathrm{AP}_{75} := \mathrm{mAP}\left(0.75\right), \quad \mathrm{AP}_{[0.5:0.95]} := \frac{1}{10}\sum_{j=0}^{9}\mathrm{mAP}\left(0.5+0.05\cdot j\right) \tag{2.144}$$

have been established where in $\mathrm{AP}_{[0.5:0.95]}$ mean average precision values are averaged over 10 IoU thresholds between 0.5 and 0.95.

2.3.2.3 Example Architectures for Object Detection

Object detection architectures also broadly follow the scheme explained in section 2.3.1.3, and consist of a backbone which extracts characteristic features from the input and a

[37] While the AP is itself not threshold-dependent and predictions with arbitrarily small objectness scores may be included, predictions with $\widehat{s} < 0.1$ oftentimes have little influence on the AP. This is so because the area gain at already small precision (due to an abundance of FPs) is proportionally small.

detection head. There is a rough division of architectures into two types of object detectors: *single-stage* and *two-stage* object detectors. In single-stage detectors, the detection head directly produces feature map activations for each anchor box which also involve a class probability distribution and an objectness score. The YOLO family of object detectors [39, 142], RetinaNet [101] and SSD [106] are standard examples of single-stage detectors. We will have a closer look at YOLOv3 [39] and RetinaNet in the next paragraph. Two-stage detectors produce activations for so-called region-proposal anchors which first only specify via an objectness score and a bounding box regression a set of boxes which likely contain an object with unspecified class. This box, together with the feature map activations, is fed through a second stage of the network. In the second stage, bounding box localizations are refined and classified by estimating a class probability distribution. The R-CNN family of object detectors [47, 48, 144] are standard examples of two-stage detectors, out of which we will focus on Faster R-CNN [144].

Single-Stage Detectors: YOLOv3 and RetinaNet. In single-stage object detectors, the last feature activations generated from an input $\boldsymbol{x} \in \mathcal{X}$ of the neural network directly give rise to the proposal boxes $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta}) \in \widehat{\mathcal{Y}}$ by undergoing simple post-processing transformations. If we denote the activation of anchor $\widetilde{a} = (\widetilde{\mathrm{x}}, \widetilde{\mathrm{y}}, \widetilde{\mathrm{w}}, \widetilde{\mathrm{h}}) \in \mathrm{Anch}(\widehat{f}\,)$ (recall section 2.3.2.1) by

$$\widehat{\phi}_{\mathrm{x}}, \widehat{\phi}_{\mathrm{y}}, \widehat{\phi}_{\mathrm{w}}, \widehat{\phi}_{\mathrm{h}}, \qquad \widehat{\phi}_{\pi_1}, \ldots, \widehat{\phi}_{\pi_C}, \qquad \widehat{\phi}_s, \tag{2.145}$$

the proposal predictions for the YOLOv3 detector are given by

$$\begin{gathered} \widehat{\mathrm{x}} := \varsigma \cdot \sigma\left(\widehat{\phi}_{\mathrm{x}}\right) + \widetilde{\mathrm{x}}, \quad \widehat{\mathrm{y}} := \varsigma \cdot \sigma\left(\widehat{\phi}_{\mathrm{y}}\right) + \widetilde{\mathrm{y}}, \quad \widehat{\mathrm{w}} := \mathrm{e}^{\widehat{\phi}_{\mathrm{w}}} \cdot \widetilde{\mathrm{w}}, \quad \widehat{\mathrm{h}} := \mathrm{e}^{\widehat{\phi}_{\mathrm{h}}} \cdot \widetilde{\mathrm{h}}, \\ \widehat{\pi}_c := \sigma\left(\widehat{\phi}_{\pi_c}\right) \quad \forall c \in [C], \quad \widehat{s} := \sigma\left(\widehat{\phi}_s\right). \end{gathered} \tag{2.146}$$

Here, σ denotes the sigmoid function. In fact, the YOLOv3 architecture makes predictions on *three anchor grids* with different strides $\varsigma_1 = 2\varsigma_2 = 4\varsigma_3$. The architecture leading to the feature map activations on these three resolutions is depicted in fig. 2.17. From the Darknet53 backbone which is the standard backbone of the YOLOv3 architecture, three routes are fed into a feature pyramid network (FPN [100]) structure. The FPN structure is designed to allow for *predictions on different resolution* feature maps, taking into account features from coarser resolutions. At each "concatenate" block, the coarser feature maps are bi-linearly interpolated to the desired resolution and concatenated along the channel dimension to the incoming route features which have the finer resolution. Activations on the three grids are obtained as the output of the three "Detection" blocks. Typically, $N_{\mathrm{Anch}} = 3$ anchors are used for each grid cell where the original paper used 9 different fixed aspect ratios of $\widetilde{\mathrm{w}}/\widetilde{\mathrm{h}}$. Ratios were obtained from a k-means cluster algorithm over MS COCO [102] ground truth bounding boxes.

Prediction proposals $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ are gathered from all three grids and NMS is performed on all proposals during the forward pass. During training, anchor matching is performed on each grid separately, leading also to grid-wise loss computation for localization, classification and objectness. The original implementation with input resolution $H \times W = 608 \times 608$ uses strides $\varsigma_1 = 32$. This leads to $3 \cdot (19^2 + 38^2 + 76^2) = 22.743$ proposal boxes in total where a separate objectness score is learned for each anchor. This moderate number

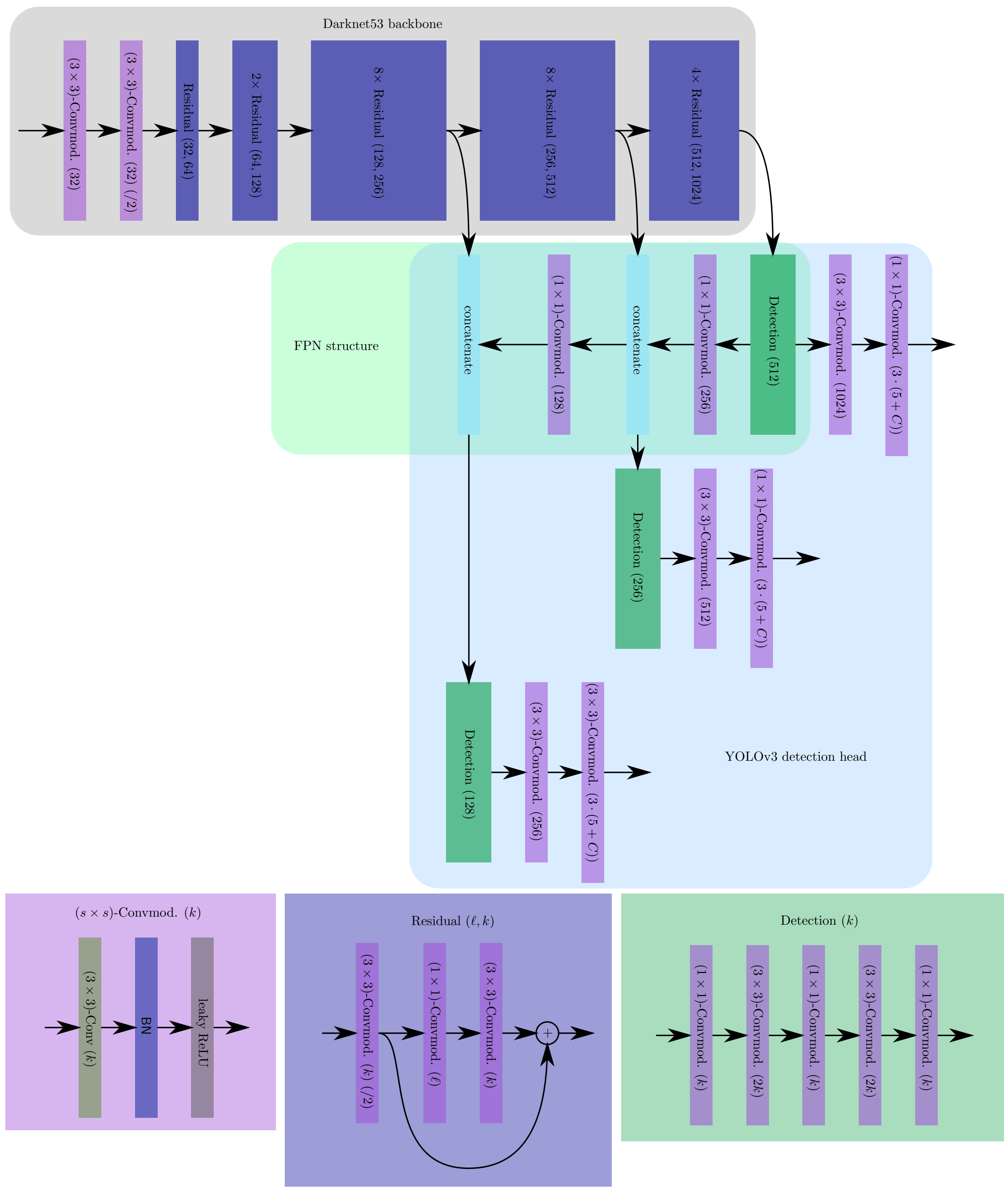


Figure 2.17: Illustration of the YOLOv3 architecture. The Darknet53 backbone consists of a sequence of simple residual blocks which reduce feature resolution (indicated by “(/2)”). Features are extracted at three different output routes and successively concatenated (FPN structure) to obtain detection results at different resolutions in the detection head.

of proposal boxes allows for an implementation with no special background sampling or weighting strategy during training as opposed to the RetinaNet detector.

The RetinaNet [101] detector follows a similar architectural design as YOLOv3 where a ResNet serves as backbone. An FPN extending the last three feature maps of different resolutions of the ResNet backbone to five feature resolutions via up-sampling. Two shared fully convolutional subnetworks consisting of five convolutional modules (see "Convmod." in fig. 2.17) take care of bounding box regression and classification for each of the cumulative ≈ 100.000 anchor boxes. Due to the inherent immense imbalance between foreground and background assignments during training, Lin et al. proposed to utilize the focal loss on each classification prediction $\widehat{\pi}_1, \ldots, \widehat{\pi}_C$ individually instead of defining a separate objectness score for each anchor. Therefore, the classifier subnetwork takes care of classification and foreground-background discrimination at the same time.

Two-Stage Detectors: Faster R-CNN. Two-stage detectors first produce a *region proposal detection* $\widehat{f}_{\mathrm{RPN}}(\boldsymbol{x}|\boldsymbol{\theta})$ which is a class-agnostic single-stage object detection where only foreground instances are detected. Training of the region proposal network (RPN) follows the same procedure as a single-stage object detector where often, $\tau_{\mathrm{IoU}}^{\mathrm{Asgn}} = 0.3$ and $\tau_{\mathrm{IoU}}^{\mathrm{Asgn}} = 0.7$. After objectness thresholding and NMS on the RPN proposals, a list of RPN detection remains for processing in the second stage. Based on extracted backbone features, the RPN detections are classified, and the localization is refined in the so-called *region of interest* (RoI) head. Assignment of the RoI output $\widehat{f}_{\mathrm{RoI}}(\boldsymbol{x}|\boldsymbol{\theta})$ to ground truth boxes follows the same procedure as in section 2.3.2.1 with $\mathrm{Anch}(\widehat{f}\,)$ replaced by the NMS output of $\widehat{f}_{\mathrm{RPN}}(\boldsymbol{x}|\boldsymbol{\theta})$.

A large amount of implementation of the Faster R-CNN architecture use a ResNet backbone with FPN as described in the last paragraph, also with five feature map resolutions. The RPN uses a single (3×3) convolution layer with ReLU activation. This generates a feature map with a fixed number of channels (oftentimes 256) which are fed through two "sibling" [144] convolution layers. The first convolution layer returns regression activations $(\widehat{\phi}_{\mathrm{x}}^{\mathrm{RPN}}, \widehat{\phi}_{\mathrm{y}}^{\mathrm{RPN}}, \widehat{\phi}_{\mathrm{w}}^{\mathrm{RPN}}, \widehat{\phi}_{\mathrm{h}}^{\mathrm{RPN}})$ for all RPN anchors. The second convolution layer gives an objectness score activation $\widehat{\phi}_s^{\mathrm{RPN}}$ for each anchor on the basis of which NMS can be performed. This entire convolutional RPN structure is used on each feature map resolution scale. Each RPN box $\widehat{\xi}^{\mathrm{RPN}} = (\widehat{\mathrm{x}}^{\mathrm{RPN}}, \widehat{\mathrm{y}}^{\mathrm{RPN}}, \widehat{\mathrm{w}}^{\mathrm{RPN}}, \widehat{\mathrm{h}}^{\mathrm{RPN}})$ is utilized to extract a small feature map patches of fixed size (oftentimes (7×7)) from the FPN feature map, see fig. 2.18. Such feature map patches are obtained from coarsened pooling (quantizing the alignment of the feature map pixels and $\widehat{\xi}^{\mathrm{RPN}}$: *RoI pooling*) or a weighted pooling (weighting the alignment of the feature map pixels with $\widehat{\xi}^{\mathrm{RPN}}$ according to coverage: *RoI alignment*). The resulting pooled feature map values are flattened and fed through two fully connected layers with ReLU activations. From there, one fully connected layer outputs classification activations $\widehat{\phi}_0^{\mathrm{RoI}}, \ldots, \widehat{\phi}_C^{\mathrm{RoI}}$ where 0 indicates the background class. Another fully connected layer yields C bounding box refinements $\widehat{\phi}_1^{\mathrm{RoI}}, \ldots \widehat{\phi}_C^{\mathrm{RoI}}$ (each being four-dimensional) for each RPN box. The ground truth box that is matched to $\widehat{\xi}^{\mathrm{RPN}}$ determines which class-dependent bounding box refinement contributes to the loss function. The region proposal

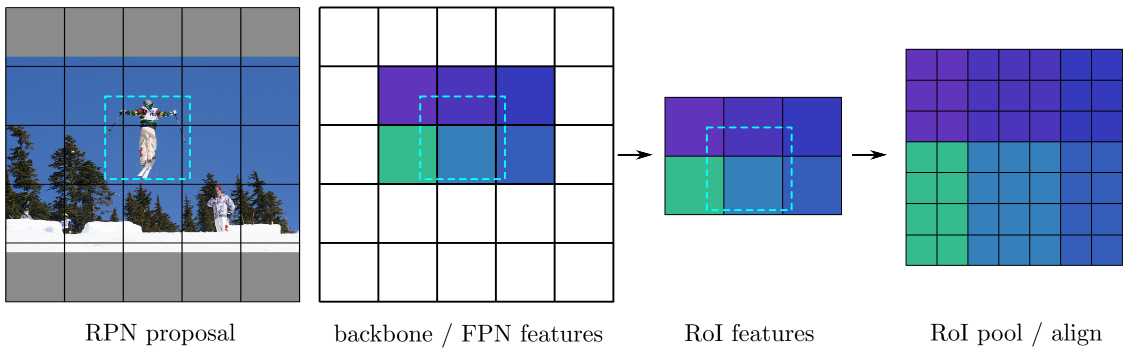


Figure 2.18: Feature extraction based on a RPN proposal box in the RoI head. Pooling or aligning of features results in a feature map. On the basis of flattened features, classification and bounding box regression for the RPN box is performed with fully connected layers. The image on the left is taken from the MS COCO dataset [102].

transformation based on the respective anchor box is

$$\widehat{\mathrm{x}}^{\mathrm{RPN}} := \widetilde{\mathrm{w}} \cdot \widehat{\phi}_{\mathrm{x}}^{\mathrm{RPN}} + \widetilde{\mathrm{x}}, \quad \widehat{\mathrm{y}}^{\mathrm{RPN}} := \widetilde{\mathrm{h}} \cdot \widehat{\phi}_{\mathrm{y}}^{\mathrm{RPN}} + \widetilde{\mathrm{y}}, \quad \widehat{\mathrm{w}}^{\mathrm{RPN}} := \mathrm{e}^{\widehat{\phi}_{\mathrm{w}}^{\mathrm{RPN}}} \cdot \widetilde{\mathrm{w}}, \quad \widehat{\mathrm{h}}^{\mathrm{RPN}} := \mathrm{e}^{\widehat{\phi}_{\mathrm{h}}^{\mathrm{RPN}}} \cdot \widetilde{\mathrm{h}} \tag{2.147}$$

as well as $\widehat{s}^{\mathrm{RPN}} :=, \sigma\left(\widehat{\phi}_s^{\mathrm{RPN}}\right)$. Further, the RoI refinement acts as further offsets to the RPN box:

$$\begin{gathered}
\left(\widehat{\mathrm{x}}^{\mathrm{RoI}}\right)_c := \widehat{\mathrm{w}}^{\mathrm{RPN}} \cdot \left(\widehat{\phi}_{\mathrm{x}}\right)_c^{\mathrm{RoI}} + \widehat{\mathrm{x}}^{\mathrm{RPN}}, \quad \left(\widehat{\mathrm{y}}^{\mathrm{RoI}}\right)_c := \widehat{\mathrm{h}}^{\mathrm{RPN}} \cdot \left(\widehat{\phi}_{\mathrm{y}}\right)_c^{\mathrm{RoI}} + \widehat{\mathrm{y}}^{\mathrm{RPN}}, \\
\widehat{\mathrm{w}}_c^{\mathrm{RoI}} := \mathrm{e}^{(\widehat{\phi}_{\mathrm{w}}^{\mathrm{RoI}})_c} \cdot \widehat{\mathrm{w}}^{\mathrm{RPN}}, \quad \widehat{\mathrm{h}}_c^{\mathrm{RoI}} := \mathrm{e}^{(\widehat{\phi}_{\mathrm{h}}^{\mathrm{RoI}})_c} \cdot \widehat{\mathrm{h}}^{\mathrm{RPN}}, \\
\widehat{\pi}_c^{\mathrm{RoI}} = \Sigma_c\left(\widehat{\phi}_0^{\mathrm{RoI}}, \dots, \widehat{\phi}_C^{\mathrm{RoI}}\right).
\end{gathered} \tag{2.148}$$

This yields the refined box $\widehat{\xi}_c^{\mathrm{RoI}} = (\widehat{\mathrm{x}}_c^{\mathrm{RoI}}, \widehat{\mathrm{y}}_c^{\mathrm{RoI}}, \widehat{\mathrm{w}}_c^{\mathrm{RoI}}, \widehat{\mathrm{h}}_c^{\mathrm{RoI}})$.

2.3.3 Semantic Segmentation of Camera Images: Pixel-wise Classification

In semantic segmentation, each pixel of a given input image $\boldsymbol{x} \in \mathcal{X} = [0,1]^{3\times H\times W}$ is assigned to one of a predefined list of categories $[C] = \{1,\dots,C\}$. Therefore, the target space in semantic segmentation is

$$\mathcal{Y} := [C]^{H\times W}. \tag{2.149}$$

While *pixel-level classification* is a technologically far more complex task than image classification, it can be theoretically treated in a very similar way. Ground truth annotations are given as a segmentation mask $\overline{y} \in [C]^{H\times W}$ which serves as labels for the pixel-wise classification. We obtain a probability distribution per pixel and, therefore, a function

$$\widehat{f}(\cdot|\boldsymbol{\theta}) : \mathcal{X} \to [0,1]^{C\times H\times W}, \qquad \widehat{f}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) : \mathcal{X} \to [0,1]^{C}. \tag{2.150}$$

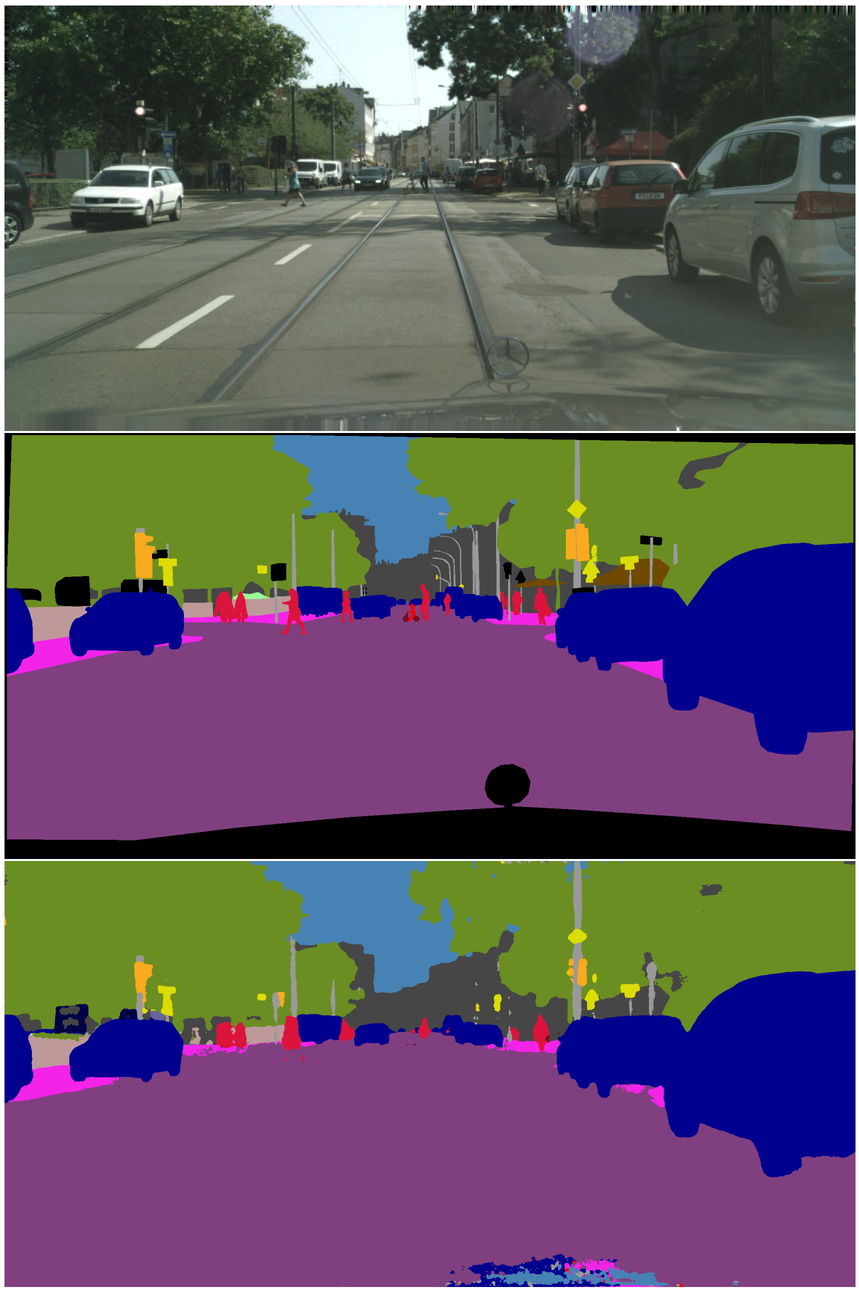


Figure 2.19: Data for semantic segmentation (taken from the Cityscapes [28] validation dataset) including the original RGB image (*top*), colorized pixel-wise category annotations (*center*) and predicted segmentation (*bottom*). The prediction was produced by the DeepLabv3+ model with SEResNeXt backbone used in the investigations in chapter 4.

Here, $\widehat{f}^{i,j}(\cdot|\boldsymbol{\theta})$ denotes the probability distribution over pixel (i,j). Again, this function is accompanied by an argmax function $\widehat{c}(\boldsymbol{x}|\boldsymbol{\theta})$ with

$$\widehat{c}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) = \underset{c\in\{1,\dots,C\}}{\operatorname{argmax}}\ \widehat{f}_c^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) \tag{2.151}$$

as well, as a softmax score function

$$\widehat{s}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) = \max_{c\in\{1,\dots,C\}} \widehat{f}_c^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}). \tag{2.152}$$

The pixel-wise class prediction $\widehat{c}(\boldsymbol{x}|\boldsymbol{\theta})$ gives rise to *connected components* in the image which are assigned the same predicted class, so-called segments. A *segment* is, therefore, a set of pixels

$$S := \{(i,j) : \exists(a,b) \in \{(i\pm 1, j\pm 1)\} \text{ with } \widehat{c}^{a,b}(\boldsymbol{x}|\boldsymbol{\theta}) = \widehat{c}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\} \subset \mathcal{I} \tag{2.153}$$

such that at least one of the eight neighboring pixels is assigned the same class.

2.3.3.1 Loss Functions in Semantic Segmentation

Since semantic segmentation is primarily a classification task on pixel-level, usually the pixel-wise *cross entropy loss*

$$\mathcal{L}_{\mathrm{CE}}^{H\times W}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right) := \frac{1}{HW}\sum_{(i,j)\in\mathcal{I}} \mathcal{L}_{\mathrm{CE}}\left(\widehat{f}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}^{i,j}\right) \tag{2.154}$$

is used for training, where pixel-wise losses are averaged over the entire image. This way, learning feedback from all pixel predictions is generated and weighted equally across the image. In some cases, a class-weighting based on $\overline{y}$ may be beneficial in order to simplify the learning of minority classes.

2.3.3.2 Evaluation Metrics in Semantic Segmentation: Mean Intersection Over Union

As discussed in section 2.3.2, the *intersection over union can* be generalized to arbitrary measurable subsets of $\mathbb{R}^2$. This plays a role in assessing how accurately the semantic segmentation prediction by a model $\widehat{f}$ covers the actual ground truth mask $\overline{y}$. To this end, for the ground truth class $c \in [C]$ fixed, we consider the IoU between predictions and ground truth pixels of class c. On a test dataset $\mathcal{D} = \{(\boldsymbol{x}_1, \overline{y}_1), \dots, (\boldsymbol{x}_{|\mathcal{D}|}, \overline{y}_{|\mathcal{D}|})\}$ we compute the overall IoU

$$\mathrm{IoU}_{\widehat{f}}(c;\mathcal{D}) := \frac{1}{|\mathcal{D}|}\sum_{l=1}^{|\mathcal{D}|} \mathrm{IoU}\left(1_{\{\widehat{c}(\boldsymbol{x}_l|\boldsymbol{\theta})=c\}}, 1_{\{\overline{y}_l=c\}}\right) \tag{2.155}$$

of class c. On a fixed-class basis, each pixel can be categorized as a TP ($\widehat{c}^{i,j} = c = \overline{y}^{i,j}$), FP ($\widehat{c}^{i,j} = c \neq \overline{y}^{i,j}$) or a FN ($\widehat{c}^{i,j} \neq c = \overline{y}^{i,j}$). With this categorization, the IoU in eq. (2.155) can also be written as

$$\mathrm{IoU}\left(1_{\{\widehat{c}(\boldsymbol{x}_l|\boldsymbol{\theta})=c\}}, 1_{\{\overline{y}_l=c\}}\right) = \frac{|\mathrm{TP}_{\widehat{c}(\boldsymbol{x}_l|\boldsymbol{\theta}),\overline{y}}(c)|}{|\mathrm{TP}_{\widehat{c}(\boldsymbol{x}_l|\boldsymbol{\theta}),\overline{y}}(c)| + |\mathrm{FP}_{\widehat{c}(\boldsymbol{x}_l|\boldsymbol{\theta}),\overline{y}}(c)| + |\mathrm{FN}_{\widehat{c}(\boldsymbol{x}_l|\boldsymbol{\theta}),\overline{y}}(c)|}. \tag{2.156}$$

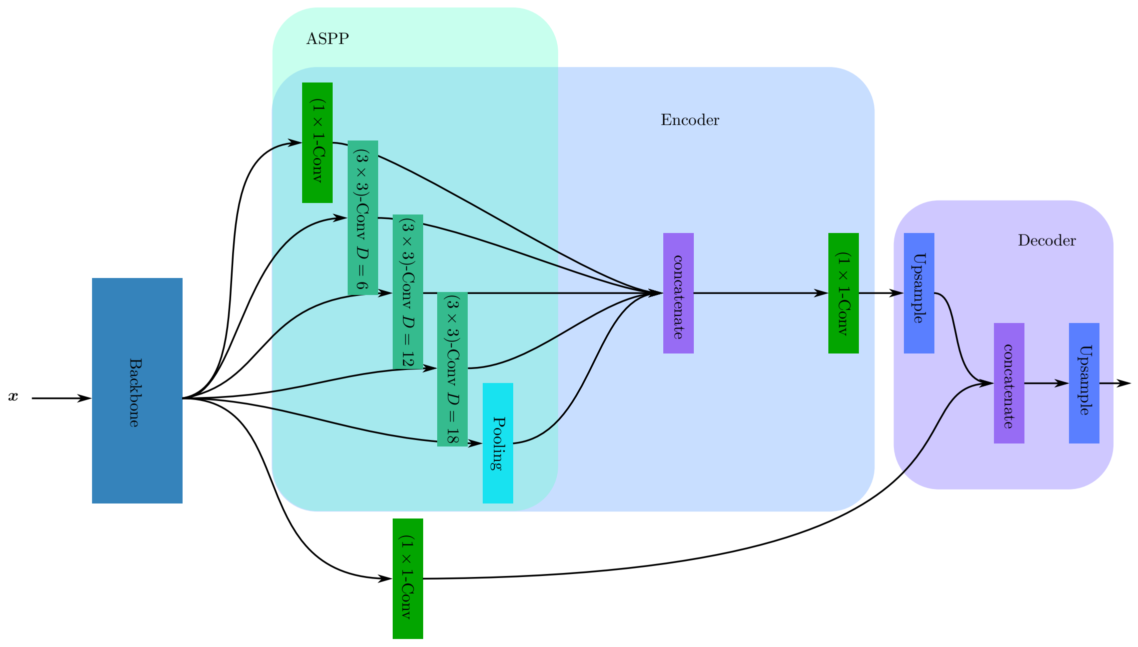


Figure 2.20: Schematic illustration of the DeepLabv3+ encoder-decoder architecture [20] involving an ASPP structure.

A common performance measure of semantic segmentation is then the *mean* IoU where an average over classes

$$\mathrm{mIoU}_{\widehat{f}}(\mathcal{D}) := \frac{1}{C} \sum_{c\in[C]} \mathrm{IoU}_{\widehat{f}}(c;\mathcal{D}) \tag{2.157}$$

is computed.

2.3.3.3 Example Architectures in Semantic Segmentation

Modern architectures in semantic segmentation are mostly *fully convolutional* [19,20,149]. However, the recent advances of vision transformers (section 2.2.1.3) have also given rise to powerful and highly expressive backbone networks which also achieve state-of-the-art performance. Typical architectures follow a similar template as classification and object detection networks and consist of a backbone network which extracts features and a segmentation head. The latter produces a pixel-wise activation feature map $\widehat{\phi}(\boldsymbol{x}|\boldsymbol{\theta}) \in \mathbb{R}^{C\times H\times W}$ of the same resolution as the input $\boldsymbol{x} \in \mathcal{X}$. $\widehat{f}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) = \Sigma(\widehat{\phi}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}))$ is obtained via the softmax function applied pixel-wise. As an example architecture we focus on the DeepLabv3+ [20] architecture in this section. It combines some important components which have lead to the success of convolutional neural networks in semantic segmentation. From the input to the final backbone layer, the resolution is typically iteratively reduced up to a factor of 16 at which point relevant information in the feature maps has been significantly compressed. The original DeepLabv3+ implementation utilized a Xception [24] backbone, whereas other implementations are built on the basis of ResNet variants. In both cases, the replacement of pooling by strided depth-wise convolution layers has proven beneficial. A novelty introduced in the original DeepLabv3 [19] architecture was the usage of *spatial pyramid pooling* via dilated ("atrous") convolutions (so-called "atrous spatial pyramid pooling" or short, ASPP) illustrated in fig. 2.20. The final feature map obtained from the backbone is fed in parallel through a (1×1) convolution, dilated (3×3) convolutions with dilations $6, 12, 18$ and a global average pooling layer. The resulting five feature maps are concatenated and fed through another (1×1) convolution to obtain activations which served as logits that were up-sampled in DeepLabv3. In the DeepLabv3+ architecture, this part without final up-sampling acts as an encoder network which is *supplemented with a decoder network*. In the decoder, feature maps from the backbone with $4\times$ reduced resolution are fed through a (1×1) convolution in order to reduce the amount of channels. The result is concatenated with the up-sampled encoder output and fed through two (3×3) convolutions with batch normalization and ReLU activation before the final (1×1) convolution. The output of the decoder is again bi-linearly up-sampled by a factor of 4 yielding the final logit activations of the DeepLabv3+ architecture.

3 Gradient Uncertainty for Deep Object Detectors

In this chapter we introduce a novel uncertainty quantification method for deep object detectors based on a version of the self-learning gradient on instance level. The presented contents are in large parts taken word-for-word from [145].

3.1 Introduction: Confidence Assignment in Deep Object Detection and Uncertainty Quantification

Deep artificial neural networks (DNNs) designed for tasks such as object detection or semantic segmentation provide a probabilistic prediction on given feature data such as camera images. Modern deep object detection architectures [12, 39, 101, 106, 144] predict bounding boxes for instances of a set of learned classes on an input image. The so-called *objectness or confidence score* indicates the probability of the existence of an object for each predicted bounding box, see section 2.3.2. Throughout this chapter, we will refer to this quantity which the DNN learns by the term *score*. For applications of deep object detectors such as automated surgery or driving, the reliability of this component is crucial. See, for example the detection in the top panel of fig. 3.1 where each box is colored from red (low score) to green (high score). Apart from the accurate, green boxes, boxes with a score below 0.3 are indicated with dashed lines. These contain true and false predictions which *cannot be reliably separated in terms of their score*. In addition, it is well-known that DNNs tend to give mis-calibrated scores [50,55,174] that are oftentimes over-confident and may also lead to unreliable predictions. Over-confident predictions might render an autonomous driving system inoperable by perceiving non-existent, FP instances. Perhaps even more detrimental, under-confidence may lead to overlooked, FN predictions possibly endangering humans outside autonomous vehicles like pedestrians and cyclists, as well as the passengers.

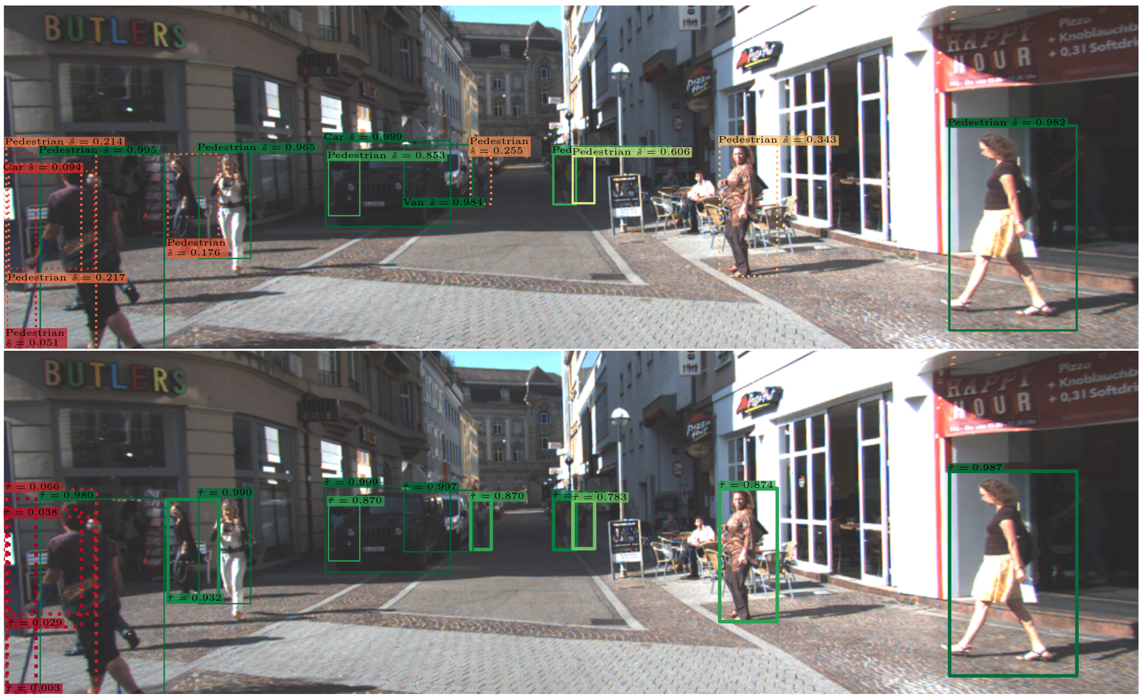


Figure 3.1: Object detection in a street scene. *Top*: DNN Score $\hat{s}$. *Bottom*: Meta classification confidence $\hat{\tau}$ involving gradient features. Dashed boxes here indicate the discarding at any confidence threshold in $[0.3, 0.85]$. The top image contains FNs which are not separable from correctly discarded boxes based on the score (lower threshold would lead to FPs). In the bottom image, those $\widehat{s}$-FNs are assigned higher confidences and there is a large range of thresholds with no FPs.

Apart from modifying and improving the detection architecture or the loss function, there exist methods to estimate prediction confidence which are more involved than the score in order to remedy these issues [109, 121, 161]. We use the term *confidence* more broadly than *score* to refer to quantities which represent the estimated probability of a detection being correct. Such a quantity should reflect the model's overall level of competency when confronted with a given input and is intimately linked to prediction uncertainty. Uncertainty for statistical models, in particular DNNs, can broadly be divided into two types [70] depending on their primary source [80, 202], see also section 2.2.4. Whereas aleatoric uncertainty is mainly founded in the stochastic nature of the data generating process, epistemic uncertainty stems from the probabilistic nature of sampling data for training, as well, as the choice of model and the training algorithm.

Due to the instance-based nature of deep object detection, modern ways of capturing epistemic uncertainty are mainly based on the instance-wise DNN output. From a theoretical point of view, Bayesian DNNs [31, 115] represent an attractive framework for capturing epistemic uncertainty for DNNs by modeling their weights as random variables. Practically, this approach introduces a large computational overhead making its application infeasible for object detection. Therefore, in variational inference approaches, weights are sampled from predefined distributions to address this. These famously include methods like Monte-Carlo (MC) dropout [45, 169] generating prediction variance by performing several forward passes under active dropout. The same idea underlies deep ensemble sampling [88] where separately trained models with the same architecture produce variational forward passes. Other methods based on the classification output can also be applied to object detection such as softmax entropy or energy methods.

A number of other, strong uncertainty quantification methods that do not only rely on the classification output have also been developed for image classification architectures [27, 116, 132, 141]. However, the *transfer of such methods to object detection* frameworks can pose serious challenges, if at all possible, due to architectural restrictions. For example, the usage of a learning gradient evaluated at the network's own prediction was proposed [132] to contain epistemic uncertainty information. This approach has been investigated for the detection of out-of-distribution (OoD) data in image classification. The method has also been applied natural language understanding [181] where gradient features and deep ensemble uncertainty were aggregated. The resulting confidence measures were well-calibrated. The epistemic content of gradient uncertainty has further been explored in [69] in the classification setting by observing shifts in the data distribution.

In this chapter, we propose a way to compute gradient features for the prediction of deep object detectors. We show that they perform on par with state-of-the-art uncertainty quantification methods and that they contain information that cannot be obtained from output- or sampling-based methods. In particular, we summarize the main contributions as follows:

- We introduce a way of generating gradient-based uncertainty features for modern object detection architectures in section 3.3.1, allowing to generate uncertainty information from hidden network layers.

- We investigate the performance of gradient features in terms of meta classification (FP detection, see section 3.3), calibration and meta regression (prediction of intersection over union IoU with the ground truth) in section 3.4. They are compared to other means to quantify or approximate epistemic uncertainty and investigate mutual redundancy as well as detection performance through gradient uncertainty.
- We explicitly investigate the tradeoff between FP and FN predictions for pedestrian detection based on the score and meta classifiers.
- A theoretical treatment of the computational complexity of gradient features is provided. The focus of this treatment is a comparison with MC dropout and deep ensembles where we show that their FLOP count is similar at worst. Explicit runtime measurements are performed for verification.

An implementation of our method is publicly available[38]. A video illustration of our method is publicly available at `https://youtu.be/L4oVNQAGiBc`.

3.2 Related Work: Uncertainty Quantification in Object Detection and Meta Classification

Epistemic Uncertainty for Deep Object Detection. Sampling-based uncertainty quantification such as MC dropout and deep ensembles have been investigated in the context of object detection by several authors in the past. They are straight-forward to implement into any architecture and yield output variance for all bounding box features. Harakeh et al. [57] employed MC dropout and Bayesian inference as a replacement of NMS to get a joint estimation of epistemic and aleatoric uncertainty. Similarly, epistemic uncertainty measures were obtained by Kraus and Dietmayer [85] from MC dropout. Miller et al. [121] investigated MC dropout as a means to improve object detection performance in open-set conditions. Different merging strategies for samples from MC dropout were investigated by Miller et al. [120] and compared with the influence of merging boxes in deep ensembles [122]. Lyu et al. [109] aggregated deep ensemble samples as if produced from a single detector to obtain improved detection performance. A variety of uncertainty measures generated from proposal box variance pre-NMS called MetaDetect was investigated by Schubert et al. [161]. In generating advanced scores and IoU estimates, it was reported that the obtained information is largely redundant with MC dropout uncertainty features. All the methods above are based on the network output and generate variance by aggregating prediction proposals in some manner. Moreover, a large amount of uncertainty quantification methods based on classification outputs can be directly applied to object detection [62, 107]. Little is known about other methods developed for image classification that are not directly transferable to object detection due to architectural constraints (e.g., activation-based [27] or gradient-based [132] uncertainty). The central difficulty lies in the fact that *different predicted instances depend on shared latent features or DNN weights* such that the base method can only estimate uncertainty for the entire prediction, i.e., for all instances, instead of individual estimates per instance. We show

[38]`https://github.com/tobiasriedlinger/gradient-metrics-od`

that gradient uncertainty information can be extracted from hidden layers in object detectors. We seek to determine how they compare to output-based methods and show that they contain orthogonal information.

Meta Classification and Meta Regression. The term meta classification refers to the discrimination of TPs from FPs on the basis of uncertainty features which was first explored by Hendrycks and Gimpel [62] to detect OoD samples based on the maximum softmax probability. Since then, the approach has been applied to natural language processing [181], semantic segmentation [16, 113, 152, 153, 155], instance segmentation in videos [114] and object detection [84, 161] to detect FP predictions on the basis of uncertainty features accessible during inference. Moreover, meta regression, the estimation of IoU based on uncertainty in the same manner, was also investigated [113,114,153,155,161] showing large correlations between estimates and the true localization quality. Chan et al. [16] have shown that meta classification can be used to improve network accuracy, an idea that so-far has not been achieved for object detection. Previous studies have overlooked class-restricted meta classification performance, e.g., when restricting to safety-relevant instance classes. Moreover, in order to base downstream applications on meta classification outputs, resulting confidences need to be statistically reliable, i.e., calibrated which has also escaped previous research.

3.3 Methods: Computing Instance-wise Loss Gradients in Object Detection

3.3.1 Gradient-Based Epistemic Uncertainty

In instance-based recognition tasks, such as object detection or instance segmentation, the *prediction*

$$\widehat{y} = \mathrm{NMS}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\right) =: \left(\widehat{y}^1, \ldots, \widehat{y}^{N_{\boldsymbol{x}}}\right) \tag{3.1}$$

consists of a list of instances like bounding boxes, section 2.3.2 for notation. The length of $\widehat{y}$ usually depends on the corresponding input $\boldsymbol{x} \in \mathcal{X} = [0,1]^{3\times H\times W}$ and on hyperparameters like confidence or overlap thresholds $\tau_s, \tau_{\mathrm{IoU}}^{\mathrm{NMS}}$. Uncertainty information which is not generated directly from instance-wise data such as activation- or gradient-based information can *at best yield statements about the entirety of* $\widehat{y}$. However, uncertainty is not immediately given for any individual instance $\widehat{y}^j$. This issue is especially apparent for uncertainty generated from deep features which potentially all contribute to an instance $\widehat{y}^j$ where $j \in [N_{\boldsymbol{x}}]$. Here, we introduce an approach to generate gradient-based uncertainty features for the instance-based setting. To this end, we sketch how gradient uncertainty is generated for classification tasks.

3.3.1.1 Gradient Uncertainty in the Classification Setting: Uncertainty via Learning Stress

Generically, given an input $\boldsymbol{x}$, a classification network predicts a class distribution

$$\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta}) \in [0,1]^C \tag{3.2}$$

of fixed length C given a set of weights $\boldsymbol{\theta}$ (recall section 2.3.1). During training, the latter is compared to the ground truth label y belonging to $\boldsymbol{x}$ by means of some loss function $\mathcal{L}(\cdot|\cdot)$ (see section 2.3.1.1), which is minimized by optimizing $\boldsymbol{\theta}$, e.g., by standard stochastic gradient descent (see section 2.2.2.1). The $\boldsymbol{\theta}$-step is proportional to the gradient $g(\boldsymbol{x}, \boldsymbol{\theta}, y) := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\widehat{f}(\boldsymbol{x}|, \boldsymbol{\theta})|y)$ which can also be regarded as a measure of *learning stress* imposed upon $\boldsymbol{\theta}$. Gradient uncertainty features are generated by substituting the non-accessible ground truth y with the network's class prediction $\widehat{c}(\boldsymbol{x}|\boldsymbol{\theta}) := \operatorname{argmax}_c \{\widehat{f}_c(\boldsymbol{x}|\boldsymbol{\theta})\}_{c\in[C]}$. Here, we disregard the dependence of the latter on $\boldsymbol{\theta}$[39], denoted $\widehat{c}(\boldsymbol{x})$. Scalar values are obtained by computing some magnitude of

$$g(\boldsymbol{x}, \boldsymbol{\theta}, \widehat{c}(\boldsymbol{x})) = \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\widehat{c}(\boldsymbol{x})\right). \tag{3.3}$$

To this end, in our experiments we employ the maps

$$\{\min(\cdot), \max(\cdot), \operatorname{mean}(\cdot), \operatorname{std}(\cdot), \|\cdot\|_1, \|\cdot\|_2\}. \tag{3.4}$$

We discuss the latter choice in section 3.4 and first illuminate some points about eq. (3.3).

***Intuition and Discussion of* (3.3).** First, eq. (3.3) can be regarded as the *self-learning gradient* of the network. It, therefore, expresses the learning stress on $\boldsymbol{\theta}$ under the condition that the class prediction $\widehat{c}(\boldsymbol{x})$ were given as the ground truth label. The collapse of the predicted distribution $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ to $\widehat{c}(\boldsymbol{x})$ implies that eq. (3.3) does not generally vanish in the classification setting. However, this consideration poses a problem for (bounding box) regression which we will address in the next paragraph. We also note that it is possible to generate fine-grained features by restricting $\boldsymbol{\theta}$ in eq. (3.3) to subsets of weights $\boldsymbol{\theta}_\ell$. These could be individual layers, convolutional filters or singular weights where we then compute partial gradients of $\mathcal{L}$.

Using eq. (3.3) as a measure of uncertainty may be understood by regarding true and false predictions. A well-performing neural network which has $\widehat{c}(\boldsymbol{x})$ already close to the true label y tends to experience little stress when trained on $(\boldsymbol{x}, y)$ with the usual learning gradient. This reflects *confidence* in the prediction $\widehat{c}(\boldsymbol{x})$ and the difference between eq. (3.3) and the true gradient is then small. In the case of false predictions $\widehat{c}(\boldsymbol{x}) \neq y$, the true learning gradient enforces large adjustments in $\boldsymbol{\theta}$. The self-learning gradient eq. (3.3) behaves differently in that it is *large for non-peaked or uncertain* (high entropy) predictions $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ and small for highly peaked distributions.

In the following, we explain how empirical findings support the hypothesis that gradient uncertainty as explained above contains empirical uncertainty.

3.3.1.2 Theoretical Link with Empirical Findings

Assuming that we draw data $z = (\boldsymbol{x}, y)$ (recall section 2.1.1.3) from a fixed distribution μ_Z, we regard $g(\boldsymbol{x}) := g(\boldsymbol{x}, \boldsymbol{\theta}, \widehat{c}(\boldsymbol{x}))$ for a parametric classification model:

$$\widehat{f}(\cdot|\boldsymbol{\theta}) : \boldsymbol{x} \mapsto \widehat{f}(\boldsymbol{x}|\boldsymbol{\theta}). \tag{3.5}$$

[39] Due to the application of the argmax operator, $\widehat{c}$ can be regarded as being piece-wise constant with respect to $\boldsymbol{\theta}$.

This model estimates $\mu_{|\boldsymbol{X}=\boldsymbol{x}}$ (see section 2.1.1). The labels y and the model's class prediction $\widehat{c}$ are categorical over a class space $\mathcal{Y} = [C]$. For a loss function $\mathcal{L} = \mathcal{L}(f(\boldsymbol{x}|\boldsymbol{\theta})|y)$, we compute the gradient

$$g(\boldsymbol{x}) = g(\boldsymbol{x}, \boldsymbol{\theta}, \widehat{c}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})|\widehat{c}) \tag{3.6}$$

where we neglect the implicit $\boldsymbol{\theta}$-dependency in $\widehat{c}(\boldsymbol{x})$ since $\widehat{c}(\boldsymbol{x})$ is locally constant with discontinuity on decision boundaries of $\widehat{f}$. This self-learning gradient coincides with the ordinary learning gradient $g(\boldsymbol{x}, \boldsymbol{\theta}, y)$ with frequency $\Pr(y = \widehat{c}(\boldsymbol{x}))$, that is, the accuracy of the model.

We can investigate whether $g(\boldsymbol{x})$ is large in the sense of some metric M, like $M(g) = \|g\|_\rho$ for $\rho \in [1, \infty]$, statistically whenever the prediction $\widehat{c}(\boldsymbol{x})$ is uncertain or prone to error. Thus, we compare conditional distributions over $M(g(\boldsymbol{x}))$ conditioned to incorrect predictions $\widehat{c}(\boldsymbol{x}) \neq y$ versus correct predictions $\widehat{c} = y$. The application of different risk functionals to the distributions can then relate the statistical magnitudes of $M(g)$ conditional to $\widehat{c}$. Investigating the *expected value* as a simple risk functional, we search conditions for

$$\mathbb{E}_{(\boldsymbol{X},Y)\sim\mu_Z}[M(g(\boldsymbol{X}))|\widehat{c}(\boldsymbol{X}) = Y] < \mathbb{E}_{(\boldsymbol{X},Y)\sim\mu_Z}[M(g(\boldsymbol{X}))|\widehat{c}(\boldsymbol{X}) \neq Y]. \tag{3.7}$$

We first call $\varepsilon(\boldsymbol{x})$ the conditional error rate and ε the total error rate of the model $\widehat{f}$ under the distribution μ_Z with

$$\varepsilon(\boldsymbol{x}) = \sum_{c \neq \widehat{c}(\boldsymbol{x})} p(c|\boldsymbol{x}), \quad \varepsilon = \mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}[\varepsilon(\boldsymbol{X})]. \tag{3.8}$$

The conditional expectations then yield

$$\begin{aligned}
\mathbb{E}_{(\boldsymbol{X},Y)\sim\mu_Z}[M(g(\boldsymbol{X}))|\widehat{c}(\boldsymbol{X}) \neq Y] &= \frac{\int \sum_{y\neq\widehat{c}(\boldsymbol{X})} M(g(\boldsymbol{X}))\,\mathrm{d}\mu_Z(\boldsymbol{X}, y)}{\Pr(\widehat{c}(\boldsymbol{X}) \neq Y)} \\
&= \frac{1}{\varepsilon}\mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}[\varepsilon(\boldsymbol{X})M(g(\boldsymbol{X}))] \\
&= \mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}[M(g(\boldsymbol{X}))] + \frac{\mathrm{Cov}(\varepsilon(\boldsymbol{X}), M(g(\boldsymbol{X})))}{\varepsilon}
\end{aligned} \tag{3.9}$$

$$\begin{aligned}
\mathbb{E}_{(\boldsymbol{X},Y)\sim\mu_Z}[M(g(\boldsymbol{X}))|\widehat{c}(\boldsymbol{X}) = Y] &= \frac{\int (1-\varepsilon(\boldsymbol{X}))M(g(\boldsymbol{X}))\,\mathrm{d}\mu_{\boldsymbol{X}}(\boldsymbol{X})}{\Pr(\widehat{c}(\boldsymbol{X}) = Y)} \\
&= \frac{1}{1-\varepsilon}\mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}[(1-\varepsilon(\boldsymbol{X}))M(\boldsymbol{X})] \\
&= \mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}[M(g(\boldsymbol{X}))] + \frac{\mathrm{Cov}(\varepsilon(\boldsymbol{X}), M(g(\boldsymbol{X})))}{\varepsilon - 1}.
\end{aligned} \tag{3.10}$$

Therefore, $\mathbb{E}_{\boldsymbol{X}\sim\mu_{\boldsymbol{X}}}[M(g(\boldsymbol{X}))]$ and the total error rate ε drop out of eq. (3.7), which is finally equivalent to

$$\mathrm{Cov}(\varepsilon(\boldsymbol{X}), M(g(\boldsymbol{X}))) > 0. \tag{3.11}$$

For accurate models $\widehat{f}$, the self-learning gradient $g(\boldsymbol{x})$ will be close to the real gradient $g(\boldsymbol{x}, \boldsymbol{\theta}, y)$. Therefore, the above covariance will be close to $\mathrm{Cov}(\varepsilon(\boldsymbol{X}), M(g(\boldsymbol{X}, \boldsymbol{\theta}, y)))$. The positivity of the latter has a *clear interpretation in terms of epistemic uncertainty*, in that learning steps are larger whenever the model performance is poor, i.e., $\varepsilon(\boldsymbol{x})$ is large. In such regions, the model still attempts to adapt strongly to new instances, whereas little adaptation is given for instances where the model performance is already good, i.e., $\varepsilon(\boldsymbol{x})$ is small. That such local improvements are in fact possible is an easy consequence of the universal approximation property of deep neural networks.

That the condition for the self-learning gradient in eq. (3.11) holds can be seen in the classification experiments conducted and shown in [132, Fig. 2] and [170, Fig. 2]. There, the distributions of $M(g(\boldsymbol{X}))$ are explicitly conditioned to true and false predictions.

3.3.1.3 Extension to Object Detectors: Instance-wise Self-Learning Gradients

We first clarify the aforementioned *complications in generating uncertainty* information for object detection. Generally, the prediction eq. (3.1) is the filtering result of a larger, often fixed number $\widehat{N}_{\text{out}} := N_{\text{Anch}} \cdot \lceil H/\varsigma \rceil \cdot \lceil W/\varsigma \rceil$ (recall section 2.3.2) of proposal bounding boxes $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$. Given a ground truth list $\overline{y}$ of bounding boxes, the loss function usually has the form

$$\mathcal{L} = \mathcal{L}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle|\,\overline{y}\right), \tag{3.12}$$

such that all $\widehat{N}_{\text{out}}$ proposal bounding boxes potentially contribute to $g(\boldsymbol{x}, \boldsymbol{\theta}, y)$. Again, when filtering $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ to a smaller number of predicted boxes $\widehat{y}$ and converting them to ground truth format $\widehat{\mathrm{b}} = (\widehat{\mathrm{b}}^1, \ldots, \widehat{\mathrm{b}}^{N_{\boldsymbol{x}}})$ with

$$\widehat{\mathrm{b}}^j := \left(\widehat{\xi}^j, \underset{c\in[C]}{\operatorname{argmax}}\, \widehat{\pi}_c^j\right), \qquad \forall j \in [N_{\boldsymbol{x}}], \tag{3.13}$$

where $\widehat{y}^j = (\widehat{\xi}^j, \widehat{\pi}^j, \widehat{s}^j)$, we can compute the self-learning gradient $g(\boldsymbol{x}, \boldsymbol{\theta}, \widehat{\mathrm{b}})$. This quantity, however, *does not refer to any individual prediction* $\widehat{y}^j$, but rather to all boxes in $\widehat{y}$ simultaneously. We take two steps to obtain meaningful gradient information for one particular box $\widehat{y}^j$ from this approach.

First, we restrict the ground truth slot to only contain the length-one list $\{\widehat{\mathrm{b}}^j\}$, regarding it as the hypothetical label. This alone is insufficient since other, correctly predicted instances in $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ would lead to a penalization and "overcorrecting" gradient $g(\boldsymbol{x}, \boldsymbol{\theta}, \{\widehat{\mathrm{b}}\})$, given $\widehat{\mathrm{b}}^j$ as label. This gradient's optimization goal is, figuratively speaking, to forget to predict everything but $\widehat{y}^j$ when presented with $\boldsymbol{x}$. Note that we cannot simply compute $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\widehat{y}^j(\boldsymbol{x}, \boldsymbol{\theta})|\widehat{\mathrm{b}}^j)$ for regression losses, such as for bounding box regression. Such losses are frequently norm-based, like L^p-losses (see section 2.3.2.1) such that the respective loss and gradient would both vanish. Therefore, we secondly mask $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ such that the result is likely to only contain proposal boxes meaning to predict the same instance as $\widehat{\mathrm{b}}^j$. Our conditions for this mask are *sufficient score*, *sufficient overlap* with $\widehat{\mathrm{b}}^j$ and *same indicated class* as $\widehat{\mathrm{b}}^j$, i.e., the predictions which would be suppressed by $\widehat{\mathrm{b}}^j$ in NMS. We call the subset of $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ that satisfies these conditions *candidate boxes* for $\widehat{\mathrm{b}}^j$, denoted $\mathrm{cand}[\widehat{y}^j]$.

We, thus, propose the candidate-restriced self-learning gradient

$$g^{\mathrm{cand}}(\boldsymbol{x}, \boldsymbol{\theta}, \widehat{y}^j) := \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\mathrm{cand}[\widehat{y}^j](\boldsymbol{x}, \boldsymbol{\theta}) \,\middle|\, \widehat{\mathrm{b}}^j\right) \tag{3.14}$$

of $\widehat{y}^j$ for computing instance-wise uncertainty. This approach is in line with the motivation for the classification setting and extends it when computing eq. (3.14) for multi-criterial loss functions in object detection.

3.3.1.4 Computational Complexity

Sampling-based epistemic uncertainty quantification methods such as MC dropout and deep ensembles tend to *generate a significant computational overhead* as several forward passes are required. Here, we provide a theoretical result on the count of floating point operations (FLOP) of gradient uncertainty features which is supported with a proof and additional detail in the following paragraphs. In our experiments, we use the gradients computed over the last one, respectively, two layers of each network architecture of different architectural branches, as well, if applicable, see section 2.3.2.3. For layer ℓ, we assume stride-1, $(2s_\ell + 1) \times (2s_\ell + 1)$-convolutional layers acting on feature maps of spatial size $H_\ell \times W_\ell$. These assumptions hold for all architectures in our experiments. We denote the number of input channels by $k_{\ell-1}$ and of output channels by k_ℓ.

Theorem 1. *The number of FLOP required to compute the last layer ($\ell = L$) gradient $\nabla_{K_L} \mathcal{L}(\mu^j \phi_L(K_L), \phi^j)$ is $\mathcal{O}(k_L H_L W_L + k_L k_{L-1}(2s_L + 1)^4)$. Similarly, for earlier layers $\ell \in [L-1]$, i.e., $\nabla_{K_\ell} \mathcal{L}(\mu^j \phi_L(K_\ell), \phi^j)$, we have $\mathcal{O}(k_{\ell+1}k_\ell + k_\ell k_{\ell-1})$, provided that we have previously computed the gradient for the consecutive layer $\ell + 1$. Performing variational inference only on the last layer, i.e., ϕ_{L-1} requires $\mathcal{O}(k_L k_{L-1} H_L, W_L)$ FLOP per sample.*

Theorem 1 provides that even for MC dropout only before the last layer, or the use of efficient deep sub-ensembles [179] sharing the entire architecture but the last layer, gradient features require fewer or at worst similar FLOP counts. Earlier sampling, especially entire deep ensembles, have even higher FLOP counts than these variants. Note, that *computing gradient features have somewhat larger computational latency* since the full forward pass needs to be computed before gradients can be computed. Moreover, while sampling strategies can in principle be implemented to run all sample forward passes in parallel, the computation of gradients can run in parallel for predicted boxes per image. We compare explicit time measurements for different methods in section 3.4 and provide a proof of theorem 1 and the notation setting in the following.

Setting. As in [166, Chapter 20.6], we regard a (convolutional) neural network as a graph of feature maps with vertices $V = \bigsqcup_{\ell=0}^{L} V_\ell$ arranged in layers V_ℓ. For our consideration it will suffice to regard them as sequentially ordered. Each layer V_ℓ contains a set number $k_\ell := |V_\ell|$ of feature map channels $\phi_\ell^c \in \mathbb{R}^{H_\ell \times W_\ell}$, $c \in [k_\ell]$. We denote the activation of V_ℓ by $\phi_\ell = (\phi_\ell^1, \dots, \phi_\ell^{k_\ell})$. The activation $\phi_{\ell+1} \in (\mathbb{R}^{H_{\ell+1} \times W_{\ell+1}})^{k_{\ell+1}}$ is obtained from ϕ_ℓ by convolutions. We have $k_\ell \times k_{\ell+1}$ quadratic filter matrices

$$(K_{\ell+1})_c^d \in \mathbb{R}^{(2s_\ell+1)\times(2s_\ell+1)}, \quad c \in [k_\ell]; \quad d \in [k_{\ell+1}]. \tag{3.15}$$

Here, s_ℓ is a (usually small) natural number, the spatial extent of the filter to either side. Also, we have respectively $k_{\ell+1}$ biases $b^d_{\ell+1} \in \mathbb{R}$, $d \in [k_{\ell+1}]$. We denote the convolution of $K \in \mathbb{R}^{(2s+1)\times(2s+1)}$ and $\phi \in \mathbb{R}^{H\times W}$ as

$$(K * \phi)_{ab} := \sum_{m,n=-s}^{s} K_{s+1+p,s+1+q}\phi_{a+p,b+q}, \tag{3.16}$$

where $a \in [H]$ and $b \in [W]$, cf. section 2.2.1.2. This is, strictly speaking, only correct for convolutions with stride 1, although a closed form can be given for the more general case (see [60]). For our goals, we will use stride 1 to upper bound the FLOPs which comes with the simplification that the feature maps' sizes are conserved. We then define

$$\psi^d_{\ell+1} = \sum_{c=1}^{k_\ell} (K_{\ell+1})^d_c * \phi^c_\ell + b^d_{\ell+1}\mathbf{1}_{H_\ell\times W_\ell}, \quad d \in [k_{\ell+1}]. \tag{3.17}$$

Finally, we apply activation functions $\alpha_\ell : \mathbb{R} \to \mathbb{R}$ to each entry to obtain $\phi_{\ell+1} = \alpha_{\ell+1}(\psi_{\ell+1})$. In practice, α_ℓ is usually a ReLU activation (recall section 2.2.1.1) or some modification like leaky ReLU. We will treat the computational complexity of this operation later. We can then determine the computational expense of computing $\psi_{\ell+1}$ from ϕ_ℓ. In the following, we will be interested in the linear convolution action (see section 2.2.1.2)

$$\mathrm{Mat}[K_\ell] : \mathbb{R}^{k_{\ell-1}\times h_{\ell-1}\times w_{\ell-1}} \to \mathbb{R}^{k_\ell\times H_\ell\times W_\ell}, \quad (\mathrm{Mat}[K_\ell]\phi_{\ell-1})^d_{ab} := \left(\sum_{c=1}^{k_\ell} (K_\ell)^d_c * \phi^c_\ell\right)_{ab}, \tag{3.18}$$

where $d \in [k_\ell]$, $a \in [H_\ell]$ and $b \in [W_\ell]$. Note that $\mathrm{Mat}[K_\ell]$ is also linear in K_ℓ. On the last layer feature map $\phi_L = \widehat{\phi}$ we define the loss function $\mathcal{L} : (\widehat{\phi}, \phi) \mapsto \mathcal{L}(\widehat{\phi}|\phi) \in \mathbb{R}$. Here, ϕ stands for the ground truth[40] transformed to feature map size $\mathbb{R}^{H_L\times W_L\times k_L}$. In order to make dependencies explicit, define the loss of the subnet starting at layer ℓ by $\mathcal{L}_\ell$, i.e.,

$$\mathcal{L}_L(\phi_L) := \mathcal{L}(\widehat{\phi}, \phi), \qquad \mathcal{L}_{\ell-1}(\phi_{\ell-1}) := \mathcal{L}_\ell(\alpha_\ell(\psi_\ell)). \tag{3.19}$$

Straight-forward calculations yield

$$\nabla_{K_L}\mathcal{L} = \nabla_{K_L}(\mathcal{L}_L \circ \alpha_L \circ \psi_L(K_L)) = D_1\mathcal{L}|_{\phi_L} \cdot D\alpha_L|_{\psi_L} \cdot \nabla_{K_L}\psi_L \tag{3.20}$$

$$\begin{aligned}\nabla_{K_{L-1}}\mathcal{L} =& \nabla_{K_{L-1}}(\mathcal{L}_L \circ \alpha_L \circ \psi_L \circ \alpha_{L-1} \circ \psi_{L-1}(K_{L-1})) \\ =& D_1\mathcal{L}|_{\phi_L} \cdot D\alpha_L|_{\psi_L} \cdot \mathrm{Mat}[K_L] \cdot D\alpha_{L-1}|_{\psi_{L-1}} \cdot \nabla_{K_{L-1}}\psi_{L-1}.\end{aligned} \tag{3.21}$$

Here, D denotes the total derivative (D_1 for the first variable, respectively) and we have used the linearity of $\mathrm{Mat}[K_L]$. Note, that in section 3.3.2, we omitted the terms $D\alpha_L|_{\psi_L}$ and $D\alpha_{L-1}|_{\psi_{L-1}}$. We will come back to them later in the discussion. For the gradient features we present in this chapter, each $\widehat{y}^j$ for which we compute gradients receives a binary mask μ^j such that $\mu^j \cdot \phi_L$ are the feature map representations of candidate boxes for $\widehat{y}^j$. The scalar loss function then becomes $\mathcal{L}(\mu_j\phi_L|\phi^j)$ for the purposes of computing gradient uncertainty. Here, ϕ^j is $\widehat{\mathrm{b}}^j$ in feature map representation, see eq. (3.13). We address next, how this masking influences eq. (3.20), eq. (3.21) and the FLOP count of our method.

[40] The transformations are listed in section 2.3.2 for the entries of $\widehat{\phi}$.

Table 3.1: Upper bounds on FLOP and elementary function evaluations performed during the computation of $D\mathcal{L}^j$ (all contributions) and post-processing for sampling-based uncertainty quantification (sampling pp) for N_{samp} inference samples.

	YOLOv3	Faster/Cascade R-CNN	RetinaNet
# FLOP $D\mathcal{L}^j$	$(9+C)N_{\mathrm{out}}$	$10N_{\mathrm{out}}^{\mathrm{RPN}} + (2+2C)N_{\mathrm{out}}$	$(18+11C)N_{\mathrm{out}}$
# FLOP sampling pp	$8N_{\mathrm{out}}N_{\mathrm{samp}}$	$(9+2C)N_{\mathrm{out}}N_{\mathrm{samp}}$	$8N_{\mathrm{out}}N_{\mathrm{samp}}$
# evaluations $D\mathcal{L}^j$	0	0	$2(1+C)N_{\mathrm{out}}$
# evaluations sampling pp	$(5+C)N_{\mathrm{out}}N_{\mathrm{samp}}$	$(3+C)N_{\mathrm{out}}N_{\mathrm{samp}}$	$(3+C)N_{\mathrm{out}}N_{\mathrm{samp}}$

Computing the mask. The complexity of determining μ^j (i.e., finding cand$[\widehat{y}^j]$) is the complexity of computing all mutual IoU values between $\widehat{y}^j$ and the $n_L := H_L \cdot W_L \cdot k_L$ other predicted boxes. Computing the IoU (recall section 2.3.2) of a box ξ_1 and ξ_2 can be done in a few steps with an efficient method exploiting the fact that:

$$U = A_1 + A_2 - I, \qquad IoU = I/U, \tag{3.22}$$

where the computation of the intersection area I and the individual areas A_1 and A_2 can each be done in 3 FLOP, resulting in 12 FLOP per pair of boxes. Note that different localization constellations of ξ_1 and ξ_2 may result in slightly varying formulas for the computation of I. However, the constellation can be easily determined by binary checks which we ignore computationally. Also, the additional check for the class and sufficient score will be ignored, so we have $12n_L$ FLOP per mask μ^j. Inserting the binary mask[41] μ^j in eq. (3.20) and eq. (3.21) leads to the replacement of $D_1\mathcal{L}|_{\phi_L} \cdot D\alpha_L|_{\psi_L}$ by $D\mathcal{L}^j := D_1\mathcal{L}(\cdot, \gamma^j)|_{\mu^j\phi_L} \cdot \mu^j \cdot D\alpha_L|_{\psi_L}$ for each relevant box $\widehat{y}^j$.

In table 3.1 we have listed upper bounds on the number of FLOP and elementary function evaluations performed for the computation of $D\mathcal{L}^j$ for the investigated loss functions. The numbers were obtained from the explicit partial derivatives computed in section 3.4.1. In principle, those formulas allow for every possible choice of $b \in [N_{\mathrm{out}}]$ which is why all counts are proportional to it. Practically, however, at most the $|\mu^j|$ candidate boxes are relevant which need to be identified additionally as foreground or background for $\widehat{y}^j$. This happens in a separate step involving a IoU computation between $\widehat{y}^j$ and the respective anchor. The total count of candidate boxes in practice is on average not larger than ~ 30. When evaluating the formulas from section 3.4.1 note, that there is only one ground truth box per gradient, and we assume here, that one full forward pass has already been performed such that the majority of the appearing evaluations of elementary functions like sigmoids or exponentials, have been computed beforehand. This is not the case for the RetinaNet classification loss eq. (3.53). In table 3.1 we also list the additional post-processing cost for the output transformations (see sections 2.3.2 and 3.4.1) required for sampling-based uncertainty quantification like MC dropout or deep ensemble samples ("sampling pp"). The latter are also proportional to N_{out}, but also to the number N_{samp} of samples.

Proof of Theorem 1. Our implementations exclusively use stride 1 convolutions for the layers indicated in section 3.4, so $W_L = W_{L-1} = W_{L-2} =: W$, resp. $H_L = H_{L-1} =$

[41]See section 3.3.2. The mask μ^j selects the feature map representation of cand$[\widehat{y}^j]$ out of ϕ_L.

$H_{L-2} =: H$. As before, we denote $n_\ell := HWk_\ell$, and regard $D\mathcal{L}^j$ as a $1 \times n_L$ matrix. Next, regard the matrix-vector multiplication to be performed in eq. (3.20). Since for all $\ell \in [L]$ we have that ψ_ℓ is linear in K_ℓ, we regard $\nabla_{K_\ell}\psi_\ell$ as a matrix acting on the filter space $\mathbb{R}^{k_{\ell-1}\times k_\ell \times (2s_\ell+1)^2}$. For $d \in [k_\ell]$, ψ_ℓ^d only depends on K_ℓ^d (see eq. (3.17)), so $\nabla_{K_\ell}\psi_\ell$ only has at most $k_{\ell-1} \cdot (2s_\ell+1)^2 \cdot n_\ell$ non-vanishing entries. Therefore, regard it as a $(n_\ell \times (k_{\ell-1}(2s_\ell+1)^2))$-matrix. We will now show that this matrix has $k_\ell(2s_\ell+1)^2$-sparse columns.

Let $c \in [k_\ell]$, $d \in [k_{\ell-1}]$, $p, q \in \{-s_\ell, \dots, s_\ell\}$, $a \in [H_\ell]$ and $b \in [W_\ell]$. One easily sees from eqs. (3.16) and (3.17) that

$$\frac{\partial}{\partial((K_\ell)_c^d)_{pq}}(\psi_\ell)_{ab}^d = (\phi_{\ell-1})_{a+p-s_\ell-1,b+q-s_\ell-1}^c, \tag{3.23}$$

where $\phi_{\ell-1}^c$ is considered to vanish for $a+p-s_\ell-1 \notin [H_\ell]$ and $b+q-s_\ell-1 \notin [W_\ell]$. Consistency with the definition of p and q requires that both the conditions

$$1 < a \leq 2s_\ell + 2, \qquad 1 < b \leq 2s_\ell + 2 \tag{3.24}$$

are satisfied, which means that $(\nabla_{K_\ell}\psi_\ell)^d$ can only have $k_\ell(2s_\ell+1)^2$ non-zero entries. Appealing to sparsity $\nabla_{K_L}\psi_L$ in eq. (3.20) is then, effectively, a $(k_{L-1}\cdot(2s_L+1)^2) \times (k_L \cdot (2s_L+1)^2)$-matrix, resulting in a FLOP count of

$$[2 \cdot k_L(2s_L+1)^2 - 1] \cdot [k_{L-1} \cdot (2s_L+1)^2] \tag{3.25}$$

for the multiplication $D\mathcal{L}^j \cdot \nabla_{K_L}\psi_L$ giving the claimed complexity considering that the computation of μ^j is $\mathcal{O}(k_L HW)$.

Next, we investigate the multiplication in eq. (3.21), in particular the multiplication $D\mathcal{L}^j \cdot \mathrm{Mat}[K_L]$ as the same sparsity argument applies to $\nabla_{K_{L-1}}\psi_{L-1}$. First, for $\ell \in [L]$, regard $\mathrm{Mat}[K_\ell]$ as a $(n_\ell \times n_{\ell-1})$-matrix acting on a feature map $\phi \in \mathbb{R}^{n_{\ell-1}}$ from the left via

$$(\mathrm{Mat}[K_\ell]\phi)_{ab}^d = \sum_{c=1}^{k_{\ell-1}} \left[(K_\ell)_c^d * \phi^c\right]_{ab} = \sum_{c=1}^{k_{\ell-1}} \sum_{m,n=-s_\ell}^{s_\ell} [(K_\ell)_c^d]_{s_\ell+1+m,s_\ell+1+n}(\phi^c)_{a+m,b+n}, \tag{3.26}$$

where $d \in [k_\ell]$, $b \in [W_\ell]$ and $a \in [H_\ell]$ indicate one particular row in the matrix representation of $\mathrm{Mat}[K_\ell]$ (recall eq. (2.49)). From this, we see the sparsity of $\mathrm{Mat}[K_\ell]$, namely the multiplication result of row (d, a, b) acts on at most $k_{\ell-1} \cdot (2s_\ell+1)^2$ components of $\phi_{\ell-1}$. That is, we have $k_{\ell-1}(2s_\ell+1)^2$-sparsity of the rows. Conversely, we also see that at most $k_\ell \cdot (2s_\ell+1)^2$ convolution products $(\mathrm{Mat}[K_\ell]\phi)_{ab}^d$ have a dependency on one particular feature map pixel $(\phi^c)_{\tilde{a}\tilde{b}}$ (i.e., $k_\ell(2s_\ell+1)^2$-sparsity of the columns). Now, let $\ell \in [L-1]$ and assume that we have already computed the gradient

$$\nabla_{K_{\ell+1}}\mathcal{L} = \nabla_{K_{\ell+1}}\ell_{\ell+1}(\phi_{\ell+1}(K_{\ell+1})) = D\ell_{\ell+1}|_{\phi_{\ell+1}} \cdot \alpha_{\ell+1}|_{\psi_{\ell+1}} \cdot \nabla_{K_{\ell+1}}\psi_{\ell+1}, \tag{3.27}$$

then by backpropagation, i.e., eq. (3.19), we obtain

$$\begin{aligned}\nabla_{K_\ell}\mathcal{L} &= \nabla_{K_\ell}[\ell_{\ell+1}\circ\alpha_{\ell+1}\circ\psi_{\ell+1}(\phi_\ell(K_\ell))] \\ &= D\ell_{\ell+1}|_{\phi_{\ell+1}}\cdot\alpha_{\ell+1}|_{\psi_{\ell+1}}\cdot \mathrm{Mat}[K_{\ell+1}]\cdot D\alpha_\ell|_{\psi_\ell}\cdot\nabla_{K_\ell}\psi_\ell.\end{aligned} \tag{3.28}$$

Here, the first two factors have already been computed, hence we obtain a FLOP count for subsequently computing $\nabla_{K_\ell}\mathcal{L}$ of

$$[2\cdot k_{\ell+1}(2s_{\ell+1}+1)^2-1]\cdot[k_\ell(2s_\ell+1)^2]+[2\cdot k_\ell(2s_\ell+1)^2-1]\cdot[k_{\ell-1}(2s_\ell+1)^2] \tag{3.29}$$

via the backpropagation step from $\nabla_{K_{\ell+1}}\mathcal{L}$. The claim in theorem 1 addressing eq. (3.21), follows for $\ell = L-1$ in eq. (3.29).

Finally, we address the computational complexity for sampling-based uncertainty quantification methods with sampling on ϕ_{L-1}. This is applicable, e.g., for dropout on the last layer (as in our experiments) or a deep sub-ensemble [179] sharing the forward pass up to the last layer. Note, that we do not use sub-ensembles in our experiments, but regular deep ensembles. Earlier sampling leads to far higher FLOP counts. Again, we ignore the cost of dropout itself as it is random binary masking together with a respective up-scaling/multiplication of the non-masked entries by a constant. The cost stated in theorem 1 results from the residual forward pass $\phi_{L-1}\mapsto\phi_L=\alpha_L(\mathrm{Mat}[K_L]\cdot\phi_{L-1}+b_L)$ where we now apply previous results. Obtaining all n_L entries in the resulting sample feature map requires a total FLOP count of

$$2n_Lk_{L-1}(2s_L+1)^2-1+n_L \tag{3.30}$$

as claimed, where we have considered the sparsity of $\mathrm{Mat}[K_L]$. The last term results from the bias addition.

Discussion. A large part of the FLOP required to compute gradient features results from the computation of the masks μ^j and the term $D\mathcal{L}^j$ for each relevant predicted box. In table 3.1 we have treated the latter separately and found that, although the counts listed for $D\mathcal{L}^j$ apply to each separate box, sampling post-processing comes with considerable computational complexity as well. In that regard, we have similar costs for gradient features and sampling over the last network layer. Note in particular, that computing $D\mathcal{L}^j$ requires no new evaluation of elementary functions, as opposed to sampling. Once $D\mathcal{L}^j$ is computed for $\widehat{y}^j$, the last layer gradient can be computed in $\mathcal{O}(k_Lk_{L-1})$ and every further gradient for layer V_ℓ in $\mathcal{O}(k_{\ell+1}k_\ell+k_\ell k_{\ell-1})$. Each sample results in $\mathcal{O}(n_Lk_{L-1})$ with sampling on ϕ_{L-1}. Sampling any earlier results in additional full convolution forward passes which also come with considerable computational costs. We note that sampling-based epistemic uncertainty can be computed in parallel with all N_{samp} forward passes being performed simultaneously. Gradient uncertainty features, in contrast, require one full forward pass for the individual gradients $\nabla_{K_\ell}\mathcal{L}(\mu^j\phi_L(K_\ell),\phi^j)$ to be computed. Therefore, gradient uncertainty features experience a slight computational latency as compared to sampling methods. We argue that in principle, all following steps (computation of μ^j and $\nabla_{K_\ell}\mathcal{L}(\mu^j\phi_L(K_\ell),\phi^j)$) can be implemented to run in parallel as no sequential order of computations is required. We have not addressed the computations of mapping the

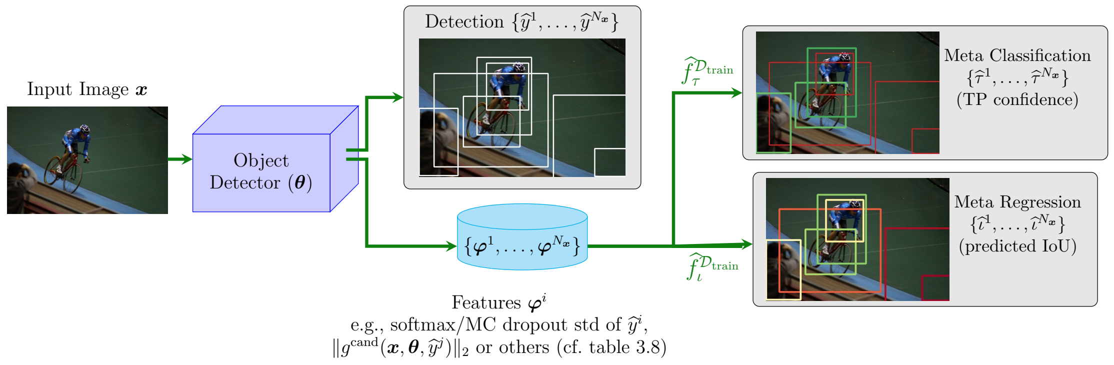

Input Image $\boldsymbol{x}$
Object Detector ($\boldsymbol{\theta}$)
Detection $\{\widehat{y}^1, \ldots, \widehat{y}^{N_{\boldsymbol{x}}}\}$
$\{\boldsymbol{\varphi}^1, \ldots, \boldsymbol{\varphi}^{N_{\boldsymbol{x}}}\}$
Features $\boldsymbol{\varphi}^i$
e.g., softmax/MC dropout std of $\widehat{y}^i$,
$\|g^{\mathrm{cand}}(\boldsymbol{x}, \boldsymbol{\theta}, \widehat{y}^i)\|_2$ or others (cf. table 3.8)
$\widehat{f}_\tau^{\mathcal{D}_{\mathrm{train}}}$
$\widehat{f}_\iota^{\mathcal{D}_{\mathrm{train}}}$
Meta Classification
$\{\widehat{\tau}^1, \ldots, \widehat{\tau}^{N_{\boldsymbol{x}}}\}$
(TP confidence)
Meta Regression
$\{\widehat{\iota}^1, \ldots, \widehat{\iota}^{N_{\boldsymbol{x}}}\}$
(predicted IoU)

Figure 3.2: Meta classification and meta regression pipeline for object detection: An uncertainty feature vector $\boldsymbol{\varphi}^j$ is assigned to each detected box $\widehat{y}^j$. During training, we fit $\widehat{f}_\tau^{\mathcal{D}_{\mathrm{train}}}$ and $\widehat{f}_\iota^{\mathcal{D}_{\mathrm{train}}}$ to map $\boldsymbol{\varphi}^j$ to τ^j (TP/FP) and max. IoU ι^j of $\widehat{y}^j$, resp. At inference, $\widehat{f}_\tau^{\mathcal{D}_{\mathrm{train}}}$ and $\widehat{f}_\iota^{\mathcal{D}_{\mathrm{train}}}$ yield confidence and IoU estimates $\widehat{\tau}^j$ and $\widehat{\iota}^j$ for $\widehat{y}^j$ based on $\boldsymbol{\varphi}^j$. Image taken from the Pascal VOC [38] dataset.

gradients to scalars from eq. (3.4) which are roughly comparable to the cost of computing the sample std for sampling-based methods, especially once the sparsity of $D\mathcal{L}^j$ has been determined in the computation of $\nabla_{K_L}\mathcal{L}$. The latter also brings a significant reduction in FLOP (from n_L to $|\mu^j|$) which cannot be estimated more sharply, however. Since $D\mathcal{L}^j$ is sparse, multiplication from the right with $D\alpha_L|_{\psi_L}$ in eqs. (3.20) and (3.21) for a leaky ReLU activation only leads to lower-order terms. The same terms were also omitted before in determining the computational complexity of sampling uncertainty methods. Also, for this consideration, we regard the fully connected layers used for bounding box regression and classification in the Faster/Cascade R-CNN RoI head as (1×1)-convolutions to stay in the setting presented here.

3.3.2 Meta Classification and Meta Regression: Prediction Quality Estimation in Post-Processing

We evaluate the efficacy of gradient scores in terms of *meta classification and meta regression*. These two approaches allow for the aggregation of potentially large feature vectors to obtain uncertainty estimates for a respective prediction like a bounding box. However, meta classification and meta regression can also *be applied to connected components in semantic segmentation*. The aim of meta classification is to detect FP predictions by generating confidence estimates for the prediction being a TP. Recall that for bounding box predictions, this usually means that the prediction has a IoU with at least one ground truth of more than 0.5. In segmentation, a TP is usually defined as a segment that has a IoU with the ground truth greater than 0, i.e., having at least one pixel class overlap with the ground truth. Meanwhile, meta regression directly estimates the prediction quality, usually in terms of IoU which is non-ambiguous for both semantic segmentation and object detection. Meta classification and meta regression allow for the *unified comparison of different uncertainty quantification methods* and combinations thereof by regarding post-processing models based on different features. Moreover, we are able to investigate the degree of mutual redundancy of different sources of uncertainty. In the following, we summarize this method for bounding box detection and illustrate the scheme in fig. 3.2.

We regard an object detector generating a list of $N_{\boldsymbol{x}}$ detections along with a vector $\boldsymbol{\varphi}^j$ for each predicted bounding box $\widehat{y}^j$. This vector $\boldsymbol{\varphi}^j \in \mathbb{R}^n$ of n "features" may contain gradient scores, but also, e.g., bounding box features, MC dropout or deep ensemble features or combinations thereof (e.g., by concatenation of dropout and ensemble feature vectors). On training data[42] $\mathcal{D}_{\mathrm{train}}$, we compute boxes $\widehat{y}$ and corresponding features $\varphi = (\boldsymbol{\varphi}^1, \dots, \boldsymbol{\varphi}^{N_{\boldsymbol{x}}})$. We evaluate each predicted instance $\widehat{y}^j$ corresponding to the features $\boldsymbol{\varphi}^j$ in terms of their maximal IoU, denoted $\iota^j \in [0,1]$ with the respective ground truth and determine FP/TP labels $\tau^j \in \{0,1\}$. A *meta classifier* is a lightweight classification model $\widehat{f}_\tau : \mathbb{R}^n \to (0,1)$ giving probabilities for the classification of $\boldsymbol{\varphi}^j$ (vicariously for the uncertainty of $\widehat{y}^j$) as TP which we fit on $\mathcal{D}_{\mathrm{train}}$. Similarly, a *meta regression* model $\widehat{f}_\iota : \mathbb{R}^n \to \mathbb{R}$ is fit to the maximum IoU ι^j of $\widehat{y}^j$ with the ground truth of $\boldsymbol{x}$. The models $\widehat{f}_\tau^{\mathcal{D}_{\mathrm{train}}}$ and $\widehat{f}_\iota^{\mathcal{D}_{\mathrm{train}}}$ can be regarded as post-processing modules which generate confidence measures given an input to an object detector leading to features φ^j. At inference time, we then obtain box-wise classification probabilities $\widehat{\tau}^j = \widehat{f}_\tau^{\mathcal{D}_{\mathrm{train}}}(\boldsymbol{\varphi}^j)$ and IoU predictions $\widehat{\iota}^j = \widehat{f}_\iota^{\mathcal{D}_{\mathrm{train}}}(\boldsymbol{\varphi}^j)$. We then determine the predictive power of $\widehat{f}_\tau^{\mathcal{D}_{\mathrm{train}}}$ and $\widehat{f}_\iota^{\mathcal{D}_{\mathrm{train}}}$ in terms of their AuROC, (recall section 2.3.1) or AP metrics and the determination coefficient R^2, respectively.

Similarly, meta classification and meta regression act in semantic segmentation on feature vectors $\boldsymbol{\varphi}^j$ for each predicted segment $\widehat{S}^j$ which contains uncertainty features specific to $\widehat{S}^j$ like the features computed in the MetaSeg framework [152], see also section 4.4.1.1. Again, a meta classifier is a post-processing model $\widehat{f}_\tau^{\mathcal{D}_{\mathrm{train}}}$ estimating the probability for $\widehat{S}^j$ having a IoU greater than 0 with the ground truth. A meta regression model $\widehat{f}_\iota^{\mathcal{D}_{\mathrm{train}}}$ estimates the IoU of $\widehat{S}^j$ with the ground truth based on $\boldsymbol{\varphi}^j$.

MetaFusion: Object Detection Post-Processing. As a direct application of uncertainty quantification, we investigate an approach inspired by [16]. We *implement meta classification into the object detection pipeline* by assigning each output box in $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ its meta classification probability $\widehat{\tau}$ as prediction confidence as shown in fig. 3.1. State-of-the-art object detectors use score thresholding in addition to NMS which we compare with confidence filtering based on meta classification. For most competitive uncertainty baselines in our experiments, computation for the entirety of the network proposals $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ is expensive. Hence, we implement a small score threshold which still allows for a large amount of predicted boxes of ~ 150 bounding boxes per image. This way, well-performing meta classifiers which accurately detect FPs, together with an increase in detection sensitivity offer a way to "trade" uncertainty information for detection performance. In most object detection pipelines, score thresholding is carried out before NMS. We choose to interchange them here as they commute for the baseline approach. The resulting predictions are compared for a range of confidence thresholds in terms of mean Average Precision (see section 2.3.2). Figure 3.3 shows a sketch of the resulting MetaFusion pipeline, where the usual object detection pipeline is shown in blue. The standard object detection

[42]Note, that this is training data for the meta classifier, respectively meta regression model. In practice, this is oftentimes disjoint from the training data of the base deep learning model which $\boldsymbol{\varphi}$ is computed with.

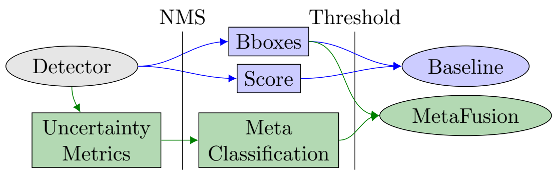


Figure 3.3: Schematic sketch of the baseline detection pipeline and the alternative MetaFusion pipeline for an object detector.

pipeline relies on filtering out false positive output boxes on the basis of their score, see also fig. 3.1. An altered confidence estimation like meta classification can improve the threshold-dependent detection quality of the object detection pipeline. This way, boxes which are falsely assigned a low score can survive the thresholding step. Similarly, FPs with a high score may be suppressed by proper predictive confidence estimation methods. This approach is not limited to meta classification, however, our experiments show that meta classification constitutes such a method.

Calibration: Reliability of Confidence Assignments. Generally, *calibration* methods aim at rectifying scores as confidences in the sense of section 3.1 such that the calibrated scores *reflect the conditional frequency of true predictions*. For example, out of 100 predictions with a confidence of 0.3, around 30 should be correct, say, TP. In formal terms, this means that

$$\Pr\big(\mathrm{TP}(\widehat{y}^j(\boldsymbol{X})|\overline{Y})\big|\,\widehat{s}^j(\boldsymbol{X}) = p\big) = p \qquad \forall p \in [0,1], \tag{3.31}$$

where we denote by $\mathrm{TP}(\widehat{y}^j(\boldsymbol{X})|\overline{Y})$ the logical statement of the prediction $\widehat{y}^j(\boldsymbol{X})$ being a TP prediction when regarding the ground truth $\overline{Y}$. Also, $\widehat{s}^j(\boldsymbol{X})$ stands representative for any confidence estimation assigned to the prediction $\widehat{y}^j$ which may be an object detector's objectness score or the confidence given by a meta classifier. Calibration can be defined in semantic segmentation analogously on segment level by what a TP is logically. Here, $\widehat{s}^j$ can again be seen as the confidence of a meta classifier for the respectively predicted segment $\widehat{S}^j$. Moreover, pixel-wise evaluation is possible in semantic segmentation, where $\widehat{s}^j$ represents some pixel-wise confidence value. Such confidence values could be the maximum softmax probability of the segmentation network or some pixel-wise gradient score, see chapter 4. Since the statement in eq. (3.31) is a continuous statement, discretized variants are usually investigated in experiments where a finite dataset is employed. One resorts to a fixed partition of the range $p \in [0,1]$ into usually equally-sized bins $\beta_1, \ldots, \beta_B$ and computes the frequency of TP predictions (accuracy) and average confidence for each bin. In our experiments, we sort the examples into bins of fixed width 0.1 according to their confidence, see fig. 3.4. For each bin β_i, we compute

$$\mathrm{acc}_i = \frac{\mathrm{TP}_i}{|\beta_i|}, \qquad \mathrm{conf}_i = \frac{1}{|\beta_i|}\sum_{j=1}^{|\beta_i|} \widehat{s}_i \tag{3.32}$$

where $|\beta_i|$ denotes the number of examples in β_i and $\widehat{s}_i$ is the respective confidence, i.e., the network's score or a meta classification probability. TP_i denotes the number of correctly classified in β_i. In standard classification tasks, this boils down to the classification

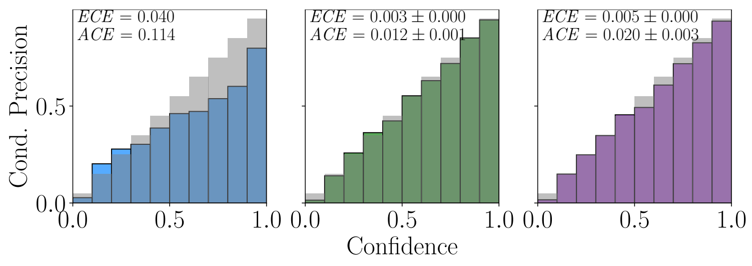


Figure 3.4: Reliability plots of the Score (*left*) and meta classifiers for MD (*center*) and $\mathrm{GS}_{\mathrm{full}}$ (*right*) on the VOC dataset (YOLOv3) with calibration errors (mean ± std). The gray diagonal shows optimal calibration.

accuracy, whereas in the object detection setting, this is the detector's precision on the bin β_i. A meta classifier performs a binary classification on detector positives, so we keep with the notation used for classifiers. Calibration metrics are usually defined as functions of the bin-wise differences between acc_i and conf_i in the manner described below. One of the definition given is specialized for object detection.

Confidence calibration methods have been previously applied to object detection in [128] where temperature scaling was found to improve calibration. In addition to considering the *expected calibration error* (*ECE*) and the *maximum calibration error* (*MCE*) [126], the authors of [128] argue that in object detection, it is important that confidences are calibrated irrespective of how many examples fall into a bin. Therefore, they introduced the *average calibration error* (*ACE*) as a new calibration metric which is insensitive to the bin counts. In section 3.4, we evaluate the calibration of meta classifiers in terms of the expected, maximum and average calibration error metrics:

$$MCE = \max_{i \in [B]} |\mathrm{acc}_i - \mathrm{conf}_i|, \;\; ACE = \frac{1}{B} \sum_{i \in [B]} |\mathrm{acc}_i - \mathrm{conf}_i|, \;\; ECE = \sum_{i \in [B]} \frac{1}{|\beta_i|} |\mathrm{acc}_i - \mathrm{conf}_i|. \tag{3.33}$$

The main difference between *ECE* and *ACE* is that the expected calibration error scales the calibration errors *according to the population of each bin.* This means that calibration errors of heavily populated bins will be scaled down proportionally. In object detection where it is common that confidences of foreground and background, instances strongly populate the outer bins, calibration errors are down-weighted due to the mass of instances per bin. The *ACE* metric treats all bins equally, irrespective of their population.

3.4 Experiments: Meta Classification, Meta Regression and Runtime

In this section, we report our numerical methods and experimental findings. We investigate meta classification and meta regression on three object detection datasets, namely Pascal VOC [38], MS COCO [102] and KITTI [46]. We investigate gradient-based meta classification and meta regression for only 2-norm scalars, denoted $\mathrm{GS}_{\|\cdot\|_2}$ (refer to section 3.3.1)

Table 3.2: Number of layers and losses utilized and resulting numbers of gradients per box. Multiplication in # layers denotes parallel output strands of the respective DNN (no additional gradients).

Architecture	# layers	# Losses	# gradients
YOLOv3	2×3	3	6
Faster R-CNN	2×4	4	8
RetinaNet	2×2	2	4
Cascade R-CNN	2×8	8	16

Table 3.3: Dataset splits used for training and evaluation of object detectors. Note, that we train meta classifiers and meta regressors on a validation part of the evaluation split and evaluate it on the complementary split.

Dataset	training	evaluation	# eval images
VOC	2007+2012 trainval	2007 test	4952
COCO	train2017	val2017	5000
KITTI	random part of training	complement part of training	2000

as well as the larger model for all maps listed in eq. (3.4), denoted $\mathrm{GS}_{\mathrm{full}}$. $\mathrm{GS}_{\mathrm{full}}$ is always computed for the last two network layers (unless specified otherwise) of each architectural branch and for each contribution to the loss function $\mathcal{L}$ separately, i.e., for classification, bounding box regression and, if applicable, objectness score (see section 2.3.2). We list the resulting counts and number of gradients per investigated architecture in table 3.2. As meta classifiers and meta regressors, we use gradient boosting models which have been shown [114, 161, 181] to perform well as such. Whenever we indicate means and standard deviations, we obtained those by 10-fold image-wise cross validation (short: "cv") for the training split $\mathcal{D}_{\mathrm{train}}$ of the meta classifier or meta regression model. Evaluation is done on the complement of $\mathcal{D}_{\mathrm{train}}$.

3.4.1 *Implementation Details*

Here, we state details of the implementations of our framework to different architectures, and on different datasets.

Datasets. In order to show a wide range of applications, we investigate our method on the following object detection datasets, see table 3.3 for the splits used. Meta classification and meta regression models are as post-processing modules fitted on a validation sample of the evaluation dataset. Their performance is evaluated on the complementary sample of the evaluation dataset in cross-validation.

The **Pascal VOC 2007+2012** [38] dataset is an object detection benchmark of everyday images involving 20 different object categories. We train on the 2007 and 2012 "trainval" splits, amounting to 16.550 train images, and we evaluate on the 2007 test split of 4.952 images. For training, we include labels marked as difficult in the original annotations.

The **MS COCO 2017** [102] dataset constitutes a second vision benchmark involving 2D bounding box detection annotations for everyday images with 80 object categories. We train on the "train2017" split of 118.287 images and evaluate on the 5.000 images of the "val2017" split.

The **KITTI** [46] vision benchmark contains 21 real world street scenes annotated with 2D bounding boxes. We randomly divide the 7.481 labeled images into a training split of 5.481 images and use the complement of 2.000 images for evaluation.

Detectors. For our experiments, we employ three common object detection architectures, namely YOLOv3 with Darknet53 backbone [39], Faster R-CNN [144] and RetinaNet [101], each with a ResNet50 [61] backbone. Moreover, we investigate a state-of-the-art detector in Cascade R-CNN [12] with a large ResNeSt200 [215] backbone. We started from PyTorch [136] reimplementations, added dropout layers and trained from scratch on the datasets in table 3.3. We list some detector-specific details.

The basis of our **YOLOv3** implementation is a publicly available GitHub repository [189]. We position dropout layers with $p = 0.5$ before the last convolutional layers of each detection head. Gradient features are computed over the last two layers in each of the three detection heads as the final network layers have been found to be most informative in the classification setting [132]. Since each output box is the result of exactly one of the three heads, we only have two layers for gradients per box resulting in 2×3 gradients per box (2 layers per 3 losses) as indicated in table 3.2. We train an ensemble of 5 detectors for each dataset from scratch.

Based on the official Torchvision implementation, our **Faster R-CNN** model uses dropout ($p = 0.5$) before the last fully connected layer of the architecture, i.e., classification and bounding box prediction in the Fast R-CNN head. We compute gradient features for the last two fully connected layers of the Fast R-CNN head as well as for the last two convolutional layers of the RPN per box, i.e., objectness and localization. This leads to 4×2 gradients per box: $2 + 2$ for localization, 2 for classification and 2 for proposal objectness.

We also employ **RetinaNet** as implemented in Torchvision with ($p = 0.5$)-dropout before the last convolutional layers for bounding box regression and classification. Gradients are computed for the last two convolutional layers for bounding box regression and classification resulting in 2×2 gradients per prediction.

For the **Cascade R-CNN** detector, we use the Detectron2 [190]-supported implementation of ResNeSt provided by the ResNeSt authors Zhang et al. [215] and the pre-trained weights on the MS COCO dataset. We train from scratch on Pascal VOC and KITTI. Since this model is primarily interesting for investigation due to its naturally strong score baseline based on cascaded regression, we do not report MC dropout results for it. Gradient uncertainty features are computed for the last two fully connected layers, i.e., bounding box regression and classification, of each of the three cascades. The loss of later cascade stages depends in principle on the weights of previous cascade stages. However, we only compute the gradients with respect to the weights in the current stage resulting in 2×6 (3 stages for bounding box regression and classification) gradients for the Cascade R-CNN head. Furthermore, we have the 2×2 RPN gradients as in Faster R-CNN.

Implemented Loss Functions. Here, we give a short account of the loss functions implemented in our experiments.

The loss function we used to train **YOLOv3** operates on the pre-transformation activations defined in eq. (2.146) to which the bounding box features $\mathrm{b} = (\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h}, \kappa)$ of the ground truth annotations relate respectively. The latter are first transformed to pseudo activations $\phi_\mathrm{x}, \phi_\mathrm{y}, \phi_\mathrm{w}, \phi_\mathrm{h}$ in the respective manner following eq. (2.146). The objectness loss utilized is the binary cross entropy loss defined in eq. (2.140) and classification follows the one-versus-all classification loss in eq. (2.133). Since the transformations defined in eq. (2.146) assume sigmoid activations in order to compute $\widehat{\mathrm{x}}$ and $\widehat{\mathrm{y}}$, they can be learned also via a binary cross entropy loss which is implemented in our framework. Since we will be interested in the number of FLOPs required to compute the loss derivatives later, we will adapt a notation here which is closer to the implementation.

$$\mathcal{L}_\xi^{\mathrm{Yv3}}(\widehat{y}, \overline{y}) = 2 \sum_{a=1}^{N_\mathrm{out}} \sum_{t=1}^{N_{\boldsymbol{x}}} \mathbb{1}_{at}^{\mathrm{obj}} \left[\mathcal{L}_\mathrm{MSE} \left(\begin{pmatrix} \widehat{\phi}_\mathrm{w}^a \\ \widehat{\phi}_\mathrm{h}^a \end{pmatrix} \middle| \begin{pmatrix} \phi_\mathrm{w}^t \\ \phi_\mathrm{h}^t \end{pmatrix} \right) + \mathcal{L}_\mathrm{BCE} \left(\sigma \begin{pmatrix} \widehat{\phi}_\mathrm{x}^a \\ \widehat{\phi}_\mathrm{y}^a \end{pmatrix} \middle| \sigma \begin{pmatrix} \phi_\mathrm{x}^t \\ \phi_\mathrm{y}^t \end{pmatrix} \right) \right], \tag{3.34}$$

$$\mathcal{L}_s^{\mathrm{Yv3}}(\widehat{y}, \overline{y}) = \sum_{a=1}^{N_\mathrm{out}} \sum_{t=1}^{N_{\boldsymbol{x}}} \left[\mathbb{1}_{at}^{\mathrm{obj}} \mathcal{L}_\mathrm{BCE} \left(\sigma(\widehat{\phi}_s^a) \middle| \mathbf{1}_{N_\mathrm{out}} \right) + \mathbb{1}_{at}^{\mathrm{noobj}} \mathcal{L}_\mathrm{BCE} \left(\sigma(\widehat{\phi}_s^a) \middle| \mathbf{0}_{N_\mathrm{out}} \right) \right], \tag{3.35}$$

$$\mathcal{L}_p^{\mathrm{Yv3}}(\widehat{y}, \overline{y}) = \sum_{a=1}^{N_\mathrm{out}} \sum_{t=1}^{N_{\boldsymbol{x}}} \mathbb{1}_{at}^{\mathrm{obj}} \mathcal{L}_\mathrm{BCE} \left(\sigma(\widehat{\phi}_p^a) \middle| \sigma(\phi_p^t) \right). \tag{3.36}$$

Here, the first sum ranges over all N_out anchors[43] a and the second sum over the total number N_gt of ground truth instances in $\overline{y}$. Note, that for the regression objective, the binary cross entropy $\mathcal{L}_\mathrm{BCE}$ must allow for non-onehot targets, i.e.,

$$\mathrm{BCE}(p|q) = -\sum_{i \in [d]} q_i \log(p_i) + (1 - q_i) \log(1 - p_i), \tag{3.37}$$

where $p, q \in (0,1)^d$ for some fixed length $d \in \mathbb{N}$. Using binary cross entropy for classification amounts to learning C binary classifiers, in particular the probabilities are in general not normalized. Note also, that each summand in eq. (3.35) only has one contribution due to the binary ground truth $\mathbf{1}_{N_\mathrm{out}}$, resp. $\mathbf{0}_{N_\mathrm{out}}$. The binary cross entropy is also sometimes used for the center location of anchor boxes when the position within each cell is scaled to $(0, 1)$, see $\mathcal{L}_\xi^{\mathrm{YOLOv3}}$.

Since **Faster R-CNN** [144] and **Cascade R-CNN** [12] are two-stage architectures, there are separate loss contributions for the Region Proposal Network (RPN) and the Region of Interest (RoI) head, the latter of which produces the actual proposals, cf. section 2.3.2. Formally, writing $\phi_\xi^\mathrm{RPN} := (\phi_\mathrm{x}^\mathrm{RPN}, \phi_\mathrm{y}^\mathrm{RPN}, \phi_\mathrm{w}^\mathrm{RPN}, \phi_\mathrm{h}^\mathrm{RPN})$ for the respectively transformed ground truth localization, similarly $\widehat{\phi}_\xi^\mathrm{RPN}$ for the RPN outputs $\widehat{\xi}^\mathrm{RPN}$ and $\widehat{\phi}_s^\mathrm{RPN}$ the proposal score output (where $\widehat{s}^a = \sigma((\widehat{\phi}_s^\mathrm{RPN})^a)$ is the proposal score):

[43]These translate directly to indices (i, j, n) as described in section 2.3.2. The tensors $\mathbb{1}^\mathrm{obj}$ and $\mathbb{1}^\mathrm{noobj}$ translate to assignments to foreground boxes and the background class, respectively.

$$
\mathcal{L}_\xi^{\mathrm{RPN}}(\widehat{\phi}^{\mathrm{RPN}}|\overline{y}) = \frac{1}{|I^+|} \sum_{a=1}^{N_{\mathrm{out}}^{\mathrm{RPN}}} \sum_{t=1}^{N_{\boldsymbol{x}}} I_a^+ \mathbb{1}_{at}^{\mathrm{obj}} \, L_{\mathrm{sm}}^1 \left((\widehat{\phi}_\xi^{\mathrm{RPN}})^a \middle| (\phi_\xi^{\mathrm{RPN}})^t \right), \tag{3.38}
$$

$$
\begin{aligned}
\mathcal{L}_s^{\mathrm{RPN}}(\widehat{\phi}^{\mathrm{RPN}}|\overline{y}) = \sum_{a=1}^{N_{\mathrm{out}}^{\mathrm{RPN}}} \sum_{t=1}^{N_{\boldsymbol{x}}} &\Big[\mathbb{1}_{at}^{\mathrm{obj}} \mathcal{L}_{\mathrm{BCE}} \left(\sigma(\widehat{\phi}_s^{\mathrm{RPN}})^a \middle| \mathbf{1}_{N_{\mathrm{out}}^{\mathrm{RPN}}} \right) \\
&+ I_a^- \mathbb{1}_{at}^{\mathrm{noobj}} \mathcal{L}_{\mathrm{BCE}} \left(\sigma(\widehat{\phi}_s^{\mathrm{RPN}})^a \middle| \mathbf{0}_{N_{\mathrm{out}}^{\mathrm{RPN}}} \right) \Big].
\end{aligned} \tag{3.39}
$$

Predictions are randomly sampled to contribute to the loss function by the tensors I^+ and I^-, which can be regarded as random variables. The constant batch size B of predictions to enter the RPN loss is a hyperparameter set to 256 in our implementation. We randomly sample $n_+ := \min\{|\tilde{\mathbb{1}}^{\mathrm{obj}}|, B/2\}$ of the $|\tilde{\mathbb{1}}^{\mathrm{obj}}|$ positive anchors (constituting the mask I^+) and $n_- := \min\{|\tilde{\mathbb{1}}^{\mathrm{noobj}}|, B - n_+\}$ negative anchors (I^-). The summation of a ranges over the $N_{\mathrm{out}}^{\mathrm{RPN}}$ outputs of the RPN (in our case, 1.000). For the smooth L^1 loss, we use the default parameter choice $\beta = \frac{1}{9}$. Denoting with $\phi_\xi := (\phi_{\mathrm{x}}, \phi_{\mathrm{y}}, \phi_{\mathrm{w}}, \phi_{\mathrm{h}})$ ground truth localization transformed relatively to the respective proposal:

$$
\mathcal{L}_\xi^{\mathrm{RoI}}(\widehat{\phi}^{\mathrm{RoI}}, \overline{y}) = \frac{1}{|\mathbb{1}^{\mathrm{obj}}|} \sum_{a=1}^{N_{\mathrm{out}}} \sum_{t=1}^{N_{\boldsymbol{x}}} \mathbb{1}_{at}^{\mathrm{obj}} \, \mathrm{sm}L_\beta \left((\widehat{\phi}_\xi^{\mathrm{RoI}})^a \middle| \phi_\xi^t \right), \tag{3.40}
$$

$$
\mathcal{L}_p^{\mathrm{RoI}}(\widehat{\phi}^{\mathrm{RoI}}, \overline{y}) = \sum_{a=1}^{N_{\mathrm{out}}} \sum_{t=1}^{N_{\boldsymbol{x}}} \Big[\mathbb{1}_{at}^{\mathrm{obj}} \mathcal{L}_{\mathrm{CE}}(\varSigma((\widehat{\phi}_p^{\mathrm{RoI}})^a)|p^t) + \mathbb{1}_{at}^{\mathrm{noobj}} \, \mathcal{L}_{\mathrm{CE}}(\varSigma((\widehat{\phi}_{p_0}^{\mathrm{RoI}})^a)|1) \Big]. \tag{3.41}
$$

The cascaded bounding box regression of Cascade R-CNN implements the smooth L^1 loss at each of three cascade stages, where bounding box offsets and scaling are computed from the previous bounding box regression results as proposals.

In the **RetinaNet** [101] architecture, score assignment is part of the classification.

$$
\mathcal{L}_\xi^{\mathrm{Ret}}(\widehat{\phi}, \overline{y}) = \frac{1}{|\mathbb{1}^{\mathrm{obj}}|} \sum_{a=1}^{N_{\mathrm{out}}} \sum_{t=1}^{N_{\boldsymbol{x}}} \mathbb{1}_{at}^{\mathrm{obj}} \frac{1}{4} \left\| \widehat{\phi}_\xi^a - \phi_\xi^t \right\|_{L^1}, \tag{3.42}
$$

$$
\begin{aligned}
\mathcal{L}_p^{\mathrm{Ret}}(\widehat{\phi}, \overline{y}) = \frac{1}{|\mathbb{1}^{\mathrm{obj}}|} \sum_{a=1}^{N_{\mathrm{out}}} \sum_{t=1}^{N_{\boldsymbol{x}}} &\Big[\mathbb{1}_{at}^{\mathrm{obj}} \sum_{j=1}^{C} \alpha (1 - \sigma(\widehat{\phi}_j^a))^{\gamma_{\mathrm{Foc}}} \cdot \mathcal{L}_{\mathrm{BCE}} \left(\sigma(\widehat{\phi}_j^a) | \delta_{j,\kappa^t} \right) \\
&+ \mathbb{1}_{at}^{\mathrm{noobj}} (1 - \alpha) \sigma(\widehat{\phi}_0^a)^{\gamma_{\mathrm{Foc}}} \cdot \mathcal{L}_{\mathrm{BCE}} \left(\sigma(\widehat{\phi}_0^a) | 0 \right) \Big].
\end{aligned} \tag{3.43}
$$

The classification loss is a version of the well-known focal loss with $\alpha = 0.25$ and $\gamma_{\mathrm{F}} = 2$. Bounding box transformation follows the maps of the RPN transformations in eq. (2.147).

Theoretical Loss Derivatives. Here, we symbolically compute the loss gradients with respect to the network outputs as obtained from our accounts of the loss functions in the previous paragraphs. We do so in order to determine the computational complexity for

$D_f\mathcal{L}(f|y)|_{f=\widehat{\mu}_{n|\boldsymbol{X}=\boldsymbol{x}}}$, see eq. (2.66) in section 3.3.1.4. Note, that for all derivatives of the cross entropy, we can use

$$\frac{\mathrm{d}}{\mathrm{d}\phi}\left[-y\log(\sigma(\phi)) - (1-y)\log(1-\sigma(\phi))\right] = \sigma(\phi) - y. \tag{3.44}$$

We then find for $b = 1, \dots, N_{\text{out}}$ and features $r \in \{\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h}, s\} \cup [C]$

$$\frac{\partial}{\partial\widehat{\phi}_r^b}\mathcal{L}_\xi^{\mathrm{Yv3}} = 2\sum_{t=1}^{N_{\boldsymbol{x}}}\mathbb{1}_{bt}^{\mathrm{obj}}\begin{cases}\widehat{\phi}_r^b - \phi_r^t & r \in \{\mathrm{w},\mathrm{h}\}\\ \sigma(\widehat{\phi}_r^b) - \sigma(\phi_r^t) & r \in \{\mathrm{x},\mathrm{y}\}\\ 0 & \text{otherwise.}\end{cases}, \tag{3.45}$$

$$\frac{\partial}{\partial\widehat{\phi}_r^b}\mathcal{L}_s^{\mathrm{Yv3}} = \delta_{rs}\sum_{t=1}^{N_{\boldsymbol{x}}}\left[\mathbb{1}_{bt}^{\mathrm{obj}}(\widehat{\phi}_s^b - 1) + \mathbb{1}_{bt}^{\mathrm{noobj}}\widehat{\phi}_s^b\right], \tag{3.46}$$

$$\frac{\partial}{\partial\widehat{\phi}_r^b}\mathcal{L}_p^{\mathrm{Yv3}} = \sum_{t=1}^{N_{\boldsymbol{x}}}\mathbb{1}_{bt}^{\mathrm{obj}}\sum_{i=1}^{C}\delta_{r,i}(\sigma(\widehat{\phi}_i^b) - \sigma(\phi_i^t)), \tag{3.47}$$

where δ_{ij} is the Kronecker symbol, i.e., $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. Further, with analogous notation for the output variables of RPN and RoI

$$\frac{\partial\mathcal{L}_\xi^{\mathrm{RPN}}}{\partial(\widehat{\phi}_r^{\mathrm{RPN}})^b} = \frac{1}{|I^+|}\sum_{t=1}^{N_{\boldsymbol{x}}}I_b^+\tilde{\mathbb{1}}_{bt}^{\mathrm{obj}}\begin{cases}(\widehat{\phi}_r^{\mathrm{RPN}})^b - (\phi_r^{\mathrm{RPN}})^t & |(\widehat{\phi}_r^{\mathrm{RPN}})^b - (\phi_r^{\mathrm{RPN}})^t| < \beta,\\ & \quad r \in \{\mathrm{x},\mathrm{y},\mathrm{w},\mathrm{h}\}\\ \mathrm{sgn}((\widehat{\phi}_r^{\mathrm{RPN}})^b - (\phi_r^{\mathrm{RPN}})^t) & |(\widehat{\phi}_r^{\mathrm{RPN}})^b - (\phi_r^{\mathrm{RPN}})^t| \geq \beta,\\ & \quad r \in \{\mathrm{x},\mathrm{y},\mathrm{w},\mathrm{h}\}\\ 0 & \text{otherwise}\end{cases} \tag{3.48}$$

$$\frac{\partial\mathcal{L}_s^{\mathrm{RPN}}}{\partial(\widehat{\phi}_r^{\mathrm{RPN}})^b} = \delta_{rs}\left[I_b^+\tilde{\mathbb{1}}_{bt}^{\mathrm{obj}}(\widehat{s}^b - 1) + I_b^-\tilde{\mathbb{1}}_{bt}^{\mathrm{noobj}}\widehat{s}^b\right]. \tag{3.49}$$

Here, sgn denotes the sign function, which is the derivative of $|\cdot|$ except for the origin. Similarly,

$$\frac{\partial}{\partial\widehat{\phi}_r^b}\mathcal{L}_\xi^{\mathrm{RoI}} = \frac{1}{|\mathbb{1}^{\mathrm{obj}}|}\sum_{t=1}^{N_{\boldsymbol{x}}}\mathbb{1}_{bt}^{\mathrm{obj}}\cdot\begin{cases}\widehat{\phi}_r^b - \phi_r^t & |\widehat{\phi}_r^b - \phi_r^t| < \beta \text{ and}\\ & \quad r \in \{\mathrm{x},\mathrm{y},\mathrm{w},\mathrm{h}\}\\ \mathrm{sgn}(\widehat{\phi}_r^b - \phi_r^t) & |\widehat{\phi}_r^b - \phi_r^t| \geq \beta \text{ and}\\ & \quad r \in \{\mathrm{x},\mathrm{y},\mathrm{w},\mathrm{h}\}\\ 0 & \text{otherwise}\end{cases}, \tag{3.50}$$

$$\frac{\partial}{\partial\widehat{\phi}_r^b}\mathcal{L}_p^{\mathrm{RoI}} = -\sum_{t=1}^{N_{\boldsymbol{x}}}\left[\mathbb{1}_{bt}^{\mathrm{obj}}\sum_{j=1}^{C}p_j^t\left(\delta_{j,r} - \sum_{k=0}^{C}\delta_{p_k r}\Sigma^k(\widehat{\phi}_\pi^b)\right) + \mathbb{1}_{bt}^{\mathrm{noobj}}\left(\delta_{0,r} - \sum_{k=0}^{C}\delta_{k,r}\Sigma^k(\widehat{\phi}_\pi^b)\right)\right]. \tag{3.51}$$

Table 3.4: Ablation on the temperature parameter T for the energy score in terms of meta classification (AuROC and AP) and meta regression (R^2).

	AuROC	AP	R^2
$T = 1$	92.52 ± 0.03	91.86 ± 0.04	62.12 ± 0.09
$T = 10$	78.42 ± 0.13	81.75 ± 0.08	32.92 ± 0.20
$T = 100$	95.66 ± 0.02	95.33 ± 0.03	71.79 ± 0.06
$T = 1.000$	95.62 ± 0.03	95.33 ± 0.04	71.78 ± 0.05
Score	96.53 ± 0.05	96.87 ± 0.03	78.86 ± 0.05
MD	$\mathbf{98.23 \pm 0.02}$	$\mathbf{98.06 \pm 0.02}$	$\mathbf{85.88 \pm 0.10}$
$\mathrm{GS}_{\mathrm{full}}$	$\underline{98.04 \pm 0.03}$	$\underline{97.81 \pm 0.06}$	$\underline{85.40 \pm 0.11}$

Note that the inner sum over j only has at most one term due to $\delta_{j,r}$. With $\sigma'(\phi) = \sigma(\phi)(1 - \sigma(\phi))$, we finally find for RetinaNet

$$\frac{\partial}{\partial \widehat{\phi}_r^b} \mathcal{L}_\xi^{\mathrm{Ret}} = \frac{1}{|\mathbb{1}^{\mathrm{obj}}|} \sum_{t=1}^{N_{\boldsymbol{x}}} \mathbb{1}_{bt}^{\mathrm{obj}} \cdot \begin{cases} \mathrm{sgn}(\widehat{\phi}_r^b - \phi_r^t) & r \in \{\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h}\} \\ 0 & \text{otherwise} \end{cases}, \tag{3.52}$$

$$\begin{aligned} \frac{\partial}{\partial \widehat{\phi}_r^b} \mathcal{L}_p^{\mathrm{Ret}} = \frac{1}{|\mathbb{1}^{\mathrm{obj}}|} \sum_{t=1}^{N_{\boldsymbol{x}}} \Bigg[& \mathbb{1}_{bt}^{\mathrm{obj}} \sum_{j=1}^{C} \delta_{j,r} \alpha (1 - \sigma(\widehat{\phi}_j^b))^{\gamma_{\mathrm{Foc}}} \cdot \\ & \cdot [-\gamma_{\mathrm{F}} \sigma(\widehat{\phi}_j^b) \mathcal{L}_{\mathrm{BCE}}(\sigma((\widehat{\phi}_\pi^b)_j), \kappa^t) + \sigma(\widehat{\phi}_j^b) - 1] \\ & + \mathbb{1}_{bt}^{\mathrm{noobj}} \delta_{0,r} (1 - \alpha) \sigma(\widehat{\phi}_0^b)^{\gamma_{\mathrm{Foc}}} \cdot [-\gamma_{\mathrm{Foc}} (1 - \sigma(\widehat{\phi}_0^b)) \log(1 - \sigma(\widehat{\phi}_0^b)) + \sigma(\widehat{\phi}_0^b)] \Bigg]. \end{aligned} \tag{3.53}$$

Uncertainty Baselines. We give a short account of the baselines implemented and investigated in our experiments.

By the **Score**, we mean the box-wise objectness score for YOLOv3 and the maximum softmax probability for Faster R-CNN, RetinaNet and Cascade R-CNN. As standard object detection pipelines discard output bounding boxes based on a score threshold, this quantity is the baseline for discriminating true against false outputs.

The **Entropy** is a common hand-crafted uncertainty measure based on the classification output $\widehat{\pi} \in [0,1]^C$ (softmax or category-wise sigmoid) and given by

$$H(\widehat{\pi}) = -\sum_{c=1}^{C} \widehat{\pi}_c \log(\widehat{\pi}_c), \tag{3.54}$$

which is simply eq. (2.91) with adjusted notation.

As an alternative to the maximum softmax probability and the entropy, Liu et al. proposed an **Energy score** depending on a temperature parameter T given by

$$E\left(\widehat{\phi}\right) = -T \log \sum_{c=1}^{C} \mathrm{e}^{\widehat{\phi}_c / T} \tag{3.55}$$

Table 3.5: Ablation on the sample count size N_{DO} for MC dropout in terms of meta classification (AuROC and AP) and meta regression (R^2). Results obtained from the sample standard deviation.

	AuROC	AP	R^2
$N_{\mathrm{DO}} = 10$	97.40 ± 0.04	96.91 ± 0.06	80.85 ± 0.10
$N_{\mathrm{DO}} = 15$	97.50 ± 0.03	97.08 ± 0.07	81.28 ± 0.09
$N_{\mathrm{DO}} = 20$	97.69 ± 0.03	97.28 ± 0.05	82.11 ± 0.09
$N_{\mathrm{DO}} = 25$	97.64 ± 0.03	97.20 ± 0.04	81.94 ± 0.12
$N_{\mathrm{DO}} = 30$	97.60 ± 0.07	97.17 ± 0.10	82.10 ± 0.11
$N_{\mathrm{DO}} = 35$	97.71 ± 0.03	97.29 ± 0.05	82.17 ± 0.13
$N_{\mathrm{DO}} = 40$	97.69 ± 0.04	97.29 ± 0.06	82.12 ± 0.13
Score	96.53 ± 0.05	96.87 ± 0.03	78.86 ± 0.05
MD	$\mathbf{98.23 \pm 0.02}$	$\mathbf{98.06 \pm 0.02}$	$\mathbf{85.88 \pm 0.10}$
$\mathrm{GS}_{\mathrm{full}}$	$\underline{98.04 \pm 0.03}$	$\underline{97.81 \pm 0.06}$	$\underline{85.40 \pm 0.11}$

Table 3.6: Ablation on the ensemble size N_{DE} for deep ensembles in terms of meta classification (AuROC and AP) and meta regression (R^2). Results obtained from the sample standard deviation.

	AuROC	AP	R^2
$N_{\mathrm{DE}} = 3$	97.53 ± 0.03	97.17 ± 0.05	82.63 ± 0.13
$N_{\mathrm{DE}} = 4$	97.79 ± 0.04	97.48 ± 0.06	83.62 ± 0.12
$N_{\mathrm{DE}} = 5$	97.92 ± 0.04	97.63 ± 0.05	84.18 ± 0.12
$N_{\mathrm{DE}} = 6$	98.04 ± 0.03	97.75 ± 0.04	84.64 ± 0.16
$N_{\mathrm{DE}} = 7$	98.06 ± 0.03	97.80 ± 0.05	84.78 ± 0.11
$N_{\mathrm{DE}} = 8$	$\underline{98.08 \pm 0.02}$	97.80 ± 0.03	84.91 ± 0.10
Score	96.53 ± 0.05	96.87 ± 0.03	78.86 ± 0.05
MD	$\mathbf{98.23 \pm 0.02}$	$\mathbf{98.06 \pm 0.02}$	$\mathbf{85.88 \pm 0.10}$
$\mathrm{GS}_{\mathrm{full}}$	98.04 ± 0.03	$\underline{97.81 \pm 0.06}$	$\underline{85.40 \pm 0.11}$

based on the probability logits $(\widehat{\phi}_1, \ldots, \widehat{\phi}_C)$. We found that $T = 100$ delivers the strongest results, see table 3.4 where we compared different values of T (like in [107]) for YOLOv3 on the KITTI dataset in terms of meta classification and meta regression performance.

We investigate an enveloping model ("**Full softmax**") of all classification-based uncertainty features by involving all probabilities $(\widehat{\pi}_1, \ldots, \widehat{\pi}_C)$ directly as co-variables in the meta classifier or meta regression model. It outperforms all purely classification-based models, which is expected due to the amount of information available.

As another common baseline, we investigate **MC dropout uncertainty** (MC), cf. section 2.2.4.2. Since we are explicitly interested in the uncertainty content of MC dropout, we only include anchor-wise standard deviations of the entire network output $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ obtained from 30 dropout samples. We found that computing more samples does not significantly improve predictive uncertainty content as seen in the ablation study on the MC sample count N_{DO} in table 3.5 for YOLOv3 on the KITTI dataset. Meta classification performance can be further improved by involving dropout means of $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$. However, MC dropout means do not carry an intrinsic meaning of uncertainty as opposed to standard deviations, so we do not include them in our main experiments but only in our extended results.

Table 3.7: Ablation on the number of network layers used in terms of meta classification (AuROC and AP) and meta regression (R^2). Gradient features per layer are accumulated to those of later layers starting from the last layer of the DNN.

		# Layers				
Metric	Score	1	2	3	4	5
AuROC	96.53 ± 0.05	98.04 ± 0.03	98.06 ± 0.02	98.18 ± 0.03	98.18 ± 0.03	98.19 ± 0.02
AP	96.87 ± 0.03	97.81 ± 0.06	97.83 ± 0.04	97.98 ± 0.05	98.00 ± 0.04	98.04 ± 0.04
R^2	78.89 ± 0.05	84.35 ± 0.05	85.40 ± 0.11	86.04 ± 0.11	86.18 ± 0.07	86.24 ± 0.09

As another common, sampling-based baseline, we investigate **deep ensemble** (E, cf. section 2.2.4.2) uncertainty obtained from ensembles of size 5. We find that larger ensembles do not significantly improve meta classification performance. For reference, we show an ablation on the ensemble size N_{DE} for YOLOv3 on the KITTI dataset in table 3.6 in terms of meta classification and meta regression. By the same motivation as for MC dropout, we only include anchor-wise standard deviations over forward passes from the ensemble and add means in the extended results.

The output-based **MetaDetect framework** (MD) computes uncertainty features for use in meta classification and meta regression from pre-NMS variance in anchor-based object detection. In our implementation, we compute the $46 + C$ MetaDetect features [161] which include the entire network output $\widehat{f^j}(\boldsymbol{x}|\boldsymbol{\theta})$. Additionally, the MetaDetect framework computes also the number of proposal boxes suppressed by $\widehat{\xi^j}$, i.e., $|\mathrm{cand}[\widehat{\xi^j}]|$ and uses the bounding box statistics given by the candidates to compute quantities like minimum, maximum, mean and standard deviations of bounding box features and IoU values between proposals. The MetaDetect framework is, therefore, an enveloping model to any uncertainty features based on the object detection output (in particular to any classification-based uncertainty) which we also find in our experiments. We include it in order to cover all such baselines.

In our experiments, we investigate two **gradient-based uncertainty** (GS) models. While $\mathrm{GS}_{\|\cdot\|_2}$ is based on the two-norms of box-wise gradients, $\mathrm{GS}_{\mathrm{full}}$ is utilizes all the six maps in eq. (3.4). While the two norms $\|\cdot\|_1$ and $\|\cdot\|_2$ directly compute the magnitude of a vector, the maps $\mathrm{mean}(\cdot)$ and $\mathrm{std}(\cdot)$ do not immediately capture a concept of length. However, they have been found in [132] to yield decent separation capabilities. Similarly, the component-wise $\min(\cdot)$ and $\max(\cdot)$ contain relevant predictive information. Note, that the latter two are related to the sup-norm $\|\cdot\|_\infty$ but together contain more information. While the last layer gradients themselves are highly informative, we allow for gradients of the last two layers in our main experiments. In table 3.7 we show meta classification and meta regression performance of gradient-based models with features obtained from different numbers of network layers of the YOLOv3 model on the KITTI dataset. Starting with the last layer gradient only (# layers is 1), the gradient features from the two last layers and so on. We see that meta classification performance quickly saturates, and no significant benefit can be seen from using more than 3 layers. However, meta regression can still be improved slightly by using up to 5 network layers.

Table 3.8: Meta classification performance in terms of AuROC and AP per meta classifier input over 10-fold cv. Model: YOLOv3 with Darknet53 backbone.

YOLOv3	Pascal VOC		COCO		KITTI	
	AuROC	AP	AuROC	AP	AuROC	AP
Score	90.68 ± 0.06	69.56 ± 0.12	82.97 ± 0.04	62.31 ± 0.05	96.53 ± 0.05	96.87 ± 0.03
Entropy	91.30 ± 0.02	61.94 ± 0.06	76.52 ± 0.02	42.52 ± 0.04	94.79 ± 0.06	94.83 ± 0.05
Energy Score [107]	92.59 ± 0.02	64.65 ± 0.06	75.39 ± 0.02	39.72 ± 0.06	95.66 ± 0.02	95.33 ± 0.03
Full Softmax	93.81 ± 0.06	72.08 ± 0.15	82.91 ± 0.06	58.65 ± 0.10	97.07 ± 0.03	96.85 ± 0.03
MC Dropout [169] (MC, $N_{\mathrm{MC}} = 30$)	$\underline{96.72 \pm 0.02}$	78.15 ± 0.09	$\mathbf{89.04 \pm 0.02}$	64.94 ± 0.11	97.60 ± 0.07	97.17 ± 0.10
Ensemble [88] (E, $N_{\mathrm{ens}} = 5$)	$\mathbf{96.87 \pm 0.02}$	77.86 ± 0.11	$\underline{88.97 \pm 0.02}$	64.05 ± 0.12	97.98 ± 0.03	97.69 ± 0.04
MetaDetect [161] (MD)	95.78 ± 0.05	$\mathbf{78.64 \pm 0.08}$	87.16 ± 0.04	$\underline{69.41 \pm 0.07}$	$\mathbf{98.23 \pm 0.02}$	$\mathbf{98.06 \pm 0.02}$
Grad. Score$_{\Vert\cdot\Vert_2}$ (GS$_{\Vert\cdot\Vert_2}$; ours)	94.76 ± 0.03	74.86 ± 0.10	86.05 ± 0.04	64.25 ± 0.06	97.31 ± 0.05	96.86 ± 0.10
Grad. Score$_{\mathrm{full}}$ (GS$_{\mathrm{full}}$; ours)	95.80 ± 0.04	$\underline{78.57 \pm 0.11}$	88.07 ± 0.03	$\mathbf{69.62 \pm 0.07}$	$\underline{98.04 \pm 0.03}$	$\underline{97.81 \pm 0.06}$
MC+E+MD	97.66 ± 0.02	85.13 ± 0.12	91.14 ± 0.02	73.82 ± 0.05	98.56 ± 0.03	98.45 ± 0.03
GS$_{\mathrm{full}}$+MC+E+MD	$\mathbf{97.95 \pm 0.02}$	$\mathbf{86.69 \pm 0.09}$	$\mathbf{91.65 \pm 0.03}$	$\mathbf{74.88 \pm 0.07}$	$\mathbf{98.74 \pm 0.02}$	$\mathbf{98.62 \pm 0.01}$

In some of our experiments, we compute gradients via the PyTorch autograd framework, iteratively for each bounding box. While this alleviates significant implementation effort, this procedure is computationally far less efficient than directly computing the gradients from the formulas in section 3.4.1 as is done in our runtime measurements.

In order to save on computational effort, we compute gradient features not for all predicted bounding boxes. We use a small score threshold of 10^{-4} (KITTI, Pascal VOC), resp. 10^{-2} (MS COCO) as a pre-filter. On average, this produces ~ 150 predictions per image. These settings lead to a highly dis-balanced TP/FP ratio post NMS on which meta classification and meta regression models are fitted. On YOLOv3, for example, these ratios are for Pascal VOC: 0.099, MS COCO: 0.158 and for KITTI: 0.464, so our models fit on significantly more FPs than TPs. However, our meta classification and meta regression models (see section 3.3.2) are gradient boosting models which tend to reflect well-calibrated confidences / regressions on the domain of training data. Our results (e.g., table 3.8) obtained from cross-validation confirm that this ratio does not constitute an obstacle for obtaining well-performing models on data not used to fit the model. For gradient boosting models, we employ the XGBoost library [22] with 30 estimators (otherwise standard hyperparameters).

3.4.2 Comparison with Output-based Uncertainty

We compare gradient-based uncertainty with various uncertainty baselines in terms of meta classification (table 3.8) and meta regression (table 3.9) for a YOLOv3 model with standard Darknet53 backbone [39]. As class probability baselines, we consider objectness score, softmax entropy, energy score [107] and the full softmax distribution per box. Since the full softmax baseline fits a model directly to all class probabilities (as opposed to relying on hand-crafted functions), it can be considered an *enveloping model* to both, entropy and energy score. Moreover, we consider other output baselines in MC dropout (MC), deep ensembles (E) and MetaDetect (MD). Since MetaDetect involves the entire network output of a bounding box, it leads to meta classifiers fitted on more variables than class probability baselines. It is, thus, an enveloping model of the full softmax baseline and, therefore, all classification baselines. The results in table 3.8 indicate that GS$_{\mathrm{full}}$ is

Table 3.9: Meta regression performance in terms of R^2 per meta classifier input over 10-fold cv. Model: YOLOv3 with Darknet53 backbone.

YOLOv3	Pascal VOC	COCO	KITTI
Score	48.29 ± 0.04	32.60 ± 0.02	78.86 ± 0.05
Entropy	43.24 ± 0.03	21.10 ± 0.04	69.33 ± 0.04
Energy Score	47.18 ± 0.03	17.94 ± 0.02	71.53 ± 0.10
Full Softmax	53.86 ± 0.11	36.95 ± 0.13	78.92 ± 0.11
MC	$\underline{61.63 \pm 0.15}$	43.85 ± 0.09	82.10 ± 0.11
E	61.48 ± 0.07	43.53 ± 0.13	84.18 ± 0.12
MD	60.36 ± 0.14	$\underline{44.22 \pm 0.11}$	$\mathbf{85.88 \pm 0.10}$
$\mathrm{GS}_{\|\cdot\|_2}$ (ours)	58.05 ± 0.13	38.77 ± 0.04	81.21 ± 0.05
$\mathrm{GS}_{\mathrm{full}}$ (ours)	$\mathbf{62.50 \pm 0.11}$	$\mathbf{44.90 \pm 0.09}$	$\underline{85.40 \pm 0.11}$
MC+E+MD	69.38 ± 0.11	54.07 ± 0.08	87.78 ± 0.11
$\mathrm{GS}_{\mathrm{full}}$+MC+E+MD	$\mathbf{72.26 \pm 0.08}$	$\mathbf{56.14 \pm 0.11}$	$\mathbf{88.80 \pm 0.07}$

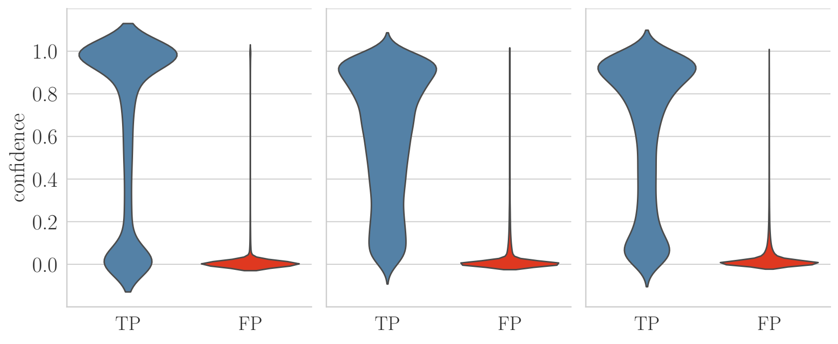


Figure 3.5: Confidence violin plots divided into TP and FP for Score (*left*), $\mathrm{GS}_{\mathrm{full}}$ (*center*) and $\mathrm{GS}_{\mathrm{full}}$+MC+E+MD (*right*). Model: YOLOv3, dataset: Pascal VOC evaluation split.

roughly in the same AuROC range as sampling-based uncertainty methods, while being consistently among the two best methods in terms of AP. The smaller gradient-based model $\mathrm{GS}_{\|\cdot\|_2}$ is consistently better than the full softmax baseline, by up to 3.14 AuROC percentage points (ppts) and up to 5.60 AP ppts. We also find that $\mathrm{GS}_{\mathrm{full}}$ tends to rank lower in terms of AuROC. Note also, that MetaDetect is roughly on par with the sampling approaches MC and E throughout. While the latter methods aim at capturing epistemic uncertainty they constitute approximations and are, not necessarily mutually redundant.

In addition, we compare the largest output-based model MC+E+MD and add the gradient features $\mathrm{GS}_{\mathrm{full}}$ to find out about the degree of redundancy between the approximated epistemic uncertainty in MC+E+MD and our method. We note significant boosts to the already well-performing model MC+E+MD across all metrics. Table 3.9 suggests that gradient uncertainty is especially informative for meta regression with $\mathrm{GS}_{\mathrm{full}}$ being consistently among the best two models and achieving R^2 scores of up to 85.4 on the KITTI dataset. Adding $\mathrm{GS}_{\mathrm{full}}$ to MC+E+MD always leads to a gain of more than one R^2 ppt indicating non-redundancy of gradient- and sampling-based features.

Figure 3.5 shows the confidence violin plots of the score (left), $\mathrm{GS}_{\mathrm{full}}$ (center) and the

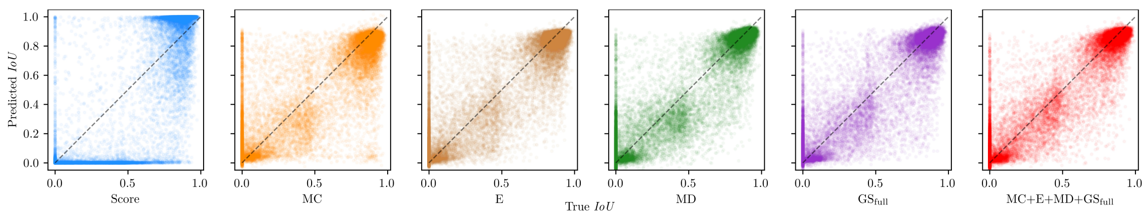


Figure 3.6: Scatter plots for samples of Score and meta regression based on MC dropout, a deep ensemble, Meta Detect, gradient features $\mathrm{GS}_{\mathrm{full}}$ and the combination model MC+E+MD+$\mathrm{GS}_{\mathrm{full}}$. We draw the optimal diagonal for reference. Model: YOLOv3, dataset: KITTI evaluation split.

composite model $\mathrm{GS}_{\mathrm{full}}$+MC+E+MD (right) conditioned on TP and FP predictions. The violin widths are normalized for increased width contrast. The score TP-violin shows especially large density at low confidences as opposed to the TP-violin plots of $\mathrm{GS}_{\mathrm{full}}$ and $\mathrm{GS}_{\mathrm{full}}$+MC+E+MD which are less concentrated around the confidence $\hat{\tau} = 0$. Instead, they have mass shifted towards the medium confidence range, i.e., the “neck” of the violin.

Extended Results: Meta Regression. We underline the meta regression results by showing samples of predicted IoU values over their true IoU in fig. 3.6. The samples are the results of one cross-validation split from table 3.9, and we indicate the diagonal of optimal regression with a dashed line in each panel. Note that the x-axis shows the true IoU values, and we indicate the uncertainty quantification method below each panel plot at a label. The y-axis shows the predicted IoU for each method. We find a large cluster for the score with low score but medium to high true IoU (from 0.1 to 0.8), the right-most part of which (predicted IoU ≥ 0.5) are false negative predictions. In this regard, we refer again to fig. 3.1 where FNs such as these become very apparent. Moreover, the score indicates very little correlation with the true IoU for true IoU ≥ 0.6 where there are numerous samples with a score between 0.4 and 0.6.

In contrast, the meta regression models show striking amounts of FPs (true IoU equal to 0 and, e.g., prediction $\iota \geq 0.3$). This phenomenon seems especially apparent for Monte Carlo dropout uncertainty. The meta regression models MD, $\mathrm{GS}_{\mathrm{full}}$ and $\mathrm{GS}_{\mathrm{full}}$+MC+E+MD show fits that are comparatively close to the optimal diagonal which is in line with the determined regression performance R^2 between 0.81 and 0.89 in table 3.9.

3.4.3 Generalization over Object Detection Architectures

We investigate the applicability and viability of gradient uncertainty for a variety of different architectures. In addition to the YOLOv3 model, we investigate two more standard object detectors in Faster R-CNN [144] and RetinaNet [101] both with a ResNet50 backbone [61]. Refer also to section 2.3.2 for additional detail on these architectures. Moreover, we investigate a stronger object detector in Cascade R-CNN [12] with a large ResNeSt200 [215] backbone which at the time of writing was ranked among the top 10 on the official MS COCO Detection Leaderboard. With an MS COCO detection AP of 49.03, this is in the state-of-the-art range for pure, non-hybrid-task object detectors. In table 3.10, we list meta classification AuROC and meta regression R^2 for the score, MetaDe-

Table 3.10: Meta classification and meta regression performance in terms of AuROC and R^2, respectively, for different object detection architectures. Results obtained from 10-fold cv as above.

	Pascal VOC		COCO		KITTI	
	AuROC	R^2	AuROC	R^2	AuROC	R^2
Faster R-CNN						
Score	89.77 ± 0.05	39.94 ± 0.02	83.82 ± 0.03	40.50 ± 0.01	96.53 ± 0.05	72.29 ± 0.02
MD	94.43 ± 0.02	47.92 ± 0.09	91.31 ± 0.02	44.41 ± 0.04	98.86 ± 0.02	79.92 ± 0.04
GS_{full}	$\mathbf{95.88 \pm 0.05}$	$\mathbf{59.40 \pm 0.03}$	$\mathbf{91.38 \pm 0.03}$	$\mathbf{50.44 \pm 0.04}$	$\mathbf{99.20 \pm 0.01}$	$\mathbf{86.31 \pm 0.07}$
GS_{full} + MD	96.77 ± 0.05	63.64 ± 0.08	92.30 ± 0.02	52.30 ± 0.04	99.37 ± 0.02	87.46 ± 0.05
RetinaNet						
Score	87.53 ± 0.03	40.43 ± 0.01	84.95 ± 0.02	39.88 ± 0.02	95.91 ± 0.02	73.44 ± 0.02
MD	89.57 ± 0.04	50.27 ± 0.10	85.09 ± 0.01	42.45 ± 0.12	96.19 ± 0.02	77.53 ± 0.08
GS_{full}	$\mathbf{91.58 \pm 0.04}$	$\mathbf{57.23 \pm 0.07}$	$\mathbf{85.59 \pm 0.02}$	$\mathbf{47.74 \pm 0.06}$	$\mathbf{97.26 \pm 0.03}$	$\mathbf{84.47 \pm 0.04}$
GS_{full} + MD	92.99 ± 0.03	64.32 ± 0.07	87.15 ± 0.05	51.07 ± 0.09	97.61 ± 0.02	85.73 ± 0.09
Cascade R-CNN						
Score	95.70 ± 0.04	57.90 ± 0.09	$\mathbf{94.11 \pm 0.01}$	56.31 ± 0.01	98.67 ± 0.02	83.31 ± 0.03
MD	96.32 ± 0.05	63.62 ± 0.12	94.10 ± 0.02	$\mathbf{58.74 \pm 0.08}$	99.18 ± 0.01	86.22 ± 0.08
GS_{full}	$\mathbf{96.66 \pm 0.05}$	$\mathbf{63.94 \pm 0.13}$	93.97 ± 0.01	57.80 ± 0.08	$\mathbf{99.34 \pm 0.01}$	$\mathbf{87.39 \pm 0.08}$
GS_{full} + MD	97.24 ± 0.05	69.78 ± 0.13	94.78 ± 0.02	62.13 ± 0.06	99.48 ± 0.01	89.59 ± 0.04

tect (representing output-based methods), GS_{full} and the combined model GS_{full}+MD. We see GS_{full} again being on par with MD, in the majority of cases even surpassing it by up to 2.01 AuROC ppts and up to 11.52 R^2 ppts. When added to MD, we find again boosts in both performance metrics, especially in R^2. On the MS COCO dataset, the high performance model Cascade R-CNN delivers a remarkably strong Score baseline completely redundant with MD and surpassing GS_{full} on its own. However, here we also find an improvement of 0.68 ppts by adding gradient information.

Extended Results: Non-Redundancy. Gradient features show significant improvements when combined with output- or sampling-based uncertainty quantification methods (see table 3.8 and table 3.9). We show additional meta classification and meta regression results in table 3.11 and in table 3.12 to further illustrate this finding. First, in table 3.11 we find that adding GS_{full} to the raw object detection output features $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ performs similarly as the combination GS_{full}+MD. In fact, when directly comparing MD with $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$, we see consistently better results on $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$, even though MD contains $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ as co-variables. We attribute this finding to overfitting of the gradient boosting classifier and regression on MD. This suggests that the information in MD is mostly redundant with the network output features. Also, for combinations of one output-based uncertainty source (i.e., one of MC, E and MD) we gain strong boosts, especially in meta regression (R^2). Note, that GS_{full}+E_{std} is almost always the second-best model, even out-performing the purely output-based model MC_{std}+E_{std}+MD. We show meta classification and meta regression performance of the sampling-based epistemic uncertainty methods MC and E when we include sampling averages of all features in addition to standard deviations which also leads to significant boosts. Finally, we show an additional subset of GS_{full} consisting of one- and two-norms ($\{\|\cdot\|_1, \|\cdot\|_2\}$) of all gradients which we abbreviate by $\text{GS}_{\|\cdot\|_{1,2}}$. We notice significant gain of the latter to $\text{GS}_{\|\cdot\|_2}$, which shows that the one-norms $\|\cdot\|_1$ con-

Table 3.11: Extended meta classification (AuROC and AP) and meta regression (R^2) performance results of baseline methods, variants of gradient metrics and different combinations of output-based uncertainty quantification methods with gradient metrics (mean $\pm$ std). We also show the results of using the entire network output $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ for meta classification and regression, as well, as adding sampling means to standard deviation features for MC and E.

	Pascal VOC			COCO			KITTI		
YOLOv3	AuROC	AP	R^2	AuROC	AP	R^2	AuROC	AP	R^2
Score	90.68 ± 0.06	69.56 ± 0.12	48.29 ± 0.04	82.97 ± 0.04	62.31 ± 0.05	32.60 ± 0.02	96.55 ± 0.04	96.87 ± 0.03	78.83 ± 0.05
Entropy	91.30 ± 0.02	61.94 ± 0.06	43.24 ± 0.03	76.52 ± 0.02	42.52 ± 0.04	21.10 ± 0.04	94.78 ± 0.03	94.82 ± 0.05	69.33 ± 0.08
Energy	92.59 ± 0.02	64.65 ± 0.06	47.18 ± 0.03	75.39 ± 0.02	39.72 ± 0.06	17.94 ± 0.02	95.46 ± 0.05	94.63 ± 0.08	70.39 ± 0.10
Full Softmax	93.81 ± 0.06	72.08 ± 0.15	53.86 ± 0.11	82.91 ± 0.06	58.65 ± 0.10	36.95 ± 0.13	97.10 ± 0.02	96.90 ± 0.04	78.79 ± 0.12
Full output $\widehat{f}(\boldsymbol{x}\|\boldsymbol{\theta})$	95.84 ± 0.04	78.84 ± 0.10	60.67 ± 0.18	86.31 ± 0.05	67.46 ± 0.07	44.32 ± 0.11	98.35 ± 0.02	98.21 ± 0.04	86.34 ± 0.07
$\mathrm{MC}_{\mathrm{std}}$	96.72 ± 0.02	78.15 ± 0.09	61.63 ± 0.15	89.04 ± 0.02	64.94 ± 0.11	43.85 ± 0.09	95.43 ± 0.04	94.11 ± 0.12	75.09 ± 0.13
$\mathrm{MC}_{\mathrm{std+mean}}$	97.42 ± 0.02	84.18 ± 0.09	68.33 ± 0.16	90.40 ± 0.03	72.63 ± 0.07	52.38 ± 0.07	98.43 ± 0.03	98.28 ± 0.04	86.86 ± 0.09
$\mathrm{E}_{\mathrm{std}}$	96.87 ± 0.02	77.86 ± 0.11	61.48 ± 0.07	88.97 ± 0.02	64.05 ± 0.12	43.53 ± 0.13	97.98 ± 0.03	97.69 ± 0.04	84.29 ± 0.12
$\mathrm{E}_{\mathrm{std+mean}}$	97.62 ± 0.02	84.87 ± 0.14	68.88 ± 0.09	90.75 ± 0.03	73.15 ± 0.06	53.09 ± 0.09	98.61 ± 0.02	$\underline{98.49 \pm 0.03}$	88.00 ± 0.08
$\mathrm{MC}_{\mathrm{std+mean}}+\mathrm{E}_{\mathrm{std+mean}}$	97.69 ± 0.02	85.30 ± 0.11	69.60 ± 0.13	91.15 ± 0.03	73.85 ± 0.05	54.12 ± 0.09	98.61 ± 0.01	$\underline{98.49 \pm 0.02}$	87.95 ± 0.10
MD	95.78 ± 0.05	78.64 ± 0.08	60.36 ± 0.14	86.23 ± 0.05	67.37 ± 0.08	44.22 ± 0.11	98.23 ± 0.03	98.07 ± 0.03	85.97 ± 0.09
$\mathrm{GS}_{\|\cdot\|_2}$	94.76 ± 0.03	74.86 ± 0.10	58.05 ± 0.13	84.90 ± 0.02	61.49 ± 0.08	38.77 ± 0.04	97.30 ± 0.05	96.82 ± 0.10	81.11 ± 0.14
$\mathrm{GS}_{\|\cdot\|_{1,2}}$	95.03 ± 0.03	76.04 ± 0.10	59.83 ± 0.10	86.21 ± 0.04	63.32 ± 0.13	41.36 ± 0.09	97.65 ± 0.04	97.21 ± 0.07	83.27 ± 0.09
$\mathrm{GS}_{\mathrm{full}}$	95.80 ± 0.04	78.57 ± 0.11	62.50 ± 0.11	86.94 ± 0.04	66.96 ± 0.06	44.90 ± 0.09	98.04 ± 0.02	97.81 ± 0.04	85.28 ± 0.07
$\mathrm{GS}_{\mathrm{full}}+\widehat{f}(\boldsymbol{x}\|\boldsymbol{\theta})$	96.51 ± 0.018	81.20 ± 0.09	65.24 ± 0.16	87.54 ± 0.04	69.05 ± 0.07	47.67 ± 0.09	98.57 ± 0.03	98.47 ± 0.04	87.83 ± 0.08
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MC}_{\mathrm{std}}$	97.65 ± 0.01	85.12 ± 0.06	70.30 ± 0.08	90.76 ± 0.02	72.50 ± 0.08	52.71 ± 0.07	98.35 ± 0.04	98.16 ± 0.04	86.48 ± 0.11
$\mathrm{GS}_{\mathrm{full}}+\mathrm{E}_{\mathrm{std}}$	$\underline{97.85 \pm 0.02}$	$\underline{85.90 \pm 0.15}$	$\underline{71.22 \pm 0.07}$	$\underline{91.27 \pm 0.03}$	73.44 ± 0.06	$\underline{54.17 \pm 0.06}$	$\underline{98.64 \pm 0.02}$	$\underline{98.49 \pm 0.03}$	$\underline{88.34 \pm 0.10}$
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MD}$	96.46 ± 0.04	81.00 ± 0.16	65.08 ± 0.14	87.51 ± 0.02	68.98 ± 0.08	47.63 ± 0.10	98.53 ± 0.03	98.42 ± 0.04	87.69 ± 0.06
$\mathrm{MC}_{\mathrm{std}}+\mathrm{E}_{\mathrm{std}}+\mathrm{MD}$	97.66 ± 0.02	85.13 ± 0.12	69.38 ± 0.11	91.14 ± 0.02	$\underline{73.82 \pm 0.05}$	54.07 ± 0.08	98.56 ± 0.03	98.45 ± 0.03	87.78 ± 0.11
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MC}_{\mathrm{std}}+\mathrm{E}_{\mathrm{std}}+\mathrm{MD}$	$\mathbf{97.95 \pm 0.02}$	$\mathbf{86.69 \pm 0.09}$	$\mathbf{72.26 \pm 0.08}$	$\mathbf{91.65 \pm 0.03}$	$\mathbf{74.88 \pm 0.07}$	$\mathbf{56.14 \pm 0.11}$	$\mathbf{98.74 \pm 0.02}$	$\mathbf{98.62 \pm 0.01}$	$\mathbf{88.80 \pm 0.07}$

Table 3.12: Extended meta classification (AuROC and AP) and meta regression (R^2) performance results of baseline methods, variants of gradient metrics and combinations of output- and gradient-based metrics for different object detection architectures.

	Pascal VOC			COCO			KITTI		
	AuROC	AP	R^2	AuROC	AP	R^2	AuROC	AP	R^2
Faster R-CNN									
Score	89.77 ± 0.05	67.71 ± 0.03	39.94 ± 0.02	83.82 ± 0.03	64.14 ± 0.03	40.50 ± 0.01	96.53 ± 0.05	93.29 ± 0.02	72.29 ± 0.02
MC	89.99 ± 0.06	44.22 ± 0.26	23.70 ± 0.17	85.80 ± 0.03	40.48 ± 0.12	23.56 ± 0.09	93.39 ± 0.07	67.82 ± 0.24	40.09 ± 0.17
MD	94.43 ± 0.02	71.18 ± 0.06	47.92 ± 0.09	91.31 ± 0.02	64.73 ± 0.05	44.41 ± 0.04	98.86 ± 0.03	94.31 ± 0.05	79.92 ± 0.04
$\mathrm{GS}_{\|\cdot\|_2}$	91.04 ± 0.07	61.66 ± 0.15	44.88 ± 0.05	89.80 ± 0.03	61.16 ± 0.06	44.93 ± 0.04	98.75 ± 0.02	93.01 ± 0.05	81.54 ± 0.05
$\mathrm{GS}_{\|\cdot\|_{1,2}}$	94.91 ± 0.04	67.73 ± 0.10	56.70 ± 0.06	90.64 ± 0.03	62.53 ± 0.07	48.27 ± 0.03	98.97 ± 0.03	93.89 ± 0.07	84.04 ± 0.04
$\mathrm{GS}_{\mathrm{full}}$	95.88 ± 0.05	68.74 ± 0.13	59.40 ± 0.03	91.38 ± 0.03	63.31 ± 0.07	50.44 ± 0.04	99.20 ± 0.01	94.60 ± 0.07	86.31 ± 0.07
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MC}$	96.59 ± 0.03	71.31 ± 0.08	60.74 ± 0.07	92.09 ± 0.02	64.59 ± 0.06	51.09 ± 0.04	99.34 ± 0.02	95.24 ± 0.05	86.85 ± 0.04
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MD}$	$\underline{96.77 \pm 0.05}$	$\underline{73.60 \pm 0.07}$	$\underline{63.64 \pm 0.08}$	$\underline{92.30 \pm 0.02}$	$\underline{65.67 \pm 0.05}$	$\underline{52.30 \pm 0.04}$	$\underline{99.37 \pm 0.02}$	$\underline{95.38 \pm 0.05}$	$\underline{87.46 \pm 0.05}$
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MC}+\mathrm{MD}$	$\mathbf{96.72 \pm 0.04}$	$\mathbf{73.51 \pm 0.10}$	$\mathbf{63.02 \pm 0.03}$	$\mathbf{92.30 \pm 0.01}$	$\mathbf{65.77 \pm 0.06}$	$\mathbf{52.21 \pm 0.04}$	$\mathbf{99.35 \pm 0.02}$	$\mathbf{95.37 \pm 0.03}$	$\mathbf{86.99 \pm 0.07}$
RetinaNet									
Score	87.53 ± 0.03	66.30 ± 0.05	40.43 ± 0.01	84.95 ± 0.04	68.58 ± 0.01	39.88 ± 0.02	95.91 ± 0.02	89.93 ± 0.02	73.44 ± 0.02
MC	72.90 ± 0.08	27.39 ± 0.11	14.17 ± 0.12	76.96 ± 0.04	43.54 ± 0.06	19.46 ± 0.06	88.13 ± 0.06	71.19 ± 0.10	50.51 ± 0.12
MD	89.57 ± 0.04	68.43 ± 0.08	50.27 ± 0.10	85.09 ± 0.01	68.32 ± 0.06	42.45 ± 0.12	96.19 ± 0.03	90.13 ± 0.04	77.53 ± 0.08
$\mathrm{GS}_{\|\cdot\|_2}$	87.86 ± 0.04	64.35 ± 0.06	46.19 ± 0.05	81.62 ± 0.04	63.95 ± 0.03	38.01 ± 0.04	95.93 ± 0.03	90.03 ± 0.05	79.17 ± 0.04
$\mathrm{GS}_{\|\cdot\|_{1,2}}$	88.77 ± 0.06	65.40 ± 0.05	49.64 ± 0.06	83.53 ± 0.05	65.88 ± 0.07	41.96 ± 0.05	96.47 ± 0.04	90.50 ± 0.03	81.35 ± 0.05
$\mathrm{GS}_{\mathrm{full}}$	91.58 ± 0.04	68.32 ± 0.06	57.23 ± 0.07	85.59 ± 0.02	67.93 ± 0.04	47.74 ± 0.06	97.26 ± 0.03	91.51 ± 0.07	84.47 ± 0.04
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MC}$	92.54 ± 0.03	70.65 ± 0.06	61.73 ± 0.04	86.87 ± 0.03	69.42 ± 0.03	50.63 ± 0.07	97.52 ± 0.02	91.98 ± 0.06	85.08 ± 0.04
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MD}$	$\mathbf{92.99 \pm 0.03}$	$\underline{72.30 \pm 0.08}$	$\mathbf{64.32 \pm 0.07}$	$\underline{87.15 \pm 0.05}$	$\underline{70.16 \pm 0.07}$	$\underline{51.07 \pm 0.09}$	$\underline{97.61 \pm 0.02}$	$\underline{92.26 \pm 0.05}$	$\mathbf{85.73 \pm 0.09}$
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MC}+\mathrm{MD}$	$\underline{92.95 \pm 0.03}$	$\mathbf{72.33 \pm 0.07}$	$\underline{63.44 \pm 0.06}$	$\mathbf{87.20 \pm 0.04}$	$\mathbf{70.21 \pm 0.03}$	$\mathbf{51.38 \pm 0.09}$	$\mathbf{97.63 \pm 0.01}$	$\mathbf{92.30 \pm 0.03}$	$\underline{85.64 \pm 0.08}$
Cascade R-CNN									
Score	95.70 ± 0.04	79.62 ± 0.10	57.90 ± 0.09	94.11 ± 0.01	81.36 ± 0.02	56.32 ± 0.02	98.67 ± 0.02	95.81 ± 0.04	83.31 ± 0.03
MD	96.32 ± 0.05	$\underline{82.11 \pm 0.12}$	63.62 ± 0.12	$\underline{94.12 \pm 0.03}$	$\underline{81.60 \pm 0.05}$	$\underline{58.84 \pm 0.04}$	99.18 ± 0.01	$\underline{96.60 \pm 0.05}$	86.22 ± 0.08
$\mathrm{GS}_{\|\cdot\|_2}$	96.46 ± 0.05	76.94 ± 0.19	61.56 ± 0.12	93.30 ± 0.02	76.40 ± 0.06	54.13 ± 0.06	99.19 ± 0.01	95.83 ± 0.06	85.80 ± 0.06
$\mathrm{GS}_{\|\cdot\|_{1,2}}$	96.54 ± 0.06	78.19 ± 0.22	62.82 ± 0.15	93.63 ± 0.02	77.95 ± 0.06	56.24 ± 0.05	99.23 ± 0.01	96.07 ± 0.05	86.33 ± 0.06
$\mathrm{GS}_{\mathrm{full}}$	$\underline{96.66 \pm 0.05}$	78.97 ± 0.19	$\underline{63.94 \pm 0.13}$	93.97 ± 0.02	79.17 ± 0.09	57.86 ± 0.05	$\underline{99.34 \pm 0.01}$	96.48 ± 0.04	$\underline{87.39 \pm 0.08}$
$\mathrm{GS}_{\mathrm{full}}+\mathrm{MD}$	$\mathbf{97.24 \pm 0.05}$	$\mathbf{84.11 \pm 0.13}$	$\mathbf{69.78 \pm 0.13}$	$\mathbf{94.78 \pm 0.02}$	$\mathbf{82.53 \pm 0.05}$	$\mathbf{62.13 \pm 0.05}$	$\mathbf{99.48 \pm 0.01}$	$\mathbf{97.27 \pm 0.04}$	$\mathbf{89.59 \pm 0.04}$

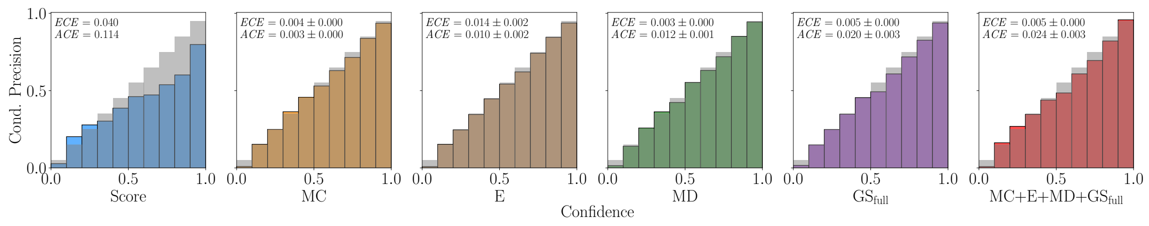


Figure 3.7: Reliability diagrams for the Score and meta classifiers based on different epistemic uncertainty features of the YOLOv3 architecture on the KITTI dataset. See table 3.13 for calibration errors of all meta classification models investigated in section 3.4.

tains important predictive information. Moreover, $\mathrm{GS}_{\mathrm{full}}$ is still significantly stronger than $\mathrm{GS}_{\|\cdot\|_{1,2}}$, showing that the other uncertainty features in eq. (3.4) lead to large performance boosts. Note that in almost all cases, combining MC dropout and deep ensemble features shows improvement over the single models even though both are epistemic (model) uncertainty. The two methods, therefore, do not contain the exact same information but still complement each other to some degree and are rather different approximations of epistemic uncertainty.

For further illustration of our method, table 3.12 shows additional meta classification and meta regression results for the architectures from table 3.10. We find similar tendencies for the purely norm-based gradient model $\mathrm{GS}_{\|\cdot\|_{1,2}}$ and see a significant degree of non-redundancy between gradient-based uncertainty and output-based uncertainty quantification methods. Note in particular, that MC stays roughly on par with the score baseline in terms of AuROC. We see significantly worse performance in terms of AP and meta regression (R^2). We attribute this to the anchor-based dropout sampling method which was also employed for the present architectures. In the case of Faster R-CNN, the aggregation approach is region proposal-based.

3.4.4 Calibration of Meta Classifiers

We evaluate the meta classifier confidences obtained above in terms of their calibration errors when divided into 10 confidence bins. Reliability plots are shown in fig. 3.7 for the Score, MC, E, MD, $\mathrm{GS}_{\mathrm{full}}$ and $\mathrm{MC{+}E{+}MDGS}_{\mathrm{full}}$ together with corresponding expected (ECE [126]) and average (ACE [128]) calibration errors. The Score is clearly over-confident in the upper confidence range and both meta classifiers are well-calibrated. Both calibration errors of the latter are about one order of magnitude smaller than for the Score.

For sake of completeness, we list in table 3.13 the calibration metrics ECE, MCE and ACE defined in section 3.3.2 for score and meta classifiers for all object detectors on all three datasets. The ECE metric is comparatively small for all meta classifiers and, therefore, insensitive and harder to interpret than MCE and ACE. As was argued in [128], the former is also less informative as bin-wise accuracy is weighted with the bin counts. In table 3.13 we can see a weakly increasing trend of calibration errors in the meta classifiers likely due to overfitting on the increasing number of co-variables. All meta classifiers

Table 3.13: Expected (*ECE*, [126]), maximum (*MCE*, [126]) and average (*ACE*, [128]) calibration errors per confidence model over 10-fold cv.

	Pascal VOC			COCO			KITTI		
YOLOv3	*ECE*	*MCE*	*ACE*	*ECE*	*MCE*	*ACE*	*ECE*	*MCE*	*ACE*
Score	0.040	0.252	0.114	0.0327	0.050	0.034	0.068	0.348	0.227
Entropy	$\underline{0.002 \pm 0.001}$	$\underline{0.021 \pm 0.010}$	$\underline{0.007 \pm 0.003}$	**0.002 ± 0.001**	0.028 ± 0.020	$\underline{0.007 \pm 0.003}$	**0.005 ± 0.001**	**0.033 ± 0.010**	**0.011 ± 0.003**
Energy Score	**0.001 ± 0.001**	**0.015 ± 0.007**	**0.005 ± 0.002**	**0.002 ± 0.001**	0.021 ± 0.003	0.008 ± 0.001	$\underline{0.006 \pm 0.002}$	$\underline{0.034 \pm 0.010}$	$\underline{0.013 \pm 0.005}$
Full Softmax	0.003 ± 0.000	0.028 ± 0.006	0.010 ± 0.002	$\underline{0.003 \pm 0.001}$	$\underline{0.018 \pm 0.003}$	$\underline{0.007 \pm 0.001}$	0.008 ± 0.001	0.048 ± 0.010	0.018 ± 0.002
MC	0.004 ± 0.000	0.033 ± 0.006	0.014 ± 0.002	0.004 ± 0.001	0.025 ± 0.003	0.010 ± 0.001	0.011 ± 0.001	0.036 ± 0.010	$\underline{0.013 \pm 0.002}$
E	0.003 ± 0.000	0.025 ± 0.005	0.010 ± 0.002	0.004 ± 0.001	0.022 ± 0.003	0.010 ± 0.001	0.013 ± 0.001	0.062 ± 0.010	0.028 ± 0.004
MD	0.003 ± 0.000	0.040 ± 0.009	0.012 ± 0.001	0.005 ± 0.001	0.033 ± 0.005	0.014 ± 0.001	0.012 ± 0.001	0.074 ± 0.020	0.028 ± 0.005
$\text{GS}_{\lVert\cdot\rVert_2}$	0.003 ± 0.000	0.036 ± 0.007	0.014 ± 0.001	**0.002 ± 0.000**	**0.013 ± 0.004**	**0.005 ± 0.001**	0.008 ± 0.001	0.054 ± 0.010	0.022 ± 0.003
GS_{full}	0.005 ± 0.000	0.055 ± 0.002	0.021 ± 0.003	0.005 ± 0.001	0.039 ± 0.003	0.015 ± 0.001	0.012 ± 0.001	0.078 ± 0.020	0.034 ± 0.006
MC+E+MD	0.005 ± 0.001	0.049 ± 0.010	0.020 ± 0.003	0.005 ± 0.000	0.031 ± 0.006	0.014 ± 0.001	0.014 ± 0.001	0.076 ± 0.010	0.034 ± 0.005
MC+E+MD+GS_{full}	0.005 ± 0.000	0.061 ± 0.010	0.024 ± 0.003	0.006 ± 0.000	0.042 ± 0.004	0.018 ± 0.001	0.015 ± 0.001	0.106 ± 0.020	0.043 ± 0.006
Faster R-CNN									
Score	0.050	0.427	0.232	0.075	0.212	0.138	0.036	0.283	0.114
MD	**0.003 ± 0.000**	0.039 ± 0.007	0.013 ± 0.002	**0.004 ± 0.000**	**0.020 ± 0.003**	**0.009 ± 0.001**	**0.009 ± 0.001**	**0.079 ± 0.020**	**0.029 ± 0.004**
GS_{full}	0.004 ± 0.000	**0.027 ± 0.007**	**0.011 ± 0.001**	**0.004 ± 0.001**	0.024 ± 0.003	**0.009 ± 0.001**	0.010 ± 0.001	0.084 ± 0.020	0.035 ± 0.004
MD+GS_{full}	0.005 ± 0.000	0.044 ± 0.007	0.018 ± 0.002	0.006 ± 0.001	0.029 ± 0.006	0.012 ± 0.001	0.011 ± 0.001	0.088 ± 0.010	0.037 ± 0.004
RetinaNet									
Score	0.068	0.212	0.123	0.089	0.192	0.106	0.027	0.097	0.043
MD	**0.003 ± 0.000**	**0.031 ± 0.008**	**0.011 ± 0.002**	**0.005 ± 0.001**	**0.022 ± 0.004**	**0.009 ± 0.001**	**0.003 ± 0.000**	**0.044 ± 0.006**	**0.016 ± 0.002**
GS_{full}	**0.003 ± 0.000**	0.044 ± 0.009	0.014 ± 0.001	**0.005 ± 0.000**	0.031 ± 0.006	0.012 ± 0.001	0.005 ± 0.001	0.060 ± 0.010	0.022 ± 0.004
MD+GS_{full}	0.005 ± 0.000	0.064 ± 0.008	0.024 ± 0.002	0.007 ± 0.001	0.032 ± 0.004	0.015 ± 0.001	0.006 ± 0.000	0.070 ± 0.010	0.028 ± 0.003
Cascade R-CNN									
Score	0.020	0.219	0.090	0.029	0.082	0.042	0.013	0.188	0.078
MD	**0.003 ± 0.000**	**0.021 ± 0.006**	**0.007 ± 0.002**	**0.003 ± 0.000**	0.019 ± 0.007	**0.006 ± 0.001**	**0.002 ± 0.000**	**0.038 ± 0.010**	**0.016 ± 0.005**
GS_{full}	0.005 ± 0.000	0.032 ± 0.010	0.012 ± 0.002	**0.003 ± 0.000**	**0.017 ± 0.003**	0.007 ± 0.001	0.003 ± 0.000	0.052 ± 0.010	0.020 ± 0.004
MD+GS_{full}	0.005 ± 0.000	0.034 ± 0.008	0.014 ± 0.002	0.004 ± 0.000	0.025 ± 0.004	0.010 ± 0.001	0.003 ± 0.000	0.046 ± 0.009	0.019 ± 0.003

are well-calibrated across the board. Table 3.13 also shows the expected (*ECE*, [126]) calibration error which was argued in [128] to be biased toward bins with large amounts of examples. *ECE* is, thus, less informative for safety-critical investigations.

3.4.5 Pedestrian Detection: Conditioning Meta Classification Results to a Specific Class

The statistical improvement seen in table 3.8 may not hold for non-majority classes within a dataset which are regularly safety-relevant. We investigate meta classification of the "Pedestrian" class in KITTI and explicitly study the FP/FN trade-off. This can be accomplished by sweeping the confidence threshold between 0 and 1 and counting the resulting FPs and FNs. We choose increments of 10^{-2} for meta classifiers and 10^{-4} for the scores as to not interpolate too roughly in the range of very small score values where a significant number of predictions cluster. The resulting curves are depicted in fig. 3.8. For applications in safety-critical environments, not all errors need to be equally important. We may, for example, demand a good trade-off at a given FN count which is usually desired to be especially small. Our present evaluation split contains a total of 1.152 pedestrian instances. Assume that we allowed for a detector to miss around 100 pedestrians ($\sim 10\%$), we see a reduction in FPs for some meta classifiers. MD and GS_{full} are very roughly on par, leading to a reduction of close to 100 FPs. The ensemble E turns out to be about as effective as the entire output-based model MC+E+MD, only falling behind above 150 FNs. This further indicates some degree of redundancy between output-based methods. Adding GS_{full} to MC+E+MD, however, reduces the number of FPs again by about 100 leading to an FP difference of about 250 as compared to the Score baseline. Observing the trend, the improvements become even more effective for smaller numbers of FNs (small

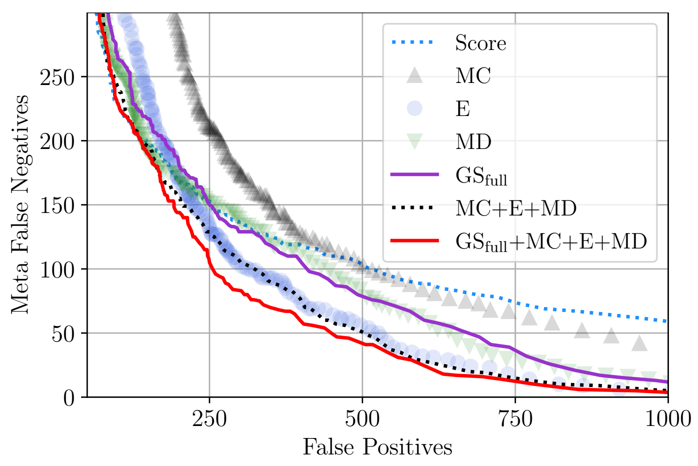


Figure 3.8: Meta classification for the class Pedestrian. Curves obtained by sweeping the threshold on score / meta classification probability. Note the FP gaps for ≤ 100 FNs.

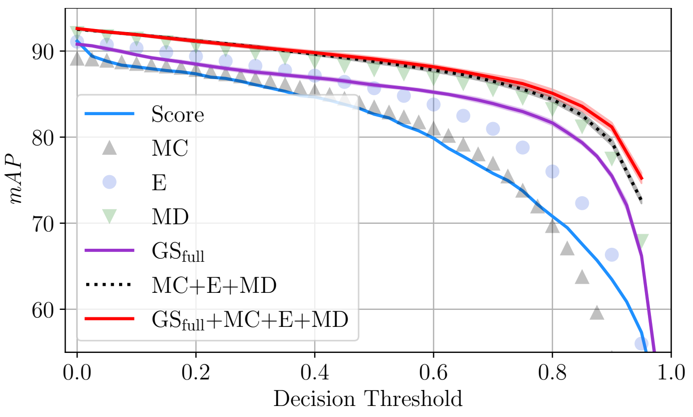


Figure 3.9: Score baseline and MetaFusion mAP. Error bands we draw around meta classifiers indicate cv-std.

thresholds) but diminish for larger numbers of above 200 FNs.

3.4.6 MetaFusion: Using Improved Confidences for Object Detection Performance

In regarding fig. 3.2, meta classifiers naturally fit as post-processing modules on top of object detection pipelines. Doing so does not generate new bounding boxes, but modifies the confidence ranking as shown in fig. 3.1 and may also lead to calibrated confidences. Therefore, the score baseline and meta classifiers are not comparable for fixed decision thresholds. We obtain a comparison of the resulting object detection performance by sweeping the decision threshold with a step size of 0.05 (resp. 0.025 for Score). The mAP curves are shown in fig. 3.9. We draw error bands showing cv-std for GS$_{\text{full}}$, MC+E+MD and GS$_{\text{full}}$+MC+E+MD. Meta classification-based decision rules are either on par (MC) with the score threshold or consistently allow for a mAP improvement of at least 1 to 2

Table 3.14: Computation timing of different methods at $\tau_s = 10^{-4}$.

Method	Parameters	AuROC	AP	R^2	FPS
Score	—	96.53	96.53	78.86	**43.48**
MC	$N = 30$, par.	97.60	97.17	82.10	31.45
E	$N = 5$, seq.	<u>97.98</u>	<u>97.69</u>	<u>84.18</u>	9.17
$\mathrm{GS}_{\mathrm{full}}$	1 layer	**98.04**	**97.81**	**84.35**	<u>34.77</u>

mAP ppts. In particular, MD performs well, gaining around 2 ppts in the maximum mAP. When comparing the addition of $\mathrm{GS}_{\mathrm{full}}$ to MC+E+MD, we still find slim improvements for thresholds ≥ 0.75. The score curve shows a kink at a threshold of 0.05 and ends at the same maximum mAP as $\mathrm{GS}_{\mathrm{full}}$ while the confidence ranking is clearly improved for MC+E+MD and $\mathrm{GS}_{\mathrm{full}}$+MC+E+MD. Note that meta classification based on $\mathrm{GS}_{\mathrm{full}}$ is less sensitive to the choice of threshold than the score in the medium range. At a threshold of 0.3 we have a mAP gap of about 1.4 ppts which widens to 5.2 ppts at 0.6.

3.4.7 Computational Runtime

We compare the runtime of our method with MC dropout and deep ensembles for YOLOv3 running on an Nvidia Quadro P6000 GPU at batch size 1. Table 3.14 shows the average performance on the KITTI dataset and throughput in frames per second (FPS). MC is batch-parallelized within dropout layers, while E runs sequentially. $\mathrm{GS}_{\mathrm{full}}$ is parallelized over the predicted boxes and backpropagation performed explicitly by convolution (see section 3.3.2). We see that at slightly better meta classification, last layer gradient scores achieve around 3 additional FPS over MC which is in line with theorem 1. This is possible due to the initial score threshold τ_s on the prediction. Computing deeper gradients amounts to performing one more transposed convolution per layer which does not obstruct parallelism.

3.5 Conclusion: Gradient-based Uncertainty Quantification for Deep Object Detection

Applications of modern DNNs in safety-critical environments demand high performance on the one hand, but also reliable confidence estimation indicating where a model is not competent. We have proposed and investigated a way of implementing gradient-based uncertainty quantification for deep object detection which complements output-based methods well and is on par with established epistemic uncertainty quantification methods. Experiments involving a number of different architectures suggest that our method can be applied to significant benefit across architectures, even for high performance state-of-the-art models. We showed that meta classification performance carries over to object detection performance when employed as post-processing and that meta classification naturally leads to well-calibrated gradient confidences improving probabilistic reliability. Equation (3.14) can in principle be augmented to fit any DNN inferring and learning on an instance-based logic like 3D bounding box detection or instance segmentation.

4 Gradient Uncertainty in Semantic Segmentation

In this chapter we introduce a method to compute pixel-wise uncertainty scores based on self-learning gradients for semantic segmentation networks ending on a convolutional layer. The presented contents are in large parts taken word-for-word from [112].

4.1 Introduction: Gradient-based Uncertainty and Out-of-Distribution Segmentation

Semantic segmentation decomposes the pixels of an input image into segments which are assigned to a fixed and predefined set of semantic classes (see section 2.3.3). In recent years, DNNs have performed excellently in this task [20, 185], providing comprehensive and precise information about the given scene. However, in safety-relevant applications like automated driving where semantic segmentation is used in open world scenarios, DNNs often fail to function properly on unseen objects for which the network has not been trained, see for example the bobby car in fig. 4.1 (left). These objects from outside the network's semantic space are called of *out-of-distribution* (OoD) objects. It is of the highest interest that the DNN identifies these objects and abstains from deciding on the semantic class for those pixels covered by the OoD object. Another case are OoD objects which might belong to a known class, however, appearing differently to substantial significance from other objects of the same class seen during training. Consequently, the respective predictions are prone to error. For these objects, as for classical OoD objects, marking them as OoD is preferable to the likely case of misclassification which may happen with high confidence. Furthermore, this additional classification task should not substantially degrade the semantic segmentation performance itself outside the OoD region. The computer vision tasks of identifying and segmenting those objects is captured by the notion of *OoD segmentation* [14, 111].

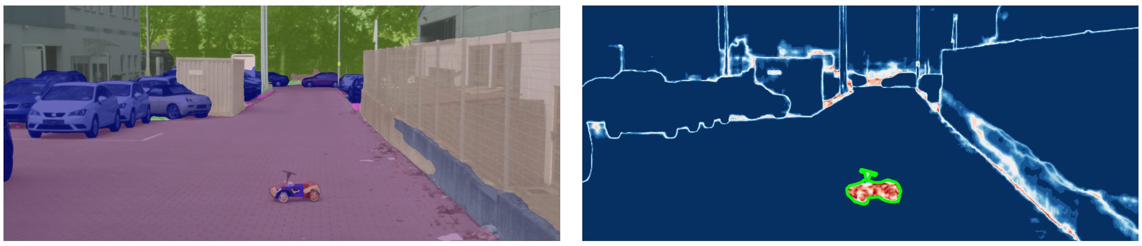


Figure 4.1: *Left*: Semantic segmentation by a state-of-the-art deep neural network. *Right*: Gradient uncertainty heatmap obtained by our method.

The recent contributions to the emerging field of OoD segmentation are mostly focused on OoD training, i.e., the incorporation of additional training data (not necessarily from the real world), sometimes obtained by large reconstruction models [34, 105]. Another line of research is the use of uncertainty quantification methods such as Bayesian models [124] or maximum softmax probability [62]. Gradient-based uncertainties are considered for OoD detection in the classification task by [132], [69] and [93] and up to now, have not been applied to OoD segmentation. [52] show that gradient norms perform well in discriminating between in- and out-of-distribution. Moreover, gradient-based features are studied by [145] for object detection to estimate the prediction quality. Loss gradients with respect to feature activations in monocular depth estimation are investigated in [65] and show correlations of gradient magnitude with depth estimation accuracy.

In this chapter, we introduce a new method for uncertainty quantification in semantic segmentation and OoD segmentation based on gradient information. Magnitude features of gradients can be computed at inference time and provide information about the uncertainty propagated in the corresponding forward pass. These features represent pixel-wise uncertainty scores applicable to prediction quality estimation and OoD segmentation. An exemplary gradient uncertainty heatmap can be found in fig. 4.1 (right). Such uncertainty scores have shown improved performance for the quality estimation task in image classification compared to uncertainties contained in the softmax output of DNNs [132]. In addition, instance-wise gradient uncertainty outperforms sampling methods like Monte-Carlo (MC) Dropout [45] and Ensembles [88] in object detection [145]. Calculating gradient uncertainty scores does not require any re-training of the DNN or computationally expensive sampling. Instead, only one backpropagation step for the gradients with respect to the final convolutional network layer is performed per inference to produce gradient scores which can be done in a highly efficient way as we show. Note, that more than one backpropagation step can be performed if deeper gradients need to be computed and other parameters of the model are considered. An overview of our approach is shown in fig. 4.2.

An application to dense predictions such as semantic segmentation has escaped previous research. Single backpropagation steps per pixel on high-resolution input images quickly become infeasible given that 10^6 gradients have to be calculated. To overcome this issue, we present a new approach to exactly compute the pixel-wise gradient scores in a batched and parallel manner applicable to a large class of segmentation architectures. We use these gradient scores to estimate the model uncertainty on pixel-level and also the prediction quality on segment-level. Segments are connected components of pixels belonging to the same class predicted by the semantic segmentation network. Moreover, the gradient uncertainty heatmaps are investigated for OoD segmentation where high uncertainties indicate possible OoD objects. Finally, we demonstrate the efficiency of our method in explicit runtime measurements and show that the computational overhead introduced is marginal compared with the standard forward pass.

We only assume a pre-trained semantic segmentation network as our method is applicable to a wide range of architectures. In our tests, we employ a state-of-the-art segmentation network [20] trained on Cityscapes [28] evaluating in-distribution uncertainty estimation and demonstrate OoD detection performance on four OoD segmentation datasets, namely LostAndFound [137], Fishyscapes [4], RoadAnomaly21 and RoadObstacle21 [14]. The source code of our method is publicly available[44]. Our contributions are summarized as follows:

- We introduce a new method for uncertainty quantification in semantic segmentation. Our gradient-based approach is applicable to a wide range of segmentation architectures.
- For the first time, we show an efficient way of computing gradients in semantic segmentation on the pixel-level. Our approach runs in a parallel manner making our method less computationally expensive than sampling-based methods which is demonstrated in explicit time measurements.
- For the first time, we demonstrate the robustness of gradient-based uncertainty quantification with respect to predictive error detection and OoD segmentation. In the OoD segmentation task, we achieve area under precision-recall curve values of up to 69.3% on the LostAndFound benchmark outperforming a variety of methods.

4.2 Related Work: Uncertainty Quantification in Semantic Segmentation and OoD Segmentation

Uncertainty Quantification. Bayesian approaches [115] are widely used to estimate model uncertainty. The well-known approximation, MC Dropout [45], has proven to be practically efficient in uncertainty estimation and is also applied to semantic segmentation by Lee et al. [94]. In addition, this method is considered to filter out predictions with low reliability by Wickstrøm et al. [188]. Pixel-wise uncertainty estimation methods based on Bayesian models or the network's softmax output have been benchmarked by Blum et al. [4]. Hoebel et al. [64] extracted uncertainty information on pixel-level by using the maximum softmax probability and MC Dropout. Prediction quality evaluation approaches are introduced by DeVries and Taylor [33] as well as Huang et al. [68] working on single objects per image. These methods are based on additional CNNs acting as post-processing mechanism. Rottmann et al. [152] present the concepts of meta classification (FP detection) and meta regression (localization quality estimation) on segment-level using features as input extracted from the segmentation network's softmax output. This line

[44]`https://github.com/tobiasriedlinger/uncertainty-gradients-seg`

of research has been extended by a temporal component [113] and transferred to object detection [145, 161] as well as to instance segmentation [110, 114].

While MC Dropout as a sampling approach is computationally expensive to create pixel-wise uncertainties, our method computes only the gradients of the last layer during a single inference run. Moreover, this method can be applied to a wide range of semantic segmentation networks without architectural changes. Compared with the work by Rottmann et al. [152], our gradient information can extend the features extracted from the segmentation network's softmax output to enhance the segment-wise quality estimation.

OoD Segmentation. Uncertainty quantification methods demonstrate high uncertainty for erroneous predictions, so they can intuitively serve for OoD detection as well. For instance, this can be accomplished via maximum softmax (probability) [62], Monte-Carlo Dropout [124] or ensembles [88] (see also section 2.2.4.2) which capture model uncertainty by averaging predictions over multiple sets of parameters. Another line of research is OoD detection training, relying on the exploitation of additional training data, not necessarily from the real world, but disjoint from the original training data [3, 5, 15, 54, 176]. In this regard, an external reconstruction model followed by a discrepancy network is considered by [34], [105], [104] and [183] and normalizing flows are leveraged by [5] and [53]. [99] and [95] perform small adversarial perturbations on the input images to improve the separation of in- and out-of-distribution samples.

Specialized training approaches for OoD detection are based on different kinds of re-training with additional data and often require generative models. Meanwhile, our method does not require OoD data, re-training or complex auxiliary models. Moreover, we do not run a full backward pass as it is required for the computation of adversarial samples. In fact, we found that it is often sufficient to only compute the gradients of the last convolutional layer. Our method is more related to classical uncertainty quantification approaches like maximum softmax, MC Dropout and ensembles. Note that the latter two are based on sampling and thus, much more computationally expensive compared to single inference.

4.3 Methods: Gradient-based Uncertainty in Semantic Segmentation and Out-of-Distribution Scores

In the following, we consider a neural network with parameters $\boldsymbol{\theta}$ yielding classification probabilities $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta}) = (\widehat{f}_1, \dots, \widehat{f}_C)$ over C semantic categories (see section 2.3.1) when presented with an input $\boldsymbol{x}$. During training on paired data $(\boldsymbol{x}, y)$, where $y \in [C]$ is the semantic label given to $\boldsymbol{x}$, such a model is commonly trained by minimizing some kind of loss function $\mathcal{L}$ between y and the predicted probability distribution $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ using gradient descent on $\boldsymbol{\theta}$. The gradient step

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta}) \middle\| y\right) \tag{4.1}$$

then indicates the *direction and strength of training feedback* obtained by $(\boldsymbol{x}, y)$. Asymptotically (in the amount of data and training steps taken), the expectation

$$\mathbb{E}_{(\boldsymbol{X},Y)\sim\mu_Z}\left[\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\widehat{f}(\boldsymbol{X}|\boldsymbol{\theta})\middle\|\, Y\right)\right] \tag{4.2}$$

of the gradient with respect to the data generating distribution μ_Z will vanish since $\boldsymbol{\theta}$ sits at a local minimum of $\mathcal{L}$. We assume, that strong models which achieve high test accuracy can be seen as an approximation of such a parameter configuration $\boldsymbol{\theta}$. Such a model has small gradients on in-distribution data which is close to samples $(\boldsymbol{x}, y)$ in the training data. Samples that differ from training data may receive larger feedback. Likewise, it is plausible to obtain large training gradients from OoD samples that are far away from the effective support of μ_Z.

In order to quantify uncertainty about the prediction $\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})$, we replace the label y from above by some fixed auxiliary label for which we make two concrete choices in our method. On the one hand, we replace y by the class prediction one-hot vector $y_k^{oh} = \delta_{k\widehat{c}}$ with $\widehat{c} = \widehat{c}(\boldsymbol{x}) = \arg\max_{k=1,\ldots,C} \widehat{f}_k(\boldsymbol{x}|\boldsymbol{\theta})$ and the Kronecker symbol δ_{ij}. The same applies to the dependency of $\widehat{c}$ on $\boldsymbol{\theta}$ as in chapter 3. This quantity correlates strongly with training labels y on in-distribution data. On the other hand, we regard a constant uniform, all-warm label $y_k^{uni} = 1/C$ for $k \in [C]$ as a replacement for y. To motivate the latter choice, we consider that classification models are oftentimes trained on the categorical cross entropy loss

$$\mathcal{L}_{\mathrm{CE}}\left(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\middle\|\, y\right) = -\sum_{k=1}^{C} y_k \log\left(\widehat{f}_k(\boldsymbol{x}|\boldsymbol{\theta})\right) \tag{4.3}$$

where we allow in principle for non-onehot labels $y \in [0,1]^C$. Since the gradient of this loss function is linear in the label y, a uniform choice y^{uni} will return the *average gradient per class*. This average gradient is expected to be large on OoD data where all possible labels are similarly unlikely. The magnitude of $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\widehat{f}|y)$ serves as uncertainty / OoD score. In the following, we explain how to compute such scores for pixel-wise classification models.

4.3.1 Efficient Computation in Semantic Segmentation

We regard a generic segmentation architecture utilizing a final convolution as the pixel-wise classification mechanism. In the following, we also assume that the segmentation model is trained via the commonly-used pixel-wise cross entropy loss

$$\mathcal{L}\left(\widehat{f}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\middle\|\, y^{i,j}\right) = -\sum_{k=1}^{C} y_k^{i,j} \log\left(\widehat{f}_k^{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\right). \tag{4.4}$$

(cf. eq. (4.3) and section 2.3.3) over each pixel $(i,j) \in \mathcal{I}$. Pixel-wise probabilities are obtained by applying the softmax function $\boldsymbol{\varSigma}$ to each output pixel, i.e., $\widehat{f}^{i,j} = \boldsymbol{\varSigma}\left(\widehat{\phi}_{ij}(\boldsymbol{\theta})\right)$. With eq. (4.3), we find for the loss gradient

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\boldsymbol{\varSigma}\left(\widehat{\phi}_{ij}(\boldsymbol{\theta})\right)\middle\|\, y^{i,j}\right) = \sum_{k=1}^{C} \varSigma_k\left(\widehat{\phi}_{i,j}\right) \cdot \left(1 - y_k^{i,j}\right) \cdot \nabla_{\boldsymbol{\theta}}\widehat{\phi}_{ij}^{k}(\boldsymbol{\theta}). \tag{4.5}$$

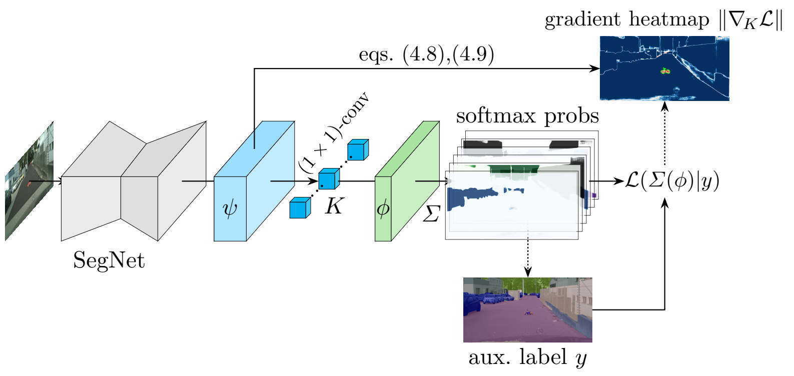


Figure 4.2: Schematic illustration of the computation of pixel-wise gradient norms for a semantic segmentation network with a final convolution layer. Auxiliary labels may be derived from the softmax prediction or supplied in any other way like as a uniform all-warm label. We circumvent back propagation via autograd for each pixel by utilizing eqs. (4.8) and (4.9).

Here, $\boldsymbol{\theta}$ is any set of parameters within the neural network. Here, $\widehat{\phi}$ is the convolution result of a pre-convolution feature map ψ (see fig. 4.2) against a filter bank K which assumes the role of $\boldsymbol{\theta}$. We postpone the derivation of eq. (4.5) and further formulas to the following paragraph. K has parameters $(K_e^h)_{fg}$ where e and h indicate in- and outgoing channels respectively and f and g index the spatial filter position. The features $\widehat{\phi}$ are linear in both, K and ψ, and depend on bias parameters β in the form known from section 2.2.1.2

$$\widehat{\phi}_{ij}^d = (K * \psi)_{ij}^d + \beta^d = \sum_{l=1}^{\kappa_{\text{in}}} \sum_{p,q=-s}^{s} (K_l^d)_{pq} \psi_{i+p,j+q}^l + \beta^d. \tag{4.6}$$

We denote by κ_{in} the total number of in-going channels and s is the spatial extent to either side of the filter K_l^d which has total size $(2s+1)\times(2s+1)$. Explicitly for the last layer gradients we find

$$\frac{\partial \widehat{\phi}_{ij}^d}{\partial (K_e^h)_{fg}} = \delta^{dh} \psi_{i+f,j+g}^e. \tag{4.7}$$

Together with eq. (4.5), we obtain an *explicit formula for computing the correct backpropagation gradients* of the loss on pixel-level for our choices of auxiliary labels which we state in the following paragraph. The computation of the loss gradient can be traced in fig. 4.2.

If we take the predicted class $\widehat{c}$ as a one-hot label per pixel as auxiliary labels, i.e., y^{oh}, we obtain for the last layer gradients

$$\frac{\partial \mathcal{L}\left(\boldsymbol{\Sigma}(\widehat{\phi}_{ij})\middle| y_{ij}^{oh}\right)}{\partial K_e^h} = \Sigma_h\left(\widehat{\phi}_{ij}\right)\cdot(1-\delta_{h\widehat{c}})\cdot \psi_{ij}^e \tag{4.8}$$

which depends on quantities that are easily accessible during a forward pass through the network. Note, that the filter bank K for the special case of (1×1)-convolutions does not require spatial indices which is a common situation in segmentation networks, albeit, not necessary for our method to be applied. Similarly, we find for the uniform label $y_j^{uni} = 1/C$

$$\frac{\partial \mathcal{L}\left(\boldsymbol{\Sigma}(\widehat{\phi}_{ij}) \middle| y_{ij}^{uni} \right)}{\partial K_e^h} = \frac{C-1}{C} \Sigma_h \left(\widehat{\phi}_{ij} \right) \psi_{ij}^e. \tag{4.9}$$

Therefore, pixel-wise gradient norms are simple to implement and particularly efficient to compute for the last layer of the segmentation model. We cover formulas for the more general case of deeper gradients from the second-to-last layer in a separate paragraph.

***Derivation of Equations* (4.5) *and* (4.7) *to* (4.9).** We assume that the final activation in the segmentation network is given by a softmax activation (see eq. (2.36)) of logit features $\widehat{\phi}$ in the last layer. Here, $\widehat{\phi} = \widehat{\phi}(\boldsymbol{\theta})$ is dependent on a set of parameters $\boldsymbol{\theta}$. In order to compute the gradients of the categorical cross entropy at pixel $(i,j) \in \mathcal{I}$ given any auxiliary label y

$$\mathcal{L}\left(\widehat{\phi}_{ij}(\boldsymbol{\theta}) \middle| y \right) = -\sum_{k=1}^{C} y_{ij}^k \log \left(\Sigma_k \left(\widehat{\phi}_{ij}(\boldsymbol{\theta}) \right) \right), \tag{4.10}$$

we require the derivative of the softmax function where we drop the pixel indices in our notation

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \Sigma_j \left(\widehat{\phi}(\boldsymbol{\theta}) \right) =& \frac{\mathrm{e}^{\widehat{\phi}^j} \nabla_{\boldsymbol{\theta}} \widehat{\phi}^j}{\sum_{i=1}^C \mathrm{e}^{\widehat{\phi}^i}} - \frac{\mathrm{e}^{\widehat{\phi}^j} \sum_{k=1}^C \mathrm{e}^{\widehat{\phi}^k} \nabla_{\boldsymbol{\theta}} \widehat{\phi}^k}{\left(\sum_{i=1}^C \mathrm{e}^{\widehat{\phi}^i} \right)^2} \\
=& \Sigma_j \left(\widehat{\phi} \right) \cdot \nabla_{\boldsymbol{\theta}} \widehat{\phi}^j \sum_{k=1}^C \Sigma_k \left(\widehat{\phi} \right) - \Sigma_j \left(\widehat{\phi} \right) \sum_{k=1}^C \Sigma_k \left(\widehat{\phi} \right) \cdot \nabla_{\boldsymbol{\theta}} \widehat{\phi}^k \\
=& \Sigma_j \left(\widehat{\phi} \right) \left(\sum_{k=1}^C \Sigma_k \left(\widehat{\phi} \right) \cdot \nabla_{\boldsymbol{\theta}} \widehat{\phi}^j - \Sigma_k \nabla_{\boldsymbol{\theta}} \widehat{\phi}^k \right) \\
=& \Sigma_j \left(\widehat{\phi} \right) \sum_{k=1}^C \Sigma_k \left(\widehat{\phi} \right) (\delta_{kj} - 1) \nabla_{\boldsymbol{\theta}} \widehat{\phi}^k.
\end{aligned} \tag{4.11}$$

Note, that the auxiliary label $y \in [0,1]^C$ may be anything, e.g., actual ground truth information about pixel (i,j), the predicted probability distribution at that location, the one-hot encoded prediction or a uniform class distribution. In the following, we regard y to be independent of $\widehat{\phi}$, i.e., in particular from $\boldsymbol{\theta}$. The gradient of the cross entropy loss as stated in eq. (4.5) is

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\widehat{\phi}_{ij}(\boldsymbol{\theta})\middle| y\right) &= -\sum_{k=1}^{C} y_{ij}^{k}\frac{1}{\varSigma_k\left(\widehat{\phi}_{ij}\right)}\nabla_{\boldsymbol{\theta}}\varSigma_k\left(\widehat{\phi}_{ij}\right)\\ &= \sum_{k=1}^{C}\sum_{l=1}^{C}\left(y_{ij}^{k}\varSigma_l\left(\widehat{\phi}_{ij}\right) - y_k\varSigma_l\left(\widehat{\phi}_{ij}\right)\delta_{lk}\right)\cdot\nabla_{\boldsymbol{\theta}}\widehat{\phi}_{ij}^{l}(\boldsymbol{\theta}) \\ &= \sum_{k=1}^{C}\varSigma_k\left(\widehat{\phi}_{ij}\right)(1-y_k)\cdot\nabla_{\boldsymbol{\theta}}\widehat{\phi}_{ij}^{k}(\boldsymbol{\theta}).\end{aligned} \tag{4.12}$$

The feature maps are the result of convolution against a filter bank

$$K \in \mathbb{R}^{\kappa_{\text{in}}\times\kappa_{\text{out}}\times(2s+1)\times(2s+1)} \tag{4.13}$$

and addition of a bias vector $\beta \in \mathbb{R}^{\kappa_{\text{out}}}$. Here, κ_{in} and κ_{out} denote the number of in- and out-going channels, respectively. Taking explicit derivatives of this expression with respect to one of the parameters in K yields eq. (4.7)

$$\frac{\partial(K*\psi)_{ij}^{d}}{\partial(K_e^h)_{fg}} = \sum_{c=1}^{\kappa_{\text{in}}}\sum_{q,r=-s}^{s}\frac{\partial(K_c^d)_{qr}}{\partial(K_e^h)_{fg}}\psi_{i+q,j+r}^{c} = \sum_{c=1}^{\kappa_{\text{in}}}\sum_{q,r=-s}^{s}\delta^{dh}\delta_{ce}\delta_{pf}\delta_{qg}\psi_{i+q,j+r}^{c} = \delta^{dh}\psi_{i+f,j+g}^{e}. \tag{4.14}$$

When utilizing the predicted one-hot vector, i.e., $y_k^{oh} = \delta_{k\hat{c}}$ and using the fact that the last-layer convolution is (1×1), we obtain eq. (4.8)

$$\frac{\partial}{\partial K_e^h}\mathcal{L}\left(\boldsymbol{\varSigma}\left(\widehat{\phi}_{ij}\right)\middle| y_{ij}^{oh}\right) = \sum_{k=1}^{C}\varSigma_k\left(\widehat{\phi}_{ij}\right)(1-\delta_{k\hat{c}})\frac{\partial\widehat{\phi}_{ij}^{k}}{\partial K_e^h} = \varSigma^h\left(\widehat{\phi}_{ij}\right)(1-\delta_{f\hat{c}})\psi_{ij}^{e} \tag{4.15}$$

while the uniform categorical label $y_k^{uni} = \frac{1}{C}$ yields eq. (4.9)

$$\frac{\partial}{\partial K_e^h}\mathcal{L}\left(\boldsymbol{\varSigma}\left(\widehat{\phi}_{ij}\right)\middle| y_{ij}^{uni}\right) = \sum_{k=1}^{C}\varSigma_k\left(\widehat{\phi}_{ij}\right)\left(1-\frac{1}{C}\right)\cdot\frac{\partial\widehat{\phi}_{ij}^{k}}{\partial K_e^h} = \frac{C-1}{C}\varSigma_h\left(\widehat{\phi}_{ij}\right)\psi_{ij}^{e}. \tag{4.16}$$

Computing Gradients for the Second-to-Last Layer. While for the last-layer gradients the computation from above simplifies significantly due to the fact that pixel-wise gradients only depend on the feature map values at the same pixel-location. Deeper layers are usually not (1×1), so the forward pass couples feature map values over a larger receptive field, e.g., (3×3). However, gradients for the second-to-last layer can still be computed with comparably small effort by extracting feature map patches via the unfold function. Here, we consider a DeepLabv3+ implementation[1] [217] for which the feature

[1]`https://github.com/NVIDIA/semantic-segmentation/tree/sdcnet`

maps depend in the following way on the weights of the second-to-last layer $L-1$ where L indicates the last layer:

$$\widehat{\phi}(K_{L-1}) = \mathcal{C}_{K_L,\beta_L} \circ \mathrm{ReLU} \circ \mathsf{BN}_L \circ \mathcal{C}_{K_{L-1},\beta_{L-1}}(\psi_{L-1}), \tag{4.17}$$

where BN_L is a batch normalization layer and ψ_{L-1} are the features prior to the convolution with K_{L-1} and β_{L-1}. With fully expanded indices, this amounts to

$$\widehat{\phi}^d_{ij} = \sum_{k_L} K^d_{k_L} \mathrm{ReLU}\left(\mathsf{BN}_L \left[\sum_{k_{L-1}} \sum_{q,r} \left((K_{L-1})^{k_L}_{k_{L-1}} \right)_{qr} (\psi_{L-1})^{k_{L-1}}_{i+q,j+r} + \beta^{k_L}_{L-1} \right] \right) + \beta^d_L \tag{4.18}$$

with k_L, k_{L-1} indexing the respective amount of channels and q, r indexing the filter coordinates of K_{L-1}. Note, that we still assume here that the last convolutional layer has spatial extent 0 into both directions, so $K_L \in \mathbb{R}^{\kappa_{\mathrm{in}} \times C \times 1 \times 1}$. The chain rule for this forward pass is then

$$\begin{aligned} \frac{\partial \widehat{\phi}^d_{ij}}{\partial ((K_{T-1})^f_e)_{gh}} &= \sum_{k_L} K^d_{k_L} \mathrm{ReLU}'(\cdot) \cdot \mathsf{BN}'_L \cdot \left(\sum_{k_{L-1}} \sum_{q,r} \frac{\partial ((K_{L-1})^{k_L}_{k_{L-1}})_{qr}}{\partial ((K_{L-1})^f_e)_{gh}} (\psi_{L-1})^{k_{L-1}}_{i+q,j+r} \right) \\ &= \sum_{k_L} K^d_{k_L} \mathrm{ReLU}'(\cdot) \cdot \mathsf{BN}'_L \cdot \sum_{k_{L-1}} \sum_{q,r} \delta^{k_L f} \delta_{k_{L-1} e} \delta_{gp} \delta_{hq} (\psi_{L-1})^{k_{L-1}}_{i+q,j+r} \\ &= K^d_f (\mathrm{ReLU}')^f_{ij} \cdot (\mathsf{BN}'_L)^f_{ij} \cdot (\psi_{L-1})^e_{i+g,j+h}. \end{aligned} \tag{4.19}$$

The term with the derivative of the ReLU activation is simply the Heaviside function evaluated at the features pre-activation which have been computed in the forward pass previously. The discontinuity at zero has vanishing probability of an evaluation. The derivative of the batch normalization layer is multiplication by the linear batch norm parameter which is applied channel-wise. The running indices g and h only apply to the last factor which can be computationally treated by extracting (3×3)-patches from ψ_{T-1} using the unfold operation. In computing the norm of $\nabla_{K_{T-1}} \widehat{\phi}^d_{ij}$ with respect to the indices e, f, g and h, we note that the expression in eq. (4.19) is a product of two tensors $S_f \cdot \psi^e_{gh}$ for each pixel (i, j).

4.3.2 Gradient Uncertainty Scores

We obtain pixel-wise scores, i.e., still depending on i and j, by computing the partial norm $\|\nabla_K \mathcal{L}\|_p$ of this tensor over the indices e and h for some fixed value of p. This can again be done in a memory efficient way by the natural decomposition of the norm of $\partial \mathcal{L} / \partial K^h_e = S_h \cdot \psi^e$. L^p-norms of such tensors $T_{ab} = S_a \psi_b$ *factorize which makes their computation feasible*:

$$\|T\|_p = \left(\sum_{a,b} |T_{ab}|^p \right)^{\frac{1}{p}} = \left(\sum_{a,b} |S_a|^p |\psi_b|^p \right)^{\frac{1}{p}} = \left(\sum_a \left[|S_a|^p \sum_b |\psi_b|^p \right] \right)^{\frac{1}{p}} = \|S\|_p \|\psi\|_p. \tag{4.20}$$

Note that the index b in this computation stands as a compound index for the triple (e, g, h). In addition to their use in error detection, these scores can be used in order to detect OoD objects in the input, i.e., instances of categories not present in the training distribution of the segmentation network. We *identify OoD regions with pixels that have a gradient score higher than some threshold* and find connected components like the one shown in Figure 4.1 (right). Different values of p are studied in section 4.4. We also consider values $0 < p < 1$ in addition to positive integer values. Note, that such choices do not strictly define the geometry of a vector space norm, however, $\| \cdot \|_p$ may still serve as a notion of magnitude and generates a partial ordering. The tensorized multiplications required in eqs. (4.8) and (4.9) are far less computationally expensive than a forward pass through the DNN. This means that computationally, this method is preferable over methods which sample predictions like MC Dropout or ensembles. We abbreviate our method using the **p**ixel-wise **g**radient **n**orms obtained from eqs. (4.8) and (4.9) by $\mathrm{PGN}_{\mathrm{oh}}$ and $\mathrm{PGN}_{\mathrm{uni}}$, respectively. The particular value of p is denoted by superscript, e.g., $\mathrm{PGN}_{\mathrm{oh}}^{p=0.3}$ for the $(p = 0.3)$-seminorm of gradients obtained from y^{oh}.

4.4 Experiments: Uncertainty Quantification on Pixel- and Segment-Level, OoD Segmentation and Runtime

In this section, we present the experimental setting first and then evaluate the uncertainty estimation quality of our method on pixel and segment level. Last, we apply our gradient-based method to OoD segmentation and show some visual results.

4.4.1 Implementation Details

Datasets. We perform our tests on the Cityscapes [28] dataset for semantic segmentation in street scenes and on four OoD segmentation datasets, namely LostAndFound [137], Fishyscapes [4], RoadAnomaly21 and RoadObstacle21 [14][1].

The **Cityscapes** dataset consists of 2,975 training and 500 validation images of dense urban traffic in 18 and 3 different German towns, respectively. The LostAndFound dataset contains 1,203 validation images with annotations for the road surface and the OoD objects, i.e., small obstacles on German roads in front of the ego-car.

In [14] this dataset is filtered (**LostAndFound test-NoKnown**) as children and bicycles are considered as OoD objects, although they are included in the Cityscapes training set.

The **Fishyscapes LostAndFound** dataset is based on the LostAndFound dataset and refines the pixel-wise annotations, i.e., it is distinguished between OoD objects, background (Cityscapes classes) and void (anything else). Moreover, frames that do not contain OoD objects as well as sequences with children and bikes are removed resulting in 100 validation images and 275 non-public test images.

[1]Benchmark: (`http://segmentmeifyoucan.com/`)

The **RoadObstacle21** dataset (412 test images) is comparable to the LostAndFound dataset as all obstacles appear on the road. However, the RoadObstacle21 dataset contains more diversity in the OoD objects as well as in the situations (night, snowy conditions and dirty roads) compared to the LostAndFound dataset where all sequences are recorded under good weather conditions. In the **RoadAnomaly21** dataset the objects (anomalies) appear anywhere on the image which makes it comparable to the Fishyscapes LostAndFound dataset.

While the latter dataset consists of frames extracted from a small number of different scenes, every image of the RoadAnomaly21 dataset (100 test images) shows a unique scene and a wider variety of anomalous objects.

Segmentation Networks. We consider a state-of-the-art DeepLabv3+ network [20] for two backbones, WideResNet38 [192] and SEResNeXt50 [66]. The network with each respective backbone is trained on Cityscapes achieving a mean IoU value of 90.58% for the WideResNet38 backbone and 80.76% for the SEResNeXt50 on the Cityscapes validation set. We use one and the same model trained exclusively on the Cityscapes dataset for both tasks, the uncertainty estimation *and* the OoD segmentation, as our method does not need to be re-trained on OoD data. Therefore, our method leaves the entire segmentation performance of the base model completely intact.

4.4.1.1 Details on the Feature Construction for Segment-wise Prediction Quality Estimation

The tasks of meta classification (false positive detection) and meta regression (prediction quality estimation) based on uncertainty measures extracted from the network's softmax output were introduced in [152]. Recall section 3.3.2 for the notions of meta classification and meta regression. The neural network provides for each pixel $(i,j) \in \mathcal{I}$ given an input image $\boldsymbol{x}$ a probability distribution $\widehat{f}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta})$ over a label space $\mathcal{Y} = [C]$, cf. section 2.3.3. The degree of randomness in the semantic segmentation prediction $\widehat{f}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta})$ is quantified by pixel-wise dispersion measures, like entropy (cf. eq. (2.91)) $H_{i,j}(\boldsymbol{x})$ and probability margin

$$M_{i,j}(\boldsymbol{x}) = 1 - \widehat{s}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) + \max_{c \in \mathcal{Y} \setminus \{\widehat{c}^{i,j}(\boldsymbol{x}|\boldsymbol{\theta})\}} f_c^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) \tag{4.21}$$

where we refer to section 2.3.3 for notation. To obtain segment-wise features from these dispersion measures which characterize uncertainty, they are aggregated over segments S (see section 2.3.3) via average pooling obtaining mean dispersion measures

$$\mu_H := \frac{1}{|S|} \sum_{(i,j)\in S} H_{i,j}(\boldsymbol{x}), \qquad \mu_M := \frac{1}{|S|} \sum_{(i,j)\in S} M_{i,j}(\boldsymbol{x}). \tag{4.22}$$

As erroneous or poor predictions oftentimes have fractal segment shapes, it is distinguished between the inner of the segment S° (consisting of all pixels whose eight neighboring pixels are also elements of this segment) and the boundary ∂S. This results in segment size $|S|$ and mean dispersion features per segment also for the inner (μ_H° and μ_M°) and the boundary (μ_H^∂ and μ_M^∂). Furthermore, relative segment sizes $\tilde{S} = |S|/|\partial S|$ and $\tilde{S}^\circ = |S^\circ|/|\partial S|$ as well as relative mean dispersion measures $\tilde{\mu}_D := \tilde{S} \cdot \mu_D$ and $\tilde{\mu}_D^\circ := \tilde{S}^\circ \cdot \mu_D^\circ$ where $D \in \{H, M\}$ are defined. Moreover, the mean class probabilities

$$\mu_{P(c)} := \frac{1}{|S|} \sum_{(i,j) \in S} \widehat{f}_c^{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) \tag{4.23}$$

for each class $c \in [C]$ are added resulting in the vector of hand-crafted features consisting of $\{|S|, |S^\circ|, |\partial S|, \tilde{S}, \tilde{S}^\circ\}$ as well as

$$\{\mu_D, \mu_D^\circ, \mu_D^\partial, \tilde{\mu}_D, \tilde{\mu}_D^\partial \,:\, D \in \{H, M\}\} \cup \{\mu_{P(c)} \,:\, c \in [C]\}\,. \tag{4.24}$$

These features serve as a baseline in our tests.

The computed gradients in section 4.3 with applied p-norm, $p \in \{0.1, 0.3, 0.5, 1, 2\}$, are denoted by $G_{p=\#}^{i,j}(\boldsymbol{x})$, $\# \in \{0.1, 0.3, 0.5, 1, 2\}$, for input $\boldsymbol{x}$. Similar to the dispersion measures, we compute the mean and additionally the variance of these pixel-wise values of a given segment to obtain

$$\mu_{G_{p=\#}} := \frac{1}{|S|} \sum_{(i,j) \in S} G_{p=\#}^{i,j}(\boldsymbol{x}), \qquad v_{G_{p=\#}} := \frac{1}{|S|-1} \sum_{(i,j) \in S} \left(G_{p=\#}^{i,j}(\boldsymbol{x}) - \mu_{G_{p=\#}} \right)^2, \tag{4.25}$$

respectively. Since the gradient uncertainties may be higher on the boundary of a segment, the mean and variance features per segment are considered also for the inner ($\mu_{G_{p=\#}}^\circ$ and $v_{G_{p=\#}}^\circ$) and the boundary ($\mu_{G_{p=\#}}^\partial$ and $v_{G_{p=\#}}^\partial$). Furthermore, we define relative mean and variance features $\tilde{\mu}_{G_{p=\#}}, \tilde{\mu}_{G_{p=\#}}^\circ, \tilde{v}_{G_{p=\#}}, \tilde{v}_{G_{p=\#}}^\circ$ to quantify the degree of fractality. This results in the following gradient features

$$\bigcup_{\# \in \{0.1,0.3,0.5,1,2\}} \{\mu_{G_{p=\#}}, \mu_{G_{p=\#}}^\circ, \mu_{G_{p=\#}}^\partial, \tilde{\mu}_{G_{p=\#}}, \tilde{\mu}_{G_{p=\#}}^\circ, v_{G_{p=\#}}, v_{G_{p=\#}}^\circ, v_{G_{p=\#}}^\partial, \tilde{v}_{G_{p=\#}}, \tilde{v}_{G_{p=\#}}^\circ\} \tag{4.26}$$

for each predicted segment. Note, these features can be computed for the gradient scores obtained from the predictive one-hot ($\mathrm{PGN_{oh}}$) and the uniform label ($\mathrm{PGN_{uni}}$) as well as for gradients of deeper layers (see section 4.3.1).

In our experiments, we apply different p-norms, $p \in \{0.1, 0.3, 0.5, 1, 2\}$, to our calculated gradients, see section 4.3.2. We provide only a selection of results in the next sections, followed by ablation studies on different values for p and further results where the gradients of deeper layers are computed.

4.4.2 Pixel-wise Uncertainty Quantification

In order to assess the correlation of uncertainty and prediction errors on the pixel-level, we resort to the commonly used sparsification graphs [72]. Sparsification graphs normalized with respect to the optimal oracle ("sparsification error") can be compared in terms of the so-called area under the sparsification error curve (AuSE). The closer the uncertainty estimation is to the oracle in terms of Brier score evaluation, i.e., the smaller the AuSE,

Table 4.1: Pixel-wise uncertainty evaluation results for both backbone architectures and the Cityscapes dataset in terms of *ECE* and AuSE.

	WideResNet		SEResNeXt	
	ECE	AuSE	*ECE*	AuSE
Ensemble	0.0173	0.4543	0.0279	0.0482
MC Dropout	0.0444	0.7056	0.0091	0.5867
Maximum Softmax	**0.0017**	0.0277	**0.0032**	**0.0327**
Entropy	0.0063	0.0642	0.0066	0.0623
$\mathrm{PGN}_{\mathrm{oh}}^{p=2}$ (ours)	0.0019	**0.0268**	0.0039	0.0365
$\mathrm{PGN}_{\mathrm{uni}}^{p=0.1}$ (ours)	0.0186	0.0784	0.0346	0.0427

the better the uncertainty eliminates false predictions by the model. The AuSE metric is capable of grading the uncertainty ranking, however, does not address the statistics in terms of given confidence. Therefore, we resort to an evaluation of calibration in terms of expected calibration error (*ECE*, see section 3.3.2 and [55]) to assess the statistical reliability of the uncertainty measure.

As baseline methods we consider the typically used uncertainty estimation measures, i.e., mutual information computed via the sampling approaches ensembles and MC Dropout. For an ensemble of N predictive probability distributions $\widehat{\pi}^1, \dots, \widehat{\pi}^N$ where each $\widehat{\pi}^i = (\widehat{\pi}^i_1, \dots, \widehat{\pi}^i_C)$, the mutual information for prediction sampling is given by (see [70])

$$I\left(\widehat{\pi}^1, \dots, \widehat{\pi}^N\right) = H\left(\frac{1}{N}\sum_{i\in[N]} \widehat{\pi}^i\right) - \frac{1}{N}\sum_{i\in[N]} H\left(\widehat{\pi}^i\right). \tag{4.27}$$

Moreover, we consider the uncertainty ranking provided by the maximum softmax probabilities native to the segmentation model and the softmax entropy. An evaluation of calibration errors requires normalized scores, so we normalize our gradient scores according to the highest value obtained on the test data for the computation of *ECE*.

The resulting metrics are compiled in table 4.1 for both architectures evaluated on the Cityscapes val split. We see that the calibration of $\mathrm{PGN}_{\mathrm{oh}}$ is roughly on par with the stronger maximum softmax baseline (which was trained to exactly that aim via the negative log-likelihood loss) and outperforms the other three baselines. Note, that there is no immediate reason for $\mathrm{PGN}_{\mathrm{oh}}$ to be calibrated in any way. Generally, we see the trend that $\mathrm{PGN}_{\mathrm{uni}}$ has higher calibration error by one order of magnitude and higher sparsification errors of up to 5.2 percentage points (pp) which may indicate that this kind of score is less indicative of in-distribution errors.

Ablations on Values of p ***and Deeper Gradients.*** In table 4.2 the results for the different p-norms in terms of these two metrics are given. We observe improved performance for the gradient scores obtained from the predictive one-hot label with respect

Formally, this is the mutual information between the parameters $\boldsymbol{\theta}$ of the model and the resulting prediction $\widehat{\pi}$ both regarded as random variables. Refer to [125] for the definition of the mutual information between random variables.

Table 4.2: Pixel-wise uncertainty evaluation results for both backbone architectures and the Cityscapes' dataset as well as for different p-norms and layers in terms of *ECE* and AuSE.

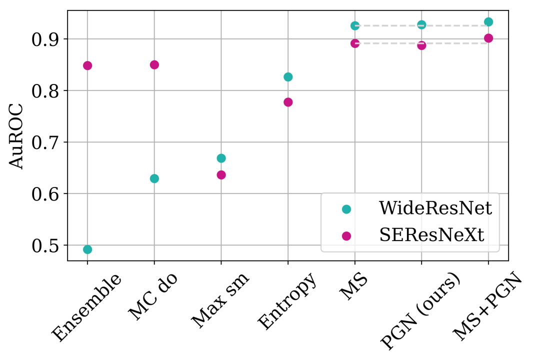
		last layer		second-to-last layer	
	p	*ECE* ↓	AuSE ↓	*ECE* ↓	AuSE ↓
	0.1	0.0187	0.0500	0.0186	$\underline{0.0235}$
Wide-	0.3	0.0183	0.0712	0.0125	0.0286
ResNet	0.5	0.0163	0.0508	0.0059	0.0280
(*one-hot*)	1	$\underline{0.0025}$	0.0307	$\underline{0.0027}$	0.0271
	2	**0.0019**	$\underline{0.0268}$	**0.0021**	0.0265
	0.1	0.0186	0.0784	0.0096	0.3347
Wide-	0.3	0.0163	0.3426	0.1846	0.6746
ResNet	0.5	0.0241	0.5857	0.3385	0.7520
(*uniform*)	1	0.3762	0.7424	0.4345	0.8028
	2	0.3868	0.8104	0.3989	0.8253
	0.1	0.0347	**0.0201**	0.0344	**0.0060**
SERes-	0.3	0.0336	0.0386	0.0290	0.0353
NeXt	0.5	0.0305	0.0399	0.0214	0.0379
(*one-hot*)	1	0.0078	0.0383	0.0079	0.0377
	2	0.0039	0.0365	0.0068	0.0365
	0.1	0.0346	0.0427	0.0295	0.1823
SERes-	0.3	0.0313	0.2484	0.1916	0.4878
NeXt	0.5	0.0076	0.5617	0.3000	0.6198
(*uniform*)	1	0.3744	0.7694	0.4075	0.7413
	2	0.4030	0.8187	0.4003	0.8116

to both metrics. These scores are better calibrated for greater values of p, whereas there is no clear trend for the AuSE metric. In contrast, for the gradient scores obtained from the uniform label the calibration ability as well as the sparsification error enhance mostly for decreasing values of p. We also find results for both layers in table 4.2. For the gradient scores obtained from the predictive one-hot, the evaluation metrics are mainly similar showing only small performance gaps. The results differ for the uniform labels, although there is no trend to which layer achieves the higher ones.

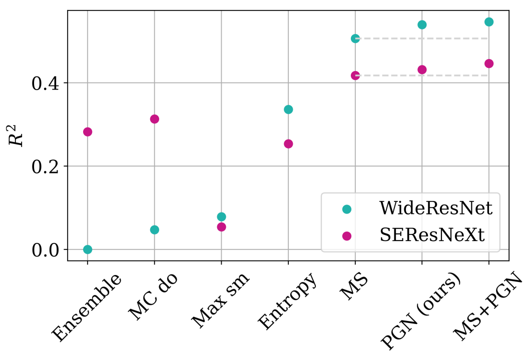
4.4.3 Segment-wise Uncertainty Quantification

To reduce the number of FP predictions and to estimate the prediction quality, we use meta classification and meta regression. As input for the post-processing models, the quantities described in section 4.4.1.1 are utilized. The degree of randomness in semantic segmentation prediction is quantified by pixel-wise quantities, like entropy and probability margin. To obtain segment-wise features from these quantities, they are aggregated over segments via average pooling. These features used in [152] serve as a baseline in our tests (called MetaSeg). Similarly, we construct segment-wise features from our pixel-wise gradient p-norms for $p \in \{0.1, 0.3, 0.5, 1, 2\}$, recall section 4.4.1.1. These hand-crafted quantities form a structured dataset where the columns correspond to features and the rows to predicted segments which serve as input to the post-processing models.

We determine the prediction accuracy of semantic segmentation networks with respect to

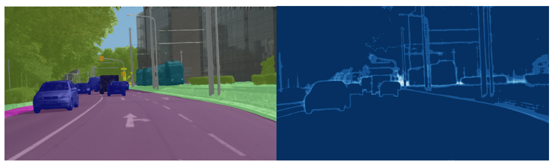


(a) Meta Classification

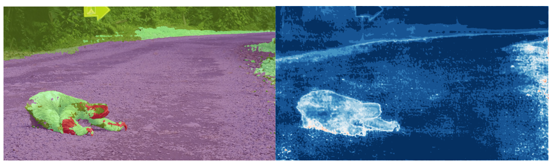


(b) Meta Regression

Figure 4.3: Segment-wise uncertainty evaluation results for both backbone architectures and the Cityscapes dataset in terms of (a) meta classification AuROC and (b) meta regression R^2 values. From left to right: ensemble, MC Dropout, maximum softmax, mean entropy, MetaSeg (MS) approach, gradient features obtained by predictive one-hot and uniform labels (PGN), MetaSeg in combination with PGN.



(a) Cityscapes



(b) RoadAnomaly21

Figure 4.4: Semantic segmentation prediction and $\text{PGN}_{uni}^{p=0.5}$ heatmap for (a) the Cityscapes dataset and (b) the RoadAnomaly21 dataset for the WideResNet backbone.

the ground truth via the segment-wise intersection over union (IoU, [74]). We use linear models as meta classifiers and meta regression models.

For the evaluation, we use AuROC for meta classification and determination coefficient R^2 for meta regression. As baselines, we employ a deep ensemble, MC Dropout, maximum softmax probability, entropy and the MetaSeg framework. A comparison of these methods and our approach for the Cityscapes dataset is given in fig. 4.3. The gradient scores outperform all baselines with the only exception of meta classification for the SEResNeXt backbone where MetaSeg achieves a marginal 0.41 pp higher AuROC value than ours. Such a post-processing model captures all the information which is contained in the network's output. Therefore, matching the MetaSeg performance by a gradient heatmap is a substantial achievement for a predictive uncertainty method. Moreover, we enhance the MetaSeg performance for both networks and both tasks combining the MetaSeg features with PGN by up to 1.02 pp for meta classification and up to 3.98 pp for meta regression. This indicates that gradient features contain information which is orthogonal to the information contained in the softmax output. Especially, the highest AuROC value of 93.31% achieved for the WideResNet backbone, shows the capability of our approach to estimate

Table 4.3: OoD segmentation benchmark results for the LostAndFound and the RoadObstacle21 dataset.

	LostAndFound test-NoKnown					RoadObstacle21				
	AuPRC ↑	FPR_{95} ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑	AuPRC ↑	FPR_{95} ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑
Ensemble	2.9	82.0	6.7	7.6	2.7	1.1	77.2	8.6	4.7	1.3
MC Dropout	36.8	35.6	17.4	34.7	13.0	4.9	50.3	5.5	5.8	1.1
Maximum Softmax	30.1	33.2	14.2	**62.2**	10.3	15.7	16.6	19.7	15.9	6.3
Entropy	52.0	30.0	40.4	$\underline{53.8}$	42.4	**20.6**	$\underline{16.3}$	$\underline{21.4}$	**19.5**	**10.4**
$\text{PGN}_{oh}^{p=0.5}$	$\underline{64.9}$	$\underline{18.4}$	$\underline{48.3}$	50.0	**46.9**	$\underline{18.8}$	**14.8**	**22.1**	$\underline{16.5}$	$\underline{9.2}$
$\text{PGN}_{uni}^{p=0.5}$	**69.3**	**9.8**	**50.0**	44.8	$\underline{45.4}$	16.5	19.7	19.5	14.8	7.4

Table 4.4: OoD segmentation benchmark results for the Fishyscapes LostAndFound and the Road-Anomaly21 dataset.

	Fishyscapes LostAndFound					RoadAnomaly21				
	AuPRC ↑	FPR_{95} ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑	AuPRC ↑	FPR_{95} ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑
Ensemble	0.3	90.4	3.1	1.1	0.4	17.7	91.1	16.4	20.8	3.4
MC Dropout	14.4	47.8	4.8	18.1	4.3	28.9	69.5	20.5	17.3	4.3
Maximum Softmax	5.6	40.5	3.5	9.5	1.8	28.0	72.1	15.5	15.3	5.4
Entropy	14.0	39.3	8.0	17.5	7.7	31.6	71.9	15.7	18.4	4.2
$\text{PGN}_{oh}^{p=0.5}$	$\underline{22.8}$	**35.5**	$\underline{12.1}$	$\underline{27.3}$	$\underline{14.1}$	$\underline{39.3}$	$\underline{66.5}$	$\underline{23.1}$	$\underline{21.5}$	$\underline{7.8}$
$\text{PGN}_{uni}^{p=0.5}$	**26.9**	$\underline{36.6}$	**14.8**	**29.6**	**16.5**	**42.8**	**56.4**	**25.8**	**21.8**	**9.7**

the prediction quality and filter out FP predictions on the segment-level for in-distribution data.

Ablations on Values of** p **and Deeper Gradients. The gradient features which are computed for each value of p, also separated for predictive one-hot and uniform labels, serve as input for the metamodels. The results are given in table 4.5 and visualized in fig. 4.5. The WideResNet backbone outperforms the SEResNeXt for meta classification and regression. Moreover, higher AuROC and R^2 performances are achieved for greater values of p independent of the architecture or the label (predictive one-hot or uniform) used for gradient scores computation. We observe the same behavior for the second-to-last layer evaluation as for the last one. Namely, that as p increases, performance improves for both metrics. Furthermore, the performance is almost equal for the second-to-last and the last layer for the 1- and the 2-norm independent of the backbone and labels to obtain the gradient scores.

4.4.4 Out-of-Distribution Segmentation

Our results in OoD segmentation are based on the evaluation protocol of the official SegmentMeIfYouCan benchmark [14]. Evaluation on the pixel-level involves the threshold-independent area under precision-recall curve (AuPRC) and the false positive rate at the point of 0.95 true positive rate (FPR_{95}). The latter constitutes an interpretable choice of operating point for each method where a minimum true positive fraction is dictated. On segment-level, an adjusted version of the mean intersection over union (sIoU) which

Table 4.5: Segment-wise uncertainty evaluation results for both backbone architectures and the Cityscapes dataset as well as for the different p-norms and layers in terms of classification AuROC and regression R^2.

		last layer		second-to-last layer	
	p	AuROC ↑	R^2 ↑	AuROC ↑	R^2 ↑
	0.1	87.66	19.40	88.40	29.45
Wide-	0.3	89.02	38.03	90.28	48.23
ResNet	0.5	90.06	46.82	90.52	49.63
(*one-hot*)	1	**91.97**	<u>50.82</u>	91.04	50.28
	2	<u>91.90</u>	50.72	**91.54**	<u>50.54</u>
	0.1	87.97	22.65	89.67	27.99
Wide-	0.3	89.84	43.72	90.84	45.96
ResNet	0.5	90.66	48.71	91.08	48.66
(*uniform*)	1	91.18	50.26	<u>91.36</u>	50.48
	2	91.77	**51.48**	**91.54**	**51.37**
	0.1	81.65	12.81	81.48	2.29
SERes-	0.3	82.91	25.73	84.80	32.56
NeXt	0.5	84.24	33.43	85.28	36.21
(*one-hot*)	1	85.68	38.57	86.01	38.03
	2	87.70	39.08	87.82	39.00
	0.1	81.90	14.41	78.49	6.06
SERes-	0.3	83.91	30.44	85.14	28.28
NeXt	0.5	85.51	35.40	85.62	33.40
(*uniform*)	1	86.46	37.61	86.38	38.05
	2	87.30	40.54	87.31	40.38

represents the accuracy of the segmentation obtained by thresholding at a particular point, positive predictive value (PPV) playing the role of binary instance-wise accuracy and the F_1-score. The latter segment-wise metrics are averaged over thresholds between 0.25 and 0.75 with a step size of 0.05 leading to the quantities $\overline{\mathrm{sIoU}}$, $\overline{\mathrm{PPV}}$ and $\overline{F_1}$.

In contrast to the previous evaluations on predictive errors, we do not report here results for both of our models, rather we compare gradient scores as an OoD score as such against other methods on the benchmark and report the best result obtained for both, $\mathrm{PGN}_{\mathrm{oh}}$ and $\mathrm{PGN}_{\mathrm{uni}}$. Results are based on evaluation files submitted to the public benchmark and are, therefore, deterministic. As baselines, we include the same methods as for error detection before since these constitute a reasonable comparison. Note, that the standard entropy baseline is not featured in the official leaderboard, so we report our own results obtained by the softmax entropy with the WideResNet backbone which performed better. We include a full table of methods in the next paragraph and find that our method is in several cases competitive with some of the stronger methods utilizing adversarial examples, auxiliary data or significant alterations of the model architecture.

The results verified by the official benchmark are compiled in table 4.3 and table 4.4. Across the board, PGN shows strong performance, almost exclusively delivering the best or second-best results with few exceptions. The only prominent exception is in the segment-based PPV metric on LostAndFound test-NoKnown where both, $\mathrm{PGN}_{\mathrm{oh}}$ and $\mathrm{PGN}_{\mathrm{uni}}$ come in behind the maximum softmax and entropy baseline. Meanwhile, the segmenta-

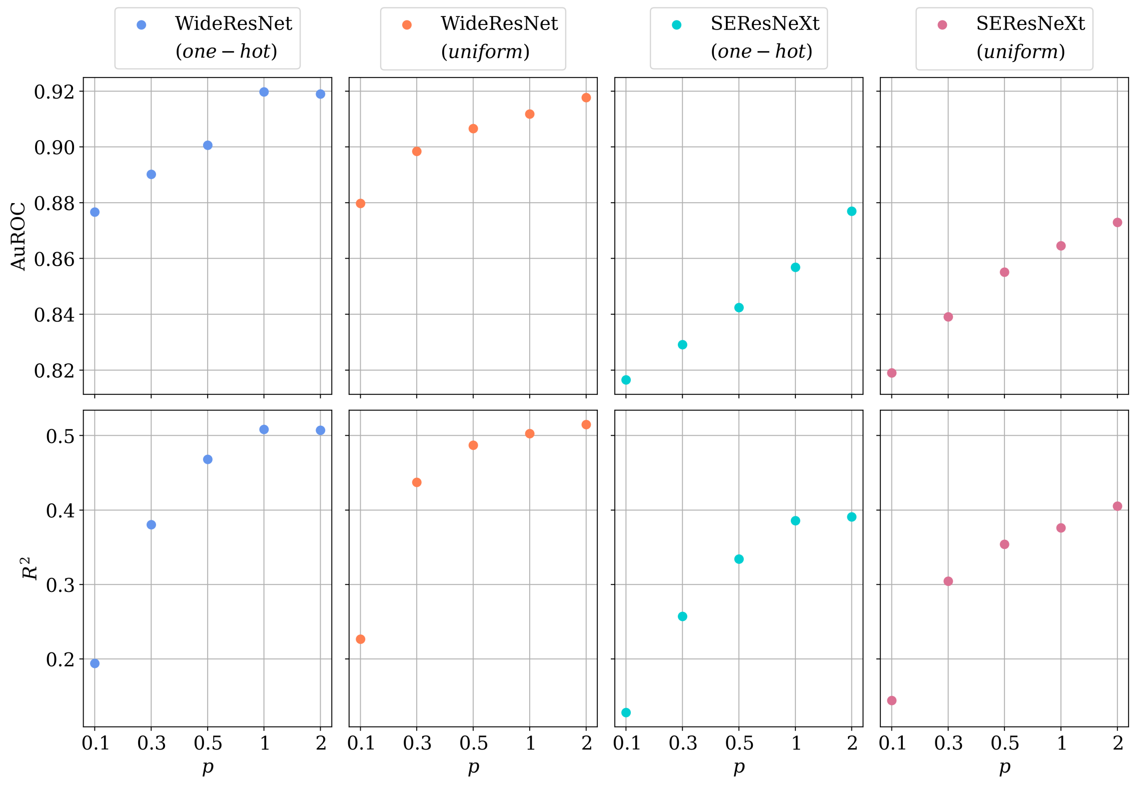


Figure 4.5: AuROC and R^2 values for both backbone architectures applied to the Cityscapes dataset and the different p-norms.

tion accuracy in terms of sIoU, as well as the F_1 score which is the harmonic mean of recall and precision are far superior to these two baselines. This indicates still better OoD segmentation quality and in particular better recall achieved by PGN. Gradient scores perform perhaps the least convincingly on the RoadObstacle21 benchmark, where in three cases only the second-best performance is achieved. Overall however, we conclude strong performance of our method in terms of out-of-distribution segmentation across different datasets. We find a slight trend of $\mathrm{PGN}_{\mathrm{oh}}$ performing better in the Obstacle track which can be seen to be closer to in-distribution data in semantic segmentation for autonomous driving. This connection would be consistent with our finding in actual in-distribution errors, while $\mathrm{PGN}_{\mathrm{uni}}$ performs better on the Anomaly track which is more clearly out-of-distribution for street scene recognition. In several cases, our method even beats some stronger OoD segmentation methods utilizing, for example on RoadAnomaly21, adversarial samples (ODIN/Mahalanobis by over 9 pp in AuPRC), OoD data (Void Classifier by over 6 pp AuPRC) or generative models (JSRNet/Embedding Density by over 5 pp AuPRC and Image Resynthesis by over 10 pp in $\overline{\mathrm{PPV}}$).

Figure 4.4 shows segmentation predictions of the pre-trained DeepLabv3+ network with the WideResNet38 backbone together with its $\mathrm{PGN}_{\mathrm{uni}}^{p=0.5}$-heatmap. The two panels on the left show an in-distribution prediction on Cityscapes where uncertainty is mainly

Table 4.6: OoD segmentation results for the LostAndFound and the RoadObstacle21 dataset.

	LostAndFound test-NoKnown					RoadObstacle21				
	AuPRC ↑	FPR$_{95}$ ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑	AuPRC ↑	FPR$_{95}$ ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑
Void Classifier	4.8	47.0	1.8	35.1	1.9	10.4	41.5	6.3	20.3	5.4
SynBoost	**81.7**	4.6	36.8	**72.3**	48.7	71.3	3.2	44.3	41.8	37.6
Maximized Entropy	77.9	9.7	45.9	63.1	49.9	85.1	0.8	**47.9**	**62.6**	48.5
PEBAL	–	–	–	–	–	5.0	12.7	29.9	7.6	5.5
DenseHybrid	78.7	**2.1**	**46.9**	52.1	**52.3**	**87.1**	**0.2**	45.7	50.1	**50.7**
Image Resynthesis	57.1	8.8	27.2	30.7	19.2	37.7	4.7	16.6	20.5	8.4
Road Inpainting	82.9	35.8	49.2	**60.7**	52.3	54.1	47.1	**57.6**	39.5	36.0
Embedding Density	61.7	10.4	37.8	35.2	27.6	0.8	46.4	35.6	2.9	2.3
NFlowJS	**89.3**	**0.7**	**54.6**	59.7	**61.8**	**85.6**	**0.4**	45.5	**49.5**	**50.4**
JSRNet	74.2	6.6	34.3	45.9	36.0	28.1	28.9	18.6	24.5	11.0
ODIN	52.9	30.0	**39.8**	**49.3**	**34.5**	**22.1**	15.3	**21.6**	18.5	**9.4**
Mahalanobis	**55.0**	**12.9**	33.8	31.7	22.1	20.9	**13.1**	13.5	**21.8**	4.7
PGN$_{oh}^{p=0.5}$	64.9	18.4	48.3	**50.0**	**46.9**	**18.8**	**14.8**	**22.1**	**16.5**	**9.2**
PGN$_{uni}^{p=0.5}$	**69.3**	**9.8**	**50.0**	44.8	45.4	16.5	19.7	19.5	14.8	7.4

concentrated around segmentation boundaries which are always subject to high prediction uncertainty. Moreover, we see some false predictions in the far distance around the street crossing which can be found as a region of high gradient norm in the heatmap. In the two panels to the right, we see an OoD prediction from the RoadAnomaly21 dataset of a sloth crossing the street which is classified as part vegetation, terrain and person. The outline of the segmented sloth can be seen brightly in the gradient norm heatmap to the right indicating clear separation.

Additional Comparisons with the SegmentMeIfYouCan Leaderboard. Above, we have compared our approach with uncertainty estimation methods such as entropy and maximum softmax. Moreover, we considered two sampling approaches, MC Dropout and deep ensembles, as baselines all of which are not explicitly designed towards OoD segmentation. In table 4.6 and table 4.7, we provide the comparison of our method with more methods from the benchmark. In detail, the first block consists of approaches using OoD data, i.e., Void Classifier [4], SynBoost [34], Maximized Entropy [15], PEBAL [176] and DenseHybrid [54]. The methods of the second block use complex auxiliary or generative models, namely Image Resynthesis [105], Road Inpainting [104], Embedding Density [4], NFlowJS [53], JSRNet [183] and ObsNet [3]. The ODIN [99] and Mahalanobis [95] baselines (in the third block) perform adversarial attacks on the input images and thus, require a full and expensive backward pass. Per block, we indicate the best method for each of the considered metrics.

Our method outperforms the two latter baselines ODIN and Mahalanobis for the LostAndFound as well as the RoadAnomaly21 dataset. In particular, we obtain AuPRC values up to 22.8 pp higher on segment-level and $\overline{F_1}$ values up to 24.8 pp higher on pixel-level. For the other two datasets we achieve similar results. Furthermore, we beat the Void Classifier method, that uses OoD data during training, in most cases. We improve the AuPRC metric by up to 64.5 pp and the $\overline{\text{sIoU}}$ metric by up to 48.2 pp, both for the LostAndFound dataset. In addition, our gradient norms outperform in most cases the

Table 4.7: OoD segmentation results for the Fishyscapes LostAndFound and the RoadAnomaly21 dataset.

	Fishyscapes LostAndFound					RoadAnomaly21				
	AuPRC ↑	FPR$_{95}$ ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑	AuPRC ↑	FPR$_{95}$ ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑
Void Classifier	11.7	**15.3**	9.2	39.1	14.9	36.6	63.5	21.1	22.1	6.5
SynBoost	**64.9**	30.9	**27.9**	**48.6**	**38.0**	56.4	61.9	34.7	17.8	10.0
Maximized Entropy	44.3	37.7	21.1	**48.6**	30.0	**85.5**	15.0	49.2	**39.5**	28.7
PEBAL	–	–	–	–	–	49.1	40.8	38.9	27.2	14.5
DenseHybrid	–	–	–	–	–	78.0	**9.8**	**54.2**	24.1	**31.1**
Image Resynthesis	5.1	**29.8**	5.1	**12.6**	4.1	52.3	**25.9**	39.7	11.0	12.5
Embedding Density	**8.9**	42.2	**5.9**	10.8	**4.9**	37.5	70.8	33.9	20.5	7.9
NFlowJS	–	–	–	–	–	56.9	34.7	36.9	18.0	14.9
JSRNet	–	–	–	–	–	33.6	43.9	20.2	29.3	13.7
ObsNet	–	–	–	–	–	**75.4**	26.7	**44.2**	**52.6**	**45.1**
ODIN	15.5	38.4	9.9	21.9	9.7	**33.1**	**71.7**	**19.5**	**17.9**	**5.2**
Mahalanobis	**32.9**	**8.7**	**19.6**	**29.4**	**19.2**	20.0	87.0	14.8	10.2	2.7
PGN$_{oh}^{p=0.5}$	22.8	**35.5**	12.1	27.3	14.1	39.3	66.5	23.1	21.5	7.8
PGN$_{uni}^{p=0.5}$	**26.9**	36.6	**14.8**	**29.6**	**16.5**	**42.8**	**56.4**	**25.8**	**21.8**	**9.7**

Embedding Density approach which is based on normalizing flows. Summing up, we have shown superior OoD segmentation performance in comparison to the other uncertainty based methods (see section 4.4) and we outperform some of the more complex approaches using OoD data, adversarial samples or generative models.

Ablations on Values of p ***and Deeper Gradients.*** Our approach provides pixel-wise uncertainty scores obtained by computing the partial norm. In fig. 4.6 the pixel-wise heatmaps for both backbones, different p-norms and the predictive one-hot as well as the uniform label are shown. We observe that for higher p values the number of uncertain pixels increases, the gradients are more sensitive to unconfident predictions. For a p value of 0.1 only a few pixels of OoD object have high uncertainty while the background is completely certain. For values of $p = 1$ and $p = 2$, in particular using the uniform label, the gradient scores show higher uncertainties in more sectors. To identify out-of-distribution regions, we perform thresholding per pixel on our gradient scores, i.e., high uncertainty corresponds to out-of-distribution. Here, the OoD objects are mostly covered (and not as many background pixels) for p values of 0.3 and 0.5. These observations are reflected in the OoD segmentation results, given in table 4.8 and table 4.9. For the LostAndFound dataset and the Fishyscapes LostAndFound dataset, the best results are achieved for the 0.3 and 0.5 p-norms. For the RoadAnomaly21 dataset, also for higher p values strong (in one case even the best) results are obtained. Across these three datasets, there is no favorability which backbone architecture or label (predictive one-hot or uniform) performs better. In comparison with the RoadObstacle21 dataset, the WideResNet backbone with gradient scores obtained from the predictive one-hot performs best. There seems to be no clear tendency which p-norm outperforms the others for the different tasks of pixel- and segment-wise uncertainty estimation as well as for OoD segmentation. However, there is a strong trend towards $p \in \{0.3, 0.5\}$ performing especially strongly.

In table 4.10 the OoD segmentation results for gradients from the second-to-last layer

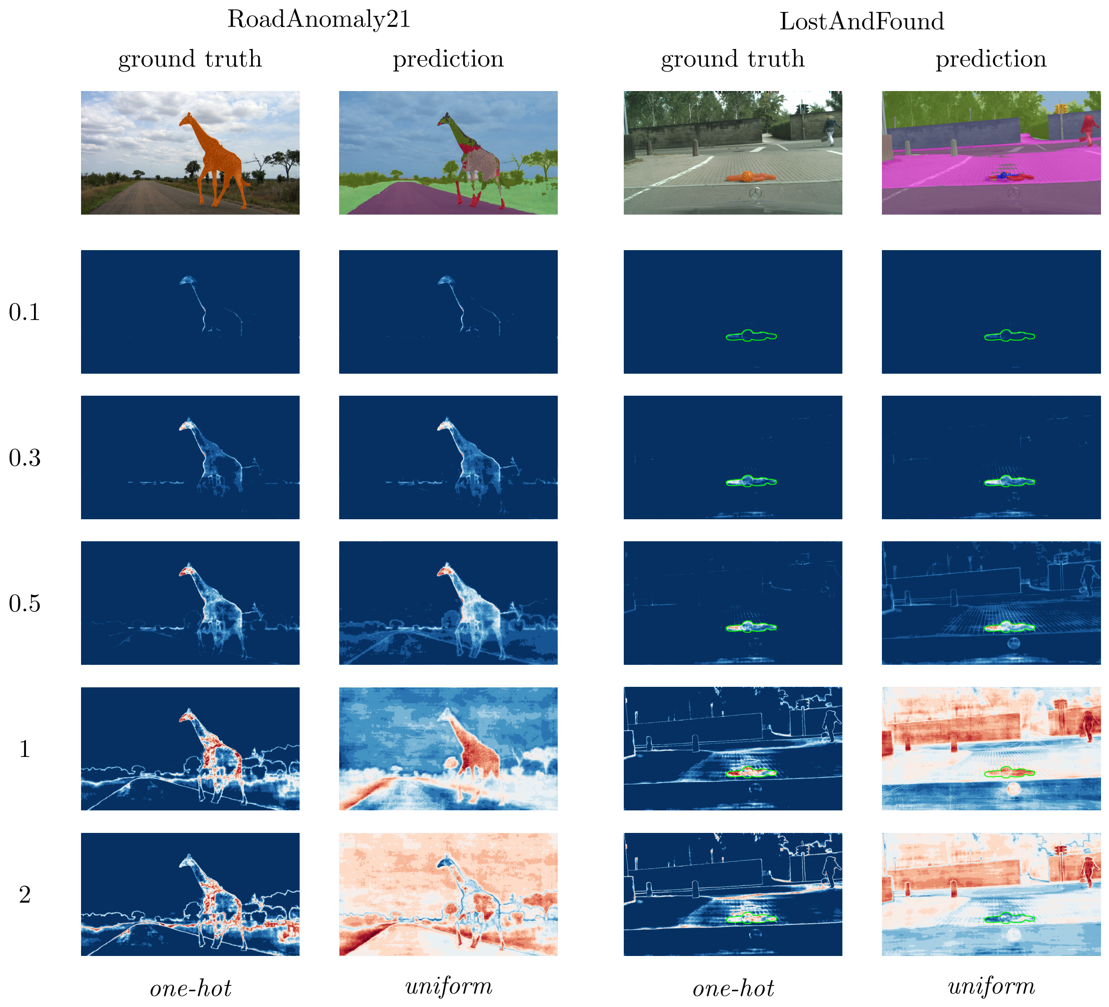


Figure 4.6: Ground truth (labeled OoD object), semantic segmentation prediction and PGN heatmaps for different p-norms as well as for the predictive one-hot and the uniform label, respectively.

Table 4.8: OoD segmentation results for the LostAndFound and the RoadObstacle21 dataset for different p-norms.

		LostAndFound					RoadObstacle21				
	p	AuPRC ↑	FPR_{95} ↓	$\overline{\mathrm{sIoU}}$ ↑	$\overline{\mathrm{PPV}}$ ↑	$\overline{F_1}$ ↑	AuPRC ↑	FPR_{95} ↓	$\overline{\mathrm{sIoU}}$ ↑	$\overline{\mathrm{PPV}}$ ↑	$\overline{F_1}$ ↑
Wide-ResNet (*one-hot*)	0.1	49.5	21.2	39.5	29.6	25.1	8.0	23.3	14.2	7.5	2.9
	0.3	64.9	15.9	**48.5**	$\underline{47.6}$	$\underline{45.8}$	16.3	$\underline{15.4}$	$\underline{20.8}$	14.4	$\underline{7.5}$
	0.5	64.9	18.4	48.3	**50.0**	**46.9**	**18.8**	**14.8**	**22.1**	$\underline{16.5}$	**9.2**
	1	44.3	25.1	25.8	40.8	21.6	$\underline{17.8}$	15.5	17.1	**16.9**	6.1
	2	11.5	30.6	26.9	26.1	16.3	12.8	16.7	19.2	13.9	5.5
Wide-ResNet (*uniform*)	0.1	47.1	21.8	38.4	27.2	22.3	7.2	25.4	13.6	6.9	2.5
	0.3	64.8	13.5	$\underline{48.4}$	44.1	43.2	14.4	17.2	18.0	13.0	6.4
	0.5	69.3	9.8	50.0	44.8	45.4	16.5	19.7	19.5	14.8	7.4
	1	57.7	10.1	33.7	35.2	27.4	8.0	62.6	5.7	8.8	1.2
	2	8.1	100.0	7.5	19.2	4.1	3.4	99.9	1.7	14.0	0.3
SERes-NeXt (*one-hot*)	0.1	66.6	5.2	43.8	35.0	34.0	3.5	39.1	5.3	6.7	1.1
	0.3	**75.1**	$\underline{4.2}$	46.2	44.2	43.8	6.7	26.8	5.3	9.7	2.5
	0.5	70.3	7.8	43.4	44.6	41.7	8.1	24.5	5.8	10.1	3.2
	1	45.1	14.5	22.6	36.5	18.5	8.5	24.7	4.2	11.8	1.8
	2	15.1	18.7	22.8	21.9	12.8	5.6	26.2	9.0	11.3	3.1
SERes-NeXt (*uniform*)	0.1	64.7	6.1	42.8	34.9	32.4	3.2	41.3	5.7	6.1	1.0
	0.3	$\underline{75.0}$	**3.9**	46.4	43.8	43.3	6.1	29.0	5.2	9.0	2.1
	0.5	73.8	6.9	45.1	44.8	42.3	7.4	28.7	5.7	10.3	2.7
	1	42.0	49.5	17.1	33.8	11.7	11.2	79.2	5.1	12.0	1.8
	2	5.8	100.0	3.3	15.7	1.4	9.2	99.8	1.7	15.5	1.7

are given. In comparison to the performance of the gradient scores of the last layer (see table 4.8 and table 4.9), the performance of the second-to-last layer is poor. Particularly, the results for all evaluation metrics are worse than these of the last layer. In some cases, there is no detection capability at all for the gradient scores obtained from the uniform label. To summarize, the gradients of the second-to-last layer improve neither the uncertainty estimation at pixel- and segment-level nor the OoD segmentation quality, rather they perform worse in some cases. The finding that deeper layer gradients contain less information than the final layer has been observed before outside the semantic segmentation setting by [69] and [145].

4.4.5 Computational Runtime

Lastly, we demonstrate the computational efficiency of our method and show runtime measurements for the network forward pass ("Softmax"), MC dropout (25 samples) and computing both of $\mathrm{PGN}_{\mathrm{oh}}$ and $\mathrm{PGN}_{\mathrm{uni}}$. While sampling MC dropout requires over twice the time per frame for both backbone networks, the computation of $\mathrm{PGN}_{\mathrm{oh}} + \mathrm{PGN}_{\mathrm{uni}}$ only leads to a marginal computational overhead of around 1% due to consisting only of tensor multiplications. The time measurements were each made on a single Nvidia Quadro P6000 GPU.

Table 4.9: OoD segmentation results for the Fishyscapes LostAndFound and the RoadAnomaly21 dataset for different p-norms.

		Fishyscapes LostAndFound					RoadAnomaly21				
	p	AuPRC ↑	FPR_{95} ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑	AuPRC ↑	FPR_{95} ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑
	0.1	23.9	22.0	13.9	29.5	15.6	28.2	75.4	18.2	14.8	4.2
Wide-	0.3	$\underline{26.9}$	31.2	**16.2**	30.0	$\underline{18.1}$	33.8	70.5	20.9	18.4	6.0
ResNet	0.5	22.8	35.5	12.1	27.3	14.1	34.5	69.5	19.4	19.3	5.5
(*one-hot*)	1	10.1	39.1	4.0	14.6	3.3	32.5	69.5	16.1	17.0	5.8
	2	1.5	41.7	11.5	2.7	1.7	25.7	70.2	15.5	15.1	5.8
	0.1	23.0	$\underline{21.7}$	13.6	27.6	14.7	27.6	75.9	17.8	14.6	4.1
Wide-	0.3	**28.3**	29.2	$\underline{15.7}$	$\underline{32.1}$	**18.4**	33.7	69.7	20.8	17.1	5.8
ResNet	0.5	$\underline{26.9}$	36.6	14.8	29.6	16.5	36.7	61.4	21.6	17.8	6.2
(*uniform*)	1	0.8	66.4	6.2	2.5	2.0	$\underline{45.2}$	$\underline{60.7}$	24.8	26.2	$\underline{9.5}$
	2	0.2	99.8	0.2	0.3	0.0	29.2	97.7	21.1	$\underline{29.0}$	6.1
	0.1	21.0	**15.6**	10.4	21.2	8.3	36.2	65.7	21.9	16.9	5.7
SERes-	0.3	23.7	23.7	10.1	26.8	11.6	40.5	64.9	24.5	19.5	8.7
NeXt	0.5	20.8	29.8	8.1	23.9	9.4	39.3	66.5	23.1	21.5	7.8
(*one-hot*)	1	8.8	36.3	4.6	12.4	3.4	33.3	69.4	17.2	16.2	8.1
	2	1.2	39.4	11.5	2.2	1.6	27.1	71.0	16.2	14.0	7.4
	0.1	20.6	**15.6**	8.1	21.6	6.6	35.6	66.2	21.3	17.0	5.3
SERes-	0.3	24.1	21.8	10.5	26.3	11.9	41.1	62.7	$\underline{24.9}$	20.0	8.5
NeXt	0.5	22.6	32.0	8.6	**32.3**	11.0	42.8	**56.4**	**25.8**	21.8	**9.7**
(*uniform*)	1	0.6	83.3	1.8	2.3	0.6	**47.4**	67.3	23.7	24.9	9.0
	2	0.2	99.8	0.2	0.3	0.0	35.0	98.1	22.3	**30.7**	8.3

4.5 Conclusion: Gradient-based Uncertainty Quantification in Semantic Segmentation

In this work, we presented an efficient method of computing gradient uncertainty scores for a wide class of deep semantic segmentation models. Moreover, we appreciate a low computational cost associated with them. Our experiments show that large gradient norms obtained by our method statistically correspond to erroneous predictions already on the pixel-level and can be normalized such that they yield similarly calibrated confidence measures as the maximum softmax score of the model. On a segment-level our method shows considerable improvement in terms of error detection. Gradient scores can be utilized to segment out-of-distribution objects significantly better than sampling- or any other output-based method on the SegmentMeIfYouCan benchmark and has competitive results with a variety of methods, in several cases clearly outperforming all of them while coming at negligible computational overhead.

Table 4.10: OoD segmentation results for the LostAndFound and the Fishyscapes LostAndFound dataset for different p-norms and the second-to-last layer.

		LostAndFound					Fishyscapes LostAndFound				
	p	AuPRC ↑	FPR$_{95}$ ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑	AuPRC ↑	FPR$_{95}$ ↓	$\overline{\text{sIoU}}$ ↑	$\overline{\text{PPV}}$ ↑	$\overline{F_1}$ ↑
	0.1	10.3	40.0	12.6	15.8	4.0	**2.0**	36.8	1.6	3.0	0.6
Wide-	0.3	**10.9**	32.9	23.1	23.0	11.3	1.7	39.4	2.3	2.2	0.5
ResNet	0.5	10.5	32.4	26.9	25.5	15.3	1.5	40.3	9.6	2.3	1.3
(*one-hot*)	1	10.0	32.2	28.9	**26.1**	17.3	1.4	41.1	12.2	2.5	1.9
	2	9.7	32.1	**30.0**	25.7	**18.3**	1.3	41.5	**13.6**	2.8	**2.4**
	0.1	0.7	98.8	0.5	0.8	0.0	0.3	96.0	0.8	0.7	0.0
Wide-	0.3	0.5	99.8	0.5	0.8	0.0	0.2	98.9	0.2	0.3	0.0
ResNet	0.5	0.5	99.9	0.5	0.8	0.0	0.2	99.5	0.2	0.3	0.0
(*uniform*)	1	0.5	100.0	0.5	0.8	0.0	0.2	99.7	0.2	0.3	0.0
	2	0.5	100.0	0.5	0.8	0.0	0.2	99.8	0.2	0.3	0.0
	0.1	3.0	66.2	4.9	7.3	1.4	1.9	41.1	5.8	**5.2**	1.8
SERes-	0.3	6.8	27.1	15.4	12.9	6.1	1.9	**36.0**	5.2	5.0	1.6
NeXt	0.5	8.1	23.9	19.5	15.2	8.6	1.7	36.8	4.1	4.3	1.0
(*one-hot*)	1	9.3	21.7	22.6	17.8	11.1	1.4	37.6	7.0	2.1	0.9
	2	10.4	**20.4**	24.1	18.4	12.7	1.2	38.3	11.5	2.3	1.7
	0.1	0.4	99.9	0.5	0.8	0.0	0.2	88.9	0.3	0.0	0.0
SERes-	0.3	0.4	99.9	0.5	0.8	0.0	0.2	93.2	0.2	0.0	0.0
NeXt	0.5	0.4	100.0	0.5	0.8	0.0	0.2	96.1	0.2	0.3	0.0
(*uniform*)	1	0.4	100.0	0.5	0.8	0.0	0.2	99.2	0.2	0.3	0.0
	2	0.5	100.0	0.5	0.8	0.0	0.2	99.7	0.2	0.3	0.0

Table 4.11: Runtime measurements in seconds per frame for each method; standard deviations taken over samples in the Cityscapes validation dataset.

sec. per frame	WideResNet	SEResNeXt
Softmax	3.02 ± 0.18	1.08 ± 0.04
MC Dropout	7.52 ± 0.35	2.31 ± 0.10
$\text{PGN}_{\text{oh}} + \text{PGN}_{\text{uni}}$	3.05 ± 0.16	1.09 ± 0.05

5 Rapid Prototyping of Active Learning for Object Detection

In this chapter we introduce and investigate a sandbox environment for developing and testing active learning strategies for deep object detection. The presented contents are in large parts taken word-for-word from [146].

5.1 Introduction: Active Learning for Deep Object Detectors

Deep learning requires large amounts of data, typically annotated by vast amounts of human labor [9, 98, 212]. In particular in complex computer vision tasks such as object detection (OD), the amount of labor per image can lead to substantial costs for data labeling. Therefore, it is desirable to avoid unnecessary labeling effort and to have a rather large variability of the database. *Active learning* (AL) [164] is one of the key methodologies that aims at labeling the data that matters for learning. AL alternates model training and data labeling as illustrated in fig. 5.1. At the core of each AL method is a query strategy that decides after each model training *which unlabeled data to query for labeling*. The computation cost of AL is in general at least an order of magnitude higher than ordinary model training and so is its development [98, 177], which comprises several AL experiments involving T query steps with different parameters, ablation studies, etc. Hence, it is notoriously challenging to develop new AL methods for applications where model training itself is already computationally costly. In the field of OD, a number of works overcame this cumbersome hurdle [8, 23, 32, 37, 58, 134, 156, 160, 172, 206, 210]. However, these works did so in very different settings which makes their comparison difficult. Hence, it is difficult to give any advice for practitioners, which AL method to choose. Besides that, AL with real-world data may suffer from other influencing factors, e.g., the quality of labels to which end fundamental research is conducted on AL in presence of label errors [6, 7, 207, 208]. These observations demand for a development environment that enables rapid prototyping,

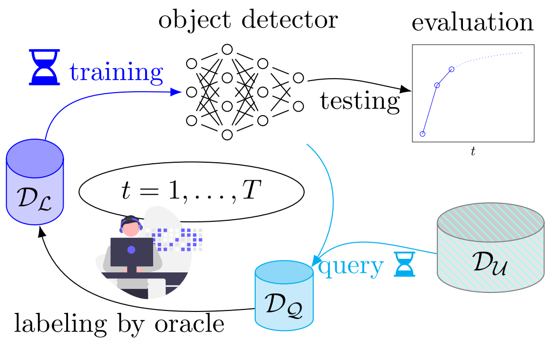


Figure 5.1: The generic pool-based AL cycle consisting of training on labeled data $\mathcal{D}_{\mathcal{L}}$, querying informative data points $\mathcal{D}_{\mathcal{Q}}$ out of a pool of unlabeled data $\mathcal{D}_{\mathcal{U}}$ and annotation by a (human) oracle. In practice, training compute time is orders of magnitude larger than evaluating the AL strategy itself.

cutting down to the huge computational efforts of AL in object detection and fostering comparability of different AL methods.

In this chapter, we propose a development environment that drastically cuts down the computational cost of developing AL methods. We do so by reducing the complexity of the deep object detectors as well as the complexity of the database, establishing a non-trivial sandbox for deep OD. To this end, we construct (a) two datasets that generalize MNIST digits [92] and EMNIST letters [26] to the setting of OD by pasting colored and transformed samples into background images from MS-COCO [102] and (b) a selection of suitable small-scale object detectors. We justify this step and underline its value for the field of AL in OD. To do so, we conduct several experiments which show that results on our datasets generalize to a similar degree to established but more complex datasets (Pascal VOC [38] and BDD100k [209]), as they generalize among each other. We also demonstrate that we reduce the computational effort of AL experiments by factors of up to 32. In addition, to establish further comparability of AL methods for OD, we propose an extensive evaluation protocol. We summarize our contributions as follows:

- We propose a sandbox environment with two datasets, multiple networks, active learning queries and an extensive evaluation protocol. This setup allows for broader comparisons and detailed and transparent experiment tracking at lowered computational effort. We demonstrate this with extensive numerical results.
- We analyze the generalization capabilities of our sandbox and find that results obtained by our sandbox generalize well to Pascal VOC and BDD100k.
- We contribute to future AL development by providing an implementation of our pipeline in a flexible environment as well as an automated framework for evaluation and visualization of results.

Code is publicly available[46]. All selected configurations and hyperparameters in addition to evaluation files for the experiments conducted will be published in order to facilitate

[46]`https://github.com/tobiasriedlinger/al-rapid-prototyping`

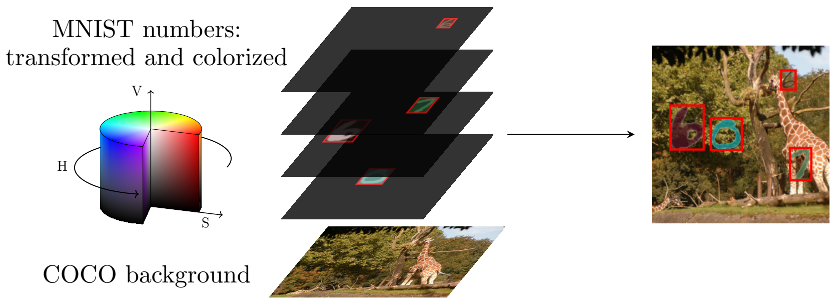


Figure 5.2: Generation scheme of semisynthetic object detection data from MNIST digits on a non-trivial background image from MS COCO.

future comparisons with the implemented baselines.

5.2 Related Work: State of Active Learning and Specializations to Object Detection

Numerous methods of AL have been, and still are, developed in the classification setting and largely fall into two main categories. Uncertainty-based query strategies rely on the informativeness of a probabilistic model's current prediction uncertainty [96]. In probabilistic classification models, popular examples include the probability margin [159], ensemble entropy [30], or committee disagreement [165]. A different approach is taken by density-based query strategies, which aim at exploiting data diversity between already known data and prospective queries. Typically, such methods are related to clustering algorithms [129,163,196] which can also be used in conjunction with uncertainty estimation methods. For a general overview of different AL approaches see for example the survey by Settles [164].

The task of OD is more complex than image classification in that the background needs to be distinguished against foreground instances and each foreground instance needs to be assigned its own localization and classification. Not only does this require the choice of additional hyperparameters for making predictions, but this also entails more complex and expensive labels. Therefore, AL plays also a large role in OD. Yoo and Kweon [206] present a task-agnostic method where a loss prediction module estimates a loss for every potentially queried image and selects the images for which the largest loss is expected. Brust et al. [8] estimate prediction-wise uncertainty by the probability margin and aggregate to image uncertainty by the summation, averaging or taking the maximum. For a black-box approach, Roy et al. [156] follow the same idea, but with using the classification entropy as prediction uncertainty. Moreover, they propose a white-box approach which is inspired by query-by-committee, making use of predictions at different scales in the object detector. In general, an ensemble of OD heads, trained independently on the same set of labeled images, can be used to estimate prediction-wise classification uncertainty, i.e., mutual information based on entropy [58], or a combination of localization and classification uncertainty due to region of interest similarities [160]. But in particular, as training a variety of detector heads in each AL step is very costly, committee or ensemble query methods tend to be approximated by Monte Carlo (MC) Dropout [45, 202]. Some methods yield to custom object detectors, i.e., a Gaussian mixture model based object detector with an adjusted loss [23], or they intervene at least in the training pipeline, e.g., by suppressing noisy instances [210]. Another class of approaches is based on semi-supervised learning, i.e., incorporating either pseudo-labels for some representative easy samples to prevent distribution drift [37]. Alternatively used "weak" labels consist only of the center coordinates [32, 134, 172]. In this chapter we compare uncertainty-based methods with each other that are exclusively based on fully supervised training [8, 23, 58, 156]. These works are difficult to compare since the datasets, network architectures, frameworks, initial dataset sizes, query sizes and hyperparameters for training and inference heavily differ from each other. Unlike the works mentioned, we aim at putting the AL task itself on equal footing between different settings to improve development speed and evaluation transparency. Here, we compare some methods mentioned above to each other based on the same configurations for frequently used datasets and network architectures. Comparative investigations of this kind has escaped previous research in the field.

5.3 Methods: A Sandbox Environment with Datasets, Models and Evaluation Metrics

In this section we describe the objective of AL and also our sandbox environment. The main setting which we propose consists of two semisynthetic OD datasets and downscaled versions of standard object detectors leaving the detection mechanism unchanged. Additionally, we introduce evaluations capturing different aspects of the observed AL curve.

5.3.1 The Active Learning Task

The term active learning refers to a setup depicted in fig. 5.1 where only a limited amount of fully annotated data $\mathcal{D}_\mathcal{L}$ is available together with a task-specific model. In addition, there is a pool or a stream of unlabeled data $\mathcal{D}_\mathcal{U}$ from which the model queries those samples $\mathcal{D}_\mathcal{Q}$ which are most informative. Success of the query strategy is measured by observing that *training on the new data pool leads to an increase of test performance* higher than another method. Afterwards, $\mathcal{D}_\mathcal{Q}$ is annotated by an oracle which is usually a human worker, added to $\mathcal{D}_\mathcal{L}$ and the model is fine-tuned or fitted from scratch on $\mathcal{D}_\mathcal{L}$ from where the cycle continues. Before each acquisition step, the current model performance is evaluated which leads to graphs like the ones in fig. 5.3. The query step can take diverse algorithmic forms, see section 5.2 or [164].

5.3.2 Sandbox Datasets

We construct an OD problem by building a synthetic overlay to images from the real-world MS COCO dataset (see fig. 5.2), which constitutes the data of our sandbox. MS COCO images with deleted annotations provide a realistic, feature-rich background on which

Table 5.1: Exemplary object detection architectures with backbone configurations employed in the experiments and associated number of parameters.

Detector	Backbone	# params	Backbone	# params
RetinaNet	ResNet50	36.5M	ResNet18	20.1M
Faster R-CNN	ResNet101	60.2M	ResNet18	28.3M
YOLOv3	Darknet53	61.6M	Darknet20	10.3M

foreground objects shall be recognized. As foreground, we utilize two sets of categories: MNIST digits and EMNIST letters. We apply randomized coloration and opacity to foreground instances such that trivial edge detection becomes unfeasible. Colors are drawn from HSV space uniformly by $(h, s, v) \sim \mathscr{U}([0.0, 1.0] \times [0.05, 1.0] \times [0.1, 1.0])$ which yielded the best optical variability. The chosen HSV-color is multiplied with the gray scale value of the original image avoiding the instances being monochromatic. Opacity is similarly drawn uniformly $\alpha \sim \mathscr{U}([0.5, 0.9])$ and in addition, we apply image translation, scaling and shearing to all numbers/letters. The number of instances per background image is Poisson-distributed with mean $\lambda = 3$. Tight bounding box and instance segmentation annotations can be obtained from the original transformed gray scale (E)MNIST images and the category label can be adopted. Compared to simple detection datasets such as SVHN [187], the geometric variety in our datasets is more similar to those of OD benchmarks such as Pascal VOC or MS COCO. The reduction in the dataset complexity allows for the *achievement of high performance even for small architectures and leads to quickly converging training* and low inference times. In the following we use the terms "MNIST-Det" and "EMNIST-Det" to refer to our OD datasets.

5.3.3 Sandbox Models

Modern OD architectures utilize several conceptually different mechanisms to solve the detection task (recall section 2.3.2). Irrespective of the amount of accessible data, some applications of OD may require high inference speed while others may require a large degree of precision or some trade-off between the two. The type of architecture as well as the underlying detection mechanism are, however, disjoint to some degree from the depth of the feature extraction in the backbone. The latter is mainly responsible for the quality and resolution of features. We use this insight to down-scale architectures for AL by reducing the network depth *while keeping the detection mechanism in the network's head unchanged.* Together with the simplified OD objectives from section 5.3.2, we obtain a well-performing, low-capacity and fast-inference setup. This allows us to study AL with the same OD mechanisms that occur in practice. table 5.1 shows the choices for a YOLOv3 [39], RetinaNet [101] and Faster R-CNN [144] setup, which we have made for our investigations together with the resulting number of parameters. The number of parameters in the standard setup was reduced by up to a factor of around 6 leading to a significant decrease in training and inference time.

5.3.4 Active Learning Methods for Object Detection

For the construction of baseline AL methods, we focus on querying whole images instead of single bounding boxes. The latter would introduce an additional dimension of complexity where unlabeled image regions need to be ignored. The frequently used uncertainty-based query strategies from image classification, such as entropy, probability margin, MC dropout, and mutual information, determine prediction-specific but not image-wise uncertainties. Hence, we introduce an aggregation step to obtain image-wise query scores.

For a given image $\boldsymbol{x}$, a neural network predicts a fixed number $\widehat{N}_{\text{out}}$ of bounding boxes

$$\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta}) = \left\{(\widehat{\mathrm{x}}_{ij}^n, \widehat{\mathrm{y}}_{ij}^n, \widehat{\mathrm{w}}_{ij}^n, \widehat{\mathrm{h}}_{ij}^n, \widehat{s}_{ij}^n, (\widehat{\pi}_1)_{ij}^n, \ldots, (\widehat{\pi}_C)_{ij}^n)\right\}_{n\in[N_{\text{Anch}}],(i,j)\in\mathcal{I}/\varsigma^2}. \tag{5.1}$$

See section 2.3.2 for the respective notation. Only the set of boxes that remain after NMS and score thresholding are used to determine prediction uncertainties. Already the choice of the parameters for the NMS and the score threshold influence the queries, since they decide which (uncertain) predictions $\widehat{y}^j$ (where $j \in N_{\boldsymbol{x}}$) remain.

Given a prediction $\widehat{y}^j$ we compute its *classification entropy*

$$H(\widehat{y}^j(\boldsymbol{x})) := -\sum_{c\in[C]} \widehat{\pi}_c^j \cdot \log\left(\widehat{\pi}_c^j\right) \tag{5.2}$$

and its *probability margin score*

$$M(\widehat{y}^j(\boldsymbol{x})) := (1 - [\widehat{\pi}_{\widehat{c}^j}^j - \max_{c\neq\widehat{c}^j} \widehat{\pi}_c^j])^2. \tag{5.3}$$

Here, $\widehat{c}^j$ denotes the class with the highest predicted probability in $\widehat{y}^j$. When dropout is implemented in the architecture, MC Dropout samples can be drawn at inference time when the output of the same anchor box j is sampled N_{DO} times

$$\widetilde{y}^j(\boldsymbol{x}) := \{\widehat{y}_1^j, \ldots, \widehat{y}_{N_{\text{DO}}}^j\}. \tag{5.4}$$

The *mutual information under MC dropout* is estimated by

$$I(\widetilde{y}^j(\boldsymbol{x})) := H\left(\frac{1}{N_{\text{DO}}}\sum_{k=1}^{N_{\text{DO}}} \widehat{y}_k^j\right) - \frac{1}{N_{\text{DO}}}\sum_{k=1}^{N_{\text{DO}}} H\left(\widehat{y}_k^j\right) \tag{5.5}$$

with the second term being the average entropy of the individual samples. We also regard the maximum feature standard deviations within $\widetilde{y}^j$ by first standardizing variances over all query predictions over the unlabeled dataset $\mathcal{D}_{\mathcal{U}}$ to treat localization and classification features on the same footing. Standardization

$$\varphi(\boldsymbol{x}) \mapsto \overline{\varphi}(\boldsymbol{x}) := \frac{\varphi(\boldsymbol{x}) - \frac{1}{|\mathcal{D}_{\mathcal{U}}|}\sum_{\boldsymbol{x}_{\mathrm{u}}\in\mathcal{D}_{\mathcal{U}}} \varphi(\boldsymbol{x}_{\mathrm{u}})}{\mathrm{std}_{\boldsymbol{x}_{\mathrm{j}}\in\mathcal{D}_{\mathcal{U}}}\varphi(\boldsymbol{x}_{\mathrm{u}})} \tag{5.6}$$

maps any $\varphi \in \{\widehat{\mathrm{x}}, \widehat{\mathrm{y}}, \widehat{\mathrm{w}}, \widehat{\mathrm{h}}, \widehat{s}, \widehat{\pi}_1, \ldots, \widehat{\pi}_C\}$ to a quantity that is standardized over $\mathcal{D}_{\mathcal{U}}$. Here, $\mathrm{std}_{\boldsymbol{x}_\mathrm{j} \in \mathcal{D}_{\mathcal{U}}} \varphi(\boldsymbol{x}_\mathrm{u})$ denotes the unbiased standard deviation estimator. The *dropout uncertainty* is then

$$D(\widetilde{y}^j(\boldsymbol{x})) := \max_{\varphi \in \{\widehat{\mathrm{x}}, \widehat{\mathrm{y}}, \widehat{\mathrm{w}}, \widehat{\mathrm{h}}, \widehat{s}, \widehat{\pi}_1, \ldots, \widehat{\pi}_C\}} \mathrm{std}_{k \in [N_{\mathrm{DO}}]} \overline{\varphi_k^j}(\boldsymbol{x}). \tag{5.7}$$

This quantity is the maximal standard deviation of bounding box features, after scale of the respective features has been accounted for by standardization. Note that for all these methods, uncertainty is only considered in the predicted foreground instances. Since the uncertainty-based selection strategies only determine prediction-based uncertainties, either the sum, average, or maximum is taken over predicted instances to obtain a final query score for the image. Summation, for instance, tends to prefer images with a large amount of instances while averaging is strongly biased by the thresholds (imagine considering many false positive predictions which could be filtered by a higher threshold). In the presented experiments, we use summation and further aggregations are investigated in an ablation study. Additionally, we also regard random acquisition as a completely uninformed query baseline.

Diversity-based methods make use of latent activation features in neural networks which heavily depend on the OD architecture. Since purely diversity-based methods have been less prominent in the literature, we focus on the more broadly established uncertainty baselines.

5.3.5 Evaluation Methods

Model Performance. In the literature, methods are frequently evaluated by counting the number of data samples needed to cross some fixed reference performance mark. For OD, performance is usually measured in terms of the mAP_{50} metric for which there is a maximum value known when training on all available data. In AL then, some percentage of this maximal performance, e.g., 90% needs to be reached with as few data points as possible. Collecting performance over amount of queried data gives rise to curves such as the one in the top right of fig. 5.1 which we call *AL curves* in the following.

Counting Annotations. "Amount of training data" usually translates to the number of queried images which is set as a hyperparameter and fixed for each method and AL cycle. Considering that in the annotation process, each bounding box needs to be labeled and there tends to be *high variance in the number of instances per image* in common benchmark datasets, it is not clear whether to measure annotated data in terms of images or instances. Therefore, we stress that the scaling of the t-axis is important especially in instance-wise prediction tasks such as OD. Both views, counting images or instances, can be argued for. Therefore, we evaluate the performance of each result not only based on the acquired images, but also transform the t-axis under the curves to the number of annotated instances. By linear interpolation between query points and averaging of individual seeds of the same experiment, we obtain image- or instance-wise standard deviations of the performance.

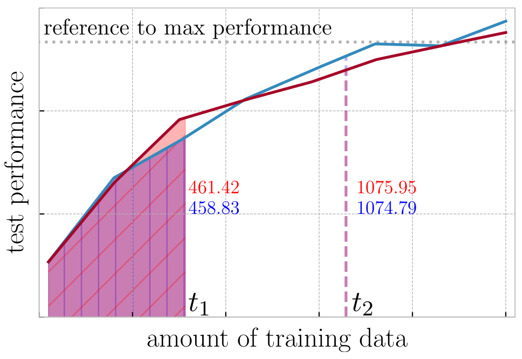


Figure 5.3: Area under AL curve (AUC) metric at different stages of an AL curve for two different query strategies (averaged, taken from fig. 5.5).

In light of the complexity of the AL problem, we propose the area under the AL curve (AUC). It constitutes a more robust performance metric compared to (horizontal or vertical) cross-sections through the learning curves. Figure 5.3 shows two AL curves and corresponding AUC metrics at two distinct points t_1 and t_2. Note that in practice, there is no fixed maximal dataset size for which a maximum performance exists. Then, the AL experiment may end and be evaluated at any given vertical section of t points of training data. Knowing the maximal performance (or the 90% mark shown in fig. 5.3) may lead to wrong conclusions in the presented case which is taken from the scenario in fig. 5.5. Ending the experiment at t_1 clearly determines the red curve (which also has a higher AUC) as preferable. Ending the experiment at t_2 favors the blue curve by just looking at the actual test performance. However, the AUC still favors the red curve, since it takes the complete AL curve into account. This is in line with our qualitative feeling of the curves when regarded up to t_2. Note that we use the AUC metric for calculating rank correlations in section 5.4.3. The raw results of the AUC metric are shown in extended experimental results.

5.4 Experiments: Benchmarking, Generalization of Results and Time Saving

In this section, we present results of experiments with our sandbox environment as well as established datasets, namely Pascal VOC and BDD100k, for the rest of this chapter abbreviated as VOC and BDD. We do so by presenting AL curves, summarizing benchmark results and discussing our observations for different evaluation metrics in section 5.4.2. We then show in section 5.4.3 quantitatively that our sandbox results generalize to the same extent to VOC and BDD as results obtained on those datasets generalize between each other. In other words, we demonstrate the dataset-wise representativity of the results obtained by our sandbox. Thereafter, this is complemented with a study on the computational speedup in section 5.4.5 when using the sandbox instead of VOC or BDD.

Table 5.2: Maximum mAP_{50} values achieved by the models in table 5.1 on the respective datasets (standard-size detectors on VOC and BDD; sandbox-size on (E)MNIST-Det). The entire available training data is used.

	YOLOv3	RetinaNet	Faster R-CNN
MNIST-Det	0.962	0.908	0.937
EMNIST-Det	0.959	0.919	0.928
Pascal VOC	0.794	0.748	0.797
BDD100k	0.426	0.464	0.525

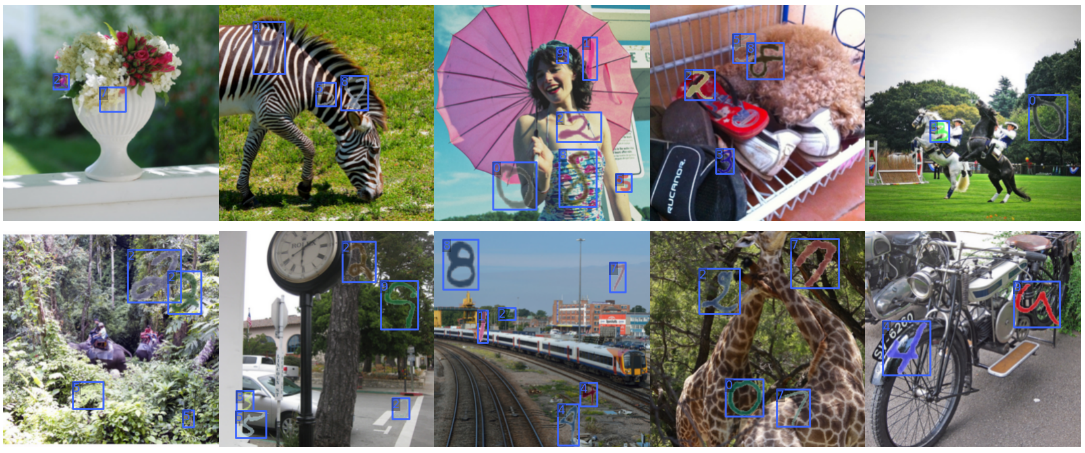
Table 5.3: Standard deviations of center coordinates, width and height (all relative to image size) of bounding boxes, as well, as number of categories in the training split for several object detection datasets.

Dataset	x	y	w	h	C
SVHN	0.099	0.059	0.048	0.161	10
Pascal VOC	0.217	0.163	0.284	0.277	20
MS COCO	0.254	0.209	0.220	0.234	80
KITTI	0.229	0.080	0.067	0.157	8
BDD100k	0.224	0.133	0.059	0.086	10
MNIST-Det	0.233	0.233	0.054	0.054	10
EMNIST-Det	0.233	0.233	0.066	0.065	26

5.4.1 Implementation Details

We implemented our pipeline in the open source MMDetection [18] toolbox. In our experiments for VOC, $\mathcal{D}_{\mathcal{U}}$ initially consists of "2007 train" + "2012 trainval" and we evaluate performance on the "2007 test"-split. At initialization, we randomly sample $\mathcal{D}_{\mathcal{L}}$ as a small portion of $\mathcal{D}_{\mathcal{U}}$. When tracking validation performance to assure convergence, we evaluate on "2007 val". Since BDD is a hard detection problem, we filtered frames with "clear" weather condition at "daytime" from the "train" split as initial pool $\mathcal{D}_{\mathcal{U}}$ yielding 12,454 images. We apply the same filter to the "val" split and divide it in half to get a test dataset (882 images for performance measurement) and a validation dataset (882 images for convergence tracking). For the (E)MNIST-Det datasets we generated 20,000 train images, 500 validation images and 2,000 test images. For reference, we collect in table 5.2 the achieved performance of the respective models for each dataset which determines the 90% mark investigated in our experiments.

Dataset Variability. When comparing to existing OD datasets, our sandbox datasets MNIST-Det and EMNIST-Det resemble in variability the common benchmarks like VOC, MS COCO, KITTI or BDD. This can be seen when looking at the variations in bounding box localization across each dataset. When normalizing to the total image resolution, we can compare the standard deviations in the annotation center coordinates x, y as well as the bounding box extent w and h, which we have collected in table 5.3. The SVHN dataset consisting of photographs of house numbers shows little variability, especially in the center localization of the object which are mostly centered on the image. Figure 5.4 shows samples from the MNIST-Det and EMNIST-Det datasets.


(a) MNIST-Det

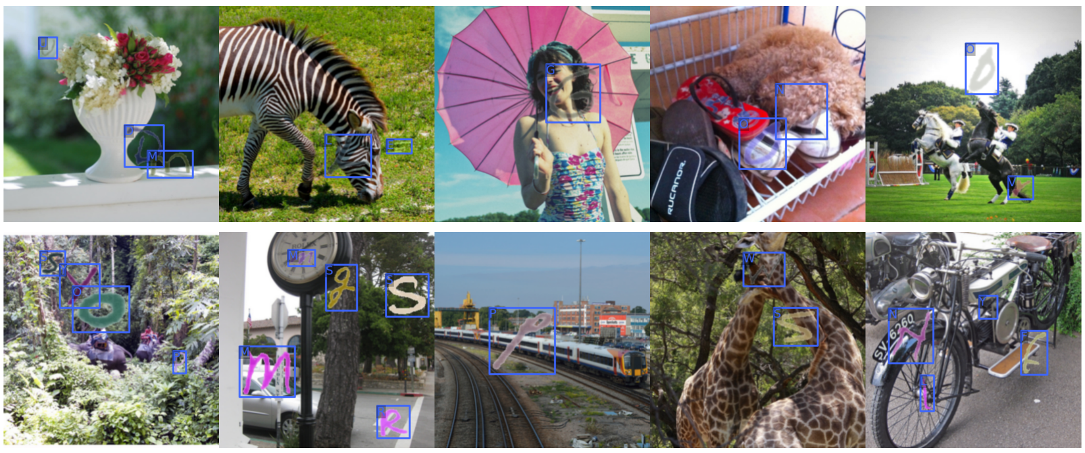

(b) EMNIST-Det

Figure 5.4: Dataset samples from the (a) MNIST-Det and (b) EMNIST-Det datasets including annotations.

Table 5.4: Configuration of Darknet20 compared with Darknet53 in analogy to [39, Tab. 1]. At equal resolution input, the feature maps also remain at the same resolution at each stage.

		Darknet53		Darknet20	
Type	Size	Blocks	Filters	Blocks	Filters
Conv	3×3		32		32
Conv	$3 \times 3/2$		64		32
Conv	1×1		32		32
Conv	3×3	$1\times$	64	$1\times$	64
Residual					
Conv	$3 \times 3/2$		128		64
Conv	1×1		64		32
Conv	3×3	$2\times$	128	$1\times$	64
Residual					
Conv	$3 \times 3/2$		256		128
Conv	1×1		128		64
Conv	3×3	$8\times$	256	$1\times$	128
Residual					
Conv	$3 \times 3/2$		512		256
Conv	1×1		256		128
Conv	3×3	$8\times$	512	$2\times$	256
Residual					
Conv	$3 \times 3/2$		1024		512
Conv	1×1		512		256
Conv	3×3	$4\times$	1024	$2\times$	512
Residual					

Table 5.5: Overview of important AL hyperparameters for querying data and model training for all datasets and architectures.

		Query					Training		
		$\lvert\mathcal{U}_{init}\rvert$	$\lvert\mathcal{Q}\rvert$	τ_s	image resolution	T	batch size	training iters	image resolution
YOLOv3	MNIST-Det	100	50	0.5	300×300	8	4	35,000	300×300
	EMNIST-Det	100	100	0.5	320×320	8	4	50,000	320×320
	Pascal VOC	200	150	0.5	608×608	15	4	18,750	$[(320, 320),\ (608, 608)]$
	BDD100k	1,100	700	0.5	608×608	8	4	160,000	$[(320, 320),\ (608, 608)]$
FRCNN	MNIST-Det	100	50	0.7	300×300	8	4	30,000	300×300
	EMNIST-Det	100	100	0.7	320×320	8	4	30,000	320×320
	Pascal VOC	100	100	0.7	1000×600	15	4	18,750	1000×600
	BDD100k	1,250	750	0.7	1000×600	8	4	170,000	1000×600
RetinaNet	MNIST-Det	100	50	0.5	300×300	8	4	25,000	300×300
	EMNIST-Det	225	125	0.5	300×300	8	4	35,000	300×300
	Pascal VOC	550	350	0.5	1000×600	8	4	60,000	1000×600
	BDD100k	1,000	500	0.5	1000×600	8	4	175,000	1000×600

Object Detection Models. In our experiments we used a YOLOv3 detector with the standard Darknet53 backbone on VOC and BDD. In our down-scaled version we replace the backbone with an analogous architecture working on the same resolutions, such that all strides remain the same and the detection mechanism works identically. Table 5.4 shows the configuration comparison between the standard Darknet53 architecture and our adapted version (here, called Darknet20) which significantly reduces depth and the number of feature channels extracted. All other model and data augmentation configurations remain unchanged. For Faster R-CNN we use a ResNet101 backbone on VOC and BDD while for RetinaNet, we use a ResNet50. Here, we use a Feature Pyramid Network (FPN) with $[256, 512, 1024, 2048]$ channels. Both are down-scaled to ResNet18 backbones with $[64, 128, 256, 512]$-channel FPN to accelerate training and inference.

For all architectures, we insert dropout layers between the two last layers (convolutional layers at all stages for YOLOv3 and RetinaNet and fully connected for Faster R-CNN). For all experiments involving sampling, i.e., MC Dropout and Mutual Information experiments, we use dropout rates of 0.5 and perform 10 forward passes.

Active Learning Parameters. Table 5.5 gives an overview of chosen hyperparameters for the AL cycle for all datasets and architectures. $\lvert\mathcal{U}_{init}\rvert$ stands for the number of initially annotated images, $\lvert\mathcal{Q}\rvert$ for the size of the selected query, ϵ_s for the score threshold for query inference (hence, determining instance-wise uncertainty) and T for number of AL steps. The hyperparameters for training are the batch size, which is always 4, the number of training iterations, and the image resolution. It follows from table 5.5 (particularly, $\lvert\mathcal{U}_{init}\rvert$, $\lvert\mathcal{Q}\rvert$ and T) that all architectures considered in our experiments need the fewest images for our sandbox datasets MNIST-Det and EMNIST-Det to reach the 90% max performance mark. Therefore, for the sandbox datasets we chose $\lvert\mathcal{U}_{init}\rvert$ and $\lvert\mathcal{Q}\rvert$ smaller than for VOC and BDD. Moreover, the sandbox datasets have lower image resolutions, which leads to faster training and inference times, even if occasionally the training iterations are lower for VOC, e.g., , for Faster R-CNN and YOLOv3. Apart from the latter case, the most iterations to obtain convergence in the training processes are needed for BDD with up to

175,000. The score threshold of 0.5 for YOLOv3 and RetinaNet, and 0.7 for Faster R-CNN was determined by ablation studies for EMNIST-Det and then adopted for the other datasets. For all query methods, we incorporate a class-weighting (the same as in [8]) for computing instance-wise uncertainty scores.

5.4.2 Benchmark Results

We first investigate differences in AL results with respect to the datasets where we fix the architecture. The comparison uses the YOLOv3 detector on two standard OD benchmarks (VOC and BDD) and our EMNIST-Det dataset. Our comparison includes the five query baselines described in section 5.3.4. We obtain AL curves averaged over four random initialization seeds and evaluated in terms of queried images as well as in terms of queried object instances, respectively. Figure 5.5 shows the test performance curves with shaded regions indicating point-wise standard deviations. The left panels show performance according to queried images while the right panels show the same curves but according to queried instances. We observe that the uncertainty-based query strategies tend to consistently outperform the Random query in image-wise evaluation. However, when regarding the number of queried bounding boxes, the situation is far less clear. For EMNIST-Det, the difference between the Random and the uncertainty-based queries decreases substantially, such that only a marginal difference is visible. In VOC and BDD, the Random baseline falls roughly somewhere in-between the uncertainty baselines in instance-wise evaluation. This indicates that greedy acquisition of images with the highest sum of uncertainty tends to query images with a large amount of ground truth boxes. Obtaining many ground truth signals improves detection performance in these cases, while the query of large amounts of boxes gives rise to a higher annotation cost on the right panels. From this observation, we conclude that comparing AL curves based only on the number of acquired training images gives an incomplete impression of the method and the associated annotation costs. In addition to the image-wise evaluation, an instance-wise evaluation should be considered. Note also, that the AL curves for EMNIST-Det and VOC have a smoother progression than those for BDD. We attribute this finding to the fact that BDD is a far more complicated detection problem which includes a large amount of small objects. However, the fluctuations in the AL curves on BDD tend to average out as we consider the AUC of the AL curve as evaluation metric. This becomes clear in light of the results in section 5.4.3, where we study the generalization across datasets in terms of the AUC of the AL curve.

In table 5.6 we show numerical benchmark results. For each detector to reach 90% of the maximum performance, the table shows the number of queried images required and respectively the number of bounding boxes for each method. These numbers were acquired as the average of four runs. We see the rankings per experiment which often favor the Entropy baseline, however, the overall rankings are rather unstable throughout the table. Note in particular, that for the arguably the hardest detection problem under consideration, BDD, the Random baseline beats the Mutual Information baseline for the YOLOv3 detector. In the analog setting for Faster R-CNN the image-wise margin of the Mutual Information becomes slim. This observation also holds for instance-wise evaluation and is even more pronounced where in six cases, Random beats an informed query baseline. For

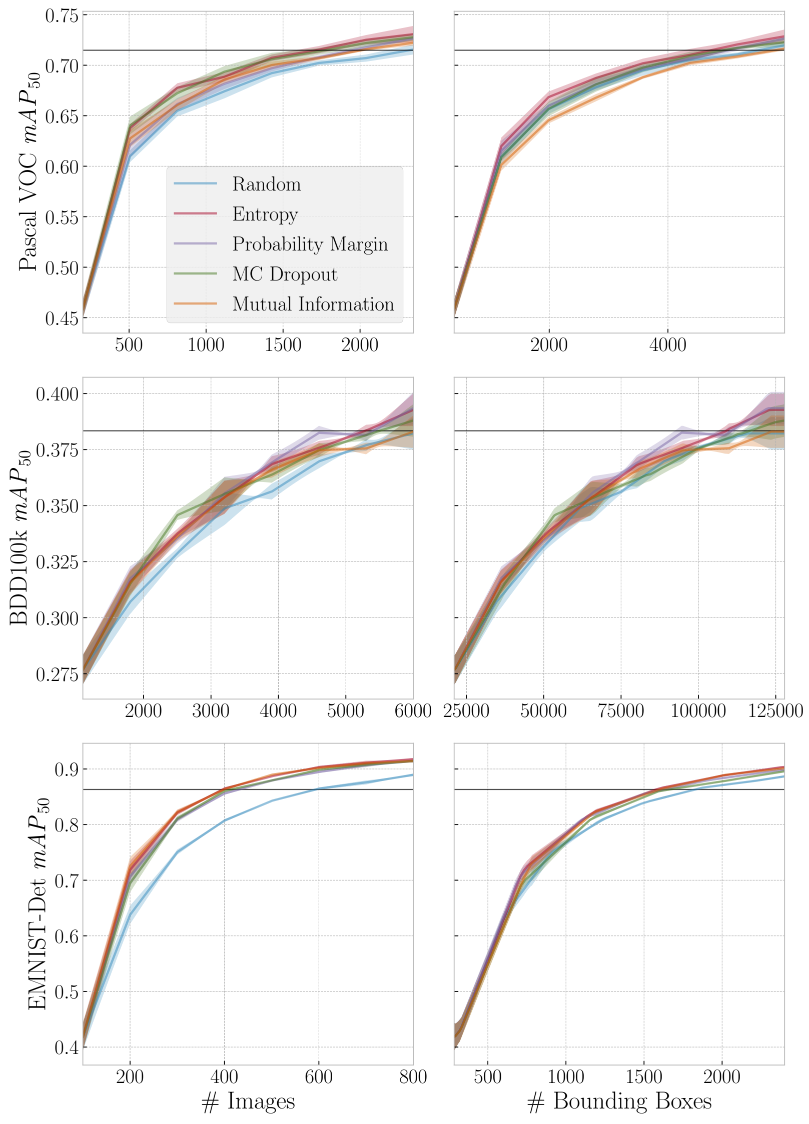


Figure 5.5: Comparison of YOLOv3 AL curves on three different datasets (Pascal VOC, BDD100k, EMNIST-Det). *Left*: image-wise evaluation, *right*: box-wise evaluation.

Table 5.6: Amount of queried images and bounding boxes necessary to cross the 90% performance mark during AL when using sum aggregation. Lower values are better. Bold numbers indicate the lowest amount of data per experiment and underlined numbers are the second lowest.

		# queried images				# queried bounding boxes			
		MNIST-Det	EMNIST-Det	Pascal VOC	BDD100k	MNIST-Det	EMNIST-Det	Pascal VOC	BDD100k
YOLOv3	Random	327.9	595.6	2236.8	5871.2	1079.1	1825.3	5344.2	116362.1
	Entropy	**245.5**	**398.8**	1732.8	5389.3	**1004.9**	**1583.0**	**4695.4**	110694.9
	Prob. Margin	256.2	429.0	1858.5	**4895.2**	1013.7	1617.1	4787.6	**100376.3**
	MC Dropout	256.3	416.2	**1679.4**	5200.5	1115.3	1671.6	4875.1	110427.6
	Mutual Inf.	249.8	399.5	1884.2	5912.9	1061.9	1602.7	5527.0	125050.1
Faster R-CNN	Random	450.0	843.4	1293.7	6434.3	2140.0	2891.7	3125.2	129219.0
	Entropy	**384.5**	**561.6**	**1030.6**	5916.7	**1608.4**	**2156.4**	**2707.0**	123008.6
	Prob. Margin	408.7	626.2	1036.5	**5761.6**	1622.9	2285.1	2711.6	**117889.3**
	MC Dropout	390.5	647.4	1127.5	6296.4	1818.1	2773.8	3624.7	130533.8
	Mutual Inf.	395.3	572.6	1080.2	6385.7	1695.6	2235.3	3026.5	132855.7
RetinaNet	Random	390.3	950.4	2555.4	3616.2	1283.8	2957.7	6220.0	69842.0
	Entropy	**288.6**	**687.7**	**1961.2**	**2866.5**	1292.0	2708.6	**5421.6**	64939.7
	Prob. Margin	310.8	733.9	2087.3	2901.5	1277.5	2721.5	5445.6	64794.9
	MC Dropout	293.3	749.6	2745.3	3027.7	1317.4	2926.4	7047.9	62395.5
	Mutual Inf.	289.6	719.0	2881.9	3124.9	**1248.0**	**2677.4**	7389.0	**61712.9**

the RetinaNet architecture, we again notice striking ranking differences between image- and instance-wise evaluation. Particularly, instance-wise evaluation has rather irregular method rankings between datasets for the RetinaNet detector which are, however, also reflected in our AUC evaluation discussed in table 5.8 in the following paragraph. These results yield further evidence that the consideration of only a single evaluation metric for active learning performance is insufficient. In light of the discussion in section 5.3.5, we conclude that in order to assess the viability of a query strategy, AL curves should be viewed from both angles: performance over number of images and over number of bounding boxes queried.

Benchmark Results in Terms of mAP$_{50}$ ***and*** AUC***.*** Table 5.7 shows the mAP$_{50}$ achieved after the final query for each method and detector-dataset constellation in the style of table 5.6. For each AL curve, the final performance (most queried images allowed according to table 5.5) is independent of an image- or instance-wise point of view. Overall, the Entropy baseline is consistently among the best two methods, however, the overall rankings show a medium degree of variance across datasets and across detectors especially when regarding instance-wise evaluation in table 5.6. Comparing vertical sections through AL curves shows overall roughly similar behavior as the results in table 5.6 (horizontal sections), however, we observe differences in the method rankings in terms of amount of data queried vs. final detection performance (e.g., Faster R-CNN on the MNIST-Det dataset).

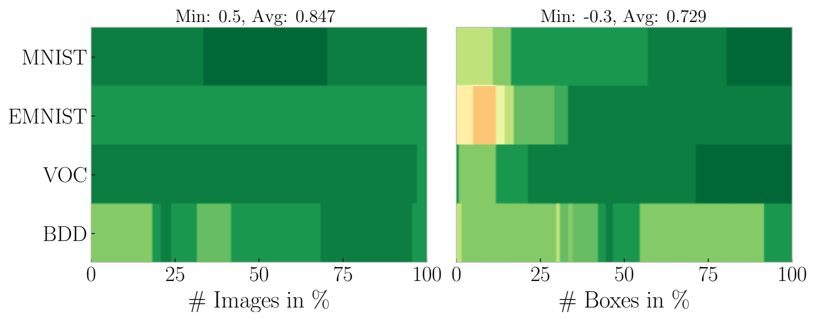
In table 5.8 we collect the values of the AUC metric. Note, that the AUC metric scales with the t-axis, i.e., results between different datasets can only be compared qualitatively. The same is true for comparisons between image- and instance-wise evaluations. When comparing with table 5.6, we see a high degree of ranking similarity with the amount of data required to cross the 90%-mark in both, image- and instance-wise evaluation. We conclude with previous findings on the rank correlations, that even in a rather late evaluation (when some fixed reference mark in performance has already been crossed), the AUC

Table 5.7: Mean average precision results per query method for maximal amount of images selected; higher values are better. Bold numbers indicate the highest performance per experiment and underlined numbers are the second highest.

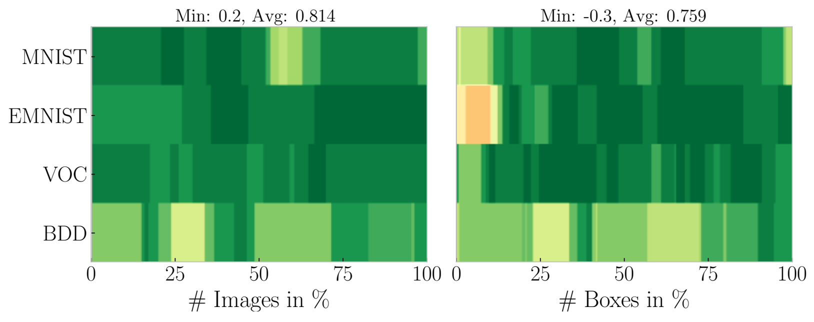
		MNIST-Det	EMNIST-Det	Pascal VOC	BDD100k
YOLOv3	Random	89.28	88.95	72.10	38.22
	Entropy	**91.53**	**91.75**	**73.08**	39.28
	Prob. Margin	91.03	91.42	72.65	**39.35**
	MC Dropout	91.08	91.42	72.78	38.80
	Mutual Inf.	91.05	91.42	72.22	38.30
Faster R-CNN	Random	83.20	83.92	74.67	47.02
	Entropy	85.27	**87.15**	75.10	48.30
	Prob. Margin	85.00	86.62	**75.40**	**48.35**
	MC Dropout	**85.28**	86.05	74.25	47.27
	Mutual Inf.	85.27	87.07	74.35	47.27
RetinaNet	Random	82.95	83.70	69.58	43.20
	Entropy	**85.35**	**86.15**	**71.80**	**43.97**
	Prob. Margin	84.38	85.78	71.55	43.75
	MC Dropout	85.02	85.60	68.17	43.50
	Mutual Inf.	85.33	86.10	68.12	43.48

Table 5.8: Area under AL curve results per query method for maximal amount of data selected; higher values are better. Bold numbers indicate the highest AUC per experiment and underlined numbers are the second highest.

		# queried images				# queried bounding boxes			
		MNIST-Det	EMNIST-Det	Pascal VOC	BDD100k	MNIST-Det	EMNIST-Det	Pascal VOC	BDD100k
YOLOv3	Random	290.9	543.3	1645.1	1691.9	1503.8	2438.7	4274.3	37327.3
	Entropy	**299.6**	577.4	**1685.2**	1728.8	**1519.0**	**2491.4**	**4324.6**	37816.6
	Prob. Margin	298.1	571.9	1664.5	**1733.6**	1516.2	2485.3	4303.4	**37931.1**
	MC Dropout	298.3	571.8	1684.9	1727.9	1503.8	2465.5	4284.6	37550.7
	Mutual Inf.	299.1	**578.0**	1666.8	1716.1	1509.4	2482.6	4233.2	37514.5
Faster R-CNN	Random	273.0	542.6	1207.9	2220.1	1448.6	2643.7	3121.5	47767.9
	Entropy	**281.3**	**567.1**	**1237.6**	2266.7	**1465.2**	**2703.6**	**3168.8**	48167.8
	Prob. Margin	279.7	561.3	1236.4	**2274.7**	**1465.2**	2692.5	3168.3	**48449.2**
	MC Dropout	280.5	557.0	1227.0	2247.3	1451.4	2619.7	3052.8	47718.9
	Mutual Inf.	280.2	566.6	1230.5	2247.8	1457.5	2697.4	3123.8	47714.8
RetinaNet	Random	270.4	677.5	1743.6	1380.5	1438.2	2896.9	4614.9	33141.5
	Entropy	**279.1**	**701.9**	**1819.7**	**1421.6**	1446.2	2932.4	4714.6	33431.4
	Prob. Margin	276.8	696.5	1802.8	1414.7	1445.6	2928.4	**4722.5**	33339.5
	MC Dropout	279.0	696.3	1730.6	1406.2	1445.6	2915.8	4535.9	33452.9
	Mutual Inf.	278.5	699.3	1734.7	1400.8	**1449.1**	**2940.7**	4543.9	**33495.2**



(a) AUC



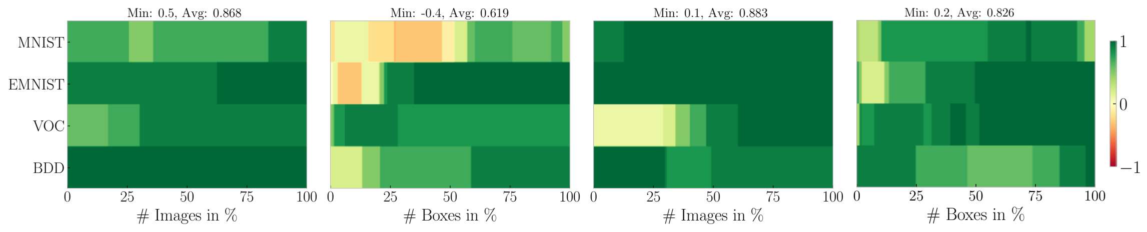
(b) mAP_{50}

Figure 5.6: Curves of rank correlations between (a) the cumulative AUC and (b) the final rankings at the 90% max performance mark for the YOLOv3 object detector.

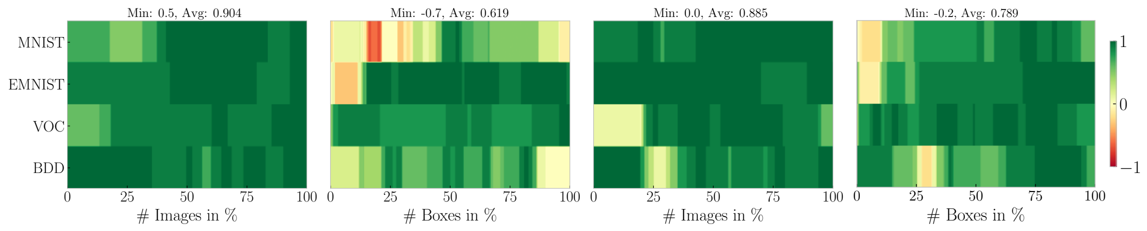
shows more similarity with the 90% ranking than raw detection performance (table 5.7). For instance, compare the Faster R-CNN row from table 5.6 with the corresponding row in table 5.8.

5.4.3 Generalization of Sandbox Results

Stability of the* AUC *Metric. Instead of evaluating the pure performance at each AL step we have proposed in section 5.3.5 to compute the corresponding AUC as a more robust metric of AL performance. With respect to the final method ranking at some fixed reference mark, we compute Spearman rank correlations with the mAP_{50} metric at each point t. We compare these with the analogous correlations with the respective AUC at each point. Figure 5.6 shows intensity diagrams representing the rank correlations both, in terms of image-wise and instance-wise evaluation. The reference mark we chose is 90% of the maximum mAP_{50} obtained (see table 5.2). The t-axes are normalized to the maximum number of images, resp. bounding boxes queried, and the color indicates the Spearman correlation of the rankings. Red represents negative correlation (i.e., partial to full inversion of the observed ranking) while green means positive correlation (i.e., a higher degree of similarity of the rankings). In fig. 5.6 both, mAP_{50} and AUC show overall high correlation with the method ranking, especially towards the end of the curves. We see that the correlations for AUC fluctuates far less. Moreover, the average correlation across entire AL curves tends to be larger for the AUC metric than mAP_{50}. Note that the final



(a) AUC

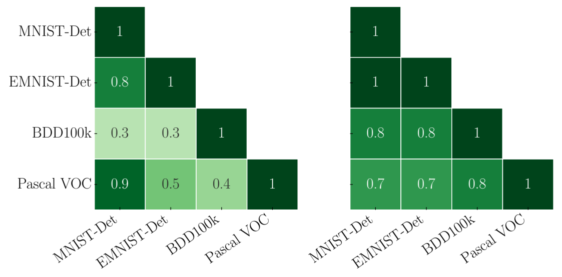


(b) mAP$_{50}$

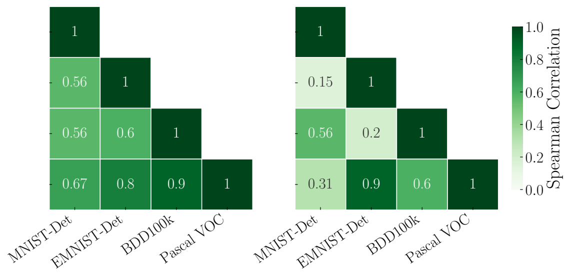
Figure 5.7: Curves of rank correlations between (a) the cumulative AUC and (b) the final rankings at the 90% max performance mark. *Left:* RetinaNet, *Right:* Faster R-CNN.

ranking of either method does not need to be perfectly correlated with the 90% max performance ranking for two reasons. Firstly, the latter does not take into consideration early performance gains and secondly, the 90% max performance ranking is a horizontal section through the curves while mAP$_{50}$ and AUC are vertical sections. For a more quantitative evaluation, we show minimal and average correlation over all curves in fig. 5.6. We observe overall high averages of upwards of 0.72 throughout the curves. The minimum AUC correlation tends to be higher than the minimum mAP$_{50}$ correlation. We conclude that AUC tends to be highly correlated with the 90% max performance ranking and is more stable with respect to t than mAP$_{50}$. Ranking correlations for mAP$_{50}$ and AUC for RetinaNet and Faster R-CNN, see fig. 5.7, tend to show similar behavior as for the YOLOv3 detector. The AUC fluctuates less than the mAP$_{50}$ and both metrics show overall high correlation with the 90% max performance ranking. For the image-wise evaluation, both metrics have correlations greater or equal than 0.5. In the instance-wise evaluation, on the other hand, the correlations increase only gradually. Even though the average correlations are identical for the mAP$_{50}$ and the AUC, the latter is clearly more stable with respect to the 90% max performance ranking and has a higher minimum correlation.

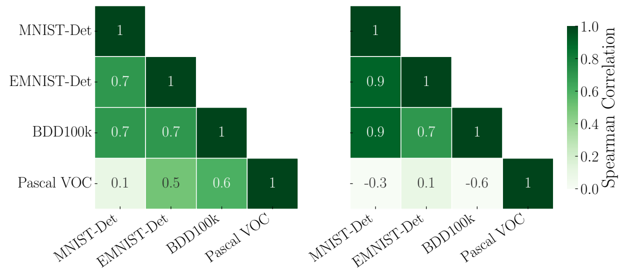
Cross-Dataset Ranking Correlations. Next, we study how comparable AL experiments are between the sandbox setting and full-complexity problems, i.e., VOC and BDD. To this end, we consider the cross-dataset correlations of the AUC score when fixing the detection architecture. Figure 5.8 shows correlation matrices for image- and instance-wise evaluation. When comparing to the VOC-BDD correlations, the MNIST-Det and EMNIST-Det method rankings tend to be similarly correlated with either one. Consider, for instance, the image-wise correlations on VOC. For the YOLOv3 detector, both MNIST-Det and EMNIST-Det have higher correlation with VOC than BDD does. Mean-



(a) YOLOv3



(b) Faster R-CNN



(c) RetinaNet

Figure 5.8: Ranking correlations between AUC values for different detectors. *Left*: Image-wise; *Right*: Instance-wise evaluation.

while, the BDD-MNIST-Det (respectively, EMNIST-Det) correlation is a bit lower than VOC-BDD but similar. For Faster R-CNN, the correlation VOC-BDD is large with 0.9, however, both EMNIST-Det-VOC and EMNIST-Det-BDD have correlations of 0.6 and 0.8 which are reasonably high. For all correlations of VOC with BDD, replacing either one of the datasets with MNIST-Det or EMNIST-Det results in similar correlation. From this, we conclude that comparing methods in the simplified setting yields a similar amount of information about the relative performances of AL as the full-complexity setting. The image-wise comparison for RetinaNet shows the highest correlation of 0.7 when comparing MNIST-Det, EMNIST-Det and BDD respectively. VOC has the highest correlation with BDD of 0.6, but the correlation with EMNIST-Det is similar with 0.5. No conclusions can be drawn between the rankings of MNIST-Det and VOC due to the low correlation of 0.1. In the instance-wise comparison, MNIST-Det has very high correlations of 0.9 with EMNIST-Det and BDD, and the comparability of the rankings of EMNIST-Det and BDD is also given by a correlation of 0.7. However, it is again noticeable that VOC is hardly comparable with any other dataset. This could be attributed to some dataset characteristics. On one hand, we observed a significant number of missing labels when looking at the VOC data (see fig. 5.9). On the other hand, the instance sizes of BDD and EMNIST-Det/MNIST-Det seem to be rather comparable as opposed to the instance sizes in VOC.

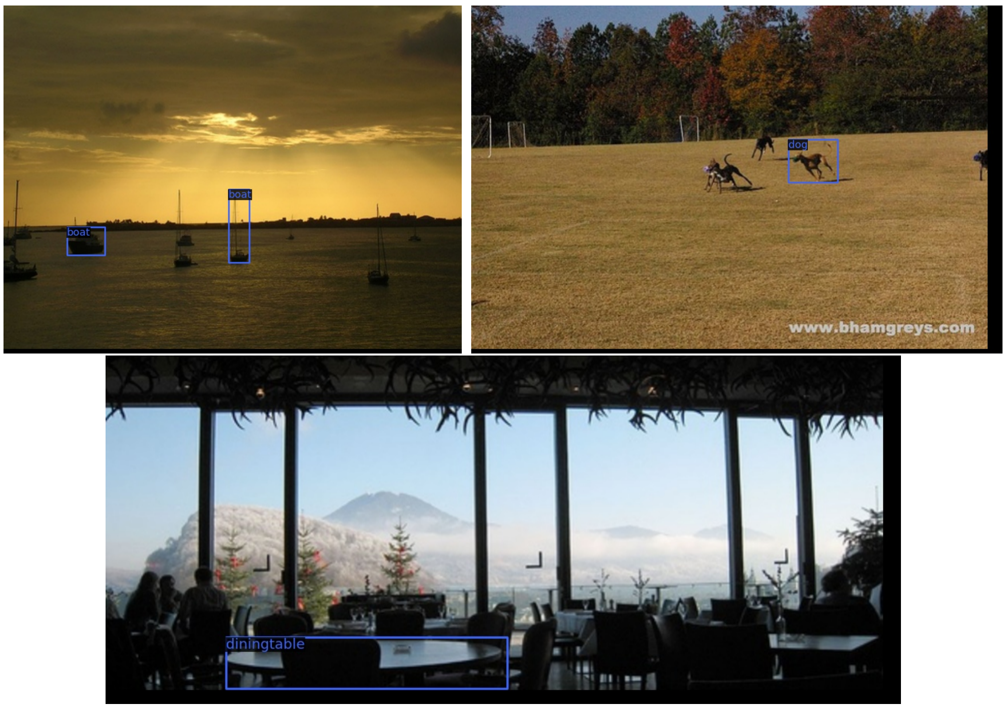


Figure 5.9: Annotation examples showing all labeled bounding boxes on each image (VOC [38] test dataset).

5.4.4 Image-Aggregation Methods

Figure 5.10 shows test mAP$_{50}$ for different image aggregations, namely sum, average and maximum, for the RetinaNet on EMNIST-Det. The left panels show mAP$_{50}$ scores as a function of the number of queried images while the right panels show mAP$_{50}$ scores as a function of the number of queried instances. For all four uncertainty baselines, the sum dominates the maximum and the maximum dominates the average in the image-wise evaluation. Nevertheless, the average remains consistently better than Random, except for Mutual Information, where both curves are almost on par. A clearly different course is obtained when considering the instance-wise evaluation. For the same number of images queried, the sum prefers images with many boxes, whereas the average queries images with even fewer boxes than the Random baseline. In terms of performance, the average outperforms the sum and maximum, which are tied, and the Random baseline for Entropy and Probability Margin. For Mutual Information, the average and sum are best, whereas for MC Dropout all curves are hardly distinguishable from each other. Comparable behaviors could also be observed on the other architectures and datasets.

Investigations of the kind presented here under normal conditions (using a full-scale standard object detector and a benchmark dataset) would require weeks of compute time and yield valuable information on sensitive parameters for querying. Using our sandbox environment makes extensive ablation studies of hyperparameters possible within a few days.

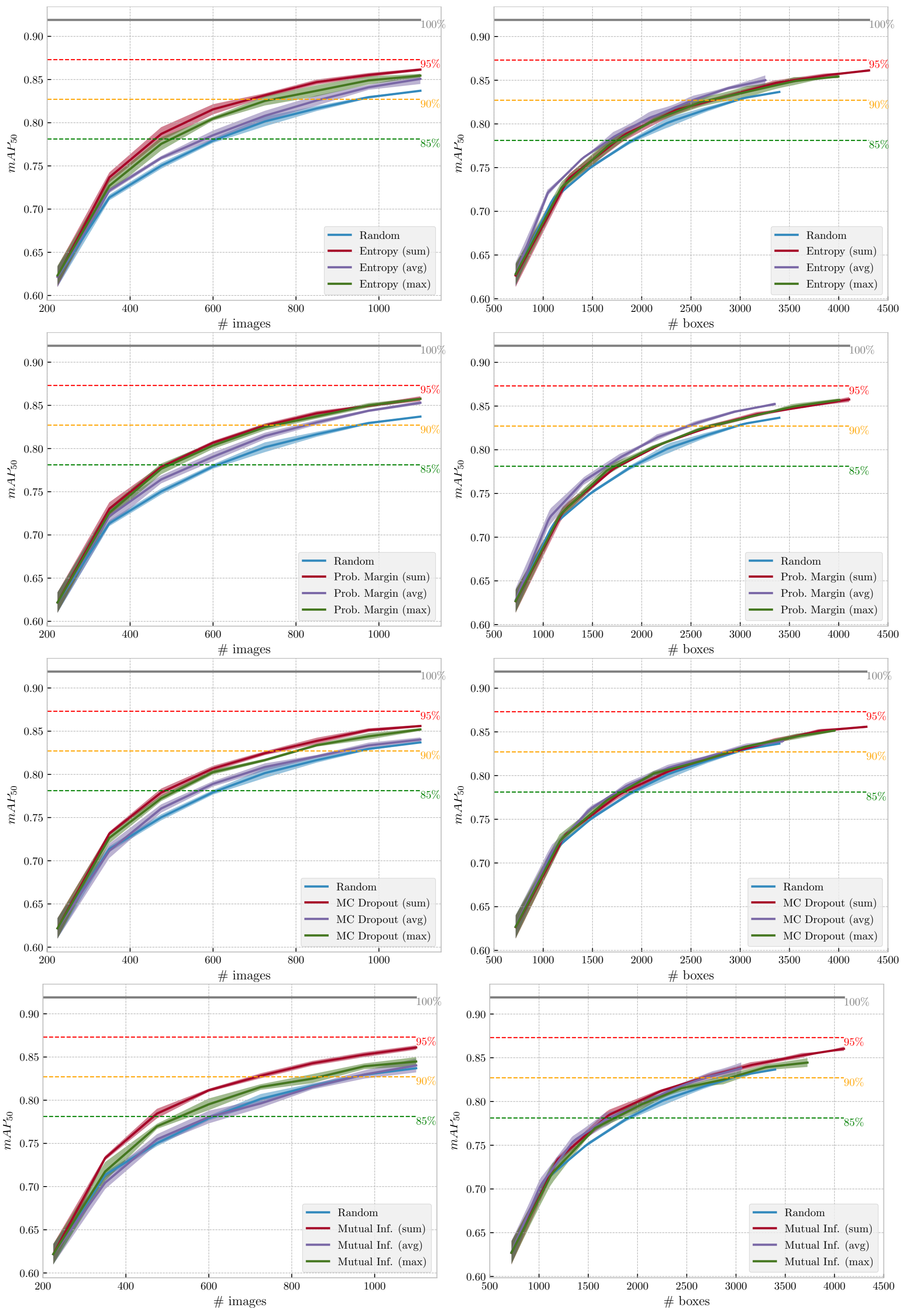


Figure 5.10: Ablation study on the aggregation method for RetinaNet on the EMNIST-Det dataset.

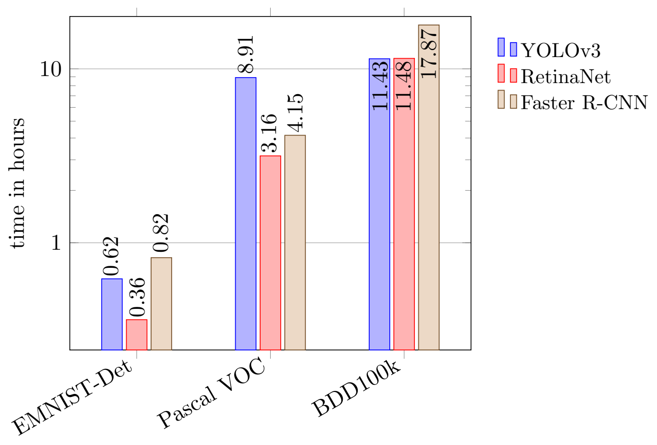


Figure 5.11: Utilized time for one AL step (training until convergence + evaluating the query) for investigated settings in hours.

5.4.5 Computational Runtime

AL for advanced image perception tasks such as OD tends to be highly time intensive, compute-heavy and energy consuming. This is due to the fact that at each AL step the model should be guaranteed to fit to convergence and there are multiple steps of (ideally) several random seeds to be executed. Figure 5.11 shows the time per AL step used in our setting when run on a Nvidia Tesla V100-SXM2-16GB GPU with a batch size of four. Note that the time-axis is scaled logarithmically, so the experiments on EMNIST-Det are always faster by at least half an order of magnitude. The training of YOLOv3 on VOC does not start from MS COCO-pretrained weights (like YOLOv3+BDD) since the two datasets VOC and MS COCO are highly similar and VOC consists of subclasses of the MS COCO dataset. In this case, we opt for ImageNet-pretrained [157] backbone weights just like for the other detectors. We overall save time up to a factor of around 14 for VOC and around 32 for the BDD dataset. Translated to AL investigations, this means that the effects of new queries can be evaluated within half a day on a single Nvidia Tesla V100-SXM2-16GB.

5.5 Conclusion: Rapid Prototyping of Active Learning for Object Detection

In this chapter, we investigated the possibility of simplifying the active learning setting in deep object detection in order to accelerate development and evaluation time. We found that active learning curves do not trivially generalize from one (dataset, detector, evaluation metric)-constellation to another and that significant qualitative differences can be observed. Even though this prohibits the prediction of final method rankings we observe that the rank correlations are primarily positive and stay high throughout the active learning experiment. In our evaluation, we also included a more direct measurement of annotation effort in counting the number of queried instances in addition to the number of queried images. Meanwhile, we can save more than an order of magnitude in total compute time due to the down-scaling of the detection architecture and simplifying the dataset complexity. Our environment allows for consistent benchmarking of active learning methods in a unified object detection framework, thereby improving transparency.

6 Label Error Identification For Object Detection Datasets

In this chapter we present a method to automatically detect annotation errors in object detection datasets by regarding instance-wise loss values. The presented contents are in large parts taken word-for-word from [162].

6.1 Introduction: Noisy Labels and the Detection of Label Errors

Nowadays, the predominant paradigm in computer vision is to learn models from data. The performance of the model largely depends on the amount of data and its quality, i.e., the diversity of input images and label accuracy [40, 71, 75, 79, 87]. DNNs are particularly data hungry [173]. In this chapter, we focus on the case of object detection where multiple objects per scene belonging to a fixed set of classes are annotated via bounding boxes [38, 146].

In many industrial and scientific applications, the labeling process consists of an iterative cycle of data acquisition, labeling, quality assessment, and model training. Labeling data is costly, time-consuming and error-prone, e.g., due to inconsistencies caused by multiple human annotators or a change in label policy over time. Therefore, at least a partial automation of the label process is desirable. One research direction that aims at this goal is automated label error detection [35, 130, 154]. The extent to which noisy labels affect the model performance is studied by [10, 191]. Wu et al. [191] observes that the model is able to tolerate a certain amount of missing annotations in training data without losing too much performance on Pascal VOC and Open Images V3 test sets. In contrast, Büttner et al. [10] show that inaccurate labels in terms of annotations size in training data yields to significant decrease of test performance for calculus detection on bitewing radiographs. Other methods model label uncertainty [121, 147] or improve robustness with respect to noisy labels [13, 44, 97, 214].

Up to now, automated detection of label errors has received less attention. There exist some works on image classification datasets [130, 131, 175], one work on semantic segmentation datasets [154] and some works for object detection [67, 83]. Label errors may affect generalization performance, which makes their detection desirable [131]. Furthermore, there is economical interest in improving and accelerating the review process by partial automation.

Here, we study the task of label error detection in object detection datasets by (a) introducing a benchmark and (b) developing a detection method and compare it against four baselines. We introduce a benchmark by simulating label errors on the BDD100k [209] and EMNIST-Det [146] dataset. The latter is a semisynthetic dataset consisting of EMNIST letters [26] pasted into MS COCO [102] images of which we expect to possess highly accurate labels. The types of label errors that we consider are missing labels (*drops*), correct localization but wrong classification (*flips*), correct classification but inaccurate localization (*shifts*), and labels that actually represent background (*spawns*). We address the detection of these errors by a novel method based on monitoring instance-wise object detection loss. We study the effectiveness of our method in comparison to four baselines. Then, we demonstrate for commonly used object detection test datasets, such as BDD100k, MS COCO, Pascal VOC [38] and KITTI [46], and also for a proprietary dataset on car part detection by the company ControlExpert. Our method detects real label errors by reviewing moderate sample sizes of 200 images per dataset. Our contributions can be summarized as follows:

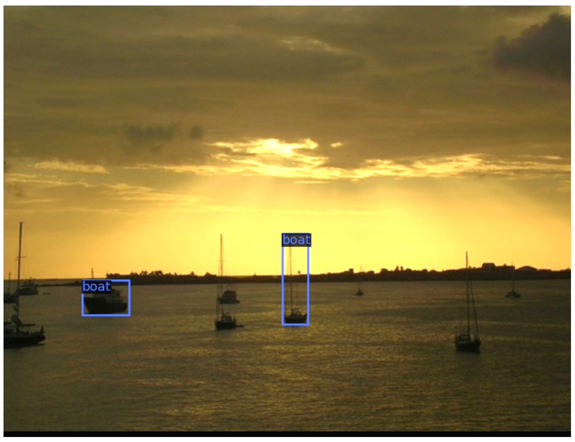
- We introduce a novel method based on the instance-wise loss for detecting label errors in object detection.
- We introduce a benchmark for identifying four types of label errors on BDD100k and EMNIST-Det.
- We apply our method to detect label errors in commonly used and proprietary object detection datasets and manually evaluate the error detection performance for moderate sample sizes.

To contribute to future development of label error detection methods and potentially cleaning up object detection datasets, we provide an implementation of our benchmark, method and baselines as well as label files that include simulated label errors and model checkpoints that allow the reproduction of our results[47].

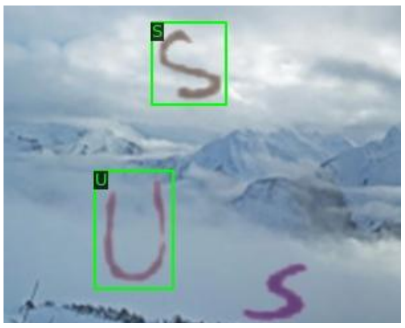
6.2 Related Work: Identification of Label Errors in Classification, Object Detection and Semantic Segmentation

The influence of noisy labels in training as well as in the test data is an active and current research topic. The labels for commonly used image classification datasets are noisy [131]

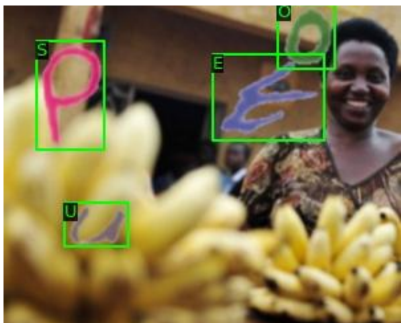
[47] `https://github.com/schubertm/identifying_label_errors_in_od`



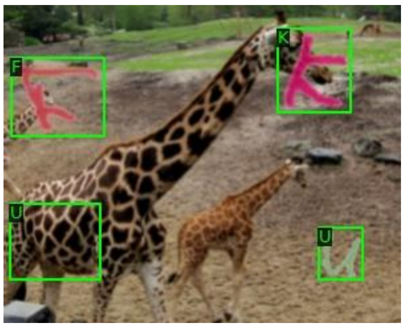
(a) Pascal VOC 2007 Error



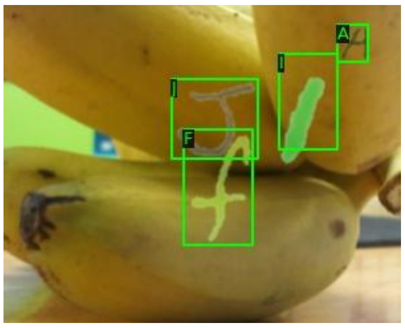
(b) Label Drop



(c) Label Flip



(d) Label Spawn



(e) Label Shift

Figure 6.1: (a) Example image from the Pascal VOC 2007 test dataset with two labeled boats marked by the blue boxes and multiple unlabeled boats. (b) – (e) Examples of the different types of simulated label errors. The images are from the EMNIST-Det test dataset [146].

and this also applies to object detection. Figure 6.1 (a) shows an image from the Pascal VOC 2007 test dataset containing just two labeled boats, but clearly more can be seen.

For the task of image classification, some learning methods exist that are more robust to label noise [49, 56, 63, 77, 130, 143, 186, 195, 213]. Also the task of label error detection has been tackled before [21, 131] and theoretically underpinned [130]. [21] filter whole samples with noisy labels, but individual label errors are not detected. Northcutt et al. present label errors in image classification datasets and study to which extent they affect benchmark results [131] followed by the introduction of the task of label error detection [130]. The latter introduces a confident learning approach, assuming that the label errors are image-independent. Then, the joint distribution between the noisy and the true labels with class-agnostic label uncertainties is estimated and utilized to find label errors. This method allows finding label errors on commonly used image classification (i.e., MNIST or ImageNet) and sentiment classification datasets, resulting in improved model performance by re-training on cleaned training data. This line of work has been recently extended to the task of multi-label classification [175], where a single object is shown per image but may carry multiple labels.

For object detection, the influence of noisy training labels on the model performance has been studied [191, 194] with the resulting observation that the model is reasonably robust when dropping labels. To counter label errors in object detection, methods that model label uncertainty [121, 147] or more robust object detectors have been developed [10, 44, 78, 97, 194, 214]. Büttner et al. [10] simulate label errors and introduce a co-teaching approach for more robust training with noisy training data. For the task of label error detection, Koksal et al. [83] simulate different types of label errors in video sequences. Predictions and labels of consecutive frames are compared and then manually reviewed to eliminate erroneous annotations. Hu et al. [67] introduce a probability differential method (PD) to identify and exclude annotations with wrong class labels during training.

For semantic segmentation, a benchmark is introduced by Rottmann and Reese [154] to detect missing labels. For this purpose, uncertainty estimates are used to predict for each false positive connected component whether a label error is present or not. Detection is performed by considering the discrepancy of the given, potentially noisy, label and the corresponding uncertainty estimate.

This chapter introduces the first benchmark with four types of label errors for label error detection methods on object detection datasets as well as a label error detection method (that detects all four types of label errors) and a number of baselines. For our benchmark, we randomly simulate four types of label errors and detect them simultaneously with a new and four baseline methods, including PD. In our method, the discrepancy between the prediction or expectation of the network and the actual labels is used to find label errors. This discrepancy is determined by the classification and regression loss from the first and second stage of the detector. This allows to find not only simulated but also real label errors on commonly used object detection test datasets.

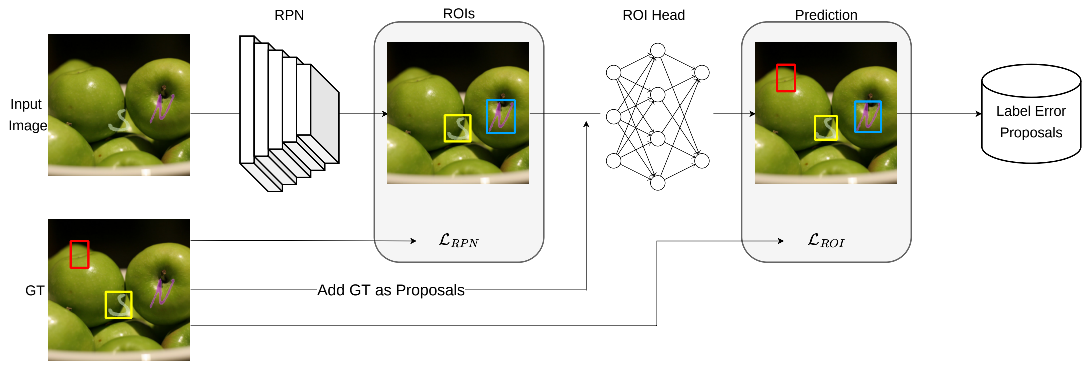


Figure 6.2: Visualization of our instance-wise loss method for detecting label errors. The red label indicates a *spawn*, the blue one a *drop* and the yellow one a correct label.

6.3 Methods: Loss-based Label Error Detection, Benchmarking and Evaluation

In this section we describe our label error benchmark as well as the setup and evaluation for real label errors on commonly used object detection test datasets and a proprietary dataset. We describe which datasets are used, which types of label errors are considered and the way we simulate label errors inspired by observations that we made in commonly used datasets and by related work. We then introduce our detection method as well as four additional baseline methods. This is complemented with evaluation metrics used to compare the methods with each other on our label error benchmark and the evaluation procedure for commonly used test datasets where we manually review the findings of our method for moderate sample sizes.

6.3.1 Benchmarking Label Error Detection

Datasets. For our benchmark we use the semisynthetic EMNIST-Det dataset and the BDD100k dataset, in the following referred to as BDD. EMNIST-Det consists of 20,000 training and 2,000 test images. To have the best possible labels for BDD, we filter the training and validation split, such that we only use daytime images with clear weather conditions. This results in 12,454 training images and the validation split is split into equally-sized test and validation sets, each consisting of 882 images.

Simulating Label Errors. We consider four different types of label errors: missing labels (*drops*), correct localization but wrong classification (*flips*), correct classification but inaccurate localization (*shifts*), and labels that actually represent background (*spawns*). Any dataset

$$\mathcal{D} := \{(\boldsymbol{x}_1, \overline{y}_1), \dots, (\boldsymbol{x}_{|\mathcal{D}|}, \overline{y}_{|\mathcal{D}|})\} \tag{6.1}$$

where $\overline{y}_k = \{\mathrm{b}_k^1, \dots, \mathrm{b}_k^{N(k)}\}$ for $k \in [|\mathcal{D}|]$ are all $N(k)$ bounding box annotations

$$\mathrm{b}_k^j = (\mathrm{x}_k^j, \mathrm{y}_k^j, \mathrm{w}_k^j, \mathrm{h}_k^j, \kappa_k^j), \qquad j \in [N(k)] \tag{6.2}$$

is equipped with a set of $G := \sum_{k\in[|\mathcal{D}|]} N(k)$ labels, i.e.,

$$\mathcal{B} := \{\mathrm{b}_k^j : \quad k \in [|\mathcal{D}|], j \in [N(k)]\}. \tag{6.3}$$

We reorder all boxes such that we identify $\mathcal{B} = \{\mathrm{b}^i : \quad i \in [G]\}$ and denote the set of bounding box indices by $\mathcal{I}_{\mathcal{B}} := [G]$. We now describe all types of label errors applied to $\mathcal{B}$, and we make the assumption that a single label is only perturbed by one type of label error instead of multiple types. We choose a parameter $\gamma \in [0,1]$ representing the relative frequency of label errors.

For **dropping** labels, we randomly choose a subset $\mathcal{I}_{\mathrm{drop}}$ of $\mathcal{I}_{\mathcal{B}}$ with cardinality $|\mathcal{I}_{\mathrm{drop}}| = \lfloor \frac{\gamma}{4} \cdot G \rfloor$. We drop all labels $\mathcal{B}_{\mathrm{drop}} = \{\mathrm{b}^i : i \in \mathcal{I}_{\mathrm{drop}}\}$ and denote $\mathcal{I}_{\setminus \mathrm{drop}} = \mathcal{I} \setminus \mathcal{I}_{\mathrm{drop}}$. Analogously, $\mathcal{B}_{\setminus \mathrm{drop}} = \mathcal{B} \setminus \mathcal{B}_{\mathrm{drop}}$.

For **flipping** class labels, we randomly choose a subset $\mathcal{I}_{\mathrm{flip}}$ of $\mathcal{I}_{\setminus \mathrm{drop}}$ with cardinality $|\mathcal{I}_{\mathrm{flip}}| = \lfloor \frac{\gamma}{4} \cdot G \rfloor$ and copy $\tilde{\mathcal{B}}_{\mathrm{flip}} = \mathcal{B}_{\mathrm{flip}} = \{\mathrm{b}^i : i \in \mathcal{I}_{\mathrm{flip}}\}$. Then, we randomly flip the class of every label in $\tilde{\mathcal{B}}_{\mathrm{flip}}$ to a different label. We denote $\mathcal{I}_{\setminus \mathrm{flip}} = \mathcal{I}_{\setminus \mathrm{drop}} \setminus \mathcal{I}_{\mathrm{flip}}$ and $\mathcal{B}_{\setminus \mathrm{flip}} = (\mathcal{B}_{\setminus \mathrm{drop}} \setminus \mathcal{B}_{\mathrm{flip}}) \cup \tilde{\mathcal{B}}_{\mathrm{flip}}$.

To insert **shifts**, we change the localization of labels. We randomly choose a subset $\mathcal{I}_{\mathrm{shift}}$ of $\mathcal{I}_{\setminus \mathrm{flip}}$ with cardinality $|\mathcal{I}_{\mathrm{shift}}| = \lfloor \frac{\gamma}{4} \cdot G \rfloor$ and copy $\tilde{\mathcal{B}}_{sh} = \mathcal{B}_{sh} = \{\mathrm{b}^i : i \in \mathcal{I}_{\mathrm{shift}}\}$. For the shift of a box $\tilde{\mathrm{b}}^i \in \tilde{\mathcal{B}}_{sh}$, the new values $\tilde{\mathrm{y}}$, $\tilde{\mathrm{h}}$ are determined analogously to $\tilde{\mathrm{x}} \sim \mathcal{N}(\mathrm{x}, 0.15 \cdot \mathrm{w})$ and $\tilde{\mathrm{w}} \sim \mathcal{N}(\mathrm{w}, 0.15 \cdot \mathrm{w})$ drawn from a normal distribution with itself as the expected value and $0.15 \cdot \mathrm{w}$ as the standard deviation. To avoid the shift being too small or too large, the parameters are repeatedly chosen until the intersection over union (IoU) of the original label $\mathrm{b}^i \in \mathcal{B}_{sh}$ and $\tilde{\mathrm{b}}^i \in \tilde{\mathcal{B}}_{sh}$ is in the interval of $[0.4, 0.7]$, $\forall i \in \mathcal{I}_{\mathrm{shift}}$. We denote $\mathcal{I}_{\setminus \mathrm{shift}} = \mathcal{I}_{\setminus \mathrm{flip}} \setminus \mathcal{I}_{\mathrm{shift}}$ and $\mathcal{B}_{\setminus \mathrm{shift}} = (\mathcal{B}_{\setminus \mathrm{flip}} \setminus \mathcal{B}_{\mathrm{shift}}) \cup \tilde{\mathcal{B}}_{\mathrm{shift}}$.

For **spawning** labels, we randomly choose a subset $\mathcal{I}_{\mathrm{spawn}}$ of $\mathcal{I}_{\setminus \mathrm{shift}}$ with cardinality $|\mathcal{I}_{\mathrm{spawn}}| = \lfloor \frac{\gamma}{4} \cdot G \rfloor$ and copy $\tilde{\mathcal{B}}_{\mathrm{spawn}} = \mathcal{B}_{\mathrm{spawn}} = \{\mathrm{b}^i : i \in \mathcal{I}_{\mathrm{spawn}}\}$. Then, we assign every label $\tilde{\mathrm{b}}^i \in \tilde{\mathcal{B}}_{\mathrm{spawn}}$ randomly to another image. Since in our experiments all images in a dataset have the same resolution, this ensures that objects do not appear in unusual positions like outside of the image. For instance, a car in BDD is more likely to be found on the bottom part of the image rather than in the sky. We denote the set of noisy labels as $\tilde{\mathcal{B}} = \mathcal{B}_{\setminus \mathrm{shift}} \cup \tilde{\mathcal{B}}_{\mathrm{spawn}}$.

One example per label error type is shown in fig. 6.1 (b)–(e). Note that the number of labels G is unchanged as the number of *drops* and *spawns* is the same.

6.3.2 Instance-wise Loss Values for Label Error Detection

Our method to detect the introduced label error types in section 6.3.1 is based on an instance-wise loss for two-stage object detectors. The NMS is no longer based on the detection score or the entropy, but on the box-wise loss of the respective stage. Every prediction in $\widehat{f}^{\mathrm{RPN}}(\boldsymbol{x}|\boldsymbol{\theta})$ is assigned with a region proposal loss $\mathcal{L}^{RPN}$, which is the sum of a classification (binary cross-entropy) and regression (smooth-L^1) loss for the labels and the prediction itself. The computation of the loss is *identical to the one in training.*

Refer to section 2.3.2.1 for the loss definitions. Since not all labels are associated with a proposal after the first stage, i.e., the model may predict only background near a label, we add the labels themselves to the set of label error proposals. After box refinement and classification, every box $\widehat{f}_{\mathrm{RoI}}$ is assigned with a region of interest loss ($\mathcal{L}^{ROI}$), which is the sum of a classification (cross entropy) and regression (smooth-L^1) loss for the labels and the prediction itself. Then $\mathcal{L}^{RPN}$ and $\mathcal{L}^{ROI}$ are summed up to obtain an instance-wise loss score. A sketch of our method is shown in fig. 6.2. We can find the *dropped* blue label for "N" since the predictions near the object should have a high detection score, resulting in a high first stage classification loss. The *spawned* red label is assigned with a high classification loss from the first and second stage, since the assigned predictions should have a score close to zero in the first stage and an almost uniform class distribution in the second stage. Whether the yellow label is a *flip* is irrelevant for the first stage, since the loss should be small either way. If the box is classified correctly according to the associated label, there is a large classification loss for a *flip* and a small one otherwise. The *shifts* are addressed by the first and second stage regression loss.

6.3.2.1 Theoretical Justification and Intuition

The intuition behind our method is that a sufficiently well-specified and fitted model has small expected loss on data sampled during training. Sufficient data sampling and moderate label error rates lead to *label errors giving rise to outlier losses* which are identified as proposals. We show that our method separates correct from incorrect labels for a classification model $\widehat{f}$ trained with the cross entropy loss $\mathcal{L}_{\mathrm{CE}}$.

Proposition 1 (Statistical Separation of the Cross Entropy Loss)**.** *Let training and testing labels be given under a stochastic flip in $\mu_{|\boldsymbol{X}=\boldsymbol{x}}$ with probability p_{F}. A correct label $y = f(\boldsymbol{x})$ is given by a true labeling function $f : \mathcal{X} \to \mathcal{Y}$ and has probability $\mu_{|\boldsymbol{X}=\boldsymbol{x}}(f(\boldsymbol{x})) = 1 - p_{\mathrm{F}}$. Incorrect labels $\widetilde{y} \neq f(\boldsymbol{x})$ are drawn with probability $\mu_{|\boldsymbol{X}=\boldsymbol{x}}(\widetilde{y}) = p_{\mathrm{F}}/(C-1)$. Let the label distribution $\mu_{|\boldsymbol{X}=(\cdot)}$ be PAC-learnable by the hypothesis space of an ERM learner $\widehat{f}(\cdot|x)$ with respect to D_{KL} (to precision ε and confidence $1-\delta$) and let $\kappa > 0$. If $p_{\mathrm{F}} < \frac{C-1}{C}(1-2\kappa)$, we obtain strict separation of the loss function*

$$\mathcal{L}_{\mathrm{CE}}(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\|f(\boldsymbol{x})) < -\log(1-p_{\mathrm{F}}-\kappa) < -\log(\kappa + \tfrac{p_{\mathrm{F}}}{C-1}) < \mathcal{L}_{\mathrm{CE}}(\widehat{f}(\boldsymbol{x}|\boldsymbol{\theta})\|\widetilde{y}) \qquad (6.4)$$

for any incorrect label $\widetilde{y} \neq f(\boldsymbol{x})$ with probability $1-\delta$ over the chosen training data and with probability $1 - \frac{2\varepsilon}{\kappa^2}$ over the choice of $\boldsymbol{x}$.

We include a proof of this statement in the following paragraph. The PAC-learnability assumption (see [166] and section 2.1.2) yields rigorous bounds for the deviation of the model $\widehat{f}$ from the label distribution p which contains label flips. Conditioned to the events of drawing correct versus drawing incorrect labels, these bounds carry over to the cross entropy. These bounds separate the two events with certain probability given in the statement above.

Setting and Proof of Proposition 1. Our goal is to show that the flip of a test label is statistically captured by the cross entropy loss evaluated at a DNN's[48] prediction

[48]Technically, it is not required that the model is a DNN as long as PAC-learnability is fulfilled.

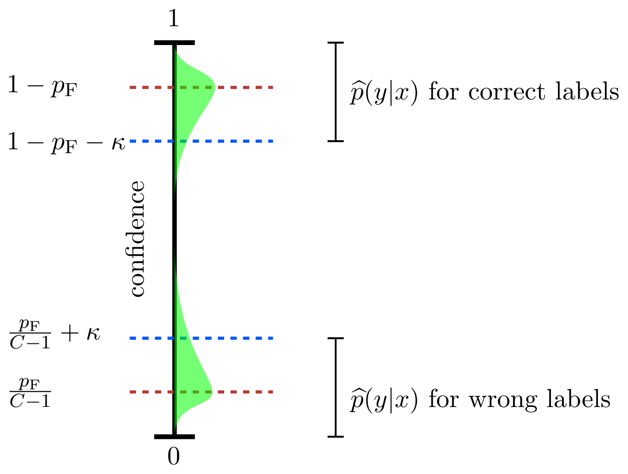

1
$1 - p_\mathrm{F}$
$1 - p_\mathrm{F} - \kappa$
confidence
$\frac{p_\mathrm{F}}{C-1} + \kappa$
$\frac{p_\mathrm{F}}{C-1}$
0
$\widehat{p}(y|x)$ for correct labels
$\widehat{p}(y|x)$ for wrong labels

Figure 6.3: Illustration of the probabilistic statement about predicted confidences conditioned to correct and incorrect given labels. PAC learning leads to concentration of the confidences around $1 - p_\mathrm{F}$ and $\frac{p_\mathrm{F}}{C-1}$, respectively. The separation on the confidences carries over to the cross entropy loss.

on a test sample $\boldsymbol{x}$ and the corresponding label y.

The rough intuition for this statement is that a probably approximate correct learner (PAC-learner, recall section 2.1.2 or [166]) $\widehat{f}$ has probabilistic bounds for having predictive distribution close to the data-generating distribution μ_Z. Therefore, sufficient data sampling and empirical loss minimization will lead to statistical concentration of confidences $\widehat{f}$ around $\mu_{|\boldsymbol{X}=(\cdot)}$. If μ_Z does not suffer from too strong label noise, we obtain separation between confidences on incorrect and confidences on correct labels. This separation then carries over to the negative log-likelihood (i.e., cross entropy) loss by monotony.

We assume data points $(\boldsymbol{x}, y) = z \sim \mu_Z$ following some noisy data generating distribution μ_Z, where $\boldsymbol{x} \sim \mu_{\boldsymbol{X}}$ follows a marginal distribution $\mu_{\boldsymbol{X}}$. In practice, training and test data originate from the same data pool, and we do not see any reason to assume that they follow different labeling procedures. However, it is sufficient to require that for testing data $(\boldsymbol{x}, y)$, $\boldsymbol{x}$ follows the same marginal distribution $\boldsymbol{x} \sim \mu_{\boldsymbol{X}}$. Our proof builds on the existence of a true labeling function $f : \boldsymbol{x} \mapsto y$ and the assumption that the data distribution μ_Z introduces stochastic flips of labels that occur with a fixed uniform rate $p_\mathrm{F} \in [0, 1)$. This flip probability p_F uniformly distributed over all $C - 1$ incorrect classes which are not $f(\boldsymbol{x})$ leads to the following constraints[49] on $\mu_{|\boldsymbol{X}=(\cdot)}$ when conditioned to $\boldsymbol{x}$:

$$\mu_{|\boldsymbol{X}=\boldsymbol{x}}(f(\boldsymbol{x})) := 1 - p_\mathrm{F}, \qquad \mu_{|\boldsymbol{X}=\boldsymbol{x}}(c) := p_\mathrm{F}/(C-1) \quad \forall c \neq f(\boldsymbol{x}). \tag{6.5}$$

[49]These assumptions seem relatively strict in that all classes except $f(\boldsymbol{x})$ are uniformly distributed. This assumption is merely a simplification which can easily be relaxed and modified. Here, we regard eq. (6.5) as a special case.

We assume that a statistical model $\widehat{f}$ learns classification on samples of the (noisy) data generating distribution μ_Z. In the present treatment, we assume PAC-learning with respect to the Kullback-Leibler divergence (recall eq. (2.6)). In the following, our goal is to show probabilistic statements about the cross entropy loss $\mathcal{L}_{\mathrm{CE}}$ on test data pairs $(\boldsymbol{x}, y)$. We show that the loss is above a certain threshold if an incorrect label is given and below some threshold in case of a correct, non-flipped label. Non-overlapping intervals indicate that the statistical separation between losses given correct and false labels seen in our experiments can be explained theoretically.

We assume PAC-learnability and ERM minimization for the proof. This assumption can be justified via the error decomposition of empirical risk minimization for the KL divergence over the hypothesis space $\mathscr{H}$ with training data $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$:

$$\begin{aligned}
\mathsf{D}(\mu_{|\boldsymbol{X}=(\cdot)} \| \widehat{f}) &:= \mathbb{E}_{\boldsymbol{x}\sim\mu_{\boldsymbol{X}}}[D_{\mathrm{KL}}(\mu_{|\boldsymbol{X}=\boldsymbol{x}} \| \widehat{f}(\boldsymbol{x}|\boldsymbol{\theta}))] \\
&\leq \inf_{\nu\in\mathscr{H}} \mathsf{D}(\mu_{|\boldsymbol{X}=(\cdot)} \| \nu_{|\boldsymbol{X}=(\cdot)}) + \left(\frac{1}{n}\sum_{j=1}^{n} \mathcal{L}_{\mathrm{CE}}(\widehat{f}(\boldsymbol{x}_j) \| y_j) - \inf_{\nu_{|\boldsymbol{X}=(\cdot)}\in\mathscr{H}} \mathcal{L}_{\mathrm{CE}}(\nu_{|\boldsymbol{X}=\boldsymbol{x}_j} \| y_j) \right) \\
&\quad + 2\cdot \sup_{\nu_{|\boldsymbol{X}=(\cdot)}\in\mathscr{H}} \left| \mathsf{D}(\mu_{|\boldsymbol{X}=(\cdot)} \| \nu_{|\boldsymbol{X}=(\cdot)}) - \frac{1}{n}\sum_{j=1}^{n} \mathcal{L}_{\mathrm{CE}}(\nu_{|\boldsymbol{X}=\boldsymbol{x}_j} \| y_j) - H(\mu_{|\boldsymbol{X}=\boldsymbol{x}_j}) \right| < \varepsilon
\end{aligned} \tag{6.6}$$

where $H(\mu_{|\boldsymbol{X}=\boldsymbol{x}}) = -\sum_{c=1}^{C} \mu_{|\boldsymbol{X}=\boldsymbol{x}}(\{c\}) \cdot \log(\mu_{|\boldsymbol{X}=\boldsymbol{x}}(\{c\}))$ is the entropy of the data generating distribution[50]. The first term is the model mis-specification error given by $\mathscr{H}$. In practice, we assume an expressive DNN with a large amount of capacity (appealing to universal approximation) which allows for this error to be negligible. In particular, in this case, no restrictions need to be made in the choice of $\mathscr{H}$. The second term measures the error of the learning algorithm with respect to an empirical risk minimizer h. Similarly to the term, an expressive DNN trained to convergence leads to small contributions by this term. Lastly, the third term is the sampling error made as compared to the loss $\mathsf{D}(\mu_{|\boldsymbol{X}=(\cdot)} \| \nu_{|\boldsymbol{X}=(\cdot)})$ in the true distribution. The third term can be controlled by application of concentration inequalities and chaining under certain assumptions (see [180]) which is why the sum of the three terms can be made smaller than some fixed $\varepsilon > 0$ given sufficient amount of data.

Proof of Proposition 1. Let $p(\cdot|\boldsymbol{x}) = \mu_{|\boldsymbol{X}=\boldsymbol{x}}$ and $\widehat{p}(\cdot|\boldsymbol{x}) = \widehat{f}(\boldsymbol{x})$. We aim at bounding

$$\max_{c=1,\dots,C} |p(c|\boldsymbol{x}) - \widehat{p}(c|\boldsymbol{x})| \tag{6.7}$$

by the total variation distance. PAC-learnability asserts that given enough data, the $\widehat{p}$-distributions illustrated in fig. 6.3 are concentrated around $1 - p_{\mathrm{F}}$ for true labels and $\frac{p_{\mathrm{F}}}{C-1}$ for incorrect labels. In particular, PAC-learnability implies

$$\mathbb{E}_{\boldsymbol{x}\sim\mu_{\boldsymbol{X}}}[D_{\mathrm{KL}}(p(\cdot|\boldsymbol{x}) \| \widehat{p}(\cdot|\boldsymbol{x}))] < \varepsilon \tag{6.8}$$

[50]Together with the cross entropy $\mathcal{L}_{\mathrm{CE}}$, the entropy H yields an unbiased risk function for D_{KL}.

with probability $1-\delta$ over the choice of training data. Let $\kappa > 0$. From this PAC result, we derive bounds for the probability of $\max_{c=1,\dots,C} |p(c|\boldsymbol{x}) - \widehat{p}(c|\boldsymbol{x})|$ exceeding κ via the total variation distance. We have

$$\begin{aligned}\mu_{\boldsymbol{X}}(\|p(\cdot|\boldsymbol{x}) - \widehat{p}(\cdot|\boldsymbol{x})\|_{\mathrm{TV}} \geq \kappa) &\leq \mu_{\boldsymbol{X}}(\sqrt{2D_{\mathrm{KL}}(p(\cdot|\boldsymbol{x})\|\widehat{p}(\cdot|\boldsymbol{x}))} \geq \kappa) \\ &\leq \mu_{\boldsymbol{X}}\left(D_{\mathrm{KL}}(p(\cdot|\boldsymbol{x})\|\widehat{p}(\cdot|\boldsymbol{x})) \geq \frac{\kappa^2}{2}\right) \\ &\leq \frac{2}{\kappa^2}\mathbb{E}_{\boldsymbol{x}\sim\mu_{\boldsymbol{X}}}[D_{\mathrm{KL}}(p(\cdot|\boldsymbol{x})\|\widehat{p}(\cdot|\boldsymbol{x}))] < \frac{2\varepsilon}{\kappa^2}\end{aligned} \tag{6.9}$$

with probability $1-\delta$ over the choice of training data. Here, the first inequality is the application of Pinsker's inequality [180, Ex. 49.] and the third due to the Markov inequality.

Assume that we are given a correct label y for $\boldsymbol{x}$, then with probability $1-\delta$ over training data and with probability $1-\frac{2\varepsilon}{\kappa^2}$ over sampling $\boldsymbol{x}$, we have that

$$|p(y|\boldsymbol{x}) - \widehat{p}(y|\boldsymbol{x})| = |(1-p_{\mathrm{F}}) - \widehat{p}(y|\boldsymbol{x})| \leq \max_y |p(y|\boldsymbol{x}) - \widehat{p}(y|\boldsymbol{x})| \tag{6.10}$$

$$\leq \|p(\cdot|\boldsymbol{x}) - \widehat{p}(\cdot|\boldsymbol{x})\|_{\mathrm{TV}} < \kappa. \tag{6.11}$$

This implies $\widehat{p}(y|\boldsymbol{x}) > 1 - p_{\mathrm{F}} - \kappa$ and therefore, by monotony of the logarithm function, $\mathcal{L}_{\mathrm{CE}}(\widehat{p}(y|\boldsymbol{x})\|y) < -\log(1-p_{\mathrm{F}}-\kappa)$. Similarly, if y is any incorrect label, we have the probabilistic statement

$$|p(y|\boldsymbol{x}) - \widehat{p}(y|\boldsymbol{x})| = \left|\frac{p_{\mathrm{F}}}{C-1} - \widehat{p}(y|\boldsymbol{x})\right| \leq \max_{y\in[C]} |p(y|\boldsymbol{x}) - \widehat{p}(y|\boldsymbol{x})| < \kappa, \tag{6.12}$$

i.e., $\widehat{p}(y|\boldsymbol{x}) < \kappa + \frac{p_{\mathrm{F}}}{C-1}$ and we have $\mathcal{L}_{\mathrm{CE}}(\widehat{p}(\boldsymbol{x})\|y) > -\log\left(\kappa + \frac{p_{\mathrm{F}}}{C-1}\right)$. Finally, we obtain separability of losses with true versus false labels in probability if

$$1 - p_{\mathrm{F}} - \kappa > \kappa + \frac{p_{\mathrm{F}}}{C-1} \iff p_{\mathrm{F}} < \frac{C-1}{C}(1-2\kappa). \tag{6.13}$$

Coming back to the assumptions on the label distribution in eq. (6.5), we note that the only thing required is that a separation regime as in eq. (6.13) exists. □

6.3.3 Evaluation Metrics

Ignoring that natural label errors exist in EMNIST-Det and BDD, we benchmark the five methods introduced in above by means of our label error simulation. To this end, we take the label error proposals of the respective method and the set of original labels $\mathcal{B}$ and decide for every proposal whether it is a label error, which corresponds to a true positive (TP_l), or no label error, which corresponds to a false positive (FP_l). Label errors that are not detected are called false negatives (FN_l). A proposal of a label error detector is a TP_l if the IoU between the proposal under consideration and a noisy label on the image is greater or equal to a threshold $1 \geq \alpha > 0$. Here, the noisy label categorizes what type of label error is detected by the proposal. If the IoU is less than α, the proposal is a FP_l. After determining this for each proposal from the dataset, AuROC and F_1 values are

Table 6.1: Validation of object detection performance on our datasets. $^{(*)}$ indicates learning with simulated label errors ($\gamma = 0.2$).

Dataset	Backbone	mAP_{50}	$\text{mAP}_{50}^{(*)}$
EMNIST-Det	Swin-T	98.2	98.0
EMNIST-Det	ResNeSt101	96.4	95.2
BDD	Swin-T	52.1	50.3
BDD	ResNeSt101	56.8	52.9
COCO	Swin-T	54.1	—
KITTI	Swin-T	38.6	—
VOC	Swin-T	83.3	—
CE	Swin-T	70.0	—

calculated according to the decision between TP_l and FP_l. F_1 values are determined with thresholding on the score of the respective method (loss/detection score/entropy/PD). We always choose the optimal threshold, i.e., the *threshold at which the F_1 value is maximized* (max F_1). Note, since the naive baseline considers images and thus label error proposals in random order, the associated AuROC values are always 0.5.

For commonly used datasets we proceed as follows. We consider for each dataset 200 proposals of our method with the highest loss and manually flag them as TP_l or FP_l, based on the label policy corresponding to the given dataset. Note that we can still compute precision values, but we are not able to determine AuROC or max F_1 values as the number of total label errors is unknown. Since several label errors can be detected with one proposal, precision describes the ratio of proposals with at least one label error and the total number of proposals considered, i.e. 200.

6.4 Experiments: Simulated and Real Label Errors

In this section we study label error detection performance on our label error benchmark as well as for real label errors in BDD, VOC, MS COCO (COCO), KITTI and the proprietary dataset (CE). The benchmark results are presented in terms of AuROC and max F_1 values for the joint evaluation of all label error types, i.e., when all label error types are present simultaneously, in section 6.4.2. For the latter, we show how many real label errors we can detect among the top-200 proposals for each real-world dataset in section 6.4.3.

6.4.1 Implementation Details

We implemented our benchmark and methods in the open source MMDetection toolbox [18]. Our models are based on a Swin-T transformer [108] and a ResNeSt101 [215] backbone, both with a CascadeRoIHead [12] as the object detection head, with a total number of trainable parameters of approx. 72M and 95M. As hyperparameters for the label error benchmark we choose relative frequency of label errors $\gamma = 0.2$, the value for score thresholding after the first stage $\tau_s^{\text{RPN}} = 0.25$, the value for score thresholding after the second stage $\tau_s^{\text{RoI}} = 0$ and the IoU-value $\alpha = 0.3$ from which a proposal for a

Table 6.2: Label error detection experiments with two different backbones; higher values are better. Bold numbers indicate the highest AuROC or max F_1 per experiment and underlined numbers are the second highest.

			AuROC				max F_1			
Dataset	Backbone	Train Labels	Loss	Detection Score	Entropy	PD	Loss	Detection Score	Entropy	PD
EMNIST-Det	Swin-T	Original	**99.46**	73.24	71.49	59.67	**95.54**	64.74	49.58	62.32
EMNIST-Det	Swin-T	Noisy	**99.40**	82.44	77.32	62.26	**93.43**	62.37	45.25	62.24
EMNIST-Det	ResNeSt101	Original	**99.84**	88.45	86.70	60.59	**94.31**	62.56	38.81	60.82
EMNIST-Det	ResNeSt101	Noisy	**99.87**	93.11	86.40	61.82	**90.74**	59.50	34.53	59.01
BDD	Swin-T	Original	**96.30**	76.82	71.73	60.59	**56.59**	31.14	22.21	52.66
BDD	Swin-T	Noisy	**92.16**	89.21	69.42	57.58	**35.97**	31.68	18.33	34.72
BDD	ResNeSt101	Original	**95.79**	87.47	83.58	60.31	**54.62**	31.99	20.37	47.16
BDD	ResNeSt101	Noisy	**92.97**	90.76	78.18	56.79	**27.85**	25.65	18.10	27.74

Table 6.3: Training hyperparameters for the Swin-T and the ResNeSt101 ($^{(*)}$) backbone.

Dataset	Batch Size	Image Resolution	# Training Iterations	Learning Rate
EMNIST-Det	24	300×300	24,000/48,000$^{(*)}$	0.02
BDD	4	1333×800	150,000/250,000$^{(*)}$	0.01
KITTI	6	1000×600	70,000	0.01
COCO	12	1000×600	250,000	0.02
VOC	6	1000×600	70,000	0.02
CE	4	1000×600	200,000	0.01

label error is considered a TP_l. We show performance results for the respective models and for each dataset in table 6.1. The upper half shows results on original (mAP_{50}) and noisy training data (mAP^*_{50}), for which $\gamma = 0.2$ also holds. The bottom half of the table presents performance of the models that we use for predicting label errors on real datasets. This happens in each case based on a model trained on unmodified, i.e., original labels and the Swin-T backbone. The performance results obtained have all been evaluated on unmodified test datasets.

Datasets. For the detection of real label errors we use the same split for BDD as introduced in section 6.3.1 as well as VOC, COCO, KITTI and CE. The training data for VOC consists of "2007 train" + "2012 trainval" and we predict label errors on the "2007 test"-split. For COCO train split is used for training and label errors are predicted on the validation split from 2017. For KITTI we use a scene-wise split, resulting in 5 scenes ($Sc = \{2, 8, 10, 13, 17\}$) and 1,402 images for evaluation as well as 16 scenes ($\{0, 1, \ldots, 20\} \setminus Sc$) and 6,407 images for training. The subset of CE data used includes 20,100 images for training and 1,070 images for evaluation. In the images, a car is in focus and the task is to do a car part detection. The labels consist of 29 different classes and divide the car into different parts, i.e., the four wheels, doors, number plate, mirrors, bumper, etc. Compared to the static academic datasets, the CE dataset is dynamic and thus of heterogeneous quality. An exemplary test image including labels for the proprietary dataset (CE) is shown in fig. 6.4. The labels divide the car into parts, such as the two wheels "WheelFrontRight" and "WheelRearRight" as well as doors, roof, etc. The example also includes a *drop* with the missing mirror "MirrorRight".

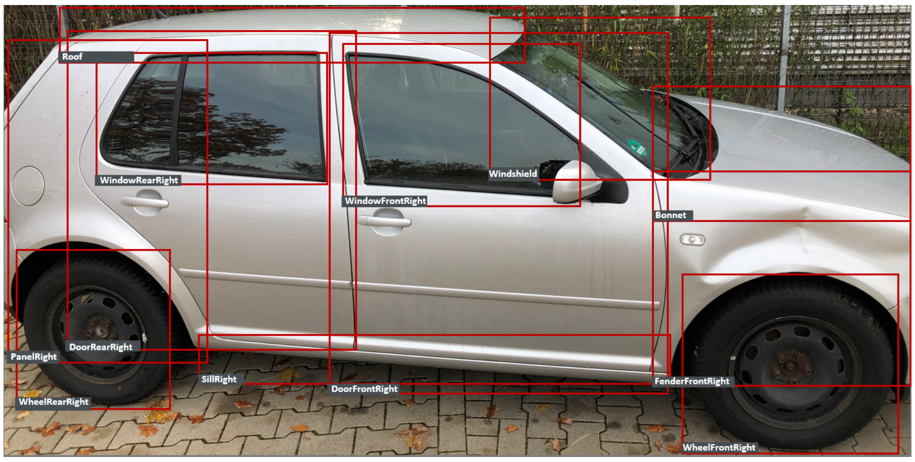


Figure 6.4: Example image from the CE test data with labels and a missing “MirrorRight”.

Dataset-dependent Parameters for Training. The dataset-dependent hyperparameters for training are stated in table 6.3. The original images from EMNIST-Det have an image resolution of 320×320 pixels, i.e., we do not artificially scale them to a higher image resolution. The BDD images also contain many small labels while having a high original resolution (1280×720), which is a challenging setup. To get the best possible label error detection, we keep this high resolution and rescale the images to 1333×800 pixels. KITTI, COCO, VOC and CE are each rescaled to an image resolution of 1000×600 pixels. The batch size for all datasets is in the range of 4-24, the initial learning rate is either 0.02 or 0.01 depending on the dataset, and the number of training iterations is in the range of 24,000 to 250,000. All numbers apply to the Swin-T backbone except the numbers $^{(*)}$ for the training iterations of EMNIST-Det and BDD, which apply to the ResNeSt101 backbone. All other hyperparameters are identical for the different architectures. The files for the configurations used in training, also containing the precise values of the above hyperparameters, are published with the code on GitHub.

6.4.1.1 Baseline Methods

The four baselines that we compare our instance-wise loss method with are based on

- inspecting the labels without the use of deep learning,
- the box-wise detection score,
- the classification entropy of the two-stage object detectors,
- the probability differential from [67].

Naive Baseline. We introduce a naive baseline to show the significant improvement of deep learning in label error detection for object detection over manual label review. We assume that all label errors can be smoothly found by taking a single look at all existing noisy labels and the (actually unknown) *drops*, i.e., by performing $\lfloor (1 + \frac{\gamma}{4}) \cdot G \rfloor$ operations. This simplified assumption is of course unrealistic, however the corresponding results can serve as a lower bound for the effort of manual label review.

Detection Score Baseline. The detection score baseline works as follows: For a given image from the set of all images of the dataset $(\boldsymbol{x}, \overline{y}) \in \mathcal{D}$, a neural network predicts a fixed number $\widehat{N}_{\text{out}}$ of bounding boxes for the first stage $\widehat{f}^{\text{RPN}}(\boldsymbol{x}|\boldsymbol{\theta})$ (cf. section 2.3.2). Then, we add the boxes $\overline{y}$ of the labels as proposals for the second stage to ensure that at least one prediction exists for each label, which is particularly important for the detection of *spawns*. For this purpose, each ground truth label from $\mathcal{B}$ is assigned with a detection score of $\widehat{s}^{\text{RPN}} = 1$ and treated like an RPN prediction. After adding the labels to $\widehat{f}^{\text{RPN}}(\boldsymbol{x}|\boldsymbol{\theta})$, only those $N_{\boldsymbol{x}}^{\text{RPN}}$ boxes that remain after class-independent NMS and score thresholding on $\widehat{s}^{\text{RPN}}$ with $\tau_s^{\text{RPN}} \geq 0$, are fed into the RoI head. After box refinement and classification as well as NMS on the detection score $\widehat{s}^{\text{RoI}} := \max_{c \in [C]} \widehat{\pi}_c^{\text{RoI}}$, $N_{\boldsymbol{x}}^{\text{RoI}}$ label error proposals remain. The remaining $N_{\boldsymbol{x}}^{\text{RoI}}$ label error proposals are defined by the localization $(\widehat{\mathrm{x}}^i, \widehat{\mathrm{y}}^i, \widehat{\mathrm{w}}^i, \widehat{\mathrm{h}}^i)$, the detection score $\widehat{s}_i^{\text{RoI}}$ and the class probabilities $\widehat{\pi}_1^i, \ldots, \widehat{\pi}_C^i$. The predicted class is given by $\widehat{c}^i := \text{argmax}_{k=1,\ldots,C}\, \widehat{\pi}_k^i$. Score thresholding is omitted here, or the threshold τ_s used for this is equal to 0, since $\tau_s > 10^{-4}$ would suppress most of the label error proposals that detect *spawns*. The detection score of these proposals is mostly very close to zero unless a second true label is nearby. After inferring each image $(\boldsymbol{x}, \overline{y}) \in \mathcal{D}$ as described above, we get label error proposals for the whole dataset.

Entropy Baseline. The entropy baseline follows the same procedure, only the NMS in the first and second stage are based on the respective box-wise entropy rather than the detection score.

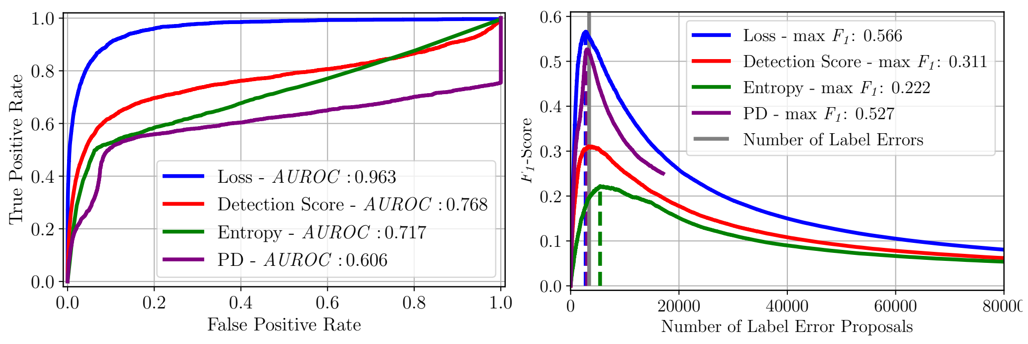
Probability Differential Baseline. For the probability differential (PD) baseline from [67], we do not add the boxes of the labels as proposals. Furthermore, score thresholding and NMS is not applied, such that every proposal box $\widehat{f}_{ij}^n$ remains in the final proposal set. After assigning every label with sufficiently overlapping predictions, the probability differential for every label $\mathrm{b} \in \mathcal{B}$ with class κ and the assigned predictions is defined as:

$$\text{PD}(\mathrm{b}) = \frac{1}{2\left|\text{Asgn}_{\widehat{f},\{\mathrm{b}\}}^{-1}(\{b\})\right|} \sum_{(i,j,n) \in \text{Asgn}_{\widehat{f},\{\mathrm{b}\}}^{-1}(\{\mathrm{b}\})} \left(1 + \max_{k \in [C] \setminus \{\kappa\}} ((\widehat{\pi}_k)_{ij}^n - (\widehat{\pi}_\kappa)_{ij}^n)\right) \quad (6.14)$$

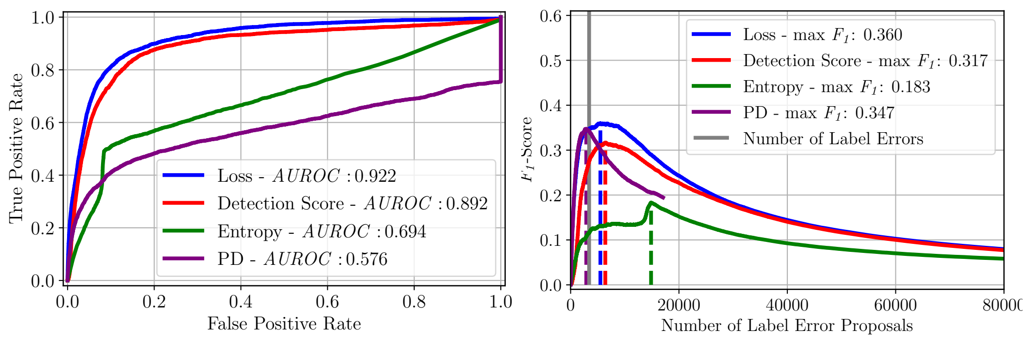
The PD of a label is in $[0, 1]$ and intuitively, the more the probabilities of the predictions and the class of the label differ (higher PD) the more likely a label error is present. Note that *drops* are always overlooked.

6.4.2 Benchmark Results for Simulated Label Errors

Table 6.1 (left) shows that although 20% of the training labels are modified, the performance in terms of mAP_{50} to mAP_{50}^* only decreases by a maximum of 1.2 percent points (pp) for EMNIST-Det and 3.9 pp for BDD. In both cases, the performance decreases more for the backbone containing more trainable parameters (ResNeSt101). This is consistent with the results for image classification by Northcutt et al. [131]. Architectures with fewer trainable parameters seem more suitable for handling label errors in the training data, possibly due to the network having less capacity to overfit the label errors. Figure 6.5 shows exemplary plots for AuROC and F_1 curves for the Swin-T backbone and BDD. On the two left plots we show results based on original training data and the two right plots



(a) Original Training Data



(b) Noisy Training Data

Figure 6.5: (a) Evaluations based on the predictions of a model trained on original training data. (b) Evaluation based on noisy training data with $\gamma = 0.2$. The number of considered label error proposals depends on threshold τ.

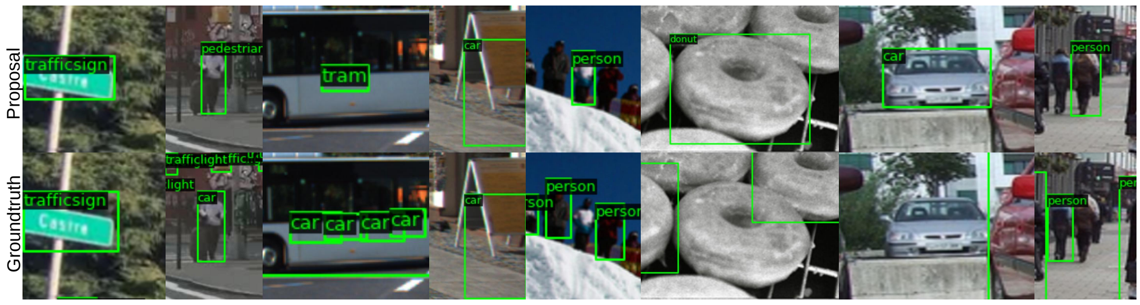


Figure 6.6: Visualization of detected label errors in real test datasets. The top row of images depicts the label error proposals and the bottom row the corresponding labels from the dataset. The image pairs belong from left to right in steps of two to BDD, KITTI, COCO and VOC.

based on noisy training data. The ranking of the methods is not identical everywhere. In terms of AuROC, loss (our method) is superior, followed by detection score, then entropy (our baselines) and finally PD. In terms of max F_1, PD outperforms the detection score and the entropy but is worse compared to the loss. Because AuROC considers rates and (max) F_1 considers absolute values and the number of label error proposals varies widely, the methods behave very differently with respect to AuROC and max F_1. However, the loss method outperforms all other methods on both metrics. Note, that the small step in the upper right of each of the AuROC plots are the false negatives according to the label errors (FN_l), i.e., the simulated label errors that are not found by the methods. This number of FN_l is vanishingly small in relation to all simulated label errors, except for PD as the method is not able to detect *drops*. The generally observed behavior for BDD also does not change when looking at the results for the ResNeSt101 backbone in table 6.2. When comparing the results for the different backbones with each other the AuROC for the loss and PD seems to remain similar, whereas detection score/entropy perform significantly better with 10.65/11.85 pp for original training data and 1.55/8.76 pp for noisy training data. The situation is different for the max F_1 values. For label error detection, loss/entropy/PD performs better with the Swin-T backbone for original training data (1.97/1.84/5.50 pp). In particular, the loss and PD seem to handle the noisy training data significantly better, resulting in 8.12 pp max F_1 difference between Swin-T and ResNeSt for the loss and 6.98 pp difference for PD. The detection score is 0.85 pp better with the ResNeSt101 backbone on original training data, but on noisy data the Swin-T outperforms the ResNeSt101 by 6.03 pp. Also, for EMNIST-Det it holds that the loss performs better than the detection score and both better than the entropy. In contrast to the results of BDD, the detection score slightly outperforms PD in all EMNIST-Det experiments also in terms of max F_1. The AuROC for loss appears to be stable across backbone and training data quality with only a maximum 0.47 pp difference overall. The detection score and entropy perform better in terms of AuROC with the ResNeSt101 backbone, but worse in terms of max F_1. Comparable to the results for BDD, the detection score performs better in terms of AuROC based on noisy training data, but the max F_1 values become worse. For PD, the AuROC seems to be rather stable comparing the two backbones, but the max F_1 is better for Swin-T compared to ResNeSt101.

Table 6.4: AuROC and max F_1 values for loss, detection score (Score), entropy and PD for all dataset-backbone-training label combinations; higher values are better. Bold numbers indicate the highest AuROC or max F_1 per experiment and underlined numbers are the second highest.

				AuROC				max F_1			
Label Error Type	Dataset	Backbone	Train Labels	Loss	Score	Entropy	PD	Loss	Score	Entropy	PD
Drop	EMNIST-Det	Swin-T	Original	98.94	**99.12**	88.16	0.00	**94.91**	89.63	56.70	0.00
	EMNIST-Det	Swin-T	Noisy	98.85	**99.19**	88.22	0.00	**93.27**	90.24	48.97	0.00
	EMNIST-Det	ResNeSt101	Original	**99.66**	99.65	78.33	0.00	**93.58**	87.02	32.82	0.00
	EMNIST-Det	ResNeSt101	Noisy	99.91	**99.94**	78.79	0.00	**86.42**	81.03	19.21	0.00
	BDD	Swin-T	Original	94.92	**96.05**	51.48	0.00	41.80	**48.38**	2.37	0.00
	BDD	Swin-T	Noisy	91.72	**93.64**	52.88	0.00	37.93	**46.93**	1.45	0.00
	BDD	ResNeSt101	Original	**94.52**	93.61	73.14	0.00	**45.89**	35.67	7.11	0.00
	BDD	ResNeSt101	Noisy	**91.89**	91.84	62.25	0.00	**26.29**	22.62	1.75	0.00
Flip	EMNIST-Det	Swin-T	Original	99.74	**99.78**	91.09	99.34	**92.89**	90.08	59.51	86.79
	EMNIST-Det	Swin-T	Noisy	99.62	**99.83**	90.79	99.51	**89.42**	88.70	49.32	87.44
	EMNIST-Det	ResNeSt101	Original	99.96	**99.97**	78.95	99.07	**90.77**	86.70	31.65	82.93
	EMNIST-Det	ResNeSt101	Noisy	99.89	**99.94**	78.50	98.83	**81.49**	80.35	18.98	80.49
	BDD	Swin-T	Original	**99.68**	98.36	50.63	98.53	**74.54**	58.79	2.75	73.86
	BDD	Swin-T	Noisy	**99.56**	98.12	50.06	98.32	60.31	58.91	2.13	**71.23**
	BDD	ResNeSt101	Original	**99.80**	98.16	75.93	97.96	**72.81**	54.38	7.12	69.95
	BDD	ResNeSt101	Noisy	**99.31**	97.24	64.34	97.13	44.94	40.15	2.18	**61.75**
Shift	EMNIST-Det	Swin-T	Original	**99.80**	51.52	93.55	40.71	**91.76**	11.14	49.41	10.61
	EMNIST-Det	Swin-T	Noisy	**99.56**	50.26	88.01	50.70	**87.86**	10.92	40.71	10.88
	EMNIST-Det	ResNeSt101	Original	**99.67**	51.28	86.14	45.54	**88.65**	11.28	30.32	10.56
	EMNIST-Det	ResNeSt101	Noisy	**99.30**	53.73	80.52	51.91	**85.97**	13.99	25.65	10.95
	BDD	Swin-T	Original	**65.49**	51.76	61.17	50.24	**16.78**	11.22	14.99	10.55
	BDD	Swin-T	Noisy	57.23	52.57	**57.91**	52.85	12.73	11.44	**12.88**	10.94
	BDD	ResNeSt101	Original	**65.84**	51.51	63.37	54.19	**17.56**	11.40	14.76	11.86
	BDD	ResNeSt101	Noisy	55.92	50.85	**56.18**	52.54	**12.58**	10.87	12.17	11.13
Spawn	EMNIST-Det	Swin-T	Original	**99.37**	75.62	97.92	97.04	**98.87**	19.89	65.08	78.97
	EMNIST-Det	Swin-T	Noisy	**99.68**	50.95	98.48	97.16	**97.77**	19.26	59.18	79.33
	EMNIST-Det	ResNeSt101	Original	**99.84**	57.98	99.40	96.12	**98.06**	18.96	37.84	74.19
	EMNIST-Det	ResNeSt101	Noisy	**99.93**	76.31	99.33	94.89	**94.93**	15.89	35.39	67.92
	BDD	Swin-T	Original	**98.48**	66.33	98.07	92.09	**74.97**	2.23	20.24	50.81
	BDD	Swin-T	Noisy	90.55	78.13	**92.98**	78.00	**17.98**	9.21	11.32	10.94
	BDD	ResNeSt101	Original	95.80	79.79	**97.00**	87.57	**60.38**	5.04	13.75	38.71
	BDD	ResNeSt101	Noisy	90.30	89.19	**95.74**	76.07	7.39	6.92	11.08	**28.55**

Results for Individual Simulated Label Error Types. In our experiments, all label errors occur simultaneously, but the evaluation can also be conditioned on the individual label error types. For *drops* or *flips* we consider only the false positives according to $\tilde{y}$, i.e., all boxes that have a maximum class-wise IoU of less than $\alpha(= 0.3)$ with all noisy labels of the associated image. Then, we can calculate AuROC and max F_1 values on this subset. We do the same for the *shifts*, except that we only consider the true positives according to $\tilde{\mathcal{B}}$. For the *spawns*, we must consider both true positives and false positives according to $\tilde{\mathcal{B}}$, since the predicted class, that overlaps sufficiently with the *spawned* label, can be the same as the class of the *spawn* itself.

The benchmark results for individual simulated label errors are stated in table 6.4. For *drops*, the detection score and instance-wise loss perform similarly well, with the AuROC values differing by at most 1.92 pp and a minimal AuROC of 91.72%. The difference in the max F_1 values is more pronounced, with the loss at EMNIST-Det outperforming the detection score by 3.03 to 6.56 pp. For BDD, the detection score of Swin-T is superior to the loss by up to 9 pp, whereas the loss for ResNeSt101 outperforms the detection score

Table 6.5: Validation of object detection performance and label error detection experiments for different noise in training data: Swin-T on BDD; higher values are better. Bold numbers indicate the highest AuROC or max F_1 per experiment and underlined numbers are the second highest.

			AuROC				max F_1			
γ	# train images	mAP$_{50}$	Loss	Detection Score	Entropy	PD	Loss	Detection Score	Entropy	PD
0	12,454	52.1	**96.30**	<u>76.82</u>	71.73	60.59	**56.59**	31.14	22.21	<u>52.66</u>
0.05	12,454	50.4	**93.44**	<u>88.09</u>	71.76	59.16	**43.36**	30.78	18.54	<u>42.98</u>
0.1	12,454	50.2	**93.21**	<u>89.05</u>	70.98	58.56	**39.36**	30.79	18.53	<u>37.92</u>
0.2	12,454	50.3	**92.61**	<u>89.21</u>	69.42	57.58	**35.97**	31.68	18.33	<u>34.72</u>

by up to 10.22 pp. The entropy reaches a maximum of 88.22%/56.70% AuROC/ max F_1 for EMNIST-Det and 73.14%/7.11% for BDD, which is far from the numbers achieved for the loss and the detection score. PD is not able to detect *drops*, as the bounding boxes of the labels are also the label error proposals itself.

A similar behavior can be observed for the *flips*, where the AuROC values for loss and detection score only differ by a maximum of 1.64 pp. In terms of max F_1 the loss outperforms the detection score and entropy in every case. PD performs worse in terms of AuROC compared to the loss, but in terms of max F_1, PD outperforms the loss for BDD based on both backbones trained on noisy data.

For the *shifts*, the detection score and PD have similar performance as the naive baseline in terms of AuROC and all max F_1 values are < 0.14. Except for BDD trained on noisy data, where entropy performs superior to the loss, loss outperforms all baselines.

For the *spawns*, the detection score performs similar compared to the *shifts*. PD performs well especially in terms of max F_1, where PD even outperforms the loss for RestNeSt101 on BDD with noisy training data by 20.16 pp, otherwise loss is superior to PD. In the cases where entropy outperforms loss, the difference is at most 5.44 pp in terms of AuROC and 3.69 pp in terms of max F_1.

The detection score can neither reliably detect the *shifts* nor the *spawns*, whereas the entropy cannot detect the *drops* and *flips* well, especially for complicated problems such as BDD. PD cannot reliably detect the *shifts* and is not able to detect *drops* by design. All in all, the loss method is the only one of those presented that can detect all four different types of label errors efficiently.

Results for Different Noise Intensity during Training. Table 6.5 shows mAP, AuROC and max F_1 values for different noise intensities for Swin-T on the BDD training dataset. In our experiments, it makes no difference whether the labels of the training data contain 5% or 20% noise, the mAP is between 50.2 and 50.4, where the model has a mAP of 52.1 due to training on the original training data. All mAP evaluations are based on the test data with original and thus unmodified labels.

On the one hand, the AuROC and max F_1 values decrease with increasing noise intensity by 3.69/20.62 pp for loss, by 2.31/3.88 pp for entropy and by 3.01/17.94 pp for PD, respectively. For the detection score, on the other hand, the AuROC value increases by 12.39 pp from 76.82 to 89.21 and the max F_1 value increases only marginally by 0.54 pp

Table 6.6: Validation of object detection performance and label error detection experiments for different noise and number of images for training Swin-T on BDD; higher values are better. Bold numbers indicate the highest AuROC or max F_1 per experiment and underlined numbers are the second highest.

			AuROC				max F_1			
γ	# train images	mAP_{50}	Loss	Detection Score	Entropy	PD	Loss	Detection Score	Entropy	PD
0	1,556	45.1	**94.79**	69.56	72.61	59.86	**58.67**	30.59	25.95	50.77
0	3,113	49.7	**95.25**	73.69	72.70	59.93	**56.48**	31.38	24.64	50.13
0	6,227	51.3	**95.18**	74.95	73.21	60.12	**55.79**	31.55	24.52	49.78
0	12,454	52.1	**96.30**	76.82	71.73	60.59	**56.59**	31.14	22.21	52.66
0.2	1,556	40.7	**94.82**	84.98	73.53	58.73	**44.47**	27.07	18.72	33.87
0.2	3,113	46.9	**93.35**	88.90	70.31	58.98	**35.45**	28.38	18.28	33.45
0.2	6,227	49.1	**93.08**	90.31	70.80	58.37	**33.60**	30.38	18.18	33.43
0.2	12,454	50.3	**92.61**	89.21	69.42	57.58	**35.97**	31.68	18.33	34.72

to 31.68. Nevertheless, the loss outperforms the detection score/entropy/PD in every case by at least 3.40/21.68/35.71 pp in terms of AuROC and by at least 4.29/17.64/0.38 pp in terms of max F_1. All AuROC/max F_1 evaluations are based on the test data with $\gamma = 0.2$ and thus on the identical label basis as for table 6.2.

Results for Different Amount of Training Images. Table 6.6 shows mAP, AuROC and max F_1 values for different amounts of training images for Swin-T on BDD. Therefore, the subsets with fewer images are always included in the subsets with more images and the identically sized subsets with different noise intensities contain the same images.

The mAP increases the more images are used for training and the less label errors exist in the training data. Here, the model trained on 6,227 and unmodified labels ($\gamma = 0$) has a 0.8 points higher mAP than the model trained on 12,454 images with $\gamma = 0.2$. In this case, after comparing the performances, it is worth to review and improve the underlying labels instead of labeling new images and add them to the training set.

The AuROC values increase as the number of images increases with $\gamma = 0$. With $\gamma = 0.2$, the values for loss and entropy decrease as the number of images increases. The max F_1 values decrease independently of γ with increasing number of images for loss and entropy, whereas the values increase for detection score. The decrease in AuROC and max F_1 values for loss and entropy could be due to overfitting of the model. For PD, AuROC and max F_1 values remain almost constant for the respective datasets. However, the loss always outperforms all baselines in terms of AuROC and max F_1. All AuROC and max F_1 evaluations are based on the test data with $\gamma = 0.2$ and thus on the identical label basis as for table 6.2.

6.4.3 Evaluation for Real Label Errors

We now aim at detecting real instead of simulated label errors. The considered real-world datasets apart from BDD (VOC, COCO, KITTI, CE) are more similar in complexity to BDD than to EMNIST-Det. For BDD we observed in section 6.4.2 that the loss method for the Swin-T backbone seems to be more stable according to label errors in the training

Table 6.7: Categorization of the top-200 proposals for real label errors with the loss method for the Swin-T backbone. $^{(*)}$ indicates the evaluation of proposals based on the detection of *drops*.

Dataset	Label Errors	Precision	Spawn	Drop	Flip	Shift
BDD	34	15.5	3	2	26	3
KITTI	96	47.5	75	0	4	17
COCO	50	24.5	14	1	18	17
COCO$^{(*)}$	125	61.0	0	125	0	0
VOC	23	11.5	13	0	10	0
VOC$^{(*)}$	175	71.5	0	175	0	0
CE$^{(*)}$	194	97.0	0	0	0	0

data. Especially the max F_1 values for the loss and noisy labels are better for Swin-T than for ResNeSt101. As we suspect label errors in the VOC, COCO, KITTI and CE training datasets, we use the Swin-T backbone to detect as many label errors as possible. Furthermore, we showed in table 6.2 that the loss method outperforms the detection score, entropy and PD in each presented experiment so in the following we detect label errors using only the loss method. Since we manually look at all proposals individually, and we are not able to look at all proposals (i.e., about 265,000 for VOC), we categorize the top-200 proposals into TP_l or FP_l. If a TP_l is found we also note which type of label error is present and if we are not sure whether the proposal is TP_l or FP_l, we conservatively interpret it as FP_l. The results are summarized in table 6.7. For BDD, there are at least 34 label errors, which mostly consist of *flips*. Since KITTI consists of image sequences, it happens that one label error appears on several consecutive frames. When this happens, it usually affects objects that are visible on previous frames but are covered by, for instance, a bus for several frames but are still labeled. Label error proposals that fall into "Don't Care" areas are not considered. In total, we find 96 label errors with a precision of 47.5% on KITTI. As COCO and VOC consist of images of different everyday scenes that really differ from image to image, the variability of the representation of objects is very high in these two datasets. Since a label error proposal is enforced for each label, this also applies to the labels that are classified as background. In a usual test setting, these labels would have been false negatives of the model, i.e., overlooked labels. The resulting loss is so high that these proposals end up in the reviewed top-200 proposals. Nevertheless, 50 label errors can be detected on COCO and 23 on VOC. When dealing with these two datasets, we noticed that *drops* are the most present label error type, although we did not find any among the top-200 proposals. We use this knowledge to restrict the proposals to those that have a class-independent IoU with the labels of the image of less than α. Using this subset and re-reviewing the top-200 proposals, we are able to find 125 *drops* with a precision of 61.0% for COCO and 175 *drops* with a precision of 71.5% for VOC. For the calculation of the precision see section 6.3.3. Prior knowledge about the label quality of the dataset and the types of label errors that occur helps to detect a specific type of label error. From the high precision values for VOC and COCO, we conclude that our method can help to correct the label errors resulting in cleaner benchmarks. Exemplary label errors for the above datasets are shown in fig. 6.6. The first proposal detects a *shift*,

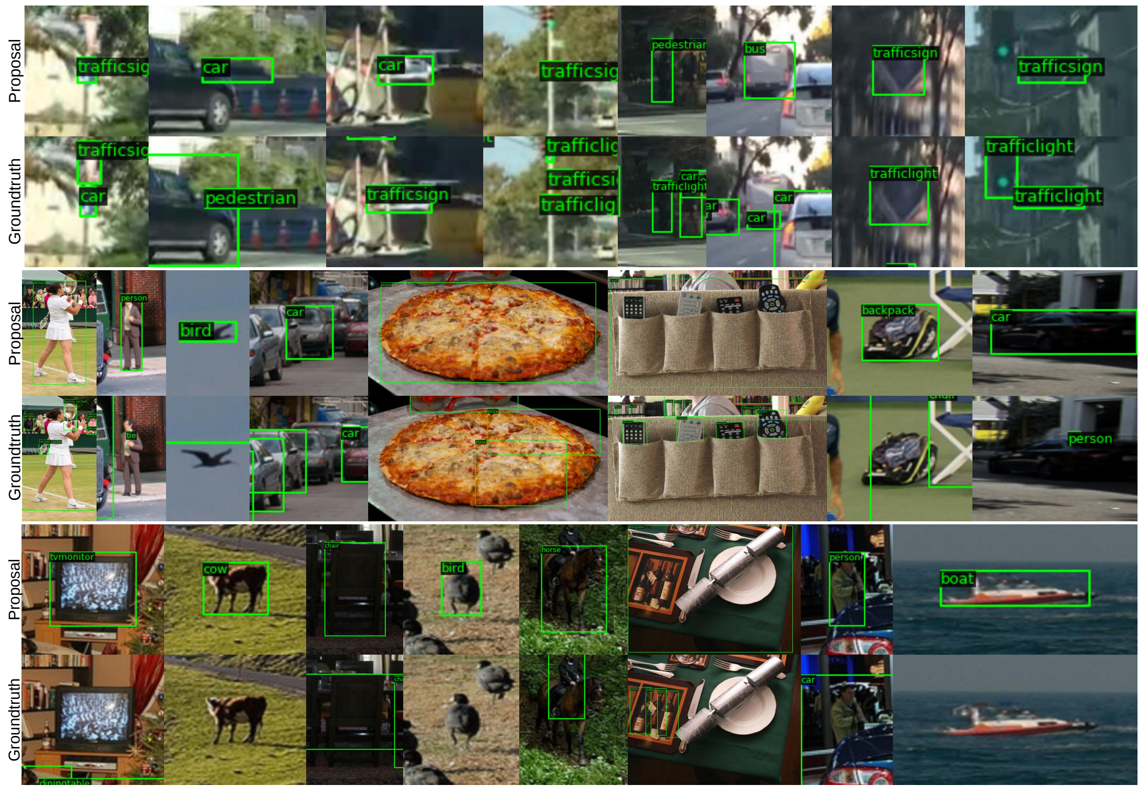


Figure 6.7: Visualization of further detected real label errors in test datasets for BDD (*top*), COCO (*center*) and VOC (*bottom*).

the second a *flip*, the third and fourth a *spawn* and the remaining proposals detect *drops*. For CE, we filter the proposals by *drops*, resulting in 194 detected *drops* with a precision of 97%.

Further Real Label Error Examples. Further, detected real label errors are presented in fig. 6.7. The top row shows examples for BDD, where all found label errors are *flips*, except for the third proposal from the right. This proposal can be interpreted as two label errors. Either the "car" label on the "bus" is wrong (*spawn*) and the bus was forgotten to be labeled (*drop*), or the localization is inaccurate (*shift*) and the label has a wrongly assigned class (*flip*). The middle and bottom rows represent detected real label errors on COCO and VOC. All proposals show *drops* and at the fourth proposal from the left in the middle row "pizza", one can even discuss two *spawns*, since the two smaller labels "pizza" should exactly reflect the label detected as *drop*.

6.5 Conclusion: Identifying Label Errors in Object Detection Data

In this chapter, we have introduced a benchmark for label error detection for object detection datasets. We for the first time simulated and evaluated four different types of label errors on two selected datasets that appear to be suitable for further method development. Furthermore, we developed a novel method based on instance-wise loss scoring and compare it with four baselines. Our method prevails by a significant margin in experiments on our simulated label error benchmark. In our experiments with real label errors, we found a number of label errors in prominent datasets as well as in a proprietary production-level dataset. With the evaluation for individual label error types we can detect real label errors on commonly used test datasets in object detection with a precision of up to 71.5%. Furthermore, we presented additional findings. Models with less parameters are more robust to label errors in training sets while models with more parameters suffer more.

7 Prediction Quality Estimation for Lidar Object Detection

In this chapter we introduce a post-processing based uncertainty quantification method for deep object detection of three-dimensional bounding boxes in Lidar point clouds. The presented contents are in large parts taken word-for-word from [148].

7.1 Introduction: Uncertainty Quantification in Lidar Object Detection

In recent years, deep learning has achieved great advances in the field of 3D object detection on Lidar data [89, 198, 199, 205]. DNN architectures for this task are well-developed, however, there is little work in the area of uncertainty quantification (UQ) for such models [17, 118, 119, 139, 201]. UQ is crucial for deployment of DNN-based object detection in the real world, since DNNs as statistical models statistically make erroneous predictions. Down-stream algorithms are supposed to further process the predictions of perception algorithms and rely on statistically accurate and meaningful UQ. Aleatoric uncertainty is usually estimated by adding variance parameters to the network prediction and fitting them to data under a specific assumption for the distribution of residuals [17, 41, 43, 118, 119]. Such approaches usually alter the training objective of the detector by appealing to the negative log-likelihood loss for normally distributed residuals. Epistemic uncertainty is oftentimes estimated via Monte-Carlo (MC) dropout [17] or deep ensembles [201] (recall section 2.2.4.2). In such approaches, model sampling leads to a significant increase in inference time. Inspired by lines of research [145, 161] in the field of 2D object detection on camera images, we develop a framework for UQ in 3D object detection for Lidar point clouds. This approach does not alter the training objective and can be applied to any pre-trained object detector and does not require prediction sampling. Our framework, called LidarMetaDetect (short LMD), performs two UQ tasks: (1) meta classification, which aims at estimating the probability of a given prediction being a true positive vs.

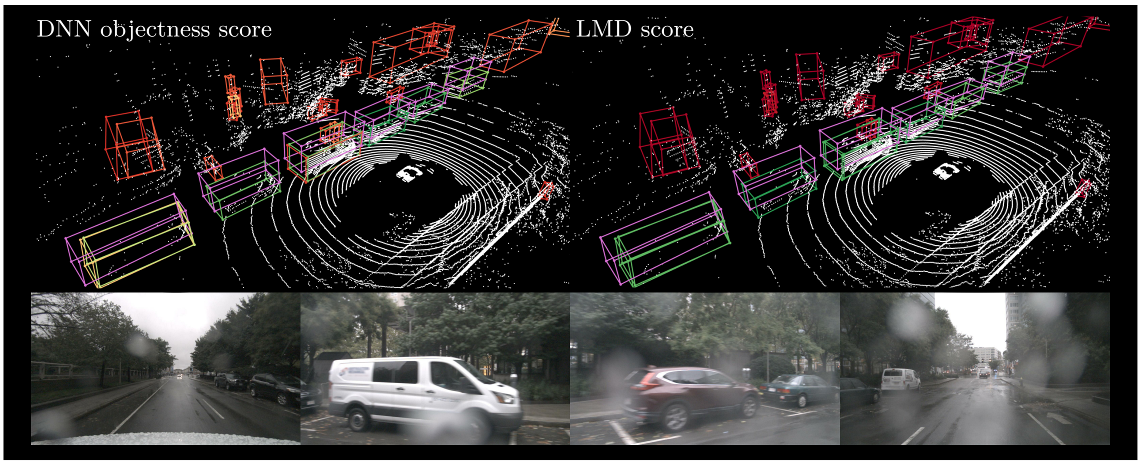


Figure 7.1: Prediction of a Lidar point cloud object detector with the native objectness score (*left*) and LMD meta classifier scores (*right*) and corresponding camera images below. Detections based on the objectness score are highly threshold-dependent and may lead to false positive detections. Detections based on LMD scores are more reliable and separate true from false predictions in a sharper way.

being a false positive; (2) meta regression, which estimates the localization quality of a prediction compared with the ground truth. Note that, outside the context of UQ for DNNs, the terms meta classification and meta regression refer to different concepts, see [103] and [171], respectively. LMD operates as a post-processing module and can be combined with any DNN without modifying it. Our methods learn on a small sample of data to assess the DNN's reliability in a frequentist sense at runtime, i.e., in the absence of ground truth. In essence, we handcraft a number of uncertainty scores on bounding box level, by which we convert both UQ tasks into structured machine learning tasks. To the best of our knowledge, our method is the first purely post-processing-based UQ method for 3D object detection based on Lidar point clouds. We conduct in-depth numerical studies on the KITTI [46], nuScenes [11] as well as a propriety dataset including comparisons of our methods with baseline methods on common uncertainty quantification benchmarks, ablation studies of relevant parameters and the relevance of our uncertainty features. This is complemented with down stream tasks where (1) we demonstrate that our UQ increases the separation of true and false predictions and leads to well-calibrated confidence estimates and (2) we show that our UQ can be utilized for the detection of erroneous annotations in Lidar object detection datasets. We evaluate our method's annotations error detection capabilities by reviewing its proposals on moderate samples from KITTI and nuScenes. Our contributions can be summarized as follows:

- We develop the first purely post-processing based UQ framework for 3D object detection in Lidar point clouds.
- We compare our UQ methods to baselines and show that they clearly outperform the DNN's built-in estimates of reliability.

- We find annotation errors in the most commonly used publicly available Lidar object detection datasets, i.e., KITTI and nuScenes.

7.2 Related Work: Reliable Lidar Object Detection via Uncertainty Quantification

In recent years, technologically sophisticated methods such as perception in Lidar point clouds have received attention in the UQ branch due to their potential industrial relevance in the autonomous driving sector. Methods for 3D object detection roughly fall into the categories of aleatoric and epistemic UQ. Aleatoric UQ methods usually build on estimating distributional noise by adding a variance output for each regression variable while epistemic UQ methods utilize some kind of model sampling either appealing to MC dropout or deep ensembles. Meyer et al. [118] estimate aleatoric uncertainty by a two-dimensional discretization scheme over the Lidar range and introducing a variance-weighted regression loss for a multimodal distributional prediction in order to improve detection performance. Meyer and Thakurdesai [119] estimate aleatoric uncertainty by adding scale regression variables to the network output, modeling Laplace-distributed residuals under a label noise assumption via a Kullback-Leibler divergence loss. Feng et al. [43] estimate heteroscedastic aleatoric uncertainty for the region proposal and the detection head of an object detector separately by modeling diagonal-covariance normally distributed bounding box regression. Feng et al. [41] achieve joint estimation of aleatoric and epistemic UQ by adding regression variables that model the covariance diagonal of a multi-variate normal distribution of the four bounding box parameters alongside MC dropout total variance for the epistemic component. Chen et al. [17] extract aleatoric uncertainty information from a self-supervised projection-reconstruction mechanism propagated to 3D object detection on camera images. Further, epistemic uncertainty of object localization is quantified via MC dropout. Yang et al. [201] perform UQ for 3D object detection on Lidar and extend the multi-input multi-output model MIMO [59] which modifies the network to be supplied simultaneously with n inputs and providing n outputs. This simulates a deep ensemble at inference time at the cost of increased memory consumption for input and output layers.

In the field of 2D object detection in camera images by DNNs, methods for UQ have been developed in a series of works [145, 161] related with research on UQ in semantic segmentation [152, 153]. Schubert et al. [161] utilize the pre-NMS anchor statistics in a post-processing approach to obtain box-wise confidence and IoU-estimates. Riedlinger et al. [145] use instance-wise gradient scores in a post-processing scheme to obtain calibrated uncertainty estimates improving detection performance. Inspired by these lines of research, we develop a framework for UQ in 3D object detection for Lidar point clouds. We use lightweight post-processing models on top of a pre-trained Lidar point cloud object detector in order to obtain improved uncertainty and IoU-estimates. In contrast to previous work, our approach has the advantage that it may be applied to any pre-trained object detector without alteration of training or architecture and does not carry the computational and memory cost of sampling weights in a Bayesian manner like MC dropout or deep ensembles. We show that this approach leads to more reliable object detection predictions and that it can be applied in an intuitive way in order to detect annotation errors in object detection datasets.

7.3 Methods: Output-based Feature Computation and Meta Classification

In this section we describe our post-processing mechanism and how it can be applied to improve detection performance and to detect annotation errors. Our method assumes an object detector $\widehat{f}(\cdot)$ which maps point clouds $\boldsymbol{x}$ to a list of N proposal bounding boxes

$$\widehat{f}(\boldsymbol{x}) = \left\{\widehat{b}^1, \dots, \widehat{b}^N\right\}. \tag{7.1}$$

Point clouds $\boldsymbol{x} = (\boldsymbol{p}_1, \dots, \boldsymbol{p}_{N_{\mathrm{pt}}})$ consist of Lidar points $\boldsymbol{p} = (x, y, z, r) \in \mathbb{R}^4$ represented by three coordinates (x, y, z) and a reflectance value r each. Bounding boxes are represented by features $\widehat{b}^j(\boldsymbol{x}) = (\widehat{x}^j, \widehat{y}^j, \widehat{z}^j, \widehat{\ell}^j, \widehat{w}^j, \widehat{h}^j, \widehat{\vartheta}^j, \widehat{s}^j, \widehat{\pi}_1^j, \dots, \widehat{\pi}_C^j)$. Here, $\widehat{x}^j, \widehat{y}^j, \widehat{z}^j, \widehat{\ell}^j, \widehat{w}^j, \widehat{h}^j, \widehat{\vartheta}^j$ define the *bounding box geometry*, $\widehat{s}^j$ is the *objectness score* and $(\widehat{\pi}_1^j, \dots, \widehat{\pi}_C^j)$ is the *predicted categorical probability distribution*. The bounding box geometry is defined by a center point $(\widehat{x}^j, \widehat{y}^j, \widehat{z}^j)$, spatial extent $(\widehat{\ell}^j, \widehat{w}^j, \widehat{h}^j)$ to either Cartesian direction and a rotation angle $\widehat{\vartheta}^j$ around the z-axis. The latter defines the predicted class $\widehat{\kappa}^j = \mathrm{argmax}_{c=1,\dots,C}\, \widehat{\pi}_c^j$ while the objectness score $\widehat{s}^j$ is the model's native confidence estimate for each prediction. Out of the N proposal bounding boxes, only a small amount N_{NMS} will be left after non-maximum suppression (NMS) filtering and contribute to the final prediction of the detector

$$\mathrm{NMS}\left[\widehat{f}(\boldsymbol{x})\right] = \left\{\widehat{b}^i : i \in I_{\mathrm{NMS}}\right\}, \tag{7.2}$$

where we let $I_{\mathrm{NMS}} \subset \{1, \dots, N\}$ denote the post-NMS index set indicating survivor boxes. NMS follows the same procedure as explained in section 2.3.2 with the bird's eye IoU as overlap measure.

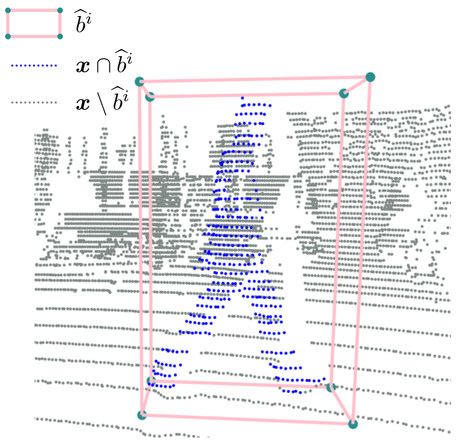
7.3.1 Computed Features in LidarMetaDetect

From this information we generate geometrical and statistical features for each $\widehat{b}^i \in \mathrm{NMS}[\widehat{f}(\boldsymbol{x})]$ for the purpose of UQ. In addition to the bounding box features

$$\widehat{\phi}^i := \{\widehat{x}^i, \widehat{y}^i, \widehat{z}^i, \widehat{\ell}^i, \widehat{w}^i, \widehat{h}^i, \widehat{\vartheta}^i, \widehat{s}^i, \widehat{\kappa}^i\} \tag{7.3}$$

of $\widehat{b}^i$ we compute the geometric features *volume* $V^i = \widehat{\ell}^i\widehat{w}^i\widehat{h}^i$, *surface area* $A^i = 2(\widehat{\ell}^i\widehat{w}^i + \widehat{\ell}^i\widehat{h}^i + \widehat{w}^i\widehat{h}^i)$, *relative size* $F^i = V^i/A^i$, *number of Lidar points* $P^i = |\boldsymbol{x} \cap \widehat{b}^i|$ within $\widehat{b}^i$ and *fraction of Lidar points* $\Phi^i = P^i/|\boldsymbol{x}|$ in $\widehat{b}^i$, see fig. 7.2 on the left for an illustration. Moreover, each Lidar point that falls into $\widehat{b}^i$ (i.e., in $\boldsymbol{x} \cap \widehat{b}^i$) has a reflectance value r. We add the maximal (ρ^i_{max}), mean (ρ^i_{mean}) and standard deviation (ρ^i_{std}) over all reflectance values of points in $\widehat{b}^i$. Lastly, for each $\widehat{b}^i$, we take the pre-NMS statistics into consideration which involves all proposal boxes in $\widehat{f}(\boldsymbol{x})$ that are NMS-filtered by $\widehat{b}^i$, i.e., the pre-image

$$\mathrm{Prop}\left(\widehat{b}^i\right) := \mathrm{NMS}^{-1}\left[\left\{\widehat{b}^i\right\}\right]. \tag{7.4}$$


$\widehat{b}^i$
$\boldsymbol{x} \cap \widehat{b}^i$
$\boldsymbol{x} \setminus \widehat{b}^i$

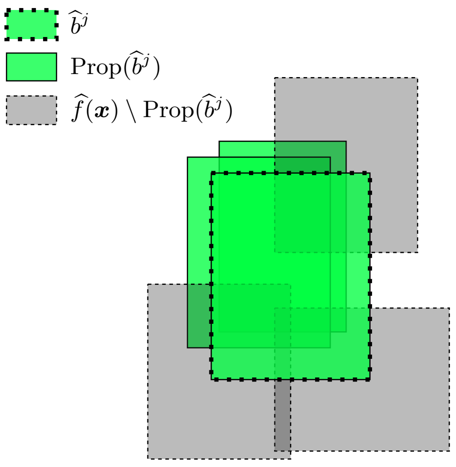

$\widehat{b}^j$
$\mathrm{Prop}(\widehat{b}^j)$
$\widehat{f}(\boldsymbol{x}) \setminus \mathrm{Prop}(\widehat{b}^j)$

Figure 7.2: *Left*: Illustration of the P^i and $\varPhi^i$ features counting Lidar points falling into a given predicted box. From the points $\boldsymbol{x} \cap \widehat{b}^i$, reflection statistics are generated. *Right*: Schematic illustration of the proposal set $\mathrm{Prop}(\widehat{b}^i)$ for a given predicted box $\widehat{b}^i$ (here, in two dimensions for simplicity). From the proposal boxes, further pre-NMS statistics are derived.

These are characterized by having a significant three-dimensional $\mathrm{IoU}_{\mathrm{3D}}$ with $\widehat{b}^i$, see fig. 7.2 on the right. The number of proposal boxes $N^i := |\mathrm{Prop}(\widehat{b}^i)|$ is an important statistics since regions with more proposals are more likely to contain a true prediction. We further derive minimum, maximum, mean and standard deviation statistics over proposal boxes $\widehat{b} \in \mathrm{Prop}(\widehat{b}^i)$ for all

$$m^i \in \widehat{\phi}^i \cup \{V^i, A^i, F^i, P^i, \varPhi^i, \rho^i_{\mathrm{max}}, \rho^i_{\mathrm{mean}}, \rho^i_{\mathrm{std}}\}, \tag{7.5}$$

as well, as the $\mathrm{IoU}_{\mathrm{3D}}$ and bird-eye intersection over union $\mathrm{IoU}_{\mathrm{BEV}}$ values between $\widehat{b}^i$ and all proposals $\mathrm{Prop}(\widehat{b}^i)$. Overall, this amounts to a vector $\boldsymbol{\varphi}^i(\boldsymbol{x})$ of length $n = 90$ consisting of co-variables (features) on which post-processing models are fit in order to predict the $\mathrm{IoU}_{\mathrm{BEV}}$ between $\widehat{b}^i$ and the ground truth or classify samples as true (TP) or false positives (FP). We call a box a TP if $\mathrm{IoU}_{\mathrm{BEV}} \geq 0.5$, otherwise we declare it FP.

7.3.2 Uncertainty Quantification in Post-Processing

On an annotated hold-out dataset $\mathcal{D}_{\mathrm{val}}$ consisting of point cloud-annotation tuples $(\boldsymbol{x}, \overline{y})$, we compute a structured dataset

$$\varphi = (\boldsymbol{\varphi}^1, \ldots, \boldsymbol{\varphi}^{N_{\mathrm{val}}}) \in \mathbb{R}^{n \times N_{\mathrm{val}}}. \tag{7.6}$$

This dataset consists of *feature vectors* for each of the N_{val} predicted boxes over all of $\mathcal{D}_{\mathrm{val}}$. The illustration of our method in fig. 7.3 shows this scheme for one particular Lidar frame $(\boldsymbol{x}, \overline{y})$ and the respective prediction on it. Further, we compute $\iota^i := \mathrm{IoU}_{\mathrm{BEV}}(\widehat{b}^i(\boldsymbol{x}), \overline{y})$ between prediction and ground truth form $\mathcal{D}_{\mathrm{val}}$ as target variables $\mathsf{y} = (\iota^1, \ldots, \iota^{N_{\mathrm{val}}}) \in \mathbb{R}^{N_{\mathrm{val}}}$. Similar to the method introduced in section 3.3.2 we then fit a light-weight (meta-) regression model $\mathcal{R} : \boldsymbol{\varphi}^i \mapsto \mathsf{y}_i$ on (φ, y). This model acts as *post-processing module of the detector* in order to produce $\mathrm{IoU}_{\mathrm{BEV}}$-estimates $\widehat{\iota}^i := \mathcal{R}(\boldsymbol{\varphi}^i)$ for each detection

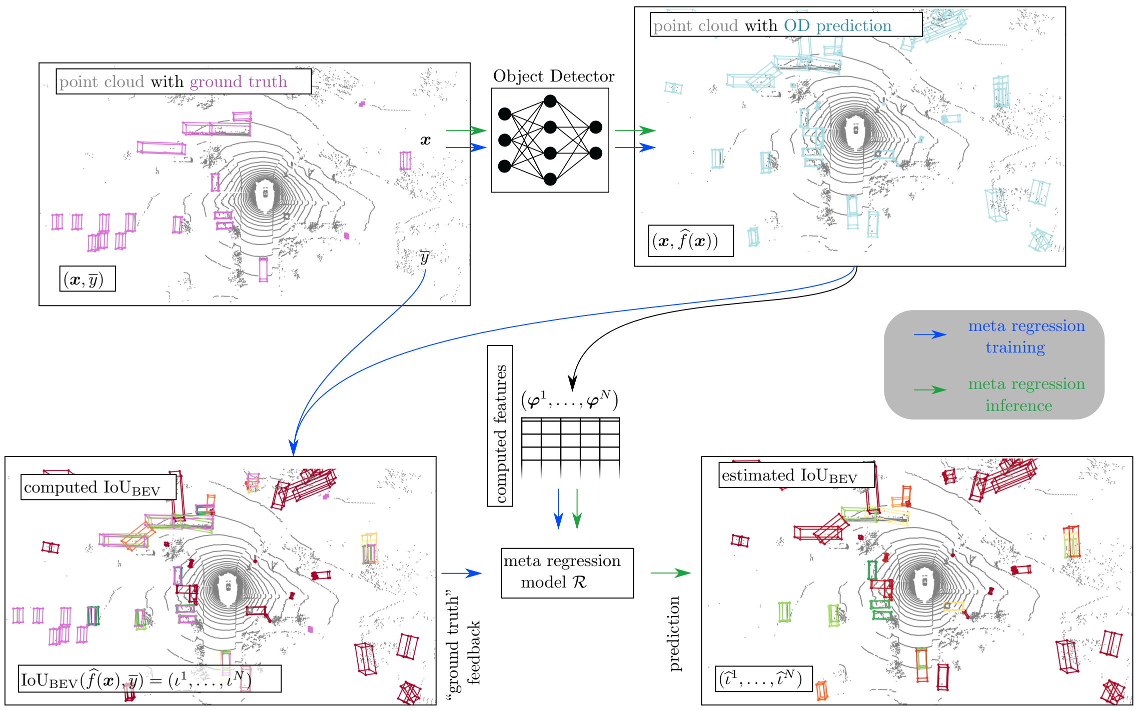

point cloud with ground truth
$(\boldsymbol{x}, \overline{y})$
$\boldsymbol{x}$
$\overline{y}$
Object Detector
point cloud with OD prediction
$(\boldsymbol{x}, \widehat{f}(\boldsymbol{x}))$
meta regression training
meta regression inference
computed features
$(\varphi^1, \ldots, \varphi^N)$
computed $\mathrm{IoU}_{\mathrm{BEV}}$
$\mathrm{IoU}_{\mathrm{BEV}}(\widehat{f}(\boldsymbol{x}), \overline{y}) = (\iota^1, \ldots, \iota^N)$
"ground truth" feedback
meta regression model $\mathcal{R}$
prediction
estimated $\mathrm{IoU}_{\mathrm{BEV}}$
$(\widehat{\iota}^1, \ldots, \widehat{\iota}^N)$

Figure 7.3: Schematic illustration of the LMD meta regression pipeline. Training of the model is based on the output $f(\boldsymbol{x})$ of a fixed (frozen) object detector and the bounding box ground truth $\overline{y}$. Meta classification follows the same scheme with binary training targets $\tau^i = 1_{\{\iota^i > 0.5\}}$.

$\widehat{b}^i$. Similarly, we fit a binary (meta-) classification model $\mathcal{C}$ obtaining the binary targets $1_{\{\mathrm{y}>0.5\}}$ which allows us to generate alternative confidence estimates $\widehat{\tau}^i := \mathcal{C}(\boldsymbol{\varphi}^i) \in [0,1]$ for each prediction $\widehat{b}^i$ in post-processing. Note that $\mathcal{C}$ is a potentially non-linear and non-monotonous function of the features $\boldsymbol{\varphi}^i$. Therefore, $\mathcal{C}$ can change the obtained confidence ranking per frame and influence detection performance as opposed to simple re-calibration methods [55, 126].

Meta classification empirically turns out to produce confidence estimates which are both, sharper (in the sense of separating TPs from FPs) and better calibrated that those produced natively by the detector, i.e., the objectness score. However, when regarding the cases of disagreement between the computed $\mathrm{IoU}_{\mathrm{BEV}}$ and $\mathcal{C}$, we frequently find that $\mathcal{C}$ is to be trusted more than the computed $\mathrm{IoU}_{\mathrm{BEV}}$ due to missing annotations. We use this observation in order to generate proposals (in descending estimation $\widehat{\tau}^i$) based on the object detector in comparison with the given ground truth (FP according to the ground truth, i.e., $\iota^i < 0.5$) that serve as suggestions of annotation errors.

7.4 Experiments: Meta Classification, Meta Regression and Label Error Detection

In this section we study meta classification and meta regression performance for two benchmark datasets as well as a proprietary dataset from Aptiv. The meta classification results are presented in terms of accuracy and AuROC and the meta regression results are presented in terms of R^2. We compare our uncertainty quantification method LidarMetaDetect (LMD) with two baseline methods (score, box features). Moreover, we find label errors on both benchmark datasets using LMD.

7.4.1 Implementation Details

Object Detection Models. We implemented our method in the open source MMDetection3D toolbox [123]. For our experiments, we consider the PointPillars [89] and CenterPoint [205] architectures. The mean average precision ($\mathrm{mAP@IoU}_{0.5}$) for KITTI based on $\mathrm{IoU}_{\mathrm{BEV}}$ is 69.0 for CenterPoint and 68.8 for PointPillars. On KITTI, the $\mathrm{mAP@IoU}_{0.5}$ based on $\mathrm{IoU}_{3\mathrm{D}}$ is 64.2 for CenterPoint and 68.8 for PointPillars and for Aptiv, the $\mathrm{mAP@IoU}_{0.5}$ based on $\mathrm{IoU}_{3\mathrm{D}}$ is 39.5 for CenterPoint and 43.7 for PointPillars. NuScenes performance is given as a weighted sum of mAP as well as the nuScenes detection score (NDS). For CenterPoint, the mAP is 57.4 and the NDS is 65.2 and for PointPillars the mAP is 40.0 and the NDS is 53.3. For KITTI and Aptiv, the models were trained individually while available public model weights from MMDetection3D are used for nuScenes. The performance results obtained have all been evaluated on respective test datasets.

Datasets. For KITTI, the images and associated point clouds are split scene-wise, such that the training set consists of 3,712, the validation set of 1,997, and the test set of 1,772 frames. For nuScenes, the validation set is split scene-wise into 3,083 validation and 2,936 test frames. The Aptivdataset consists of 50 sequences, split into $27, 14, 9$ sequences with about 145K, 75K, 65K cuboid annotations for training, validation and

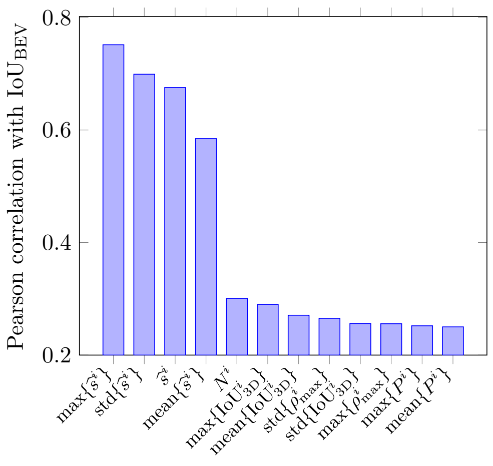


Figure 7.4: Strongest correlation coefficients for constructed box-wise features and $\mathrm{IoU}_{\mathrm{BEV}}$ for the CenterPoint architecture on the nuScenes test dataset and a score threshold $\tau = 0.1$.

Table 7.1: Strongest correlation coefficients for constructed box-wise features and $\mathrm{IoU}_{\mathrm{BEV}}$ for the CenterPoint (*left*) and PointPillars (*right*) architecture on the KITTI test dataset and a score threshold $\tau = 0.1$.

$\max\{\hat{s}^i\}$	0.8056	$\hat{s}^i$	0.8050
$\mathrm{std}\{\hat{s}^i\}$	0.7456	$\mathrm{mean}\{\hat{s}^i\}$	0.7289
$\mathrm{mean}\{\rho^i_{\max}\}$	0.5769	$\max\{\rho^i_{\max}\}$	0.5768
$\rho^i_{\max}$	0.5739	$\min\{\rho^i_{\max}\}$	0.5381
$\mathrm{mean}\{\mathrm{IoU}^i_{\mathrm{BEV}}\}$	0.5083	$\min\{\mathrm{IoU}^i_{3D}\}$	0.4984
$\max\{\mathrm{IoU}^i_{3D}\}$	0.4974	$\max\{P^i\}$	0.4593

$\max\{\hat{s}^i\}$	0.8493	$\hat{s}^i$	0.8490
$\mathrm{mean}\{\hat{s}^i\}$	0.8309	$\mathrm{std}\{\hat{s}^i\}$	0.7116
N^i	0.6258	$\max\{\ell^i\}$	0.5631
$\max\{w^i\}$	0.5174	$\mathrm{mean}\{\ell^i\}$	0.5159
$\max\{\rho^i_{\max}\}$	0.5144	$\mathrm{mean}\{w^i\}$	0.5129
$\max\{F^i\}$	0.5120	$\mathrm{mean}\{F^i\}$	0.5089

testing, respectively. Every sequence is about two minutes long while every fifth point cloud is annotated. The covered locations are countryside, highway and urban from and around Wuppertal, Germany. The dataset includes four classes: 1. smaller vehicles like cars and vans, 2. larger vehicles like busses and trucks, 3. pedestrians and 4. motorbikes and bicycles.

7.4.2 Correlation of Box-wise Features with the $\mathrm{IoU}_{\mathrm{BEV}}$.

Figure 7.4 shows the Pearson correlation coefficients of the constructed box-wise dispersion measures with the $\mathrm{IoU}_{\mathrm{BEV}}$ of prediction and ground truth for CenterPoint on the nuScenes test dataset. The score features have strong correlations (> 0.5) with the $\mathrm{IoU}_{\mathrm{BEV}}$. Note that, although the four score-related features show the highest individual correlation, these features may be partially redundant. The number of candidate boxes N^i is also reasonably correlated with the $\mathrm{IoU}_{\mathrm{BEV}}$ (0.3007), whereas the remaining features only show a minor correlation (< 0.3). However, they may still contribute to higher combined explanatory information in meta classification. The strongest correlation for the other network-dataset combinations are shown in the following paragraph.

Table 7.2: Strongest correlation coefficients for constructed box-wise features and $\mathrm{IoU}_{\mathrm{BEV}}$ for the CenterPoint (*left*) and PointPillars (*right*) architecture on the nuScenes test dataset and a score threshold $\tau = 0.1$.

$\max\{\widehat{s}^i\}$	0.7516	$\mathrm{std}\{\widehat{s}^i\}$	0.6991
$\widehat{s}^i$	0.6755	$\mathrm{mean}\{\widehat{s}^i\}$	0.5847
N^i	0.3007	$\max\{\mathrm{IoU}^i_{3\mathrm{D}}\}$	0.2900
$\mathrm{mean}\{\mathrm{IoU}^i_{3\mathrm{D}}\}$	0.2707	$\mathrm{std}\{\rho^i_{\max}\}$	0.2652
$\mathrm{std}\{\mathrm{IoU}^i_{3\mathrm{D}}\}$	0.2560	$\max\{\rho^i_{\max}\}$	0.2556
$\max\{P^i\}$	0.2519	$\mathrm{mean}\{P^i\}$	0.2500

$\max\{\widehat{s}^i\}$	0.7554	$\mathrm{std}\{\widehat{s}^i\}$	0.7344
$\mathrm{mean}\{\widehat{s}^i\}$	0.7161	N^i	0.6629
$\widehat{s}^i$	0.6244	$\mathrm{std}\{\widehat{w}^i\}$	0.4535
$\mathrm{std}\{F^i\}$	0.4228	$\mathrm{std}\{\widehat{y}^i\}$	0.3879
$\mathrm{std}\{\widehat{\ell}^i\}$	0.3794	$\mathrm{std}\{\mathrm{IoU}^i_{3\mathrm{D}}\}$	0.3689
$\mathrm{std}\{\mathrm{IoU}^i_{\mathrm{BEV}}\}$	0.3581	$\mathrm{std}\{P^i\}$	0.3433

Table 7.3: Strongest correlation coefficients for constructed box-wise features and $\mathrm{IoU}_{\mathrm{BEV}}$ for the CenterPoint (*left*) and PointPillars (*right*) architecture on the Aptiv test dataset and a score threshold $\tau = 0.1$.

$\max\{\widehat{s}^i\}$	0.7649	$\widehat{s}^i$	0.7358
$\mathrm{std}\{\widehat{s}^i\}$	0.6598	$\mathrm{mean}\{\widehat{s}^i\}$	0.5913
$\max\{\mathrm{IoU}^i_{3\mathrm{D}}\}$	0.5656	$\mathrm{mean}\{\mathrm{IoU}^i_{3\mathrm{D}}\}$	0.5596
$\max\{\mathrm{IoU}^i_{\mathrm{BEV}}\}$	0.5573	$\mathrm{mean}\{\mathrm{IoU}^i_{\mathrm{BEV}}\}$	0.5555
$\widehat{\ell}^i$	0.3736	$\min\{\widehat{\ell}^i\}$	0.3731
$\mathrm{mean}\{\widehat{\ell}^i\}$	0.3724	$\max\{\widehat{\ell}^i\}$	0.3694

$\max\{\widehat{s}^i\}$	0.7596	$\mathrm{mean}\{\widehat{s}^i\}$	0.7577
$\mathrm{std}\{\widehat{s}^i\}$	0.7570	$\widehat{s}^i$	0.7108
N^i	0.5226	$\mathrm{std}\{\widehat{\ell}^i\}$	0.5225
$\mathrm{std}\{F^i\}$	0.4488	$\mathrm{std}\{\widehat{h}^i\}$	0.3998
$\mathrm{std}\{\mathrm{IoU}^i_{3\mathrm{D}}\}$	0.3879	$\mathrm{std}\{\rho^i_{\max}\}$	0.3802
$\mathrm{std}\{\widehat{z}^i\}$	0.3727	$\max\{\mathrm{IoU}^i_{3\mathrm{D}}\}$	0.3443

Extended Correlation Results. Tables 7.1 to 7.3 show the Pearson correlation coefficients of the constructed box-wise dispersion measures with the $\mathrm{IoU}_{\mathrm{BEV}}$ of prediction and ground truth for the KITTI, nuScenes and Aptiv test datasets. Comparable to the results from fig. 7.4 of the main paper, the score features have strong correlations (> 0.5) with the $\mathrm{IoU}_{\mathrm{BEV}}$, independently of the underlying dataset or architecture. Especially for the PointPillars architecture, the number of proposal boxes N^i has a correlation > 0.6 for nuScenes and KITTI and > 0.5 for Aptiv, whereas for CenterPoint, N^i shows minor correlations (< 0.45). Moreover, the overlaps of the proposal boxes (different IoU features), as well as the localization of the box (especially $\widehat{\ell}$, $\widehat{h}$ and $\widehat{w}$) and the maximal reflectance value of points within the box ($\rho^i_{\max}$) seem to be reasonably correlated with the $\mathrm{IoU}_{\mathrm{BEV}}$. Although the other features have rather smaller correlations with the $\mathrm{IoU}_{\mathrm{BEV}}$, they may still contribute to a more informative set of features for meta classification and regression.

7.4.3 Comparison of Different Meta Classifiers and Meta Regressors.

Different models come into question as post-processing modules for meta classification ($\mathcal{C}$) and meta regression ($\mathcal{R}$, see section 7.3). For meta classification, the metamodels under consideration are logistic regression (LogReg), random forests (RF), gradient boosting (GB) and a multilayer perceptron (MLP) with two hidden layers. For meta regression, analogous regression models are used, only the logistic regression is replaced with a ridge regression (RR).

The respective metamodels are trained on the box-wise features $\boldsymbol{\varphi}^i$ of the validation sets $\mathcal{D}_{\mathrm{val}}$ and evaluated on the features of the test sets which are disjoint from $\mathcal{D}_{\mathrm{val}}$. LMD uses all available features to train the metamodels, whereas in the score baseline only the score

Table 7.4: Comparison of meta classification accuracy and AuROC as well as meta regression R^2 values for the score baseline, bounding box features and LMD for CenterPoint and nuScenes test dataset with score threshold $\tau_s = 0.1$. Models used are Logistic Regression (LR), Ridge Regression (RR), Random Forests (RF), Gradient Boosting (GB) and a Multi Layer Perceptron (MLP).

	Meta Classification $\mathrm{IoU}_{\mathrm{BEV}} \geq 0.5$ vs. $\mathrm{IoU}_{\mathrm{BEV}} < 0.5$								Meta Regression $\mathrm{IoU}_{\mathrm{BEV}}$			
	Accuracies				AuROCs				R^2			
Method	LogReg	RF	GB	MLP	LogReg	RF	GB	MLP	RR	RF	GB	MLP
Score	**0.8777**	0.8524	0.8772	0.8773	**0.8644**	0.8617	0.8623	0.8640	0.4641	0.4675	0.4733	**0.4751**
Box Features	0.8877	0.9049	**0.9203**	0.8975	0.9056	0.9454	**0.9529**	0.9293	0.5292	0.6681	**0.6792**	0.6249
LMD	0.9118	0.9166	**0.9297**	0.9200	0.9450	0.9581	**0.9628**	0.9530	0.6451	0.7242	**0.7296**	0.7122

of the prediction $\widehat{s}^i$ is used to fit the metamodel. For the bounding box features baseline, as the name implies, the box features of the prediction $\widehat{\phi}^i$ are used, in which the score $\widehat{s}^i$ is also included. Table 7.4 presents meta classification accuracy and AuROC as well as meta regression R^2 for the CenterPoint architecture on the nuScenes dataset. For the score baseline, all metamodels perform similarly well. For the meta classification accuracy there are differences of up to 2.53 percent points (pp), for the AuROC of at most 0.27 pp and for meta regression R^2 of up to 1.10 pp. In the box features the maximum differences increase to 3.26 pp in terms of accuracy, to 4.73 pp for AuROC and to 15.00 pp for R^2. In particular, for the box features and LMD, the non-linear models (RF, GB, MLP) outperform the linear model in both learning tasks. LMD outperforms the baselines box features/score by 0.94/5.20 pp in terms of accuracy, by 0.99/9.84 pp in terms of AuROC and by 5.04/25.45 pp in terms of R^2. If overfitting of the metamodel is made unlikely by choosing appropriate hyperparameters, the performance of the metamodel only benefits from adding more features, since the available information and number of parameters for fitting are increased. Overall, GB outperforms all other metamodels, especially when multiple features are used to train and evaluate the respective learning task. Therefore, only results based on GB are shown in the following experiments. An overview of the different metamodels for all network+dataset combinations are shown in the following paragraph.

Extended Comparisons of Different Post-Processing Models. Table 7.5 presents meta classification accuracy and AuROC for different meta classification models for the KITTI, nuScenes and Aptiv test datasets. For the score baseline, except for PointPillars and nuScenes, the linear model (LogReg) outperforms the random forest, gradient boosting and the MLP, where the difference is at most 2.96 percentage points (pp) in terms of meta classification accuracy. In terms of meta classification AuROC, the linear model outperforms all other metamodels of at most 0.8 pp. For the bounding box features and LMD, random forest or gradient boosting are in most cases the best metamodels in terms of meta classification accuracy and AuROC. In general, the bounding box features outperform the score baseline and LMD outperforms the bounding box features.

Table 7.6 states meta regression R^2 for different meta regression models for the KITTI, nuScenes and Aptiv test datasets. Random forest and gradient boosting outperforms the linear model (ridge regression) and the MLP in every case for the bounding box features and LMD. Except for CenterPoint on Aptiv, both MLP and gradient boosting are the

Table 7.5: Comparison of meta classification accuracy and AuROC for the score baseline, bounding box features and LMD for all available network-dataset combinations with IoU_{BEV} threshold 0.5 and score threshold $\tau_s = 0.1$. Models used for meta classification are Logistic Regression (LogReg), Gradient Boosting (GB), Random Forest (RF) and a Multi Layer Perceptron (MLP). Higher values are better. Bold numbers indicate the highest performance and underlined numbers represent the second highest (row-wise).

			Accuracies				AuROCs			
Dataset	Network	Method	LogReg	RF	GB	MLP	LogReg	RF	GB	MLP
KITTI	PointPillars	Score	**0.8955**	0.8848	0.8921	0.8940	**0.9566**	0.9497	0.9530	**0.9566**
		Box Features	**0.8961**	0.8958	0.8931	0.8846	**0.9564**	0.9548	0.9537	0.9482
		LMD	0.9000	**0.9028**	0.9004	0.8844	0.9589	**0.9621**	0.9592	0.9479
	CenterPoint	Score	**0.8726**	0.8651	0.8688	0.8725	**0.9322**	0.9242	0.9274	**0.9322**
		Box Features	**0.8727**	0.8719	0.8691	0.8608	0.9244	**0.9387**	0.9343	0.9271
		LMD	0.8818	**0.8847**	0.8806	0.8700	0.9421	**0.9522**	0.9466	0.9362
nuScenes	PointPillars	Score	0.8402	0.8106	0.8398	**0.8403**	**0.8151**	0.8130	0.8129	0.8150
		Box Features	0.8409	0.8613	**0.8708**	0.8653	0.8499	**0.9018**	0.9002	0.8957
		LMD	0.8875	0.8842	**0.8915**	0.8908	0.9208	0.9257	**0.9280**	0.9252
	CenterPoint	Score	**0.8777**	0.8524	0.8772	0.8773	**0.8644**	0.8617	0.8623	0.8640
		Box Features	0.8877	0.9049	**0.9203**	0.8975	0.9056	0.9454	**0.9529**	0.9293
		LMD	0.9118	0.9166	**0.9297**	0.9200	0.9450	0.9581	**0.9628**	0.9530
Aptiv	PointPillars	Score	**0.7956**	0.7929	0.7939	0.7954	**0.8582**	0.8555	0.8558	0.8580
		Box Features	0.8184	**0.8508**	0.8489	0.8472	0.8933	**0.9289**	0.9274	0.9255
		LMD	0.8506	0.8599	**0.8615**	0.8515	0.9273	0.9382	**0.9396**	0.9300
	CenterPoint	Score	**0.8279**	0.8149	0.8265	**0.8279**	**0.8946**	0.8900	0.8914	**0.8946**
		Box Features	0.8340	**0.8452**	0.8440	0.8439	0.9029	**0.9155**	0.9134	0.9085
		LMD	0.8478	**0.8559**	0.8548	0.8529	0.9187	**0.9294**	0.9275	0.9227

Table 7.6: Comparison of meta regression R^2 for the score baseline, bounding box features and LMD for all available network-dataset combinations using a IoU_{BEV} threshold of 0.5 and score threshold of 0.1. Models used for meta regression are Ridge Regression (RR), Gradient Boosting (GB), Random Forest (RF) and a Multi Layer Perceptron (MLP). Higher values are better. Bold numbers indicate the highest performance and underlined numbers represent the second highest (row-wise).

			R^2			
Dataset	Network	Method	RR	RF	GB	MLP
KITTI	PointPillars	Score	0.6901	0.6726	0.7108	**0.7146**
		Box Features	0.6973	0.7044	**0.7131**	0.6819
		LMD	0.7151	**0.7301**	0.7287	0.6837
	CenterPoint	Score	0.6220	0.5877	0.6235	**0.6273**
		Box Features	0.6260	0.6446	**0.6472**	0.6274
		LMD	0.6631	**0.6863**	0.6840	0.6538
nuScenes	PointPillars	Score	0.3903	0.4006	**0.4055**	0.4054
		Box Features	0.4187	0.5586	**0.5593**	0.5356
		LMD	0.6105	0.6346	**0.6413**	0.6244
	CenterPoint	Score	0.4641	0.4675	0.4733	**0.4751**
		Box Features	0.5292	0.6681	**0.6792**	0.6249
		LMD	0.6451	0.7242	**0.7296**	0.7122
Aptiv	PointPillars	Score	0.5005	0.5013	0.5096	**0.5106**
		Box Features	0.5469	0.6484	**0.6568**	0.6482
		LMD	0.6401	0.6830	**0.6924**	0.6614
	CenterPoint	Score	0.5458	0.5329	0.5456	**0.5469**
		Box Features	0.5749	0.6200	**0.6286**	0.6136
		LMD	0.6210	0.6541	**0.6591**	0.6332

Table 7.7: Comparison of meta classification accuracy and AuROC as well as meta regression R^2 for the score baseline, bounding box features and LMD for all available network+dataset combinations with $\mathrm{IoU}_{\mathrm{BEV}}$ threshold 0.5, score threshold $\tau_s = 0.1$ and GB as metamodel.

		Meta Classification $\mathrm{IoU}_{\mathrm{BEV}} \geq 0.5$ vs. $\mathrm{IoU}_{\mathrm{BEV}} < 0.5$						Meta Regression $\mathrm{IoU}_{\mathrm{BEV}}$		
		Accuracies			AuROCs			R^2		
Dataset	Network	Score	Box	LMD	Score	Box	LMD	Score	Box	LMD
KITTI	PointPillars	0.8921	0.8931	**0.9004**	0.9530	0.9537	**0.9592**	0.7108	0.7131	**0.7287**
	CenterPoint	0.8688	0.8691	**0.8806**	0.9274	0.9343	**0.9466**	0.6235	0.6472	**0.6840**
nuScenes	PointPillars	0.8398	0.8708	**0.8915**	0.8129	0.9002	**0.9280**	0.4055	0.5593	**0.6413**
	CenterPoint	0.8772	0.9203	**0.9297**	0.8623	0.9529	**0.9628**	0.4732	0.6792	**0.7296**
Aptiv	PointPillars	0.7939	0.8489	**0.8615**	0.8558	0.9274	**0.9396**	0.5096	0.6568	**0.6924**
	CenterPoint	0.8265	0.8440	**0.8548**	0.8914	0.9134	**0.9275**	0.5456	0.6286	**0.6591**

superior metamodels (compared to random forest and ridge regression) in terms of meta regression R^2 for the score baseline. Comparable to the results of table 7.4 and the results for meta classification in table 7.5, the bounding box features outperform the score baseline and LMD outperforms the bounding box features.

7.4.4 Generalization over Datasets and Networks.

Table 7.7 shows meta classification accuracy and AuROC as well as meta regression R^2 for all network+dataset combinations based on GB models. In all cases LMD outperforms both baselines and the bounding box features outperform the score baseline. This is to be expected, since the score is contained in the box features and the box features are contained in the set of features of LMD. The improvement from the score baseline to LMD ranges from 0.83 to 6.76 pp in terms of meta classification accuracy, from 0.62 to 10.51 pp in terms of AuROC and from 1.79 to 25.64 pp in terms of meta regression R^2. The improvement from the bounding box features to LMD ranges from 0.73 to 2.07 pp in terms of meta classification accuracy, from 0.55 to 2.78 pp in terms of AuROC and from 1.56 to 8.20 pp in terms of meta regression R^2. This illustrates that the addition of features, other than just the bounding box features of the prediction itself, has a significant impact on meta classification and meta regression performances and, therefore, separation of TP and FP predictions.

For CenterPoint and nuScenes, the confusion matrix fig. 7.5 shows that the GB classifier based on LMD identifies most TPs and true negatives. Therefore, predictions that are in fact FPs are also predicted as FPs. Note, that here we regard “meta” true negatives conditional on the detector’s prediction (each example is a detection TP or FP that is binarily classified). The values on the off-diagonals indicate the errors of the meta classifier. 7,002 predictions are predicted as FPs even though they are TPs. In contrast, 4,484 predictions are predicted as TPs, even though they are actually FPs. Figure 7.6 shows a scatter plot of the true $\mathrm{IoU}_{\mathrm{BEV}}$ of prediction and ground truth and the $\mathrm{IoU}_{\mathrm{BEV}}$ estimated by LMD meta regression based on a GB model, where each point represents one prediction. Well-concentrated points around the identity (dashed line) indicate well-calibrated

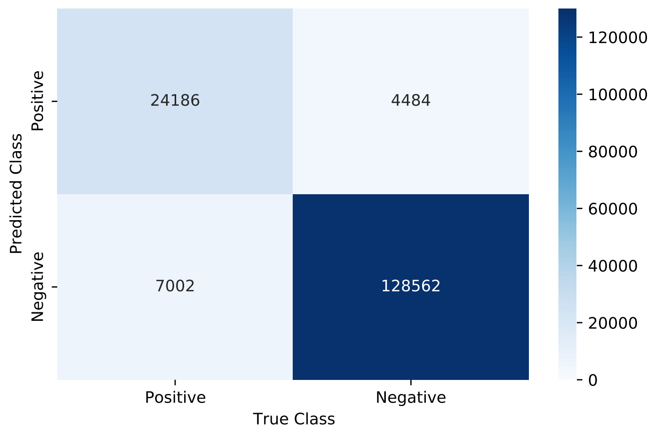


Figure 7.5: Confusion matrix of a GB classifier for LMD on CenterPoint, nuScenes and score threshold $\tau_s = 0.1$.

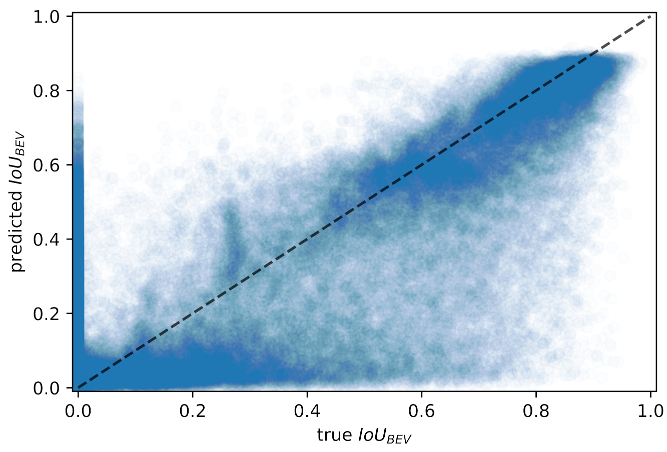
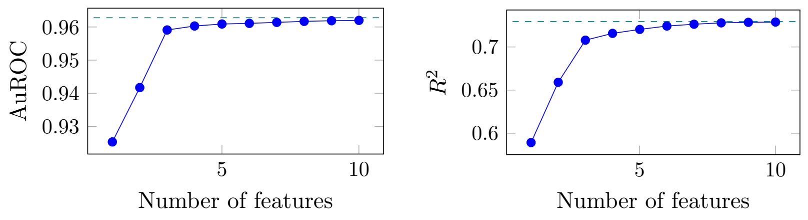


Figure 7.6: Box-wise scatter plot of true $\mathrm{IoU}_{\mathrm{BEV}}$ and predicted $\mathrm{IoU}_{\mathrm{BEV}}$ values for LMD on CenterPoint, nuScenes and score threshold $\tau_s = 0.1$. The predictions are based on a GB regressor.



(a) Meta Classification (b) Meta Regression

Figure 7.7: Feature selection using a greedy heuristic for CenterPoint, nuScenes and score threshold $\tau_s = 0.1$. (a) contains meta classification AuROC and (b) contains meta regression R^2. The dashed line shows the performance when incorporating all features (LMD).

Table 7.8: Feature selection for meta classification AuROC using a greedy heuristic for all network-dataset combinations, score threshold $\tau_s = 0.1$ and a GB classifier. The right-most column shows the performance when incorporating all features (LMD).

Meta Classification												
		Number of Features										
Network	Meta Model	1	2	3	4	5	6	7	8	9	10	All
KITTI	PointPillars	0.9482	0.9512	0.9530	0.9543	0.9564	0.9576	0.9580	0.9584	0.9587	0.9588	0.9592
	CenterPoint	0.9149	0.9268	0.9372	0.9405	0.9420	0.9439	0.9452	0.9458	0.9462	0.9463	0.9466
nuScenes	PointPillars	0.9117	0.9175	0.9214	0.9248	0.9253	0.9257	0.9259	0.9263	0.9268	0.9273	0.9280
	CenterPoint	0.9253	0.9417	0.9591	0.9603	0.9609	0.9611	0.9614	0.9617	0.9619	0.9620	0.9628
Aptiv	PointPillars	0.8940	0.9182	0.9233	0.9288	0.9312	0.9330	0.9339	0.9348	0.9352	0.9356	0.9396
	CenterPoint	0.9051	0.9150	0.9177	0.9195	0.9209	0.9226	0.9242	0.9255	0.9259	0.9268	0.9275

$\mathrm{IoU}_{\mathrm{BEV}}$-estimates and, therefore, object-wise quality estimates.

7.4.5 Feature Selection for Meta Classification and Meta Regression.

Overall, LMD is based on 90 features, which partly describe very similar properties. In order to get a subset of features which contains as few redundancies as possible but is still powerful, we apply a greedy heuristic. Starting with an empty set, a single feature that improves the meta prediction performance maximally is added iteratively. Figure 7.7 shows results in terms of AuROC for meta classification and in terms of R^2 for meta regression for CenterPoint on nuScenes. The tests for the meta classification and the meta regression are independent of each other, i.e., the selected features of the two saturation plots do not have to match. When using five selected features, the associated metamodels perform already roughly as well as when using all features (LMD), i.e., 0.19 pp worse in terms of meta classification AuROC and 0.92 pp worse in terms of meta regression R^2. With ten features used, the respective differences with the results obtained by LMD are < 0.1 pp and thus negligible. The tests for the greedy selection heuristic for all network+dataset combinations are shown in the following paragraph.

Extended Feature Selection Results. Table 7.8 shows feature selection results in terms of meta classification AuROC and table 7.9 shows feature selection results in terms of meta regression R^2. For both tasks, meta classification and regression, a few features

Table 7.9: Feature selection for meta regression R^2 using a greedy heuristic for all network-dataset combinations, score threshold $\tau_s = 0.1$ and a GB regressor. The right-most column shows the performance when incorporating all features (LMD).

		Meta Regression										
		Number of Features										
Network	Meta Model	1	2	3	4	5	6	7	8	9	10	All
KITTI	PointPillars	0.7053	0.7121	0.7165	0.7195	0.7217	0.7229	0.7237	0.7248	0.7260	0.7267	0.7287
	CenterPoint	0.6134	0.6449	0.6626	0.6746	0.6776	0.6791	0.6808	0.6815	0.6824	0.6833	0.6840
nuScenes	PointPillars	0.5931	0.6069	0.6174	0.6265	0.6318	0.6359	0.6383	0.6390	0.6397	0.6401	0.6413
	CenterPoint	0.5892	0.6591	0.7079	0.7158	0.7204	0.7243	0.7264	0.7280	0.7286	0.7289	0.7296
Aptiv	PointPillars	0.5860	0.6300	0.6454	0.6615	0.6679	0.6740	0.6774	0.6806	0.6845	0.6875	0.6924
	CenterPoint	0.5856	0.6301	0.6358	0.6400	0.6472	0.6500	0.6537	0.6558	0.6564	0.6572	0.6591

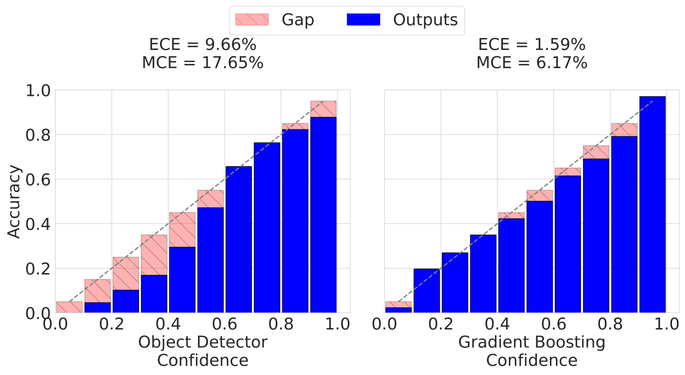
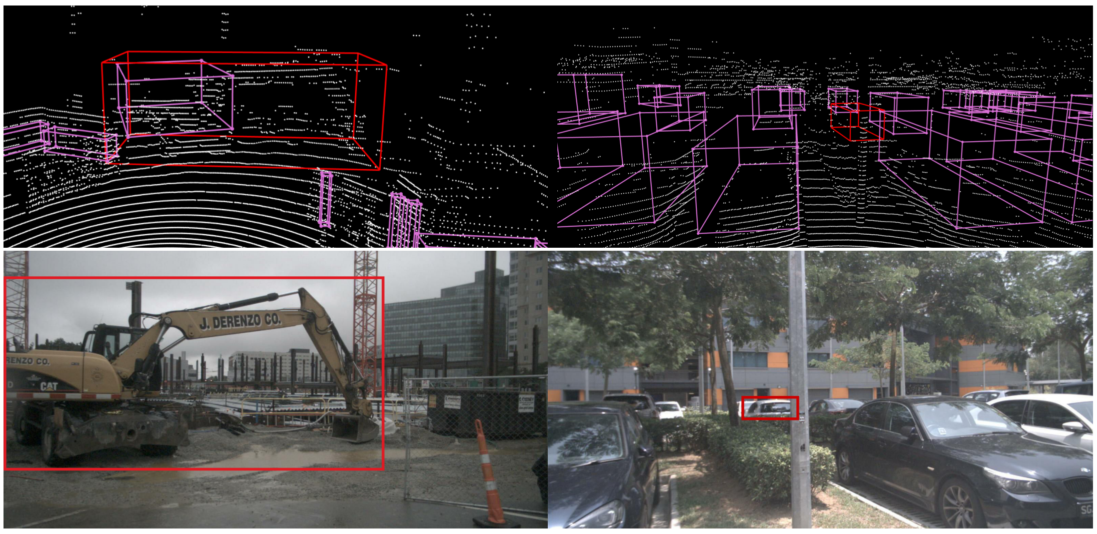


Figure 7.8: Reliability plots of the score (*left*) and GB classifier for LMD (*right*) with calibration errors (*ECE*, *MCE*) for CenterPoint, nuScenes test dataset, score threshold $\tau_s = 0.1$ and IoU_{BEV} threshold 0.5.

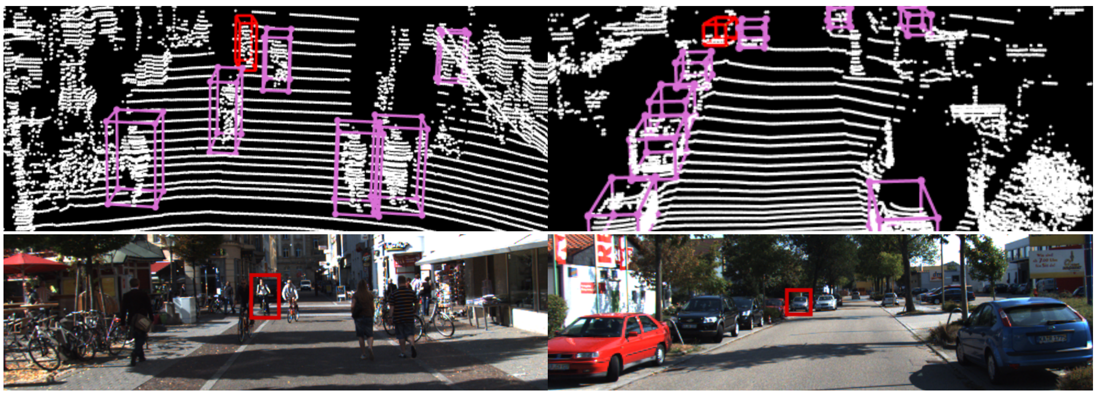
are sufficient to reach roughly the same performance as when using all features (LMD). Metamodels using five or more selected features are at most 0.84 pp below the performance of LMD in terms of meta classification AuROC and at most 2.45 pp in terms of meta regression R^2. With ten features used, the respective differences to the performance results achieved by LMD are ≤ 0.4 pp in terms of meta classification AuROC and < 0.5 pp in terms of meta regression R^2.

7.4.6 Confidence Calibration

The score and the meta classifier confidences are divided into 10 confidence bins to evaluate their calibration errors. Figure 7.8 shows exemplary reliability plots for the object detector score and LMD based on a GB classifier with corresponding expected (*ECE* [126]) and maximum calibration error (*MCE* [126]). Refer to section 3.3.2 for a treatment of these calibration metrics. The score is over-confident in the lower confidence ranges and well-calibrated in the upper confidence ranges, whereas the GB classifier for LMD is well-calibrated over all confidence ranges. This observation is also reflected in the corresponding calibration errors, as the GB classifier for LMD outperforms the score by 8.07 pp in terms of *ECE* and by 11.48 pp in terms of *MCE*. This indicates that LMD improves the statistical



(a) nuScenes



(b) KITTI

Figure 7.9: Proposed label errors in (a) nuScenes and (b) KITTI. Top images show point clouds with annotations in purple and the proposal in red. Camera images aid the evaluation.

reliability of the confidence assignment.

7.4.7 Label Error Detection as an Application of Meta Classification.

The task of annotations error detection with LMD is inspired by fig. 7.6. There are a number of predictions with $\mathrm{IoU}_{\mathrm{BEV}} = 0$ but with high predicted $\mathrm{IoU}_{\mathrm{BEV}}$. After looking at these FPs it has been noticed that not the prediction is incorrect itself but the corresponding ground truth. More precisely, incorrect ground truth corresponds to missing labels, labels with a wrong assigned class or the location of the annotation is inaccurate, i.e., the 3D bounding box is not correctly aligned with the point cloud. Annotation error detection with LMD works as follows: all FP predictions, i.e., predictions that have $\mathrm{IoU}_{\mathrm{BEV}} < 0.5$ with the ground truth, are sorted by the predicted $\mathrm{IoU}_{\mathrm{BEV}}$ in descending

Table 7.10: Comparison of detected annotation errors for the KITTI test dataset using object detectors as baselines and RF as best LMD classifier with a score threshold of 0.1 and a IoU_{BEV} threshold of 0.5.

KITTI Annotation Error Analysis (RF)				
Network	Classes	Random	Score	LMD
PointPillars	Pedestrian	8/53	1/1	2/4
	Cyclist	3/22	1/1	2/2
	Car	9/25	76/98	88/94
	Overall	**20/100**	**78/100**	**92/100**
CenterPoint	Pedestrian	4/25	25/40	7/7
	Cyclist	1/11	8/12	1/1
	Car	22/64	38/48	89/92
	Overall	**27/100**	**71/100**	**97/100**

order across all images. Then, the first 100 predictions, i.e., the top 100 FPs with the highest predicted IoU_{BEV}, are manually reviewed, see fig. 7.9 for examples of proposals by this method. In this case, a GB classifier is used to predict the box-wise IoU_{BEV}. We compare LMD against a score baseline which works in the same way, except that the FPs are sorted by the objectness score. As a random baseline, 100 randomly drawn FPs are considered for review which provides an insight into how well the respective test dataset is labeled. In general, if it was unclear whether an annotation error was present or not, this case was not marked as annotation error, i.e., the following numbers are a conservative (under-)estimation. LMD finds 43 annotation errors from 100 proposals and, in contrast, the score only 6 out of 100. Even the random baseline still finds 3 annotation errors, which indicates that there are a significant number of annotation error in the nuScenes test dataset and that these can be found at far smaller effort with LMD than with the score. Explicit annotation error detection counts for PointPillars on nuScenes and for both networks on the KITTI test dataset are shown in the following section.

Extended Results on Annotation Error Detection. Table 7.10 presents annotation error detection results for the KITTI test dataset. In each experiment, we manually reviewed 100 candidates provided by a given detection method. For the random baseline (randomly reviewing FPs of the network) applied to PointPillars, we discover 20 annotation errors. Using CenterPoint, this number increases to 27 annotations errors with the random baseline and CenterPoint, which indicates that there might be a significant number of annotation errors in the KITTI test dataset. This is already confirmed by the score baseline. Combining it with PointPillars, we find 78 annotation errors and with CenterPoint, we find 71. Although these numbers seem enormous, LMD is capable of detecting even more annotation errors. With PointPillars we detect 92 and with CenterPoint 97 annotation errors.

Table 7.11 shows annotation error detection results for the nuScenes test dataset. We detect only 6 annotation errors using the score baseline and CenterPoint, whereas we can find 43 annotation errors using LMD and CenterPoint. Considering PointPillars, the gap between the detection results of the score baseline and LMD vanishes. With the

Table 7.11: Comparison of detected annotation errors for the nuScenes test dataset using object detectors as baselines and GB as best LMD classifier with a score threshold of 0.1 and a IoU_{BEV} threshold of 0.5.

nuScenes Annotation Error Analysis (GB)				
Network	Classes	Random	Score	LMD
PointPillars	Car	1/27	10/55	13/49
	Pedestrian	1/29	4/10	–
	Barrier	0/18	–	–
	Traffic Cone	0/10	–	–
	Truck	0/6	13/23	12/30
	Trailer	0/1	0/10	1/6
	Bicycle	–	–	–
	Construction Vehicle	0/4	–	0/6
	Bus	0/2	0/2	2/9
	Motorcycle	0/3	–	–
	Overall	**2/100**	**27/100**	**28/100**
CenterPoint	Car	2/51	0/8	31/62
	Pedestrian	–	5/14	2/2
	Barrier	0/8	1/15	0/1
	Traffic Cone	–	0/4	–
	Truck	0/14	0/3	8/30
	Trailer	0/15	0/21	0/1
	Bicycle	–	–	–
	Construction Vehicle	0/14	–	2/3
	Bus	1/8	0/33	0/1
	Motorcycle	–	0/2	–
	Overall	**3/100**	**6/100**	**43/100**

score baseline we detect 27 and with LMD 28 annotation errors. This observation is in agreement with table 7.5, where CenterPoint achieves superior AuROC values compared to PointPillars on the nuScenes test dataset. Supplementing the samples of our annotation error proposal method shown above, we show additional proposals for the nuScenes test dataset in fig. 7.10 and respectively for the KITTI test dataset in fig. 7.11.

7.5 Conclusion: Prediction Quality Estimation for Lidar Object Detection

In this chapter we have introduced a purely post-processing-based uncertainty quantification method (LMD). A post-processing module which is simple to fit and can be plugged onto any pre-trained Lidar object detector allows for swift estimation of confidence (meta classification) and localization precision (meta regression) in terms of $\mathrm{IoU}_{\mathrm{BEV}}$ at inference time. Our experiments show that separation of true and false predictions obtained from LMD is sharper than that of the base detector. Statistical reliability is significantly improved in terms of calibration of the obtained confidence scores and $\mathrm{IoU}_{\mathrm{BEV}}$ is estimated to considerable precision at inference time, i.e., without knowledge of the ground truth. In addition to statistical improvement in decision-making, we introduce a method for detecting annotation errors in real-world datasets based on our uncertainty estimation method. Error counts of hand-reviewed proposals which are shown for broadly used public benchmark datasets suggest a highly beneficial industrial use case of our method beyond improving prediction reliability.

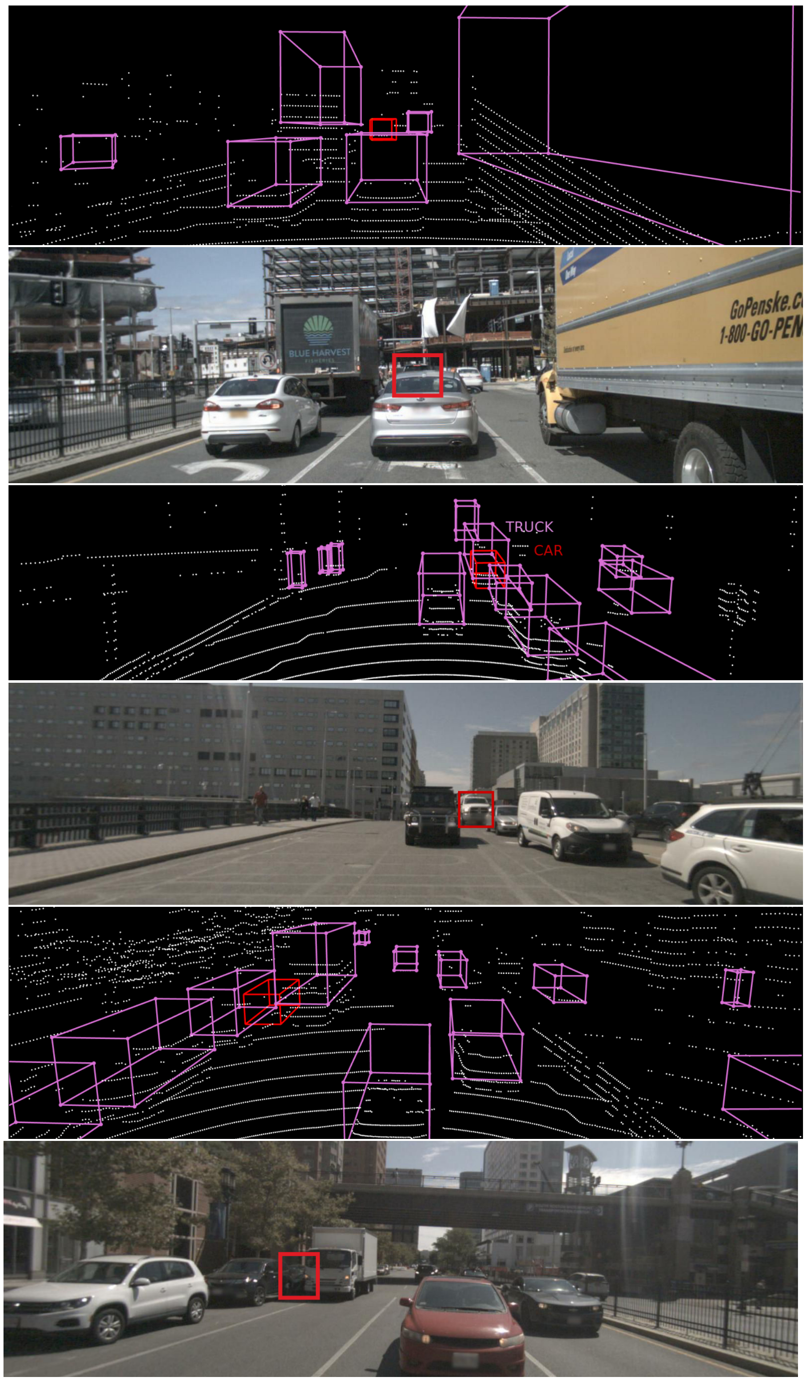


Figure 7.10: Additional annotation error proposals on the nuScenes test dataset.

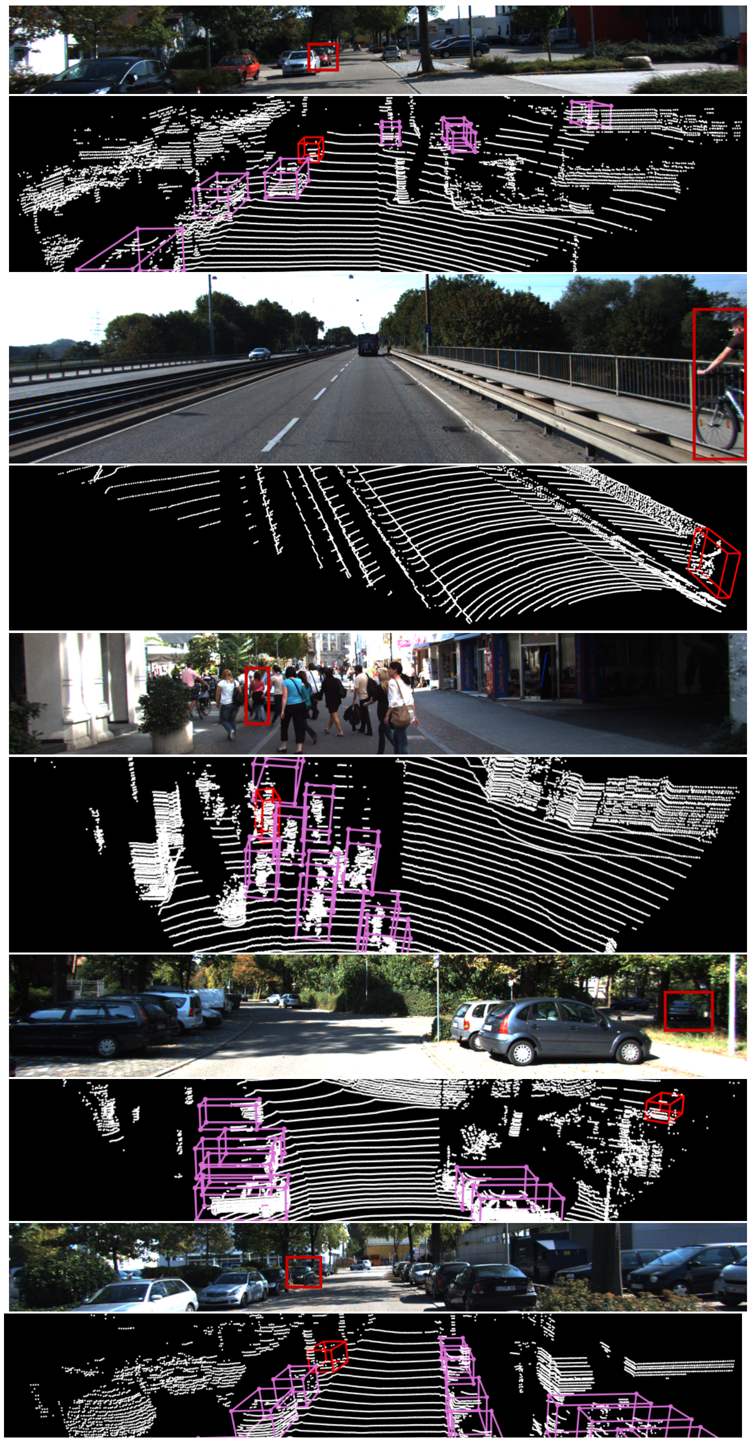


Figure 7.11: Additional annotation error proposals on the KITTI test dataset.

8 Conclusion and Outlook

In this thesis, we presented new approaches to quantify prediction uncertainty for deep neural networks in object detection (chapters 3 and 7) and semantic segmentation (chapter 4). The developed methods have some immediate applications in detecting prediction errors (chapters 3, 4 and 7) like false positives and false negatives and the segmentation of out-of-distribution objects (chapter 4). Additionally, secondary applications of uncertainty quantification were also investigated. Such secondary applications of predictive uncertainty quantification are active learning (chapter 5) and the automated detection of label errors (chapters 6 and 7).

The following paragraphs contain relations and comparisons between the new insights gained by the novel approaches in the presented studies. Starting from commonalities and differences between the presented methods, we spotlight open questions and spaces of interaction. With industrial use cases in mind we identify potential areas of future research and follow-up work.

Uncertainty Quantification Methods. In chapter 3 we investigated the usage of loss gradients computed per predicted instance in object detection as a measure or predictive uncertainty. Gradient uncertainty can be seen as epistemic as discussed in section 3.3.1.2 which is supported by empirical findings already present in the literature. Investigations in terms of meta classification and meta regression (cf. section 3.3.2) show that the proposed method is on par with and oftentimes superior to sampling-based approaches. Confidence scores from meta classification turn out to be well-calibrated since meta classification can be regarded as multivariate re-calibration based on the training dataset of the meta classifier. The computation of loss gradients, while being computationally expensive, can be shown to be still less computationally complex than sampling-based approaches which can also be seen in explicit runtime measurements. The investigation of the theoretical computational cost of instance-wise gradients in convolutional architectures led to insights into efficient computations of gradients on a feature map level. These insights eventually prompted the investigation of pixel-wise gradient scores in semantic segmentation in chapter 4. The computation of pixel-wise gradients is achieved by the particularly simple form of the gradient for the last convolutional layer of a segmentation neural network. Our investigations show that some gradient scores yield remarkably well-calibrated confidences when evaluated pixel-wise, i.e., even without training a meta classifier on validation data. When aggregated over segments in the same manner as in the MetaSeg framework, gradient score heatmaps give rise to strong meta classifiers. Gradient scores have been found to give rise to strong out-of-distribution detectors in previous work [69, 132], so our investigation also contains a heavy focus on out-of-distribution segmentation. The proposed method achieves high out-of-distribution recognition performance on the SegmentMeIfYouCan benchmark. This is so especially when considering its small computational overhead and absence of additional requirements like out-of-distribution data, specialized training or architectural changes. Considering these discoveries and the *recent emergence of OoD detection in the object detection* context opens the question whether gradient uncertainty can be applied in this domain. For example, regarding the self-learning gradient of the instance-wise classification loss might be applied in object detection to recognize OoD objects. Perhaps anchor-free object detection models are particularly well-suited to this task due to their similarity to segmentation architectures. The approach presented in chapter 3 itself is likely less suited to this task due to the meta classifier being optimized to favor true foreground instances. Note that the meta classifier is trained on in-distribution validation data. When testing on data which has a *large domain shift* compared to this validation data, deterioration of meta classification performance is to be expected due to the model's training bias. This can be presumed to influence meta classification performance and calibration on segment level in semantic segmentation and also in object detection. Although provably more efficient than sampling approaches, the *implementation of computing gradient uncertainty can come at large effort* due to the complexity of the multi-criterial loss function which is much simpler in semantic segmentation. Such an explicit implementation can be circumvented by computing gradients via the autograd framework which is, in turn, slow when done instance-wise in an iterative manner. Then, gradient uncertainty may be less beneficial in online supervision and more useful in offline applications like active learning or as an additional input to meta classification applied in label error detection. These applications allow the combination of different sources of uncertainty which also goes for the gradient heatmaps developed in chapter 4 as supplement for the MetaSeg framework. While we have tested our uncertainty quantification methods with a focus on everyday scene images and street scenes, they may find *applications in other domains* of industrial importance such as medical imaging and robotics as well. Also, the *true pixel-wise learning gradient with respect to the ground truth* may contain valuable information for label error detection. Regions where training on potentially incorrect labels may lead to large parameter updates may be well-suited candidates for label error proposals.

The proposed post-processing-based uncertainty quantification method for lidar object detection presented in chapter 7 was inspired by the original MetaDetect framework which was investigated as a baseline in chapter 3. Both approaches are conceptually similar in that they use meta classification and meta regression on features which use the network output before non-maximum suppression. However, LidarMetaDetect deals with the peculiarities of object detection in point clouds. In addition to accessing features from three-dimensional bounding boxes, also the information of the input that lies in the spatial region of the prediction is inherently different. Whereas on camera images, the amount of information from the input within a bounding box is always directly proportional to the area of the bounding box (all pixels overlapping with the prediction), a variable number of lidar points may fall into the three-dimensional volume of a predicted bounding box. This allows for additional uncertainty scores which can be derived from the prediction and used for meta classification. Comparing meta classification and meta regression results on the KITTI dataset for the original MetaDetect and LidarMetaDetect, we see that the original MetaDetect on camera images still achieves higher values in AuROC and R^2. However, there may be several reasons for this observation. Detecting two-dimensional bounding boxes on camera images may be a *much simpler task than lidar object detection*. Further, the base performance of the object detector may have an influence on both, meta classification and meta regression. The latter part is not only connected to the network architecture but also the training pipeline involving data augmentation, schedule and optimizer details. Investigations into what kind of meta classification or meta regression model is most suitable, both the results in MetaDetect and in LidarMetaDetect agree that tree-based models like random forests and gradient boosting models outperform linear models and shallow neural networks. Our investigations involving an industrial proprietary dataset show that LidarMetaDetect yields well-performing confidence scores on lidar object detections. This suggests an *industrial use case in the automotive sector* for light-weight post-processing mechanisms.

Uncertainty Queries in Active Learning for Object Detection. The development of active learning strategies based on prediction uncertainty in object detection is highly sensitive to hyperparameters such as IoU and confidence thresholds which determine the prediction. The reason for this is that commonly, uncertainty is meaningful on the level of each predicted instance separately and the previously mentioned hyperparameters, therefore, quantify the uncertainties considered in the query mechanism. The sandbox environment introduced in chapter 5 allows for quick prototyping, implementation and hyperparameter development of new active learning strategies. Especially with increasing model and dataset complexity, the time reduction achieved makes ablation studies on vital parameters feasible. While the focus of the presented investigations was on uncertainty-based query methods, other approaches such as diversity-based querying can be investigated just the same. However, such methods have not been widely studied in the context of object detection at the time of writing. The presented datasets are generated from (E)MNIST ground truth and MS COCO background and follow certain sampling and transformation prescriptions of foreground instances. This can, in principle, be modified in order to steer the dataset by *modelling it after desired criteria on the data distribution.* This way, for example, certain localization or class imbalances can be introduced into the dataset in a controlled manner. The latter may be especially interesting in order to understand the differences in query strategies between datasets when the object detection architecture is fixed.

Identifying Label Errors in Object Detection Datasets. In chapter 6, we presented a novel approach to detecting annotation errors in object detection datasets which uses the loss function. In contrast, the approach in chapter 7 utilizing a meta classifier for lidar object detection is similar to previous work in semantic segmentation of camera images [154]. Chapter 7 and [154] share similarities in using disagreements between meta classifiers and the given ground truth as proposals, in descending order in the confidence assigned. Note that the same can be done with MetaDetect in application to object detection on camera images. However, the focus on predicted instances means that only missing annotations or annotations with incorrect category can be expected to be found. In special cases, an annotation box can be so badly localized that a prediction in its vicinity counts as a false positive leading to the annotation error proposal. Incorrectly given annotations ("spawned" annotations), are not found by purely prediction-based methods such as the application of LidarMetaDetect. The KITTI dataset shows that this is not an artificial problem since here, annotations are generated by tracking. Thus, objects sometimes completely vanish visually from one frame to the next behind other foreground instances while the annotations remain. In contrast, the loss-based method has been shown to capture such spawned annotations since each ground truth annotation generates a loss and can thus be regarded separately. Prediction-based methods can, however, be modified to also involve matching ground truth annotations with predictions, which requires to regard low-confidence predictions which was investigated in chapter 6 as a baseline. Then, also spawned annotations can be detected, however, not as sharply as via the loss approach. Both approaches, the one on camera images from chapter 6 and the one on lidar point clouds from chapter 7, find numerous annotation errors in the KITTI dataset which seem systematic. In both cases, instances that have a certain distance to the ego car are not annotated, albeit that they are visible in the camera or lidar signal. The success of both methods *suggests potential industrial use cases* in the automotive and insurance branches. Based on the label error benchmark for camera images, the injection of annotation errors into high quality datasets might provide a way to *systematically learn the detection of annotation errors* with deep learning models For such a model, the injected mismatch in the ground truth acts as the learning target.

List of Figures

List of Tables

List of Notations

The following abbreviations and notations are used across all chapters:

$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers
	2.1 Statistical Learning Theory
$(\Omega, \mathscr{A}, \mathrm{Pr})$	Generic probability space
$\mathscr{B}_{\mathcal{X}}$	Borel-σ-algebra over $\mathcal{X}$ w.r.t. some metric
$\mathscr{M}_1(\Omega, \mathscr{A})$	Space of normalized (probability) measures over $(\Omega, \mathscr{A})$
$\mathscr{K}(\mathcal{X}; (\mathcal{Y}, \mathscr{A}_{\mathcal{Y}}))$	Set of Markov kernels over $(\mathcal{Y}, \mathscr{A}_{\mathcal{Y}})$ taking values in $\mathcal{X}$
$\mathcal{X}$	Data/input space
$\mathcal{Y}$	Label/target space
$\boldsymbol{x}$	Vector in $\mathbb{R}^d$ for some $d \in \mathbb{N}$
X, Y	Random variables, $\boldsymbol{X}$ if vector-valued, (x, y) realizations
μ_X	Push-forward measure of the random variable X
$X \sim \mu_X$	X follows the probability distribution μ_X
$\mathbb{E}_{X \sim \mu_X}$	Expectation operator over random variable $X \sim \mu_X$
χ_n	Training sample $\{X_1, \ldots, X_n\}$ of size n
$\mathsf{d}(\cdot\|\cdot), \mathsf{D}(\cdot\|\cdot)$	Distance measures on $\mathscr{M}_1$ and $\mathscr{K}$, respectively
$\widehat{f}$	Predictor function $\widehat{f} : \mathcal{X} \to \mathcal{Y}$
$\widehat{\mu}_n$	Estimate measure given χ_n, $\{\widehat{\mu}_n\}_{n \in \mathbb{N}}$ learning algorithm
$\mathscr{H}$	Hypothesis space of learning algorithm $\mathscr{H}_n := \mathrm{Img}\,(\widehat{\mu}_n)$
$\mathscr{L}(\chi_n \| \nu)$	Likelihood of χ_n under ν
	2.2 Deep Learning
$[n]$	$\{1, \ldots, n\}$ for $n \in \mathbb{N}$
$[n : m]$	$\{n, n+1, \ldots, m-1, m\}$ for $n < m \in \mathbb{N}$
H, W	Height (# rows) and width (# columns) of an image
$K * \psi$	Convolution of filter K over feature map ψ
X	Embedded vector $\mathsf{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ of n tokens
$\boldsymbol{\Sigma}$	Softmax activation vector

$\varTheta$	Parameter space $\varTheta \subseteq \mathbb{R}^q$ of model parameters $\boldsymbol{\theta} \in \varTheta$
$D_{\boldsymbol{\theta}}$	"Total" differential with respect to variable $\boldsymbol{\theta}$
$C(I_d)$	Space of continuous functions over $I_d = [0,1]^d$
$\mathscr{M}(I_d)$	Space of regular signed measures over I_d
$W^{n,k}(I_d)$	Space of Sobolev functions of regularity n and integrability k
$\\| \cdot \\|_{L^p}$	Lebesgue norm $\\|f\\|_{L^p} = \big(\int_{\mathbb{R}^d} \\|f(x)\\|_p^p \,\mathrm{d}x\big)^{1/p}$, $f \in L^p(\mathbb{R}^d; \mathbb{R}^n)$
	2.3 Advanced Computer Vision Tasks
$\mathcal{I}$	Image pixels $\mathcal{I} = [H] \times [W]$
$\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h}$	Bounding box coordinates: center (x, y), size (w, h)
b	Bounding box $\mathrm{b} = (\xi, \kappa)$ with $\xi = (\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h})$ and class $\kappa \in [C]$
$\widehat{\pi}, \widehat{s}$	Predicted class distribution $\widehat{\pi} = (\widehat{\pi}_1, \ldots, \widehat{\pi}_C)$ and objectness score $\widehat{s}$

Bibliography

[1] M. Angus, K. Czarnecki, and R. Salay, *Efficacy of Pixel-Level OOD Detection for Semantic Segmentation*, arXiv preprint arXiv:1911.02897, (2019).

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer Normalization*, arXiv preprint arXiv:1607.06450, (2016).

[3] V. Besnier, A. Bursuc, D. Picard, and A. Briot, *Triggering Failures: Out-Of-Distribution detection by learning from local adversarial attacks in Semantic Segmentation*, in 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, Oct. 2021, IEEE, pp. 15681–15690.

[4] H. Blum, P.-E. Sarlin, J. Nieto, R. Siegwart, and C. Cadena, *Fishyscapes: A Benchmark for Safe Semantic Segmentation in Autonomous Driving*, in 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Seoul, Korea (South), Oct. 2019, IEEE, pp. 2403–2412.

[5] ——, *The fishyscapes benchmark: Measuring blind spots in semantic segmentation*, International Journal of Computer Vision, 129 (2021), pp. 3119–3135.

[6] M.-R. Bouguelia, Y. Belaïd, and A. Belaïd, *Identifying and mitigating labelling errors in active learning*, in Pattern Recognition: Applications and Methods: 4th International Conference, ICPRAM 2015, Lisbon, Portugal, January 10-12, 2015, Revised Selected Papers 4, Springer, 2015, pp. 35–51.

[7] M.-R. Bouguelia, S. Nowaczyk, KC. Santosh, and A. Verikas, *Agreeing to disagree: Active learning with noisy labels without crowdsourcing*, International Journal of Machine Learning and Cybernetics, 9 (2018), pp. 1307–1319.

[8] C.-A. Brust, C. Käding, and J. Denzler, *Active Learning for Deep Object Detection*, arXiv preprint arXiv:1809.09875, (2018).

[9] S. Budd, E. C. Robinson, and B. Kainz, *A survey on active learning and human-in-the-loop deep learning for medical image analysis*, Medical Image Analysis, 71 (2021), p. 102062.

[10] M. Büttner, L. Schneider, A. Krasowski, J. Krois, B. Feldberg, and F. Schwendicke, *Impact of noisy labels on dental deep Learning—Calculus detection on bitewing radiographs*, Journal of Clinical Medicine, 12 (2023), p. 3058.

[11] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, *nuScenes: A Multimodal Dataset for Autonomous Driving*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, June 2020, IEEE, pp. 11618–11628.

[12] Z. Cai and N. Vasconcelos, *Cascade R-CNN: Delving Into High Quality Object Detection*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, June 2018, IEEE, pp. 6154–6162.

[13] S. Chadwick and P. Newman, *Training object detectors with noisy data*, in 2019 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2019, pp. 1319–1325.

[14] R. Chan, K. Lis, S. Uhlemeyer, H. Blum, S. Honari, R. Siegwart, P. Fua, M. Salzmann, and M. Rottmann, *SegmentMeIfYouCan: A benchmark for anomaly segmentation*, in Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, J. Vanschoren and S. Yeung, eds., vol. 1, Curran, 2021.

[15] R. Chan, M. Rottmann, and H. Gottschalk, *Entropy Maximization and Meta Classification for Out-of-Distribution Detection in Semantic Segmentation*, in 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, Oct. 2021, IEEE, pp. 5108–5117.

[16] R. Chan, M. Rottmann, F. Huger, P. Schlicht, and H. Gottschalk, *Controlled False Negative Reduction of Minority Classes in Semantic Segmentation*, in 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, July 2020, IEEE, pp. 1–8.

[17] H. Chen, Y. Huang, W. Tian, Z. Gao, and L. Xiong, *MonoRUn: Monocular 3D Object Detection by Reconstruction and Uncertainty Propagation*, in 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, June 2021, IEEE, pp. 10374–10383.

[18] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, *MMDetection: Open MMLab detection toolbox and benchmark*, arXiv preprint arXiv:1906.07155, (2019).

[19] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, *Rethinking Atrous Convolution for Semantic Image Segmentation*, arXiv preprint arXiv:1706.05587, (2017).

[20] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*, in Computer Vision – ECCV 2018, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds., vol. 11211, Springer International Publishing, Cham, 2018, pp. 833–851.

[21] P. Chen, B. B. Liao, G. Chen, and S. Zhang, *Understanding and utilizing deep neural networks trained with noisy labels*, in International Conference on Machine Learning, PMLR, 2019, pp. 1062–1070.

[22] T. Chen and C. Guestrin, *XGBoost: A Scalable Tree Boosting System*, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco California USA, Aug. 2016, ACM, pp. 785–794.

[23] J. Choi, I. Elezi, H.-J. Lee, C. Farabet, and J. M. Alvarez, *Active Learning for Deep Object Detection via Probabilistic Modeling*, in 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, Oct. 2021, IEEE, pp. 10244–10253.

[24] F. Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, July 2017, IEEE, pp. 1800–1807.

[25] G. Citovsky, G. DeSalvo, C. Gentile, L. Karydas, A. Rajagopalan, A. Rostamizadeh, and S. Kumar, *Batch active learning at scale*, Advances in Neural Information Processing Systems, 34 (2021), pp. 11933–11944.

[26] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, *EMNIST: Extending MNIST to handwritten letters*, in 2017 International Joint Conference on Neural Networks (IJCNN), May 2017, pp. 2921–2926.

[27] C. Corbière, N. Thome, A. Bar-Hen, M. Cord, and P. Pérez, *Addressing failure prediction by learning model confidence*, Advances in Neural Information Processing Systems, 32 (2019).

[28] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, *The Cityscapes Dataset for Semantic Urban Scene Understanding*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, June 2016, IEEE, pp. 3213–3223.

[29] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of control, signals and systems, 2 (1989), pp. 303–314.

[30] I. Dagan and S. P. Engelson, *Committee-based sampling for training probabilistic classifiers*, in Machine Learning Proceedings 1995, Elsevier, 1995, pp. 150–157.

[31] J. Denker and Y. LeCun, *Transforming neural-net output levels to probability distributions*, Advances in neural information processing systems, 3 (1990).

[32] S. V. Desai, A. C. Lagandula, W. Guo, S. Ninomiya, and V. N. Balasubramanian, *An adaptive supervision framework for active learning in object detection*, in Proceedings of the British Machine Vision Conference (BMVC), K. Sidorov and Y. Hicks, eds., BMVA Press, Sept. 2019, pp. 177.1–177.13.

[33] T. DeVries and G. W. Taylor, *Leveraging Uncertainty Estimates for Predicting Segmentation Quality*, arXiv preprint arXiv:1807.00502, (2018).

[34] G. Di Biase, H. Blum, R. Siegwart, and C. Cadena, *Pixel-wise Anomaly Detection in Complex Driving Scenes*, in 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, June 2021, IEEE, pp. 16913–16922.

[35] M. Dickinson and D. Meurers, *Detecting errors in part-of-speech annotation*, in 10th Conference of the European Chapter of the Association for Computational Linguistics, 2003.

[36] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, in 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021.

[37] I. Elezi, Z. Yu, A. Anandkumar, L. Leal-Taixe, and J. M. Alvarez, *Not All Labels Are Equal: Rationalizing The Labeling Costs for Training Object Detection*, in 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, June 2022, IEEE, pp. 14472–14481.

[38] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The Pascal Visual Object Classes (VOC) Challenge*, International Journal of Computer Vision, 88 (2010), pp. 303–338.

[39] A. Farhadi and J. Redmon, *Yolov3: An incremental improvement*, in Computer Vision and Pattern Recognition, vol. 1804, Springer Berlin/Heidelberg, Germany, 2018, pp. 1–6.

[40] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Glaeser, F. Timm, W. Wiesbeck, and K. Dietmayer, *Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges*, IEEE Transactions on Intelligent Transportation Systems, 22 (2020), pp. 1341–1360.

[41] D. Feng, L. Rosenbaum, and K. Dietmayer, *Towards Safe Autonomous Driving: Capture Uncertainty in the Deep Neural Network For Lidar 3D Vehicle Detection*, in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Nov. 2018, pp. 3266–3273.

[42] D. Feng, L. Rosenbaum, C. Glaeser, F. Timm, and K. Dietmayer, *Can We Trust You? On Calibration of a Probabilistic Object Detector for Autonomous Driving*, arXiv preprint arXiv:1909.12358, (2019).

[43] D. Feng, L. Rosenbaum, F. Timm, and K. Dietmayer, *Leveraging Heteroscedastic Aleatoric Uncertainties for Robust Real-Time LiDAR 3D Object Detection*, in 2019 IEEE Intelligent Vehicles Symposium (IV), June 2019, pp. 1280–1287.

[44] D. Feng, Z. Wang, Y. Zhou, L. Rosenbaum, F. Timm, K. Dietmayer, M. Tomizuka, and W. Zhan, *Labels are not perfect: Inferring spatial uncertainty in object detection*, IEEE Transactions on Intelligent Transportation Systems, (2021).

[45] Y. Gal and Z. Ghahramani, *Dropout as a bayesian approximation: Representing model uncertainty in deep learning*, in International Conference on Machine Learning, PMLR, 2016, pp. 1050–1059.

[46] A. Geiger, P. Lenz, and R. Urtasun, *Are we ready for autonomous driving? The KITTI vision benchmark suite*, in 2012 IEEE Conference on Computer Vision and Pattern Recognition, June 2012, pp. 3354–3361.

[47] R. Girshick, *Fast R-CNN*, in 2015 IEEE International Conference on Computer Vision (ICCV), Dec. 2015, pp. 1440–1448.

[48] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587.

[49] J. Goldberger and E. Ben-Reuven, *Training deep neural-networks using a noise adaptation layer*, in International Conference on Machine Learning, 2017.

[50] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, arXiv preprint arXiv:1412.6572, (2014).

[51] H. Gottschalk, *Hochdimensionale Wahrscheinlichkeitstheorie Und Maschinelles Lernen*, Private Communications.

[52] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, M. Norouzi, and K. Swersky, *Your classifier is secretly an energy based model and you should treat it like one*, in International Conference on Learning Representations, 2020.

[53] M. Grcić, P. Bevandić, Z. Kalafatić, and S. Šegvić, *Dense anomaly detection by robust learning on synthetic negative data*, arXiv preprint arXiv:2112.12833, (2021).

[54] M. Grcić, P. Bevandić, and S. Šegvić, *Densehybrid: Hybrid anomaly detection for dense open-set recognition*, in Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXV, Springer, 2022, pp. 500–517.

[55] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, *On calibration of modern neural networks*, in International Conference on Machine Learning, PMLR, 2017, pp. 1321–1330.

[56] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama, *Co-teaching: Robust training of deep neural networks with extremely noisy labels*, Advances in neural information processing systems, 31 (2018).

[57] A. Harakeh, M. Smart, and S. L. Waslander, *BayesOD: A Bayesian Approach for Uncertainty Estimation in Deep Object Detectors*, in 2020 IEEE International Conference on Robotics and Automation (ICRA), May 2020, pp. 87–93.

[58] E. Haussmann, M. Fenzi, K. Chitta, J. Ivanecky, H. Xu, D. Roy, A. Mittel, N. Koumchatzky, C. Farabet, and J. M. Alvarez, *Scalable Active Learning for Object Detection*, in 2020 IEEE Intelligent Vehicles Symposium (IV), Oct. 2020, pp. 1430–1435.

[59] M. Havasi, R. Jenatton, S. Fort, J. Z. Liu, J. Snoek, B. Lakshminarayanan, A. M. Dai, and D. Tran, *Training independent subnetworks for robust prediction*, in International Conference on Learning Representations, 2021.

[60] J. He and J. Xu, *MgNet: A Unified Framework of Multigrid and Convolutional Neural Network*, Science China Mathematics, 62 (2019), pp. 1331–1354.

[61] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, June 2016, IEEE, pp. 770–778.

[62] D. Hendrycks and K. Gimpel, *A baseline for detecting misclassified and out-of-distribution examples in neural networks*, in 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.

[63] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, *Using trusted data to train deep networks on labels corrupted by severe noise*, Advances in neural information processing systems, 31 (2018).

[64] K. Hoebel, V. Andrearczyk, A. Beers, J. Patel, K. Chang, A. Depeursinge, H. Müller, and J. Kalpathy-Cramer, *An exploration of uncertainty information for segmentation quality assessment*, Medical Imaging 2020: Image Processing, 11313 (2020), p. 113131K.

[65] J. Hornauer and V. Belagiannis, *Gradient-Based Uncertainty for Monocular Depth Estimation*, in Computer Vision – ECCV 2022, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, eds., vol. 13680, Springer Nature Switzerland, Cham, 2022, pp. 613–630.

[66] J. Hu, L. Shen, and G. Sun, *Squeeze-and-excitation networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7132–7141.

[67] Z. Hu, K. Gao, X. Zhang, J. Wang, H. Wang, and J. Han, *Probability differential-based class label noise purification for object detection in aerial images*, IEEE Geoscience and Remote Sensing Letters, 19 (2022), pp. 1–5.

[68] C. Huang, Q. Wu, and F. Meng, *Qualitynet: Segmentation quality evaluation with deep convolutional networks*, in 2016 Visual Communications and Image Processing (VCIP), IEEE, 2016, pp. 1–4.

[69] R. Huang, A. Geng, and Y. Li, *On the importance of gradients for detecting distributional shifts in the wild*, Advances in Neural Information Processing Systems, 34 (2021), pp. 677–689.

[70] E. Hüllermeier and W. Waegeman, *Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods*, Machine Learning, 110 (2021), pp. 457–506.

[71] R. Hussain and S. Zeadally, *Autonomous cars: Research results, issues, and future challenges*, IEEE Communications Surveys & Tutorials, 21 (2018), pp. 1275–1313.

[72] E. Ilg, Ö. Çiçek, S. Galesso, A. Klein, O. Makansi, F. Hutter, and T. Brox, *Uncertainty Estimates and Multi-hypotheses Networks for Optical Flow*, in Computer Vision – ECCV 2018, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds., vol. 11211, Springer International Publishing, Cham, 2018, pp. 677–693.

[73] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in International Conference on Machine Learning, pmlr, 2015, pp. 448–456.

[74] P. Jaccard, *The distribution of the flora in the alpine zone*, The New Phytologist, 11 (1912), pp. 37–50.

[75] P. F. Jaeger, S. A. Kohl, S. Bickelhaupt, F. Isensee, T. A. Kuder, H.-P. Schlemmer, and K. H. Maier-Hein, *Retina U-Net: Embarrassingly simple exploitation of segmentation supervision for medical object detection*, in Machine Learning for Health Workshop, PMLR, 2020, pp. 171–183.

[76] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang, *Acquisition of Localization Confidence for Accurate Object Detection*, in Computer Vision – ECCV 2018, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds., vol. 11218, Springer International Publishing, Cham, 2018, pp. 816–832.

[77] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, *Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels*, in International Conference on Machine Learning, PMLR, 2018, pp. 2304–2313.

[78] D. Kang, N. Arechiga, S. Pillai, P. D. Bailis, and M. Zaharia, *Finding label and model errors in perception data with learned observation assertions*, in Proceedings of the 2022 International Conference on Management of Data, 2022, pp. 496–505.

[79] A. Kaur, Y. Singh, N. Neeru, L. Kaur, and A. Singh, *A survey on deep learning approaches to medical images and a systematic look up into real-time object detection*, Archives of Computational Methods in Engineering, (2021), pp. 1–41.

[80] A. Kendall and Y. Gal, *What uncertainties do we need in bayesian deep learning for computer vision?*, Advances in neural information processing systems, 30 (2017).

[81] S. C. Kleene et al., *Representation of events in nerve nets and finite automata*, Automata studies, 34 (1956), pp. 3–41.

[82] A. Klenke, *Probability Theory: A Comprehensive Course*, Universitext, Springer, London, 2014.

[83] A. Koksal, K. G. Ince, and A. Alatan, *Effect of annotation errors on drone detection with YOLOv3*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2020, pp. 1030–1031.

[84] K. Kowol., M. Rottmann., S. Bracke., and H. Gottschalk., *YOdar: Uncertainty-based sensor fusion for vehicle detection with camera and radar sensors*, in Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,, SciTePress / INSTICC, 2021, pp. 177–186.

[85] F. Kraus and K. Dietmayer, *Uncertainty Estimation in One-Stage Object Detection*, in 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Oct. 2019, pp. 53–60.

[86] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, Communications of the ACM, 60 (2017), pp. 84–90.

[87] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, *A survey of deep learning applications to autonomous vehicle control*, IEEE Transactions on Intelligent Transportation Systems, 22 (2020), pp. 712–733.

[88] B. Lakshminarayanan, A. Pritzel, and C. Blundell, *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles.*

[89] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, *PointPillars: Fast Encoders for Object Detection From Point Clouds*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, June 2019, IEEE, pp. 12689–12697.

[90] M. T. Le, F. Diehl, T. Brunner, and A. Knoll, *Uncertainty Estimation for Deep Neural Object Detectors in Safety-Critical Applications*, in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Nov. 2018, pp. 3873–3878.

[91] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.

[92] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[93] Lee, Jinsol, M. Prabhushankar, and G. AlRegib, *Gradient-based adversarial and out-of-distribution detection*, in International Conference on Machine Learning (ICML) Workshop on New Frontiers in Adversarial Machine Learning, 2022.

[94] H. J. Lee, S. T. Kim, H. Lee, N. Navab, and Y. M. Ro, *Efficient ensemble model generation for uncertainty estimation with Bayesian approximation in segmentation*, arXiv preprint arXiv:2005.10754, (2020).

[95] K. Lee, K. Lee, H. Lee, and J. Shin, *A simple unified framework for detecting out-of-distribution samples and adversarial attacks*, Advances in neural information processing systems, 31 (2018).

[96] D. D. Lewis and J. Catlett, *Heterogeneous Uncertainty Sampling for Supervised Learning*, in Machine Learning Proceedings 1994, Elsevier, 1994, pp. 148–156.

[97] H. Li, Z. Wu, C. Zhu, C. Xiong, R. Socher, and L. S. Davis, *Learning from noisy anchors for one-stage object detection*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 10588–10597.

[98] M. Li and I. Sethi, *Confidence-based active learning*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 28 (2006), pp. 1251–1261.

[99] S. Liang, Y. Li, and R. Srikant, *Enhancing the reliability of out-of-distribution image detection in neural networks*, in International Conference on Learning Representations, 2018.

[100] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, *Feature Pyramid Networks for Object Detection*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, July 2017, IEEE, pp. 936–944.

[101] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal loss for dense object detection*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2980–2988.

[102] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, *Microsoft COCO: Common objects in context*, in European Conference on Computer Vision, Springer, 2014, pp. 740–755.

[103] W.-H. Lin and A. Hauptmann, *Meta-classification: Combining Multimodal Classifiers*, in Mining Multimedia and Complex Data, O. R. Zaïane, S. J. Simoff, and C. Djeraba, eds., Lecture Notes in Computer Science, Berlin, Heidelberg, 2003, Springer, pp. 217–231.

[104] K. Lis, S. Honari, P. Fua, and M. Salzmann, *Detecting Road Obstacles by Erasing Them*, arXiv preprint arXiv:2012.13633, (2021).

[105] K. Lis, K. K. Nakka, P. Fua, and M. Salzmann, *Detecting the Unexpected via Image Resynthesis*, in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), Oct. 2019, IEEE, pp. 2152–2161.

[106] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*, vol. 9905, 2016, pp. 21–37.

[107] W. Liu, X. Wang, J. Owens, and Y. Li, *Energy-based out-of-distribution detection*, Advances in neural information processing systems, 33 (2020), pp. 21464–21475.

[108] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*, in 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, Oct. 2021, IEEE, pp. 9992–10002.

[109] Z. Lyu, N. Gutierrez, A. Rajguru, and W. J. Beksi, *Probabilistic object detection via deep ensembles*, in Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16, Springer, 2020, pp. 67–75.

[110] K. Maag, *False Negative Reduction in Video Instance Segmentation using Uncertainty Estimates*, in 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI), Nov. 2021, pp. 1279–1286.

[111] K. Maag, R. Chan, S. Uhlemeyer, K. Kowol, and H. Gottschalk, *Two Video Data Sets for Tracking and Retrieval of Out of Distribution Objects*, in Computer Vision – ACCV 2022, L. Wang, J. Gall, T.-J. Chin, I. Sato, and R. Chellappa, eds., vol. 13845, Springer Nature Switzerland, Cham, 2023, pp. 476–494.

[112] K. Maag and T. Riedlinger, *Pixel-wise gradient uncertainty for convolutional neural networks applied to out-of-distribution segmentation*, arXiv preprint arXiv: 2303.06920, (2023).

[113] K. Maag, M. Rottmann, and H. Gottschalk, *Time-Dynamic Estimates of the Reliability of Deep Semantic Segmentation Networks*, in 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), Nov. 2020, pp. 502–509.

[114] K. Maag, M. Rottmann, S. Varghese, F. Hüger, P. Schlicht, and H. Gottschalk, *Improving Video Instance Segmentation by Light-weight Temporal Uncertainty Estimates*, in 2021 International Joint Conference on Neural Networks (IJCNN), July 2021, pp. 1–8.

[115] D. J. C. MacKay, *A Practical Bayesian Framework for Backpropagation Networks*, Neural Computation, 4 (1992), pp. 448–472.

[116] A. Malinin and M. Gales, *Predictive uncertainty estimation via prior networks*, Advances in neural information processing systems, 31 (2018).

[117] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics, 5 (1943), pp. 115–133.

[118] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington, *LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, June 2019, IEEE, pp. 12669–12678.

[119] G. P. Meyer and N. Thakurdesai, *Learning an Uncertainty-Aware Object Detector for Autonomous Driving*, in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2020, pp. 10521–10527.

[120] D. Miller, F. Dayoub, M. Milford, and N. Sünderhauf, *Evaluating Merging Strategies for Sampling-based Uncertainty Techniques in Object Detection*, in 2019 International Conference on Robotics and Automation (ICRA), May 2019, pp. 2348–2354.

[121] D. Miller, L. Nicholson, F. Dayoub, and N. Sünderhauf, *Dropout Sampling for Robust Object Detection in Open-Set Conditions*, in 2018 IEEE International Conference on Robotics and Automation (ICRA), May 2018, pp. 3243–3249.

[122] D. Miller, N. Sünderhauf, H. Zhang, D. Hall, and F. Dayoub, *Benchmarking sampling-based probabilistic object detectors.*, in CVPR Workshops, vol. 3, 2019, p. 6.

[123] MMDetection3D Contributors, *OpenMMLab's Next-generation Platform for General 3D Object Detection*, July 2020.

[124] J. Mukhoti and Y. Gal, *Evaluating Bayesian Deep Learning Methods for Semantic Segmentation*, arXiv preprint arXiv:1811.12709, (2019).

[125] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press, Mar. 2022.

[126] M. P. Naeini, G. Cooper, and M. Hauskrecht, *Obtaining well calibrated probabilities using bayesian binning*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, 2015.

[127] I. Namatēvs, L. Aleksejeva, and I. Poļaka, *Neural network modelling for sports performance classification as a complex socio-technical system*, Information Technology and Management Science, 19 (2016), pp. 45–52.

[128] L. Neumann, A. Zisserman, and A. Vedaldi, *Relaxed softmax: Efficient confidence auto-calibration for safe pedestrian detection*, in Machine Learning for Intelligent Transportation Systems Workshop, NIPS, 2018.

[129] H. T. Nguyen and A. Smeulders, *Active learning using pre-clustering*, in Proceedings of the Twenty-First International Conference on Machine Learning, 2004, p. 79.

[130] C. Northcutt, L. Jiang, and I. Chuang, *Confident learning: Estimating uncertainty in dataset labels*, Journal of Artificial Intelligence Research, 70 (2021), pp. 1373–1411.

[131] C. G. Northcutt, A. Athalye, and J. Mueller, *Pervasive label errors in test sets destabilize machine learning benchmarks*, arXiv preprint arXiv:2103.14749, (2021).

[132] P. Oberdiek, M. Rottmann, and H. Gottschalk, *Classification uncertainty of deep neural networks based on gradient information*, in Artificial Neural Networks in Pattern Recognition: 8th IAPR TC3 Workshop, ANNPR 2018, Siena, Italy, September 19–21, 2018, Proceedings 8, Springer, 2018, pp. 113–125.

[133] J. A. Pandian, V. D. Kumar, O. Geman, M. Hnatiuc, M. Arif, and K. Kanchanadevi, *Plant disease detection using deep convolutional neural network*, Applied Sciences, 12 (2022), p. 6982.

[134] D. P. Papadopoulos, J. R. R. Uijlings, F. Keller, and V. Ferrari, *Training Object Class Detectors with Click Supervision*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, July 2017, IEEE, pp. 180–189.

[135] J. Parham, J. Crall, C. Stewart, T. Berger-Wolf, and D. I. Rubenstein, *Animal population censusing at scale with citizen science and photographic identification*, in AAAI Spring Symposium-Technical Report, 2017.

[136] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library.*

[137] P. Pinggera, S. Ramos, S. Gehrig, U. Franke, C. Rother, and R. Mester, *Lost and Found: Detecting small road hazards for self-driving vehicles*, in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2016, pp. 1099–1106.

[138] A. Pinkus, *Approximation theory of the MLP model in neural networks*, Acta Numerica, 8 (1999), pp. 143–195.

[139] M. Pitropov, C. Huang, V. Abdelzad, K. Czarnecki, and S. Waslander, *LiDAR-MIMO: Efficient Uncertainty Estimation for LiDAR-based 3D Object Detection*, in 2022 IEEE Intelligent Vehicles Symposium (IV), June 2022, pp. 813–820.

[140] L. Plaza, *Collision Between Vehicle Controlled by Developmental Automated Driving System and Pedestrian, Tempe, Arizona, March 18, 2018.*

[141] T. Ramalho and M. Miranda, *Density estimation in representation space to predict model uncertainty*, in Engineering Dependable and Secure Machine Learning Systems: Third International Workshop, EDSMLS 2020, New York City, NY, USA, February 7, 2020, Revised Selected Papers 3, Springer, 2020, pp. 84–96.

[142] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, June 2016, IEEE, pp. 779–788.

[143] S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, *Training deep neural networks on noisy labels with bootstrapping*, arXiv preprint arXiv:1412.6596, (2014).

[144] S. Ren, K. He, R. Girshick, and J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 39 (2017), pp. 1137–1149.

[145] T. Riedlinger, M. Rottmann, M. Schubert, and H. Gottschalk, *Gradient-Based Quantification of Epistemic Uncertainty for Deep Object Detectors*, in 2023 IEEE/ CVF Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, Jan. 2023, IEEE, pp. 3910–3920.

[146] T. Riedlinger, M. Schubert, K. Kahl, H. Gottschalk, and M. Rottmann, *Towards rapid prototyping and comparability in active learning for deep object detection*, arXiv preprint arXiv:2212.10836, (2022).

[147] T. Riedlinger, M. Schubert, K. Kahl, and M. Rottmann, *Uncertainty quantification for object detection: Output-and gradient-based approaches*, in Deep Neural Networks and Data for Automated Driving, Springer, Cham, 2022, pp. 251–275.

[148] T. Riedlinger, M. Schubert, S. Penquitt, J. Kezmann, P. Colling, K. Kahl, L. Roese-Koerner, M. Arnold, U. Zimmermann, and M. Rottmann, *LMD: Light-weight Prediction Quality Estimation for Object Detection in Lidar Point Clouds*, June 2023.

[149] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, in Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds., Lecture Notes in Computer Science, Cham, 2015, Springer International Publishing, pp. 234–241.

[150] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain.*, Psychological review, 65 (1958), p. 386.

[151] R. Rothe, M. Guillaumin, and L. Van Gool, *Non-maximum Suppression for Object Detection by Passing Messages Between Windows*, in Computer Vision – ACCV 2014, D. Cremers, I. Reid, H. Saito, and M.-H. Yang, eds., vol. 9003, Springer International Publishing, Cham, 2015, pp. 290–306.

[152] M. Rottmann, P. Colling, T. Paul Hack, R. Chan, F. Hüger, P. Schlicht, and H. Gottschalk, *Prediction Error Meta Classification in Semantic Segmentation: Detection via Aggregated Dispersion Measures of Softmax Probabilities*, in 2020 International Joint Conference on Neural Networks (IJCNN), July 2020, pp. 1–9.

[153] M. Rottmann, K. Maag, R. Chan, F. Hüger, P. Schlicht, and H. Gottschalk, *Detection of False Positive and False Negative Samples in Semantic Segmentation*, in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Mar. 2020, pp. 1351–1356.

[154] M. Rottmann and M. Reese, *Automated detection of label errors in semantic segmentation datasets via deep learning and uncertainty quantification*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2023, pp. 3214–3223.

[155] M. Rottmann and M. Schubert, *Uncertainty Measures and Prediction Quality Rating for the Semantic Segmentation of Nested Multi Resolution Street Scene Images*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Long Beach, CA, USA, June 2019, IEEE, pp. 1361–1369.

[156] S. Roy, A. Unmesh, and V. P. Namboodiri, *Deep active learning for object detection.*, in BMVC, vol. 362, 2018, p. 91.

[157] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., *Imagenet large scale visual recognition challenge*, International journal of computer vision, 115 (2015), pp. 211–252.

[158] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, *How does batch normalization help optimization?*, Advances in neural information processing systems, 31 (2018).

[159] T. Scheffer, C. Decomain, and S. Wrobel, *Active hidden markov models for information extraction*, in Advances in Intelligent Data Analysis: 4th International Conference, IDA 2001 Cascais, Portugal, September 13–15, 2001 Proceedings 4, Springer, 2001, pp. 309–318.

[160] S. Schmidt, Q. Rao, J. Tatsch, and A. Knoll, *Advanced Active Learning Strategies for Object Detection*, in 2020 IEEE Intelligent Vehicles Symposium (IV), Oct. 2020, pp. 871–876.

[161] M. Schubert, K. Kahl, and M. Rottmann, *MetaDetect: Uncertainty Quantification and Prediction Quality Estimates for Object Detection*, in 2021 International Joint Conference on Neural Networks (IJCNN), July 2021, pp. 1–10.

[162] M. Schubert, T. Riedlinger, K. Kahl, D. Kröll, S. Schoenen, S. Šegvić, and M. Rottmann, *Identifying label errors in object detection datasets by loss inspection*, arXiv preprint arXiv:2303.06999, (2023).

[163] O. Sener and S. Savarese, *Active learning for convolutional neural networks: A core-set approach*, in International Conference on Learning Representations, 2018.

[164] B. Settles, *Active learning literature survey*, Machine Learning, 15 (1994), pp. 201–221.

[165] H. S. Seung, M. Opper, and H. Sompolinsky, *Query by committee*, in Proceedings of the Fifth Annual Workshop on Computational Learning Theory, 1992, pp. 287–294.

[166] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, first ed., May 2014.

[167] C. E. Shannon, *A mathematical theory of communication*, The Bell System Technical Journal, 27 (1948), pp. 379–423.

[168] S. Sinha, S. Ebrahimi, and T. Darrell, *Variational Adversarial Active Learning*, in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), Oct. 2019, IEEE, pp. 5971–5980.

[169] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.*

[170] N. Ståhl, G. Falkman, A. Karlsson, and G. Mathiason, *Evaluation of uncertainty quantification in deep learning*, in Information Processing and Management of Uncertainty in Knowledge-Based Systems: 18th International Conference, IPMU 2020, Lisbon, Portugal, June 15–19, 2020, Proceedings, Part I 18, Springer, 2020, pp. 556–568.

[171] T. D. Stanley and S. B. Jarrell, *Meta-Regression Analysis: A Quantitative Method of Literature Surveys*, Journal of Economic Surveys, 19 (2005), pp. 299–308.

[172] A. Subramanian and A. Subramanian, *One-Click Annotation with Guided Hierarchical Object Detection*, arXiv preprint arXiv:1810.00609, (2018).

[173] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, *Revisiting unreasonable effectiveness of data in deep learning era*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 843–852.

[174] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, *Intriguing properties of neural networks*, in 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2014.

[175] A. Thyagarajan, E. Snorrason, C. Northcutt, and J. Mueller, *Identifying incorrect annotations in multi-label classification data*, arXiv preprint 2211.13895, (2022).

[176] Y. Tian, Y. Liu, G. Pang, F. Liu, Y. Chen, and G. Carneiro, *Pixel-wise energy-biased abstention learning for anomaly segmentation on complex urban driving scenes*, in Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIX, Springer, 2022, pp. 246–263.

[177] A. Tsvigun, A. Shelmanov, G. Kuzmin, L. Sanochkin, D. Larionov, G. Gusev, M. Avetisian, and L. Zhukov, *Towards computationally feasible deep active learning*, in Findings of the Association for Computational Linguistics: NAACL 2022, 2022, pp. 1198–1218.

[178] D. Ulyanov, A. Vedaldi, and V. Lempitsky, *Instance Normalization: The Missing Ingredient for Fast Stylization*, arXiv preprint arXiv:1607.08022, (2017).

[179] M. Valdenegro-Toro, *Deep sub-ensembles for fast uncertainty estimation in image classification*, arXiv preprint arXiv:1910.08168, (2019).

[180] R. Van Handel, *Probability in high dimension*, tech. rep., PRINCETON UNIV NJ, 2014.

[181] V. T. Vasudevan, A. Sethy, and A. R. Ghias, *Towards better confidence estimation for neural models*, in ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 7335–7339.

[182] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is All you Need*.

[183] T. Vojir, T. Sipka, R. Aljundi, N. Chumerin, D. O. Reino, and J. Matas, *Road Anomaly Detection by Partial Image Reconstruction with Segmentation Coupling*, in 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, Oct. 2021, IEEE, pp. 15631–15640.

[184] S. Wallelign, M. Polceanu, and C. Buche, *Soybean plant disease identification using convolutional neural network.*, in FLAIRS Conference, 2018, pp. 146–151.

[185] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao, *Deep High-Resolution Representation Learning for Visual Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 43 (2021), pp. 3349–3364.

[186] Y. Wang, X. Ma, Z. Chen, Y. Luo, J. Yi, and J. Bailey, *Symmetric cross entropy for robust learning with noisy labels*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 322–330.

[187] YNT. Wang, A. Coates, A. Bissacco, B. Wu, and AY. Ng, *Reading digits in natural images with unsupervised feature learning*, in NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

[188] K. Wickstrøm, M. Kampffmeyer, and R. Jenssen, *Uncertainty and interpretability in convolutional neural networks for semantic segmentation of colorectal polyps*, Medical image analysis, 60 (2020), p. 101619.

[189] H. Wu, *YOLOv3-in-PyTorch*, 2018.

[190] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, 2019.

[191] Z. Wu, N. Bodla, B. Singh, M. Najibi, R. Chellappa, and L. S. Davis, *Soft sampling for robust object detection*, arXiv preprint arXiv:1806.06986, (2018).

[192] Z. Wu, C. Shen, and A. Van Den Hengel, *Wider or deeper: Revisiting the resnet model for visual recognition*, Pattern Recognition, 90 (2019), pp. 119–133.

[193] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms*, 2017-08-28, 2017.

[194] M. Xu, Y. Bai, B. Ghanem, B. Liu, Y. Gao, N. Guo, X. Ye, F. Wan, H. You, D. Fan, et al., *Missing labels in object detection.*, in CVPR Workshops, vol. 3, 2019, p. 5.

[195] Y. Xu, P. Cao, Y. Kong, and Y. Wang, *L_dmi: A novel information-theoretic loss function for training deep nets robust to label noise*, Advances in neural information processing systems, 32 (2019).

[196] Z. Xu, R. Akella, and Y. Zhang, *Incorporating diversity and density in active learning for relevance feedback*, in European Conference on Information Retrieval, Springer, 2007, pp. 246–257.

[197] R. Yampolskiy, *Incident number 52*, AI Incident Database, (2016).

[198] Y. Yan, Y. Mao, and B. Li, *Second: Sparsely embedded convolutional detection*, Sensors, 18 (2018), p. 3337.

[199] B. Yang, W. Luo, and R. Urtasun, *PIXOR: Real-time 3D Object Detection from Point Clouds*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, June 2018, IEEE, pp. 7652–7660.

[200] F. Yang, H.-z. Wang, H. Mi, C.-d. Lin, and W.-w. Cai, *Using random forest for reliable classification and cost-sensitive learning for medical diagnosis*, BMC Bioinformatics, 10 (2009), p. S22.

[201] Q. Yang, H. Chen, Z. Chen, and J. Su, *Uncertainty Estimation for Monocular 3D Object Detectors in Autonomous Driving*, in 2021 6th International Conference on Robotics and Automation Engineering (ICRAE), Nov. 2021, pp. 55–59.

[202] G. Yarin, *Uncertainty in deep learning*, University of Cambridge, Cambridge, (2016).

[203] D. Yarotsky, *Error bounds for approximations with deep ReLU networks*, Neural Networks, 94 (2017), pp. 103–114.

[204] ——, *Universal Approximations of Invariant Maps by Neural Networks*, Constructive Approximation, 55 (2022), pp. 407–474.

[205] T. Yin, X. Zhou, and P. Krahenbuhl, *Center-based 3D Object Detection and Tracking*, in 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, June 2021, IEEE, pp. 11779–11788.

[206] D. Yoo and I. S. Kweon, *Learning Loss for Active Learning*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, June 2019, IEEE, pp. 93–102.

[207] T. Younesian, D. Epema, and L. Y. Chen, *Active Learning for Noisy Data Streams Using Weak and Strong Labelers*, arXiv preprint 2010.14149, (2020).

[208] T. Younesian, Z. Zhao, A. Ghiassi, R. Birke, and L. Y. Chen, *Qactor: Active learning on noisy labels*, in Asian Conference on Machine Learning, PMLR, 2021, pp. 548–563.

[209] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, June 2020, IEEE, pp. 2633–2642.

[210] T. Yuan, F. Wan, M. Fu, J. Liu, S. Xu, X. Ji, and Q. Ye, *Multiple Instance Active Learning for Object Detection*, in 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, June 2021, IEEE, pp. 5326–5335.

[211] C. Yun, S. Bhojanapalli, A. S. Rawat, S. Reddi, and S. Kumar, *Are Transformers universal approximators of sequence-to-sequence functions?*, in International Conference on Learning Representations, 2020.

[212] X. Zhan, Q. Wang, K.-h. Huang, H. Xiong, D. Dou, and A. B. Chan, *A Comparative Survey of Deep Active Learning*, arXiv preprint 2203.13450, (2022).

[213] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, *Mixup: Beyond empirical risk minimization*, arXiv preprint arXiv:1710.09412, (2017).

[214] H. Zhang and J. Wang, *Towards adversarially robust object detection*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 421–430.

[215] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha, M. Li, and A. Smola, *ResNeSt: Split-Attention Networks*, in 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), New Orleans, LA, USA, June 2022, IEEE, pp. 2735–2745.

[216] W. Zhang, J. Tanida, K. Itoh, and Y. Ichioka, *Shift-invariant pattern recognition neural network and its optical architecture*, in Proceedings of Annual Conference of the Japan Society of Applied Physics, Montreal, CA, 1988, pp. 2147–2151.

[217] Y. Zhu, K. Sapra, F. A. Reda, K. J. Shih, S. Newsam, A. Tao, and B. Catanzaro, *Improving Semantic Segmentation via Video Propagation and Label Relaxation*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, June 2019, IEEE, pp. 8848–8857.