Uncertainty Quantification Methods and their Applications in Object Detection

BERGISCHE
UNIVERSITÄT
WUPPERTAL

Dissertation

University of Wuppertal
Faculty of Mathematics and Computer Science

submitted by
Marius Schubert, M. Sc.
for the degree of

Doctor of Natural Sciences (Dr. rer. nat.)

supervised by
PD Dr. Matthias Rottmann
PD Dr. Karsten Kahl

Wuppertal, October 10, 2023

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisors, Matthias Rottmann and Karsten Kahl, for their unlimited support during my entire time as a doctoral student. Our conversations and discussions always took place in a peaceful and friendly atmosphere, and I was privileged to benefit from your experience and knowledge.

Furthermore, I would like to thank my colleagues of the Applied Computer Science and BUW-KI Group for an awesome and memorable time.

Finally, I would like to thank my family, friends and especially my future wife for their continuous mental support.

FOREWORD

The contents of the following chapters 4 to 8 are taken almost word-by-word from the publications listed below with a short description of contributions. Notations were made consistent, redundancies were replaced by references to the first appearances in this thesis, as well as the respective results of the supplementary materials were included in the main results of the chapter for enhanced readability.

- M. Schubert, K. Kahl, and M. Rottmann, *MetaDetect: Uncertainty Quantification and Prediction Quality Estimates for Object Detection,* in International Joint Conference on Neural Networks (IJCNN), IEEE, 2021, pp. 1-10

  Carried out independently

- 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)

  Implementation of active learning and the associated baselines, benchmark evaluation of active learning experiments, related work, active learning methods in object detection, implementation, benchmark results

- 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)

  Carried out independently except for the formulation of theoretical considerations on the method's viability and proofs for statistical loss separation for classification

- M. Schubert, T. Riedlinger, K. Kahl, and M. Rottmann, *Deep Active Learning with Noisy Oracle in Object Detection,* arXiv preprint arXiv: 2310.00372, (2023)

  Implementation of review in active learning cycle, experiment logistics and management, statistical evaluation of results

- T. Riedlinger, M. Schubert, S. Penquitt, J.-M. 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,* arXiv preprint arXiv: 2306.07835, (2023)

  Joint co-supervision of the original Master's thesis, experiments, numerical results and review

CONTENTS

CHAPTER 1 INTRODUCTION

These days, artificial intelligence is en vogue and neural networks exhibit excellent performance in computer vision tasks like image classification [153, 158], object detection [39, 123], semantic segmentation [25, 138] or instance segmentation [5, 61]. Its fields of applications are diverse, such as chatbots on websites [146], speech recognition and voice output on cell phones [150], as well as in safety-critical tasks such as automated driving [20] and medical diagnosis [78]. Probably the best-known chatbot at the moment is ChatGPT, which is based on artificial intelligence and communicates with users via text-based messages and images, where the user receives an answer to every question or task. For the task of automated driving, more and more technical assistance systems are installed in vehicles, such as an intelligent brake assistant, driver fatigue detection, parking assistance, etc. For this purpose, the environment is recorded by different sensors, e.g., by Cameras, Radar or Lidar sensors, and then combined for a unified output. The rapid development has been made possible by the use of graphics processing units (GPUs), as well as the amount of (annotated) data and advanced architectures.

Even though artificial intelligence became very famous in recent years, the origins of neural networks date back to the early 1940s. In 1943, McCulloch and Pitts described the interconnection of elementary units as a kind of network similar to the interconnection of neurons in the brain, which can be used to compute virtually any logical or arithmetic function and can be used, for instance, for spatial pattern recognition [104]. Six years later, Donald Olding Hebb presents the first neural learning method in which the change in synaptic transmission is represented as a change in the weight of the neural graph [65]. The first successful applications have been the Mark I Perceptron [129] published in 1958 by Frank Rosenblatt and Charles Wightman, which could recognize simple digits, and the ADALINE [166]

(ADAptive LInear NEuron) published in 1960 by Bernard Widrow and Marcian E. Hoff, which has been used in analogue telephones for real-time echo filtering. The late 1960s witnessed a temporary end to research and funding in the field of neural networks as Marvin Minsky and Seymour Papert showed that important problems could not be solved with the perceptron model of Rosenblatt and Wightman. Over the next decades, there have been several publications in the field of artificial neural networks, and in 1985, Minsky's view was disproved by making non-linearly separable problems solvable by multilayer perceptrons through the *backpropagation of error* learning procedure [68]. Nevertheless, the renaissance in neural network research came along in 2012, when Alex Krizhevsky et al. [84] won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC-2012) in image classification with AlexNet. AlexNet is a convolutional neural network (CNN) that achieved a top-5 error rate of 15.3%, over ten percentage points lower than the runner-up. The main reason for the strong performance was the depth of the AlexNet, which could be realized by using GPUs during training. This was followed by several milestones in image classification: VGGNet [149] in 2014, studying the effect of the convolutional network depth on its accuracy, GoogLeNet [155] in 2015, increasing the depth and width of the network while keeping the computational budget constant, ResNets [63] in 2016, applying a residual learning framework to ease the training of deep networks, as well as vision transformers [35] in 2020, applying attentions in image classification frameworks.

The task of object detection is to locate and classify objects of interest, and its beginnings date back to the early 2000s, where, for instance, human faces were detected with hand-crafted features and a sliding window approach [164]. The era of deep learning in object detection began in 2014 with the proposal of regions with CNN features (R-CNN [49]). This two-stage detection approach is comprised of the extraction of region proposals via selective search and the rescaling of these proposals to a fixed size in order to classify them with linear support vector machines. Improvements to this framework are the Fast R-CNN (2015) [48] and the Faster R-CNN [123] (2015). The former makes use of the entire image as input for feature extraction rather than of each region proposal individually. Furthermore, it introduces region of interest (RoI) pooling to feed a concatenation of extracted features in a fully-connected layer for the classification of regions. The latter replaces the selective search by a separate network that learns to predict region proposals, resulting in a reduced number of proposals while maintaining strong performance. In 2017, feature pyramid networks [95] (FPNs) have been introduced, which allow the detection of smaller objects by rescaling the input images to different sizes. One-stage approaches like Yolo [121] "You only look once" (2016) and SSD [99] (2016) get rid of the determination of region proposals by dividing the input image into cells and placing predefined proposals in these cells. In recent years, there have been several extensions of the original

Yolo, with the aim to maximize the performance while maintaining the real-time applicability by reducing the trainable parameters, computation effort and the amount of proposals. The currently most powerful object detectors are based on transformers, i.e., the DETR architectures [15], which introduced attention for object detectors, as well as swin transformers [100], which are hierarchical vision transformers that use shifted windows.

All neural networks are statistical models and therefore error-prone, which can have disastrous consequences, especially in safety-critical applications such as automated driving. Probably the most famous accident of a self-driving car dates back to 2018 [111], in which the car struck a pedestrian pushing a bicycle laden with shopping bags due to an incorrect prediction. This example demonstrates that the errors produced by a neural network are of extreme interest, especially for safety-relevant tasks. Of particular interest in object detection are not only incorrectly located or misclassified predictions and overlooked objects, but also incorrectly labeled data used for training or evaluation of the underlying CNNs. Therefore, it is desirable to develop methods that can prevent or at least detect the presence of these errors.

In image classification, predicted class probabilities exist for each output, which can be used to determine uncertainties. A first approach is to use dispersion measures, such as classification entropy [148], which is a measure of disorder, or probability margin, which is one minus the difference between the highest and second highest class probability. The greater these dispersions are, the more difficult it is to decide between classes and the more uncertain the prediction. Furthermore, there are sampling approaches, such as deep ensembles [88], where the same network is trained several times on the identical input data. Since the training of such networks is not deterministic, we obtain different outputs for the identical input, and we can determine uncertainties about the different predictions, such as the mutual information, which is a dispersion measure based on the classification entropy. Since for deep ensembles we have to train a network several times, which is costly in terms of computations and thus time intensive, they are often approximated by Monte-Carlo dropout [44]. Therefore, a network is trained only once, where a number of neurons are randomly switched on and off during training, resulting in non-deterministic predictions and an uncertainty determination analogous to the deep ensembles during inference.

In the present thesis, we address methods for uncertainty quantification (UQ) in object detection and their applications for the tasks of active learning and automated label error detection. However, UQ methods still find applications in many other domains in object detection, such as confidence calibration, performance increase, more robust object detectors, out-of-distribution detection, etc. In object detection, each prediction is given an *objectness score*, which is oftentimes interpreted as the confidence of the presence of an object. This score is oftentimes over-confident, i.e., the score tends to be high even if no object is nearby. Confidence calibration is the task of calibrating the score, such that the average of the confidence values matches the average correctness of the predictions, i.e., if a prediction gets a score of 90 percent, it should also be a correct prediction nine times out of ten. Some works use UQ methods for this task [83, 86, 118, 144], whereas in [125], calibrated uncertainty estimates are used to increase the test performance of the underlying object detector. The robustness of object detectors can be improved in various circumstances, such as training with missing labels [167], or resisting adversarial attacks [22, 34, 186]. Adversarial attacks alter images so that no differences from the original image are visually apparent, but cause the object detector to make incorrect predictions [23, 101, 168]. Out-of-distribution is the task of detecting unusual objects of interest that have not been seen during training and which the object detector cannot predict reliably. One application is the open-world object detection, where the task is to detect a known set of object categories while simultaneously identifying unknown objects [54, 108].

The following paragraphs provide an overview of related work and a brief summary of our work in each research area of interest, concluding with a paragraph that introduces the structure of the following chapters.

Uncertainty Quantification in Object Detection Issues addressed by UQ are active learning, label error detection, as well as active learning incorporating label errors. Oftentimes, the input for 2D object detection [39, 123] are camera images and the input for 3D object detection [89, 172] are Lidar point clouds. Here, the errors of interest are incorrect predictions, i.e., misclassified or incorrectly located predictions (*false positives*), or overlooked objects (*false negatives*). The occurrence of these errors can have disastrous consequences in safety-related tasks. In each area of the input, predictions are distributed, and each prediction is equipped with a confidence score, which is pushed towards 1 in training if an object is in the immediate surroundings and towards 0 otherwise. Then, a threshold on the confidence score is used to decide which instances are foreground and which are background. In general, it is impossible to choose a threshold so that no errors occur, i.e., no false positives and false negatives. Perhaps the simplest approach of UQ is to interpret the prediction-based confidence score as the quality of the prediction. In practice, this does not always work well, e.g., oftentimes the confidence score only considers the localization and not the classification, such that a misclassified prediction still has a high confidence score, although it is a false positive. Since classification also takes place in object detection, each prediction also consists of class probabilities. Therefore, the dispersion measures for classification can be transferred to the instances in object detection, i.e., classification entropy and probability margin can be calculated for each prediction without further adjustment. Furthermore, uncertainties about the class probabilities can be determined by the sampling approach, e.g., by the mutual information. Other UQ methods are based on Monte-Carlo dropout [107, 108] and receive for each input multiple outputs, where the predictions are the mean of the dropout outputs, and the standard deviations in the localization, confidence score and class probabilities are interpreted as uncertainty. In [41, 57, 83, 90], the output was extended by further neurons to estimate the uncertainty of individual output variables, resulting in a modified loss function.

In Chapter 4, we introduce a post-processing UQ method for 2D object detection termed MetaDetect. We compute prediction-wise uncertainty metrics based on predictions that were suppressed in the forward pass. Once we trained a meta model based on these metrics and the quality of the respective predictions, for which we need the ground truth, we obtain a quality estimation for each instance, especially without any ground truth information. These quality estimates provide better statistical separability between true positives and false positives compared to the naive confidence score of the underlying object detector and the uncertainties obtained by Monte-Carlo dropout.

In Chapter 8, we introduce LidarMetaDetect (LMD), inspired by MetaDetect (Chapter 4), which is a post-processing UQ method for 3D object detection based on Lidar point clouds. We also get prediction-wise quality estimates and in addition we show that the quality estimates obtained by a meta model are well-calibrated compared to the over-confident score of the object detector.

Application of Uncertainty Quantification Methods in Active Learning for Object Detection One application of UQ in object detection is active learning (AL). AL is an alternating process of training a model and labeling data. The AL cycle starts with an initially labeled set of images. Iteratively, a model is trained, images are determined from previously unlabeled images that are labeled, and then added to the current training dataset. Overall, this iteration continues until either a number of AL steps or a performance limit is reached. At each step, the test performance is measured to compare different AL methods, which differ in the determination of the images to be labeled, called *query*. The most naive baseline method is random query, in which the images to be labeled are drawn randomly from all previously unlabeled images. A single AL experiment requires multiple AL steps for each query method, as well as multiple runs of each of these methods to statistically validate the experiment, which leads to very high computational costs and thus in very high research and development costs. Nevertheless, there are some works that have introduced new AL methods in object detection. Instance-wise uncertainties are thereby determined by the application of dispersion measures of class probabilities [10, 135] or Monte-Carlo dropout [59]. In addition, there are methods that either predict an image-wise loss [179] or modify the loss function to obtain uncertainty estimates [24, 183].

Due to the high cost and duration of AL experiments, previous works have often shown AL results for only one dataset and/or architecture. In addition, the results of the individual works are difficult to compare with each other, since no general AL setup exists and each work chooses its own setup, such as the dataset, the architecture, the number of initially labeled images, the size of the query, etc. Therefore, a fast and general AL development environment is desirable.

In Chapter 5, we introduce a sandbox environment which enables faster development and fair comparability of AL methods. This sandbox contains two object detection datasets, which require a comparatively small amount of training data to obtain a good performance. Therefore, only down-scaled versions of state-of-the-art object detectors are used for the AL experiments. This results in accelerated AL experiments with a factor of up to 32 compared to experiments on commonly used datasets in object detection and state-of-the-art object detectors. We show with correlation coefficients over the entire AL course, that different query strategies on conventional AL setups have similar rankings compared to the results obtained using our sandbox, i.e., new methods can be developed cost-efficiently using our sandbox and generalize well to common setups.

Uncertainty Quantification Methods for Label Error Detection Labeling data by a human is a dull and time-consuming task, as well as error-prone, resulting in erroneous benchmark datasets [113, 132]. Once data is labeled, networks are trained and evaluated based on it, and labels are typically not questioned. By inspecting some false positive predictions (see Chapter 4), we repeatedly find actual true positives but categorized as false positives based on incorrect labels. Although the idea of automated label error detection has been around since the early 2000s in the context of part-of-speech annotation, this area has only been little explored. In terms of image classification, real label errors were found on the MNIST [91] dataset due to prediction uncertainties [113] and studied to which extent they affect benchmark results [114]. For the task of semantic segmentation, a post-processing UQ method [132] identifies missing labels or labels with an incorrect class affiliation in the Cityscapes [27] dataset. In object detection, a UQ method based on the class probabilities is applied to find labels with an incorrect class affiliation [70]. Note, that these label errors were simulated and are no real label errors.

In Chapter 6, we introduce a label error detection method for object detection. This method is based on the loss of the detector, i.e., the calculated label error scores are based not only on the prediction itself, but on the agreement and disagreement between prediction and the potentially erroneous labels. We consider four different types of label errors: missing labels, correctly localized labels with an incorrect class affiliation, labels with correct classification but inaccurate localization, and labels that actually represent background. We introduce a benchmark with simulated label errors, and we also detect real label errors on commonly used datasets in object detection. Compared to baselines methods, such as the UQ method based on class probabilities [70], our method is the only one that detects all four types of label errors efficiently.

Furthermore, in Chapter 8, we use the prediction-wise uncertainty estimates obtained by LMD to identify missing labels, labels with an incorrect class affiliation and labels with an inaccurate localization for 3D bounding box annotation with Lidar point clouds. In all experiments shown, label error detection with LMD outperforms a random baseline as well as the naive confidence score of the object detector significantly.

Uncertainty Quantification Methods in Active Learning with Noisy Oracle Since labeling data is error-prone, it is not surprising that the commonly used datasets in object detection contain label errors (Chapter 6). Therefore, many published AL results in object detection are based on erroneous labels, i.e., label errors are present during training and evaluation. Nevertheless, no AL methods in object detection incorporate label errors so far, whereas some works exist for the classification task. In [80], an active label correction algorithm is introduced that robustly estimates label confidence values via classification entropy, while preventing redundancies during the cleaning procedure. In [180], suspicious noisy samples are filtered via comparison of predicted class probabilities and the current (possibly incorrect) label. Then, these samples are reviewed by either a weak and more cost-saving or a strong and more cost-intensive labeler, which provide either binary or categorical feedback. The QActor module, presented in [181], introduces a label cleaning module in the AL cycle, that filters proposals for noisy labels by the highest loss between prediction and actual label, which are then reviewed by the oracle.

In Chapter 7, we introduce for the first time an AL method that incorporates label errors within the AL cycle in object detection, where we present a review module which becomes active after querying and labeling data. During review, we discriminate between correct label and label error, where re-labeling the latter results in a cleaner dataset for training the model in the next AL step. To generate label error proposals, we apply the label error detection method from Chapter 6 and compare it to a random review baseline. In total, we detect two types of label errors: missing labels and labels with a correct localization but an incorrect class affiliation. Note, that the choice of the query strategy is independent of the choice of the review method. We show that if label errors occur during the labeling process, the review can increase the test performance significantly. However, this requires a review with high precision in terms of label error detection, such as the instance-wise loss (Chapter 6), whereas the random review even leads to a slight reduction in test performance compared to AL methods without reviewing labels.

Structure The remainder of the present thesis is structured as follows: in Chapter 2, we introduce theoretical foundations of neural networks, the task of object detection and uncertainty quantification applied to object detection. The introduced theoretical basics and the corresponding notations should help to understand the developed methods discussed in the following chapters. In more detail, we present the structure of neural networks, how they are trained, which learning tasks they can approximate and which prerequisites have to be met. Furthermore, we introduce convolutional neural networks (CNNs), since the architectures of modern object detectors are based on CNNs. Afterwards, we present the task of object detection and lead step-by-step through the training and inference process, in order to establish the foundations for the methods that follow and later show at which point in the training and inference process each method applies. Finally, the topic of uncertainty and its application to object detection will be highlighted, where all the presented methods will be employed in at least one of the following chapters. Then, in Chapter 3, we summarize the basic ideas and most important results of the following chapters to present an overview of all topics covered and their relationships to each other. Chapter 4 - Chapter 8 have all the same structure: the introduction is followed by related work, method description, the numerical results and the respective chapter closes with a conclusion. In Chapter 4, we present a post-processing UQ method for 2D object detection. Chapter 5 introduces an AL sandbox that enables fast development and fair comparability of AL methods and generalizes well to common AL setups with state-of-the-art datasets and architectures. A loss-based label error detection method for 2D object detection is introduced in Chapter 6, which is able to identify four different types of label errors efficiently. The ideas of Chapter 5 and Chapter 6 are merged together in Chapter 7, where we assume label errors in the generic AL cycle. We present a review module that allows label errors to be reviewed and corrected during the experiment, leading to a significant increase in performance throughout the runtime. In Chapter 8, the idea of Chapter 4 is extended to 3D object detection with Lidar point clouds, where we use the generated uncertainty estimates to identify real label errors on commonly used datasets as a direct application of this post-processing UQ method. Finally, we close with a conclusion in Chapter 9, comparing the obtained results and discuss further research and application areas for future work.

CHAPTER 2 REVIEW OF BASIC MATERIAL

This chapter provides an overview of the foundations that are prerequisites for the chapters that follow. The idea of supervised learning for classification and regression is introduced in Section 2.1. The basics of neural networks and especially convolutional neural networks for the task of image classification are presented in Section 2.2. Both subsections are mainly based on the textbooks [51, 147]. The structure of the networks with the division into different layers, the application of activation functions, as well as convolution and pooling operations and their transposed are explained. Furthermore, we introduce how neural networks learn from data and why they are able to approximate different function classes, such as polynomials. Then, in Section 2.3, the generic object detection training and inference are explained, as well as the general object detection architecture. Afterwards, two state-of-the-art architectures, YoloV3 and Faster R-CNN, are discussed followed by the commonly used evaluation metrics in object detection, such as precision, recall and mean average precision. The chapter concludes with the task of uncertainty quantification, the distinction between aleatoric and epistemic uncertainty, and the application of uncertainty-based methods in object detection in Section 2.4, which is mainly based on [72].

2.1 Supervised Learning

Machine learning algorithms learn from data. These algorithms can be broadly categorized into supervised, unsupervised and reinforcement learning tasks.
In reinforcement learning, an agent learns a strategy based on a reward system,

such as learning to play backgammon from scratch [157]. For supervised and unsupervised tasks, learning is based on a given dataset

$$\mathcal{X} = \big\{x_{i,j},\, i \in \{1, \dots, n\},\, j \in \{1, \dots, d\}\big\}, \tag{2.1}$$

which contains n data points, each described by d different features. For a dataset consisting of camera images, n is the amount of different images and the number of corresponding features d is the number of pixels within a single image. With $\mathcal{X}$ as input, unsupervised learning algorithms aim to learn useful properties about the structure of the dataset, i.e., the probability distribution that generated $\mathcal{X}$. For supervised learning tasks, there exists a true probability distribution $p(x, y)$ that assigns to each input $x \in \mathbb{R}^d$ a probability for each label $y \in D_{\mathcal{Y}}$. Here, $D_{\mathcal{Y}}$ represents the target domain, which can be a set of C classes $D_{\mathcal{Y}} = \{1, \dots, C\}$ (classification) or the set of real numbers $D_{\mathcal{Y}} = \mathbb{R}$ (regression). Often it is simplistically assumed that there is a unique ground truth function $f^* : \mathbb{R}^d \to D_{\mathcal{Y}}$, which provides a unique label to each input, e.g., a one-hot probability vector for classification. The overall goal of supervised learning algorithms is to learn to predict the labels $\mathcal{Y}$ from the input $\mathcal{X}$, i.e., a function f is learned as an approximation for f^*. This approximation can be learned by different algorithms, such as linear or logistic regression, support vector machines or neural networks, where the selection of the appropriate model depends on the underlying problem.

2.2 Neural Networks

Artificial neural networks are statistical models inspired by the functionality of the human brain. Based on this structure, a neural network consists of many individual but interconnected neurons that can be used to learn and predict highly complex tasks on a given data basis $\mathcal{X}$. As has been observed in recent years, such neural networks have applications in computer vision, such as perception in automated driving or medical diagnosis. The following subsection is mainly based on [147].

Feed Forward Neural Networks

A neural network can be interpreted as a directed graph $G = (V, E)$, with neurons as nodes and the interconnections of the neurons as edges. The strength of the connection between neurons is described by a weight function $w : E \to \mathbb{R}$. The input of a neuron is the weighted sum of the outputs of the neurons with an incoming edge. In addition, a bias can be added due to a single neuron that does not receive any input. Furthermore, neural networks can be divided into L

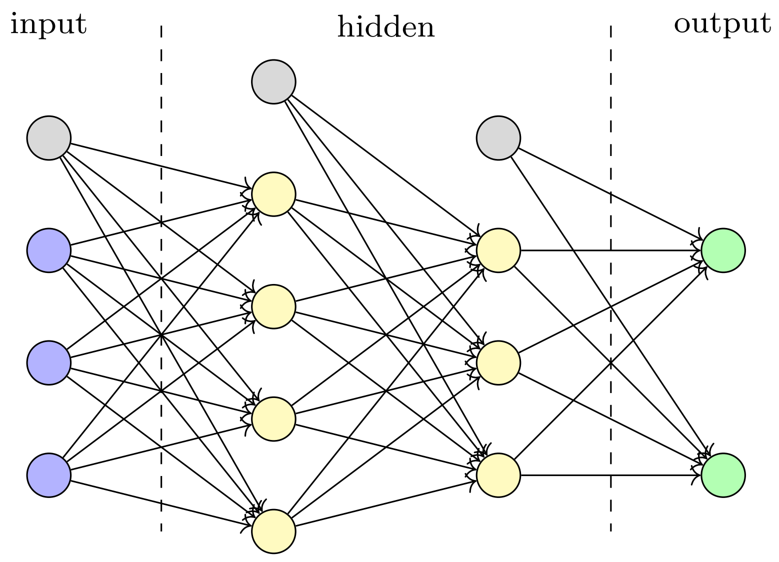


Figure 2.1: An exemplary structure of an FFN with one input layer (blue), two hidden layers (yellow), one output layer (green) and bias units (gray).

different layers, i.e., each neural network has one input layer, a various number of hidden layers, and one output layer. Thus, the nodes of the graph G can be decomposed into $V = \dot\cup_{l=0}^{L} V^{(l)}$, such that every node in $V^{(l-1)}$ is connected to some nodes in $V^{(l)}\ \forall\, l \in \{1, \ldots, L\}$. If the neural network does not include cycles, it is called *feedforward*. An example structure of a feedforward neural network (FFN) is shown in fig. 2.1. An FFN is called *fully-connected*, if each neuron is connected to all neurons of the previous layer and all neurons of the following layer, see fig. 2.1. The architecture of a neural network is fixed in advance of the training process, thus neural networks learn only by weight adjustments. The number of trainable parameters is determined by the complexity of the network. The complexity depends on the number of layers L (*depth*) and the maximum number of neurons per layer $\max\limits_{l\in\{0,\ldots,L\}} N^{(l)}$ (*width*). The output of a neural network is determined by a *forward pass*, i.e., by a chain of functions whose number corresponds to the depth L of the network:

$$f(x, w) = (f^{(L)} \circ f^{(L-1)} \circ \ldots \circ f^{(1)})(x, w) = f^{(L)}(f^{(L-1)}(\ldots(f^{(1)}(x, w)))), \quad (2.2)$$

with given input x, weights w and the number of layers L. The output of the network is determined iteratively and thus the output of layer l can be interpreted as a function of the output of the previous layer $l-1$:

$$h^{(l)} = f^{(l)}(h^{(l-1)}) = \phi^{(l)}(W^{(l)} h^{(l-1)} + b^{(l)}) = \phi^{(l)}(a^{(l)}),\ \forall l \in \{1, \ldots, L\}, \quad (2.3)$$

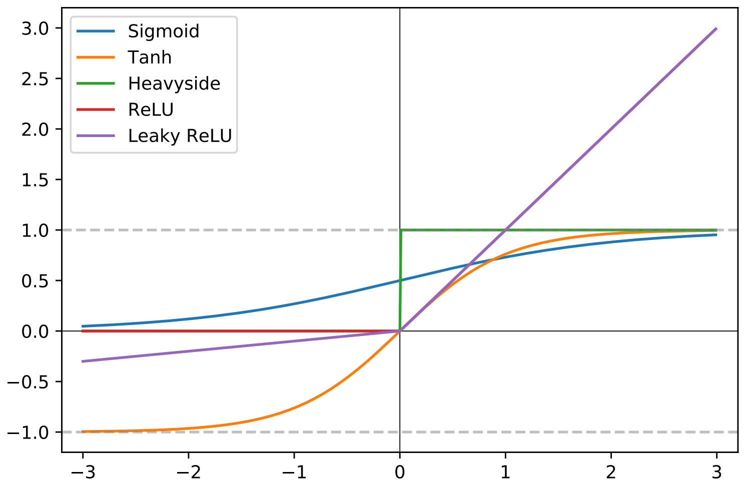


Figure 2.2: Activation functions on an interval of $[-3, 3]$.

with $f^{(l)} : \mathbb{R}^{N^{(l-1)}} \to \mathbb{R}^{N^{(l)}}$, $W^{(l)} \in \mathbb{R}^{N^{(l-1)} \times N^{(l)}}$ as weights, $b^{(l)} \in \mathbb{R}^{N^{(l)}}$ as bias and $\phi : \mathbb{R}^{N^{(l)}} \to \mathbb{R}^{N^{(l)}}$ as activation function for the input $a^{(l)}$ of layer l. Note, that the weights w can be rewritten as

$$\tilde{w} = \begin{pmatrix} 0 & 0 & \dots & 0 \\ W^{(1)} & 0 & \dots & 0 \\ \vdots & \ddots & 0 & \vdots \\ 0 & 0 & W^{(L)} & 0 \end{pmatrix}. \tag{2.4}$$

Activation Functions In (2.3), a layer-dependent activation function $\phi^{(l)}$ is applied to the input $a^{(l)}$ of the corresponding layer $l \in \{1, \dots, L\}$. Equipping an FFN with activation functions can happen on a trial and error basis. See fig. 2.2 for an overview of the activation functions presented in the following.
The *Heavyside* function is defined as:

$$\text{Heavyside}(a) = \mathbb{1}_{a>0}. \tag{2.5}$$

This function activates the neuron when the input is greater than zero and sets the output of the neuron to zero otherwise. Neural networks usually learn with gradient-based optimization methods and backpropagation, assuming differentiability of the activation functions. Since the Heavyside function is not differentiable, and the derivative even disappears completely, differentiable approximations of the Heavyside function with non-disappearing gradients are applied, i.e., the *sigmoid*, or also called *logistic*, function.

This function and the corresponding derivate are defined by:

$$\operatorname{sig}(a) = \frac{1}{1+e^{-a}} = \frac{e^a}{e^a+1} \quad \text{and} \quad \operatorname{sig}'(a) = \frac{e^{-a}}{(1+e^{-a})^2}. \tag{2.6}$$

Analogous to the Heavyside function, the function values of the sigmoid function range from zero to one. Moreover, the function is monotonically increasing and, in particular, differentiable. Furthermore, the curve of the *sigmoid* function flattens very quickly, see fig. 2.2. The gradients are determined with the chain rule during backpropagation, resulting in activations within the interval of $(0, 1)$. This can lead to *vanishing gradients*, especially in the first layers of a deep neural network, and in the worst case the gradients become zero and the neural network cannot continue learning. An improvement is obtained by the *hyperbolic tangent* function, which is a scaled version of the sigmoid function:

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \quad \text{and} \quad \tanh'(a) = 1 - \left(\frac{e^a - e^{-a}}{e^a + e^{-a}}\right)^2. \tag{2.7}$$

The range of the function is in $[-1, 1]$ and the values of the derivative around the origin are also larger compared to the sigmoid function. However, the gradients also tend quickly towards zero, such that the problem of vanishing gradients still exists.
The *rectified linear unit* (ReLU) prevents this phenomenon:

$$\mathrm{g}(a) = \max\{0, a\} \quad \text{and} \quad \mathrm{g}'(a) = \begin{cases} 0, & \text{if } a < 0 \\ 1, & \text{if } a > 0 \\ \text{undefined}, & \text{if } a = 0. \end{cases} \tag{2.8}$$

If $\mathrm{g}'(0) = 0$ is chosen for the derivative of the ReLU function, g is differentiable everywhere and $\mathrm{g}'(a) = \text{Heavyside}(a)$. Thus, the computation of the derivative is not only very efficient, but also avoids the problem of vanishing gradients. Neurons whose gradients are equal to zero do not provide training feedback for upstream layers during backpropagation. If this phenomenon happens in various neurons, the capacity of the neural network will decrease. This problem can be solved with the *LeakyReLU* activation, which allows a small gradient even when a neuron is not activated:

$$\tilde{\mathrm{g}}(a) = \begin{cases} 0.01 \cdot a, & \text{if } a \leq 0 \\ a, & \text{if } a > 0. \end{cases} \quad \text{and} \quad \tilde{\mathrm{g}}'(a) = \begin{cases} 0.01, & \text{if } a \leq 0 \\ 1, & \text{if } a > 0. \end{cases} \tag{2.9}$$

However, the LeakyReLU is rarely applied due to a worse performance compared to the ReLU activation.
Activation functions are also applied in the output layers of neural networks. For

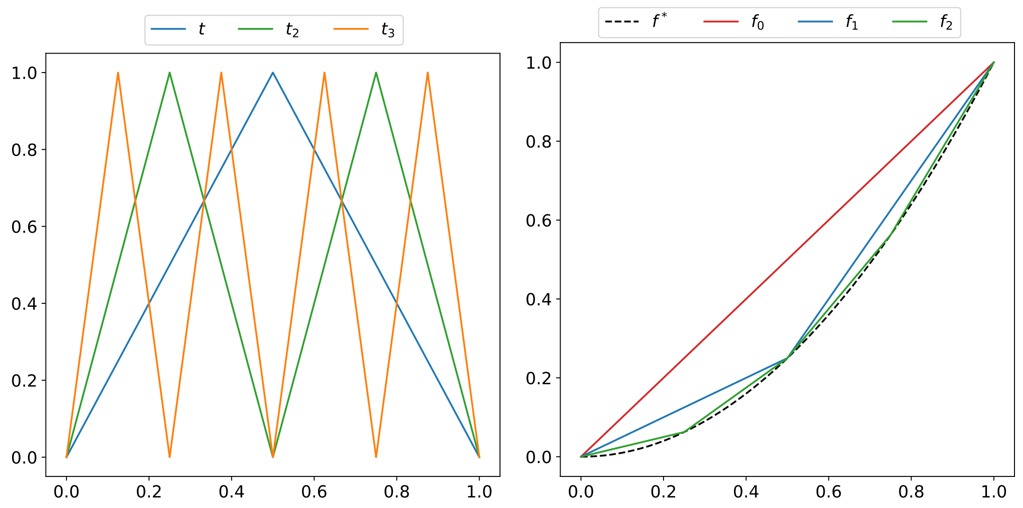


Figure 2.3: Left: the "tooth" function t from (2.13) and the iterated "sawtooth" functions t_2 and t_3 from (2.14). Right: $f^*(x) = x^2$ and the approximation functions f_0, f_1 and f_2.

instance, the *softmax* function is often applied as an activation function in the output layer of a multi-class classification model, in order to make the output interpretable. Assuming that the input has to be categorized into one out of C classes, the output of neuron a_i is defined as:

$$\text{softmax}(a_i) = \frac{e^{a_i}}{\sum\limits_{i=1}^{C} e^{a_i}}. \tag{2.10}$$

Since $\sum\limits_{i=1}^{C} \text{softmax}(a_i) = 1$, $\text{softmax}(a_i)$ can be interpreted as the probability that the input belongs to class $i \in \{1, \dots, C\}$.

Universal Approximation A neural network can be described by the parameters (V, E, ϕ, w). Here, (V, E, ϕ) is the *architecture* of the neural network or the *hypothesis class*:

$$\mathcal{H}_{V,E,\phi} = \{h_{V,E,\phi,w} : \mathbb{R}^{N^{(1)}} \to \mathbb{R}^{N^{(L)}} \quad \text{with} \quad w : E \to \mathbb{R}\}. \tag{2.11}$$

One of the fundamental questions is how well can the best function $f : \mathbb{R}^d \to D_{\mathcal{Y}}$ of the hypothesis class $\mathcal{H}_{V,E,\phi}$ approximate the ground truth function $f^* : \mathbb{R}^d \to D_{\mathcal{Y}}$ with $f^*(\mathcal{X}) = \mathcal{Y}$. The universal approximation theorem states that FFNs with a linear output layer and at least one hidden layer with a non-linear activation

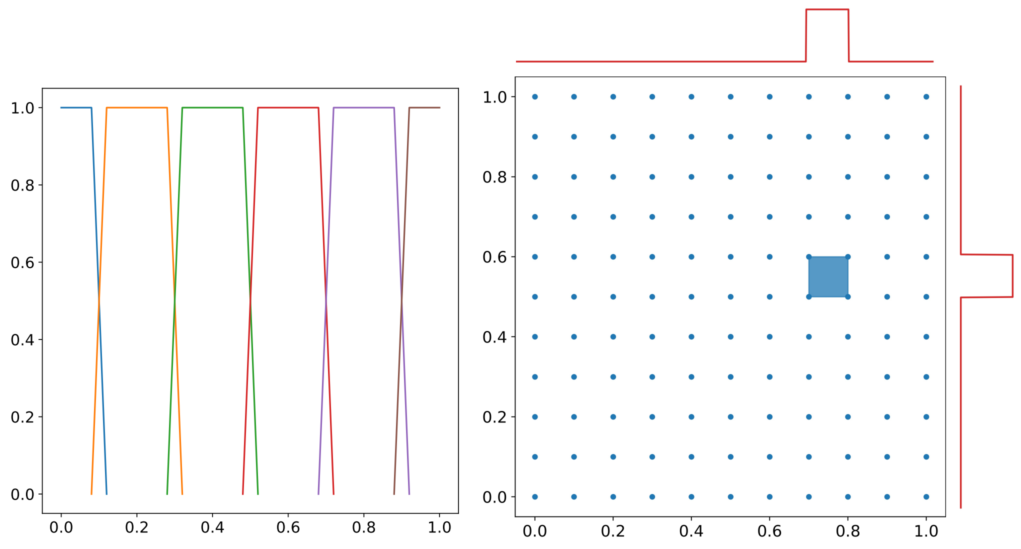


Figure 2.4: Left: unity partitioning functions $\psi(x)$. Right: partitioning of a 2D-unity square with an activated region in blue.

function can approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$ ([69]). These theorems have been proven for different activation functions, see [28] for the sigmoid function ($\mathcal{H}_{V,E,\text{sig}}$) and for the sketch of the proof for the ReLU function ($\mathcal{H}_{V,E,\text{g}}$), we follow [176]:

We assume an FFN with several input neurons, an amount of hidden layers with ReLU as activation function and one output neuron. The *approximation error* for the ground truth function $f^* : [0,1]^d \to \mathbb{R}$ and its approximation $f : [0,1]^d \to \mathbb{R}$ is given by:

$$\|f^* - f\|_\infty = \max_{x \in [0,1]^d} |f^*(x) - f(x)|. \tag{2.12}$$

First, it is shown that the function $f^*(x) = x^2$ can be approximated with any error $\epsilon > 0$ on the segment $[0,1]$ by a ReLU network. Therefore, we define the "tooth" function $t : [0,1] \to [0,1]$:

$$t(x) = \begin{cases} 2x, & x < \frac{1}{2} \\ 2(1-x), & x \geq \frac{1}{2} \end{cases} \tag{2.13}$$

and the iterated functions:

$$t_s(x) = \underbrace{t \circ t \circ \ldots \circ t}_{s\times}(x). \tag{2.14}$$

The latter function t_s is a “sawtooth” function with 2^{s-1} uniformly distributed “teeth”, see [156]:

$$t_s(x) = \begin{cases} 2^s(x - \frac{2k}{2^s}), & x \in [\frac{2k}{2^s}, \frac{2k+1}{2^s}], k = 0, 1, \dots, 2^{s-1} - 1, \\ 2^s(\frac{2k}{2^s} - x), & x \in [\frac{2k-1}{2^s}, \frac{2k}{2^s}], k = 0, 1, \dots, 2^{s-1}. \end{cases} \tag{2.15}$$

If s is increased by one in (2.14), the number of teeth doubles, see fig. 2.3 (left) with $t(x) = t_1(x)$. Let f_m be piece-wise linear interpolations of $f^*(x) = x^2$, then f^* can be approximated by linear combinations of t_s, see fig. 2.3 (right). The resulting $2^m + 1$ breakpoints have the identical function values as f^* at the given points. If m is increased by one, the residual between f_{m-1} and f_m can be described as a sawtooth function:

$$f_{m-1} - f_m = \frac{t_m(x)}{2^{2m}} \tag{2.16}$$

and therefore applies:

$$f_m(x) = x - \sum_{s=1}^{m} \frac{t_s(x)}{2^{2s}}. \tag{2.17}$$

The function f_m approximates f^* with error $\epsilon_m = 2^{-2m-2}$, which decreases with increasing m. Considering (2.17), f_m consists only of the input x and the linear interpolations $t_1, \dots, t_m$. The latter can be implemented by a finite ReLU network, since the single application of the “tooth” function from (2.13) can be rewritten as $t(x) = 2g(x) - 4g(x - \frac{1}{2}) + 2g(x - 1)$, and the “sawtooth” function from (2.14) can be implemented by nesting layers applying the tooth function.
As of now, it is shown that we can approximate x^2 arbitrarily well, and therefore also y^2 and $(x + y)^2$. Since

$$xy = \frac{1}{2}((x + y)^2 - x^2 - y^2), \tag{2.18}$$

a ReLU network is also able to multiply multiple input neurons, i.e., polynomials of higher order, as well as polynomials of higher dimension can be approximated by a ReLU network (with sufficient depth and width). Not only polynomials of any form can be approximated with ReLU networks, but also different classes of functions, e.g., functions which have a local Taylor expansion up to degree n at each point with a sufficiently small residual term. These local Taylor series, that locally approximate the underlying function, are polynomials up to degree n and can be approximated by a ReLU network. To limit the local interpolation of the Taylor series to a certain range, the unity is decomposed as a product of step functions ψ:

$$\psi(x) = \begin{cases} 1, & |x| < 1, \\ 0, & 2 < |x|, \\ 2 - |x|, & 1 \leq |x| \leq 2, \end{cases} \tag{2.19}$$

see fig. 2.4 (left). With this partition of the unity, the interpolations of the Taylor series can be locally restricted, see fig. 2.4 (right), and implemented by a ReLU network. Furthermore, fig. 2.4 also shows how to realize higher dimensions of the feature space.

Note, that the approximation of more complex functions with basis functions is not a new concept, but the argument goes back to finite element theory [9]. Yarotsky [176] transferred this idea to neural networks with ReLU as activation function, where error bounds can also be given depending on the width and depth of the neural network.

Learning from Data

After demonstrating which classes of functions can be approximated by neural networks, we introduce a learning algorithm for neural networks, i.e., how the adjustments of the weights in the neural network are realized. The training procedure of a neural network is based on a given learning function, called *loss function*. The ground truth function f^* is approximated by f, which is learned based on the loss function and the *stochastic gradient descent*, requiring *backpropagation* to determine the gradients. Furthermore, various regularizations are used during the learning process to prevent the network from memorizing the training data (*overfitting*).

Loss Functions The loss function determines the discrepancy between predictions and labels based on the training data $\mathcal{S} = \{(x_1, y_1^*), \dots, (x_n, y_n^*)\}$, with inputs x_i and labels y_i^*, $i = 1, \dots, n$. Thereby, x is generated by a probability distribution D over a domain set $\mathcal{X}$ and then labeled by $f^*(x) = y^*$, where y^* is determined from the set of potential labels $D_{\mathcal{Y}}$. The learner has to output a hypothesis $h_{\mathcal{S}} : \mathcal{X} \to \mathcal{Y}$ to predict the labels, especially for new data points. Usually, D and f^* are unknown and the task is to learn the latter. In order to measure the performance of the current hypothesis, the *error of the hypothesis* is defined by the probability that the hypothesis predicts a wrong label on a random data point generated by D:

$$\mathcal{L}_{D,f^*}(h_{\mathcal{S}}) = \mathbb{P}_{x \sim D}[h_{\mathcal{S}}(x) \neq f^*(x)] = D(\{x : h_{\mathcal{S}}(x) \neq f^*(x)\}). \tag{2.20}$$

$\mathcal{L}_{D,f^*}(h_{\mathcal{S}}) = \mathcal{L}(h_{\mathcal{S}})$ is also called *generalization error*, but is rather a theoretical expression, since in practice D and f^* are unknown. Since only the training dataset $\mathcal{S}$ is available as input to the learner, the generalization error is approximated by the *empirical risk*:

$$\mathcal{L}_{\mathcal{S}}(h_{\mathcal{S}}) = \frac{|i \in [n] : h_{\mathcal{S}}(x_i \neq y_i^*)|}{n}, \tag{2.21}$$

where a hypothesis that minimizes $\mathcal{L}_{\mathcal{S}}(h_{\mathcal{S}})$ is called *empirical risk minimizer* (ERM) [162].
In general, there exists a hypothesis $h_{\mathcal{S}}$ with $\mathcal{L}_{\mathcal{S}}(h_{\mathcal{S}}) = 0$, i.e., perfect performance on the training data, but a high generalization error. This memorization of the training data is called *overfitting* and can be mitigated or, in the best-case, totally avoided by an *inductive bias*, e.g., one possibility is to choose the network architecture independently of the training data, i.e., to determine a hypothesis class $\mathcal{H}_{V,E,\phi}$. With this inductive bias, the $\text{ERM}_{\mathcal{H}}$ learner uses the ERM rule and determines the hypothesis $h_{\mathcal{S}} \in \mathcal{H}$ with the lowest possible error for the training dataset $\mathcal{S}$:

$$\text{ERM}_{\mathcal{H}}(\mathcal{S}) = \underset{h_{\mathcal{S}} \in \mathcal{H}}{\operatorname{argmin}} \mathcal{L}_{\mathcal{S}}(h_{\mathcal{S}}). \tag{2.22}$$

If the hypothesis $h_{\mathcal{S}} \in \mathcal{H}$ that minimizes $\mathcal{L}_{\mathcal{S}}(h_{\mathcal{S}})$ is not unique, one of these hypotheses can be freely chosen. The difference of the generalization error and the empirical risk is defined as the *estimation error* ($|\mathcal{L}(h_{\mathcal{S}}) - \mathcal{L}_{\mathcal{S}}(h_{\mathcal{S}})|$) and the difference of the empirical risk and the lowest possible error with inductive bias is defined as the *approximation error* ($|\mathcal{L}_{\mathcal{S}}(h_{\mathcal{S}}) - \min_{h_{\mathcal{S}} \in \mathcal{H}} \mathcal{L}_{\mathcal{S}}(h_{\mathcal{S}})|$). Restricting the hypothesis class reduces the possibility of overfitting (lower estimation error), but also results in a stronger inductive bias (higher approximation error). This tradeoff is also called *bias-complexity-tradeoff* [47].

Since the empirical risk of misclassification from (2.22) is not differentiable, other objective functionals are used in practice. Maximum likelihood estimation [2] (MLE) determines such an empirical risk minimizer, where the parameters θ of a probability distribution D are estimated based on a given dataset $\mathcal{S}$. In general, the goal of MLE is to determine the model parameters that maximize the likelihood function. Furthermore, MLE is an empirical risk minimizer for the log loss function of the generalization error:

$$-\log\big(D_\theta(x)\big), \tag{2.23}$$

with input $x \sim D$ and the distribution D_θ, which is dependent of the parameter θ. Assuming that the input x is independently and identically distributed according to D_θ, (2.23) is called *negative log-likelihood.* Since the underlying probability distribution is unknown, and only the training data $\mathcal{S}$ is available, the generalization error from (2.23) is replaced with the empirical risk:

$$\operatorname*{argmin}_{w} \sum_{i=1}^{n} \Big(-\log\big(D_\theta(x_i)\big) \Big). \tag{2.24}$$

Note, that $x \sim D$, and not $x \sim D_\theta$, and assuming that x_i, $i = 1, \dots, n$, are independently and identically distributed according to D, the expected generalization error of θ is given by:

$$\begin{aligned}
&\mathbb{E}_x[-\log(D_\theta(x))] && (2.25)\\
&= \mathbb{E}_x\left[\sum_{i=1}^{n} \Big(-\log\big(D_\theta(x_i)\big) \Big)\right] && (2.26)\\
&= -\sum_{i=1}^{n} \Big(D(x_i)\log\big(D_\theta(x_i)\big) \Big) && (2.27)\\
&= \sum_{i=1}^{n} \left(D(x_i)\log\left(\frac{1}{D_\theta(x_i)} \right)\right) && (2.28)\\
&= \sum_{i=1}^{n} \left(D(x_i)\log\left(\frac{D(x_i)}{D_\theta(x_i)} \frac{1}{D(x_i)} \right)\right) && (2.29)\\
&= \sum_{i=1}^{n} \left(D(x_i)\log\left(\frac{D(x_i)}{D_\theta(x_i)} \right)\right) + \sum_{i=1}^{n} \left(D(x_i)\log\left(\frac{1}{D(x_i)} \right)\right) && (2.30)\\
&= \mathrm{KL}(D(x)||D_\theta(x)) + \mathrm{H}(D(x)). && (2.31)
\end{aligned}$$

KL represents the Kullback-Leibler divergence [85], which is a measure of the difference between two probability distributions, with $\mathrm{KL}(D(x)||D_\theta(x)) \geq 0$ and equality if and only if $D(x) = D_\theta(x)$. H represents the Shannon entropy [148], which is a measure of average information contained in a distribution.

The general goal is to have the output of the neural network be identical to the labels $\mathcal{Y}$. For considering a multi-class classification problem with C classes, the output from the last layer L, $\hat{y} = \mathrm{softmax}(a^{(L)})$, is a probability vector $\hat{y}^i \in [0,1]^C$, $\forall i \in \{1, \dots, n\}$, and the labels $\mathcal{Y}$ are present as unit vectors $y^{*^i} \in \{0,1\}^C$, $\forall i \in \{1, \dots, n\}$. In (2.27), by replacing the true distribution $D(x)$ by the labels $\mathcal{Y}$ and the predicted distribution $D_\theta(\mathcal{X})$ by the output $\hat{y}$, the *cross-entropy* is obtained by:

$$\mathcal{L}(\hat{y}, y^*) = -\sum_{i=1}^{C} y^{*^i} \log(\hat{y}^i). \tag{2.32}$$

By minimizing cross-entropy, the output $\hat{y}$ converges to the given probability distribution of the labels y.

Stochastic Gradient Descent For some classes of hypotheses, such as linear regression, the ERM can be computed with a closed form solution, which becomes infeasible for neural networks. Therefore, the following two paragraphs refer only to the training of neural networks. Using the loss function, which measures the difference between output and labels, *stochastic gradient descent* (SGD) [6] minimizes this difference. SGD is an iterative algorithm that takes a hypothesis class $\mathcal{H}$ with an initial hypothesis $h_{w_0} \in \mathcal{H}$ as input, which is determined by the initial weights w_0, as well as the direction of descent, the step size $\epsilon > 0$ and a stopping condition. The stopping condition can be a number of iterations T, or the arrival at an (approximate) local minimum (*early stopping*). The direction of descent is defined as the direction of the steepest descent and the weights w are adjusted every iteration:

$$w_{i+1} = w_i - \epsilon \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_{w_i} \mathcal{L}(\hat{y}, y) \right). \tag{2.33}$$

The output of the network $\hat{y}$ and thus the gradient of the loss function depends on the weights w_i of the current iteration i. In order to circumvent memory restrictions, only a randomly chosen *batch* $m \ll n$ of data points is used in a single iteration, rather than the whole training dataset, resulting in:

$$w_{i+1} = w_i - \epsilon \left(\frac{1}{m} \sum_{i=1}^{m} \nabla_{w_i} \mathcal{L}(\hat{y}, y) \right). \tag{2.34}$$

To ensure that each training data point has an impact on the learning process, the iterations can be divided into epochs, in which each data point enters a batch at least once. The step size ϵ is a hyperparameter that controls the stagnation of the SGD, which is mostly tuned by trial and error. In practice, ϵ is not identical for each iteration, but decreases as the training process progresses, resulting in step size schedules ϵ_i, $i \in \{1, \dots, T\}$ that adjust ϵ depending on the current training progress [29]. A further hyperparameter is the *momentum* $\alpha \in [0, 1)$, which increases the rate of stagnation dramatically [120]. Momentum uses decreasing averaged gradients from the previous iterations for the current weight update:

$$\nu_{i+1} = \alpha \nu_i - \epsilon \left(\frac{1}{m} \sum_{i=1}^{m} \nabla_{w_i} \mathcal{L}(\hat{y}, y) \right) \tag{2.35}$$

$$w_{i+1} = w_i + \nu_{i+1}, \tag{2.36}$$

with $\nu_0 = 0$. A commonly used SGD algorithm with momentum is the Adam algorithm [81], which is often used in applications.

In general, gradient-based algorithms are motivated by ideas for convex optimization problems. These algorithms are applied due to simplicity and straightforward application to neural networks, although most loss surfaces are non-convex in deep learning. Moreover, gradient-based algorithms are likely to reach approximate local minima, since finding the global minimum (empirical risk minimizer) is NP-hard [147].

Backpropagation The gradients in SGD are computed by *backpropagation* [136]. The backpropagation, and thus the adjustment of the weights, in iteration i is based on the current loss $\mathcal{L}_{w_i}(\hat{y}, y^*)$. Therefore, the partial derivatives have to be determined for each weight. When determining the partial derivatives for the weights $W^{(l)}$, which are located on the edges from layer $l-1$ to layer l, all other weights of the network are fixed. Let $\ell^{(l)} : \mathbb{R}^{N^{(l)}} \to \mathbb{R}$ be the loss function of the subnetwork defined by layers $\{l, \dots, L\}$ as a function of the neurons in layer l. The inputs of the neurons in layer l are $a^{(l)} = W^{(l)} h^{(l-1)}$ and the outputs are $h^{(l)} = \phi^{(l)}(a^{(l)})$. Then, the loss can be defined as a function of $W^{(l)}$:

$$g^{(l)}(W^{(l)}) = \ell^{(l)}(h^{(l)}) = \ell^{(l)}(\phi^{(l)}(a^{(l)})) = \ell^{(l)}(\phi^{(l)}(W^{(l)} h^{(l-1)})). \tag{2.37}$$

Let

$$\tilde{W}^{(l)} = [W^{(l)}_{1,1}, W^{(l)}_{1,2}, \dots, W^{(l)}_{1,N^{(l-1)}}, W^{(l)}_{2,1}, \dots, W^{(l)}_{N^{(l)},N^{(l-1)}}]^T \in \mathbb{R}^{N^{(l-1)} \cdot N^{(l)}} \tag{2.38}$$

be a column vector of the weights and

$$\tilde{H}^{(l-1)} = \begin{pmatrix} \tilde{h}^{(l-1)} & 0 & \dots & 0 \\ 0 & \tilde{h}^{(l-1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \tilde{h}^{(l-1)} \end{pmatrix} \in \mathbb{R}^{N^{(l)} \times (N^{(l-1)} \cdot N^{(l)})} \tag{2.39}$$

be a matrix of the outputs $h^{(l-1)}$ with $\tilde{h}^{(l-1)} = [h^{(l-1)}_1, h^{(l-1)}_2, \dots, h^{(l-1)}_{N^{(l-1)}}] \in \mathbb{R}^{N^{(l-1)}}$. Then, (2.37) can be reformulated:

$$g^{(l)}(\tilde{W}^{(l)}) = \ell^{(l)}(\phi^{(l)}(\tilde{H}^{(l-1)} \tilde{W}^{(l)})) \tag{2.40}$$

Applying the chain rule, as well as $a^{(l)} = \tilde{H}^{(l-1)} \tilde{W}^{(l)}$ and $h^{(l)} = \phi^{(l)}(a^{(l)})$, yields

$$J_{\tilde{W}^{(l)}}(g^{(l)}) = J_{h^{(l)}}(\ell^{(l)}) \mathrm{diag}((\phi^{(l)})'(a^{(l)})) \tilde{H}^{(l-1)}, \tag{2.41}$$

Algorithm 1: Backpropagation

```
   Input          : data point $(x, y)$, weights $w$, graph $(V, E)$, activation $\phi$
   Initialization: layers $V^{(0)}, \dots, V^{(L)}$ where $V^{(l)} = \{v_1^{(l)}, \dots, v_{N^{(l)}}^{(l)}\}$, weight
                   $w_{i,j}^{(l+1)}$ of $(v_j^{(l)}, v_i^{(l+1)})$
   Forward        :
 1 set $h^{(0)} = x$
 2 for $l = 1, \dots, L$ do
 3     for $i = 1, \dots, N^{(l)}$ do
 4         set $a_i^{(l)} = \sum_{j=1}^{N^{(l-1)}} w_{i,j}^{(l)} h_j^{(l-1)}$
 5         set $h_i^{(l)} = \phi^{(l)}(a_i^{(l)})$
 6     end
 7 end
   Backward       :
 8 set $\delta^{(L)} = (\phi^{(L)})'(\hat{y})$
 9 for $l = L - 1, \dots, 1$ do
10     for $i = 1, \dots, N^{(l)}$ do
11         $\delta_i^{(l)} = \sum_{j=1}^{N^{(l+1)}} w_{j,i}^{(l+1)} \delta_j^{(l+1)} (\phi^{(l)})'(a_j^{(l+1)})$
12     end
13 end
   Output         : partial derivative $\delta_i^{(l)} (\phi^{(l)})'(a_i^{(l)}) h_j^{(l-1)}$ for each edge
                   $(v_j^{(l-1)}, v_i^{(l)}) \in E$
```

with J as the Jacobian. With defining $\delta^{(l)} = J_{h^{(l)}}(\ell^{(l)})$, (2.41) can be rewritten as:

$$J_{\tilde{W}^{(l)}}(g^{(l)}) = [\delta_1^{(l)}(\phi^{(l)})'(a_1^{(l)})\tilde{h}^{(l-1)}, \dots, \delta_{N^{(l)}}^{(l)}(\phi^{(l)})'(a_{N^{(l)}}^{(l)})\tilde{h}^{(l-1)}] \tag{2.42}$$

It remains to determine $\delta^{(l)} = J_{h^{(l)}}(\ell^{(l)})$. For the last layer L applies:

$$\ell^{(L)} = \mathcal{L}(\hat{y}, y^*) \quad \text{and} \quad \delta^{(L)} = J_{h^{(L)}}(\ell^{(L)}) = (\phi^{(L)})'(\hat{y}). \tag{2.43}$$

With $\ell^{(l)}(h^{(l)}) = \ell^{(l+1)}(\phi^{(l+1)}(W^{(l+1)}h^{(l)}))$ and the chain rule, it follows for all hidden layer $l \in \{1, \dots, L-1\}$:

$$\begin{aligned}
\delta^{(l)} &= J_{h^{(l)}}(\ell^{(l)}) && (2.44)\\
&= J_{\phi^{(l+1)}(W^{(l+1)}h^{(l)})}(\ell^{(l+1)})\mathrm{diag}((\phi^{(l+1)})'(W^{(l+1)}h^{(l)}))W^{(l+1)} && (2.45)\\
&= J_{h^{(l+1)}}(\ell^{(l+1)})\mathrm{diag}((\phi^{(l+1)})'(a^{(l+1)}))W^{(l+1)} && (2.46)\\
&= \delta^{(l+1)}\mathrm{diag}((\phi^{(l+1)})'(a^{(l+1)}))W^{(l+1)} && (2.47)
\end{aligned}$$

Due to the iterative use of intermediate terms, backpropagation is very efficient in computing the gradients of a layer l. A summary of the backpropagation is shown in alg. 1.

Convolutional Neural Networks

Fully-connected FFNs perform well on many tasks, but face the *curse of dimensionality* [4] on higher dimensional inputs. Adding more features, such as higher resolution images, exponentially increases the amount of data needed to ensure proper generalization. To prevent this overparametrization, *convolutional neural networks* (CNNs) replace the matrix multiplication with a *convolution* layer in at least one layer of the FFN. This results in *sparse interactions*, since not all neurons of one layer are connected to all neurons of the subsequent layer, as well as in *parameter sharing*, since only one set of kernel-parameters is learned, rather than a set of parameters for every interconnection of the neurons of both layers. Furthermore, the convolution layer is equivariant to translation due to parameter sharing. Usually, a convolutional layer consists of a convolution operation, followed by a nonlinear activation function and a pooling operation. The following subsection is mainly based on [51].

Convolution The convolution operation for two integrable functions $I : \mathbb{R} \to \mathbb{R}$ and $K : \mathbb{R} \to \mathbb{R}$ is defined by:

$$(I * K)(t) = \int_{\mathbb{R}} I(a)K(t-a)\mathrm{d}a. \tag{2.48}$$

The first argument I is often referred to as the *input*, the second argument K as the *kernel* and the output $(I * K)$ as the *feature map*. In general, data is given in discrete form, e.g., evaluations/measurements for specific points in time. Then, with assuming that $I : \mathbb{Z} \to \mathbb{R}$ and $K : \mathbb{Z} \to \mathbb{R}$, the discrete convolution is defined as:

$$(I * K)(t) = \sum_{a=-\infty}^{\infty} I(a)K(t-a). \tag{2.49}$$

In machine learning applications, I is the input of a layer and the parameters of the kernel K with predefined size are learned by the underlying algorithm, where a two-dimensional kernel is required, if a two-dimensional image is used as input. Then, (2.49) can be reformulated as:

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.50}$$

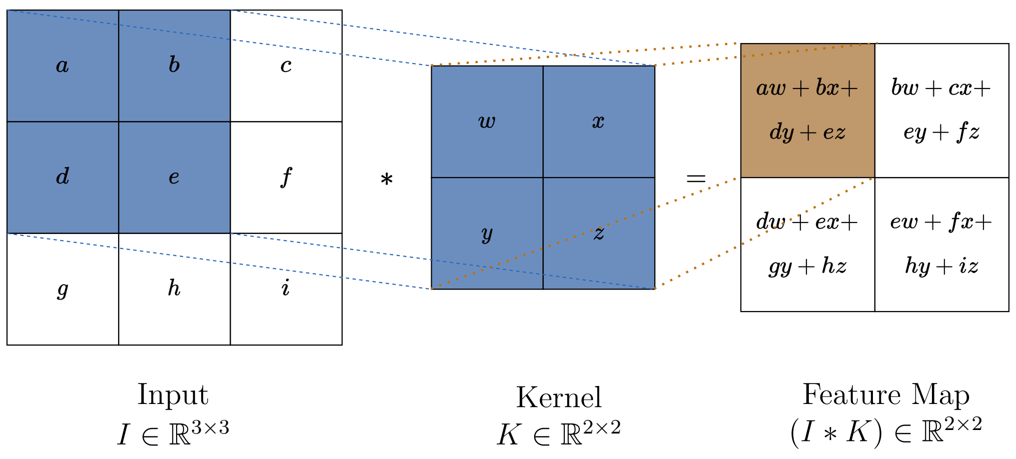


Figure 2.5: An example of a convolution on two-dimensional input data. The values of the blue square in I and the kernel K are used to compute the corresponding brown square in the feature map $(I * K)$.

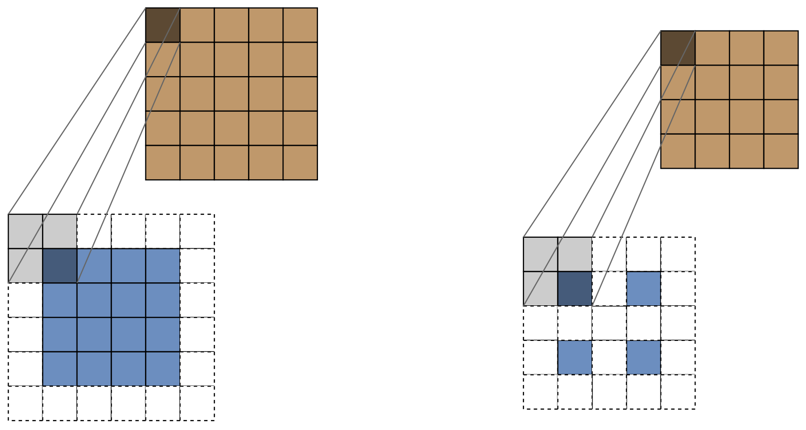


Figure 2.6: An example of two deconvolution operations on two-dimensional input data (blue squares). The higher dimensions of the outputs (brown squares) result from the zero paddings (dashed squares).

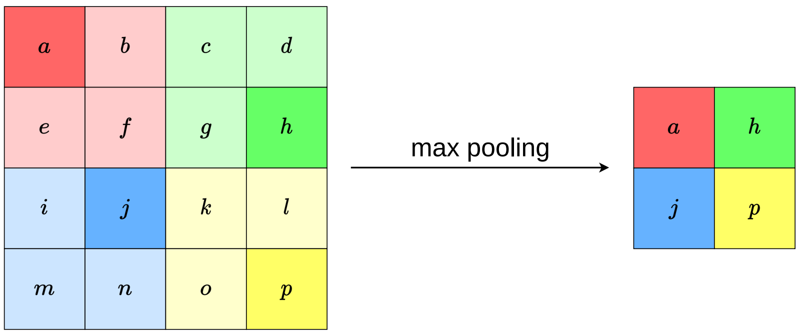


Figure 2.7: An example of max pooling operation on two-dimensional input data with a kernel size of 2×2 and stride of 2. Higher color intensities indicate the maximum value of the corresponding square.

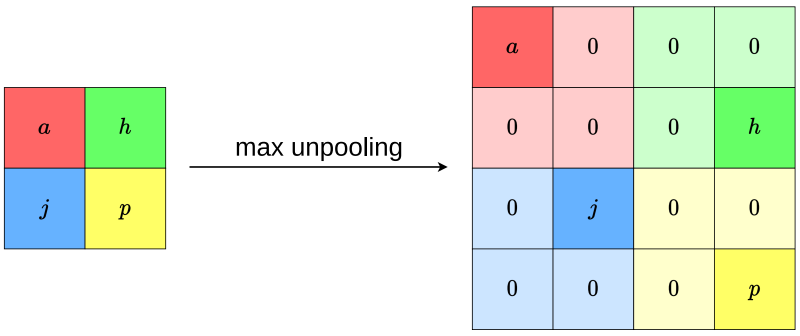


Figure 2.8: An example of a max unpooling operation with stored locations from fig. 2.7.

Due to the fact that the convolution operation is commutative, and with flipping the kernel K relative to the input I, (2.50) can be rewritten as:

$$(K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n). \tag{2.51}$$

In practice, the *cross-correlation* is used as implementation of the convolution operation:

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n). \tag{2.52}$$

Deconvolution Deconvolution [124] reverses the convolution operation, i.e., the dimensions of the input and output of the associated convolution are swapped. Thus, deconvolution leads to an increase in dimension, which is especially required if the input and the output should have identical dimensions. In fig. 2.6, two deconvolutions are shown with two inputs (blue squares) and the respective outputs

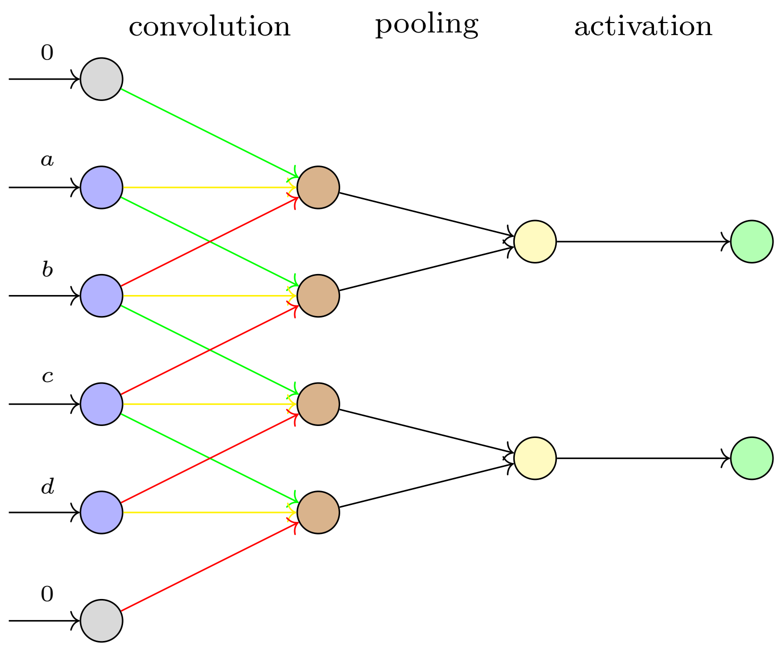


Figure 2.9: An exemplary structure of a convolutional layer consisting of the input (blue) with zero padding (white), a convolution operation (brown), one pooling operation (yellow) and an activation function (green). The colors for the convolution operation indicate shared weights.

(brown), where the increases in dimension are achieved due to *zero padding*, i.e., the input is surrounded by zeros (left), as well as individual input entries are separated by zeros (right).

Pooling A pooling operation partitions the 2D input into regular blocks, typically of size 2×2, where each block is replaced by a local statistical summary. There are different types of pooling, e.g., average, min or max pooling, as well as the L^2-norm of a block. However, in practice, the *max pooling* operation [189] is commonly used, see fig. 2.7. Max pooling operations always select the maximum value of a regular block and are therefore approximately invariant to small translations of the input. Using max pooling, it is more important whether a feature is active, rather than the exact location of this feature.

Unpooling The unpooling operation reverses the pooling operation, i.e., the input dimension for the associated pooling operation has to be identical to the output dimension of the unpooling operation. Figure 2.8 shows the max unpooling operation for the corresponding max pooling operation from fig. 2.7. The locations of the maximum entries of each regular block from the pooling operation are stored to ensure that the values for the max unpooling operation can be

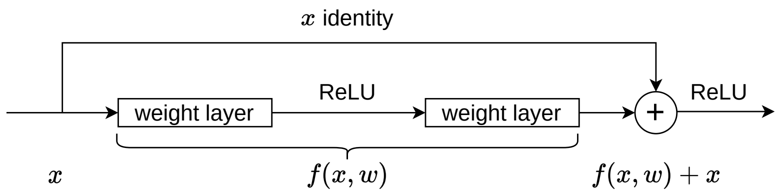


Figure 2.10: An exemplary architecture of a residual block with two weighted layers and ReLU as activation function.

assigned to the correct locations. The remaining entries are padded with zeros for max unpooling.
A simplified example of a convolution layer of a CNN with an input (with zero padding), a convolution and a pooling operation as well as a layer-specific activation function is shown in fig. 2.9.

Residual Block Intuitively, a CNN performs superior with increasing depth, as more and more complex functions can be approximated [48, 49]. However, this assumption is only conditionally valid. In practice, the performance saturates at a certain depth and then decreases rapidly, respectively the training error increases with increasing depth of the CNN [62]. This *degradation problem* is not caused by overfitting and implies that smaller neural networks are easier to optimize compared to more complex ones. Therefore, residual learning has been introduced [63] to prevent this problem. In common networks, the output of each layer is passed directly to the next layer. In networks with residual blocks (ResNet), the output of a layer is not only passed directly to the next layer, but also to further subsequent layers realized by skip connections. These skip connections apply identity mappings and merge the current output with the outputs of previous layers, see fig. 2.10. Assuming that $f(x,w)$ is the output of the subnetwork enclosed by the skip connection, these stacked layers do not learn a desired underlying mapping $\mathcal{R}(x)$ directly, but explicitly fit a residual mapping

$$f(x,w) = \mathcal{R}(x) - x, \tag{2.53}$$

where the original mapping is reformulated as $f(x,w)+x$. Residual mappings are easier to optimize than the original mappings, since it is easier to push the residual towards zero than to learn the identity with a set of nonlinear layers. The possibility of skipping entire layers allows that simple tasks can be solved efficiently with more complex networks. Since the introduction of skip connections does not lead to any additional parameters or higher computational effort, the weights of the network can be trained with SGD and backpropagation without any further adjustments.

Transformer Layer A *transformer* layer [163] learns an *attention* between the t-dimensional input *tokens* $h = [h_1, \dots, h_{\tilde{n}}] \in \mathbb{R}^{t \times \tilde{n}}$. To realize this, three weight matrices are learned: the value weights W_V, the query weights W_Q and the key weights W_K with corresponding dimension d_K. The attention $A(W_Q, W_K, W_V)$ is defined as:

$$A(W_Q, W_K, W_V) = W_V \cdot h \cdot \text{softmax}\left(\frac{(W_Q \cdot h)^T \cdot (W_K \cdot h)}{\sqrt{d_K}}\right) \tag{2.54}$$

with $W_V \in \mathbb{R}^{d \times t}$, where d is the output dimension of the previous layer, $W_Q \in \mathbb{R}^{d_K \times t}$, $W_K \in \mathbb{R}^{d_K \times t}$, and a column-wise softmax activation. The dimension d_K of W_Q and W_K can be interpreted as a hyperparameter of the transformer layer. Except for the bias, $W_V \cdot h$ corresponds to a fully-connected layer and the softmax normalizes the attention weights between the input tokens. The attention allows the neural network to assign higher weights to the relevant tokens instead of assigning the same importance to all tokens. For instance, with an image as input, disjoint image areas are the input tokens for the transformer layer. In general, attention will assign a higher weight to tokens that contain one (or more) objects than tokens that contain only background.

Universal Approximation The universal approximation for FFNs was introduced in Section 2.2, and some works present universal approximation properties for CNNs. In [117], the authors show that every FFN is equivalent to a CNN and thus also the approximation properties. Note, that all presented results are based on CNNs without any pooling operation and with convolutions that are not based on zero padding. In [188], it is shown that a CNN with a sufficient depth can approximate any continuous function to any degree of accuracy on a closed and bounded subset of $\mathbb{R}^n$. Assuming that no pooling operations are present, any continuous and translation-equivariant function can be approximated by CNNs [177].

2.3 Object Detection

Object detection is the task of identifying individual objects on input data. These objects belong to different predefined classes, e.g., different road users in traffic scenes (cars, pedestrians, etc.) or everyday objects such as animals, cutlery or similar. Object detection includes the recognition of the presence (*objectness*), the *localization* and the *classification* of objects. Deep learning techniques and in particular CNNs are able to perform object detection without the need to predefine specific features. An object detector has to be trained on the underlying task with as much data as possible to perform well. To obtain a training dataset,

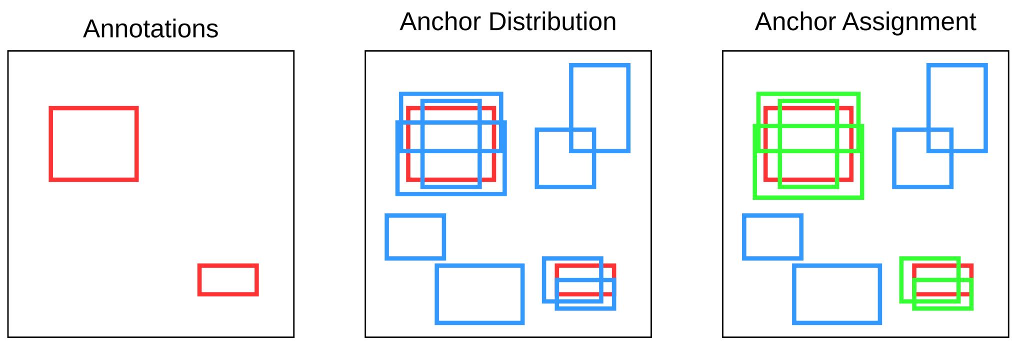


Figure 2.11: Generic training of an object detector. Left: image with annotated boxes in red, center: image with annotations and anchor distribution in blue, right: image with annotations and anchor assignment. The green boxes visualize positive anchors and the blue boxes negative anchors.

images have to be annotated beforehand, either manually or automatically. Some annotated public datasets exist for this purpose, consisting for instance of street scenes (KITTI [46], Berkeley DeepDrive [182]) or everyday situations (Pascal VOC [38], MS-COCO [97]). Nevertheless, object detection still finds applications in many other fields.

For benchmarking, the task of an object detector is to locate and classify the annotated objects correctly and, if possible, prevent false predictions, i.e., predicting objects without annotation nearby. Object detectors consist of a *backbone*, a *neck* and a *detection architecture*. The backbone is responsible for image-wise feature extraction and is adapted during training. The neck connects the backbone with the detector head by collecting feature maps from different stages of the backbone. The detection architecture trains and infers based on the extracted feature maps from the neck. The combination of backbone, neck and detector architecture determines the object detector. Note, that the detection architecture is often interpreted as an object detector, where the backbone can be freely replaced. Although the training and inference differs depending on the choice of the detection architecture, the generic training and inference procedure follow the same conceptual idea independent of the architecture.

Generic Object Detection Training

Any image x from the training dataset is equipped with a set $\mathcal{Y}$ containing G ground truth bounding boxes (also called *annotations* or *labels*), i.e.,

$$\mathcal{Y} = \{b^i,\, i = 1, ..., G\}\,, \tag{2.55}$$

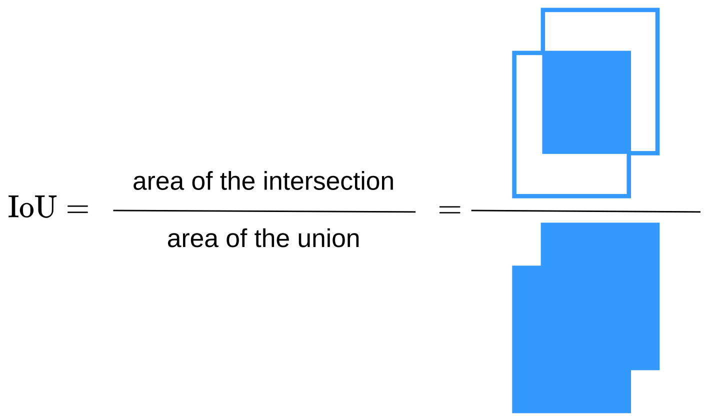


Figure 2.12: Intersection over union for two boxes.

where each ground truth box is a tuple

$$b^i = (x^i, y^i, w^i, h^i, c^i) \tag{2.56}$$

containing the box center (x^i, y^i), the box extent (w^i, h^i) and a class index c^i from the set of classes $\{1, ..., C\}$. Figure 2.11 (left) shows an image with annotated boxes, where the corresponding classes of the annotations are not visualized. Since the task of object detection consists of three subtasks (objectness, localization, classification), the overall loss $\mathcal{L}$ consists of three different loss terms:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{reg}} + \mathcal{L}_{\text{obj}}, \tag{2.57}$$

where $\mathcal{L}_{\text{cls}}$ is the classification loss, $\mathcal{L}_{\text{reg}}$ the regression loss and $\mathcal{L}_{\text{obj}}$ the objectness loss. The object detector is trained as follows: given an image x, the neck extracts image-dependent features of the backbone and the detector head distributes a fixed number N_0 of bounding boxes called *anchors*

$$\mathcal{B}_a = \{b_a^i = (x_a^i, y_a^i, w_a^i, h_a^i),\ i = 1, \dots, N_0\} \tag{2.58}$$

all over the image. Figure 2.11 (center) shows an image with annotated boxes in red and distributed anchors in blue. For the sake of clarity, the number of distributed anchors is greatly reduced compared to the real application, where the number of anchors is usually of order 10^5. The architecture is decisive in how the anchors look and how they are distributed on the image. Then, the anchors are assigned to the ground truth boxes w.r.t. the *intersection over union*.

Intersection over Union For the task of object detection, the *intersection over union* (*IoU*, also called Jaccard index [74]) describes the degree of overlap of two boxes. The $IoU \in [0, 1]$ of two boxes b_1, g_1 is defined as the number of pixels that are located in both boxes divided by the number of pixels that are located in at least one box:

$$IoU(b_1, g_1) = \frac{|b_1 \cap g_1|}{|b_1 \cup g_1|} \tag{2.59}$$

The IoU is equal to zero if the two boxes do not intersect with each other and is equal to one if the localization of the two boxes is identical. The definition of the IoU is visualized in fig. 2.12.

The prediction and ground truth assignment in training is exemplary shown in fig. 2.11 (right). The green boxes are positive anchors, i.e., anchors that have an IoU with at least one ground truth box greater or equal than a threshold (mostly between 0.5 and 0.7), and the blue boxes are negative anchors, i.e., anchors that have IoU values smaller than the threshold with all ground truth boxes. In practice, there are still anchors that are completely ignored during training, e.g., if the threshold for negative anchors is 0.3 and that for positive ones is at 0.7, then all anchors that have their highest IoU with a ground truth box between 0.3 and 0.7 are completely suppressed. Therefore, we focus only on negative and positive anchors in the following. After the assignment, all anchors are given a score $s_a^i \in \{0, 1\}$ and class probabilities $(p_1^i, \ldots, p_C^i)$ to learn:

$$\mathcal{B}_a = \{b_a^i = (x_a^i, y_a^i, w_a^i, h_a^i, s_a^i, p_1^i, \ldots, p_C^i),\ i = 1, \ldots, N_0\} \tag{2.60}$$

with

$$\begin{cases} s_a^i = 1, & \text{if } b_a^i \text{ is a positive anchor,} \\ s_a^i = 0, & \text{if } b_a^i \text{ is a negative anchor,} \end{cases} \tag{2.61}$$

and

$$\begin{cases} (p_1^i \ldots, p_C^i) = \vec{e}_c, & \text{if } b_a^i \text{ is a positive anchor and c is the class of the} \\ & \text{assigned ground truth box } b \in \mathcal{Y} \\ (p_1^i \ldots, p_C^i) = \vec{0}, & \text{if } b_a^i \text{ is a negative anchor.} \end{cases} , \tag{2.62}$$

Since the object detector is initialized with weights, e.g., randomly or pre-trained on another dataset, it is able to infer even before the first training iteration. The predictions are denoted as:

$$\mathcal{B} = \{\hat{b}^i = (\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i, \hat{s}^i, \hat{p}_1^i, \ldots, \hat{p}_C^i),\ i = 1, \ldots, N_0\}, \tag{2.63}$$

where $(\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i)$ represent the predicted localization, $\hat{s}^i$ the predicted objectness and $(\hat{p}_1^i, \ldots, \hat{p}_C^i)$ the predicted class probabilities.

Here, the objectness of the prediction $\hat{b}^i_{\text{obj}} = \{\hat{s}^i,\, i = 1, \dots, N_0\}$ is in $[0, 1]$ and is pushed to one during training if the corresponding anchor b^i_a is positive ($s^i_a = 1$) and to zero otherwise ($s^i_a = 0$). The binary cross-entropy (BCE) is an exemplary objectness loss $\mathcal{L}_{\text{obj}}$ which is defined as:

$$\mathcal{L}_{\text{obj}} = \sum_{i=1}^{N_0} s^i_a \cdot \log(\hat{s}^i) + (1 - s^i_a) \cdot \log(\hat{s}^i). \tag{2.64}$$

Further objectness loss functions are for instance the mean squared error (MSE) or the focal loss. The latter is a BCE with dynamic weighting, where especially the errors of the current training iteration are strongly weighted.

The localization of the prediction $\hat{b}^i_{\text{reg}} = \{(\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i),\, i = 1, \dots, N_0\}$ is inferred as a normalized scaling of the corresponding i-th anchor $b^i_a \in \mathcal{B}_a$:

$$\hat{t}^i_x = \frac{\hat{x}^i - x^i_a}{w^i_a},\;\; \hat{t}^i_y = \frac{\hat{y}^i - y^i_a}{h^i_a},\;\; \hat{t}^i_w = \log\left(\frac{\hat{w}^i}{w^i_a}\right),\;\; \hat{t}^i_h = \log\left(\frac{\hat{h}^i}{h^i_a}\right). \tag{2.65}$$

If the anchor is positive, exchanging the prediction $\hat{b}^i$ with the assigned ground truth box $b \in \mathcal{Y}$ in (2.65) yields the true offsets t^i_x, t^i_y, t^i_w and t^i_h. Otherwise, we define $t^i_x = t^i_y = t^i_w = t^i_h = 0$. The *smooth-*$L_1$ loss ([48]) is an example for a regression loss $\mathcal{L}_{\text{reg}}$ and is defined for one coordinate $t_k \in \{t_x, t_y, t_w, t_h\}$ as:

$$\mathcal{L}_{\text{reg},t_k} = \sum_{i=1}^{N_0} s^i_a \cdot \begin{cases} 0.5(\hat{t}^i_k - t^i_k)^2, & \text{if } |\hat{t}^i_k - t^i_k| < 1, \\ |\hat{t}^i_k - t^i_k| - 0.5, & \text{else,} \end{cases} \tag{2.66}$$

where s^i_a provides a masking such that localization is learned only from the positive anchors. The overall regression loss $\mathcal{L}_{\text{reg}}$ is the sum of the regression losses for one coordinate from (2.66):

$$\mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{reg},t_x} + \mathcal{L}_{\text{reg},t_y} + \mathcal{L}_{\text{reg},t_w} + \mathcal{L}_{\text{reg},t_h}. \tag{2.67}$$

Further regression losses are for instance the MSE or the L_1 loss.

The classification of the prediction $\hat{b}^i_{\text{cls}} = \{(\hat{p}^i_1, \dots, \hat{p}^i_C),\, i = 1, \dots, N_0\}$ consists of predicted class probabilities with

$$\sum_{c=1}^{C} \hat{p}^i_c = 1 \quad \forall i = 1, \dots, N_0. \tag{2.68}$$

The cross-entropy (CE, see (2.27)) is a commonly used classification loss $\mathcal{L}_{\text{cls}}$ and

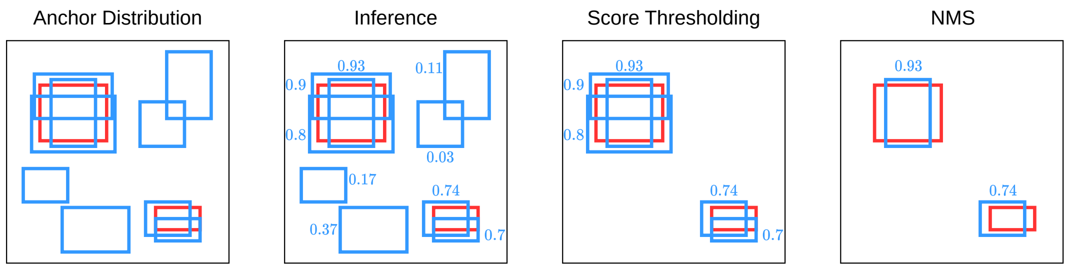


Figure 2.13: Generic inference of an object detector. Left: image with annotations in red and anchor distribution in blue, center left: image with annotations and predictions with score, center right: image with annotations and predictions after score thresholding with $s_\epsilon = 0.5$, right: image with annotations and predictions after non-maximum suppression (NMS).

is adapted to:

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{N_0} s^i \cdot \sum_{c=1}^{C} p_c^i \cdot \log(\hat{p}_c^i). \tag{2.69}$$

Analogous to the localization, classification is learned from positive anchors only.

Generic Object Detection Inference

After training, the underlying object detector is able to infer a test dataset where no ground truth has to be available. In most academic datasets, annotations exist for the test images, or at least for a validation set, to ensure an automated evaluation of developed methods. For generating image-wise predictions, the inference follows a conceptually similar idea to the training. Anchors are distributed over the entire image (2.58) and the object detector generates predictions as in (2.63). Figure 2.13 (left) shows an image with annotations in red and distributed anchors in blue. The predictions are inferred as offsets from the anchor boxes (which is not visualized due to clarity) and every prediction has an assigned objectness score, see fig. 2.13 (center left). There are many predicted boxes that do not significantly overlap any ground truth box and therefore represent background. These predictions should have a low objectness *score*, since background boxes are always marked as negative anchors during training. In order to suppress these background predictions, a score threshold $s_\epsilon \in [0, 1]$ is chosen, such that mostly foreground boxes remain as predictions:

$$\mathcal{B}_s = \{\hat{b}^i \in \mathcal{B} : \hat{s}^i \geq s_\epsilon\}, \tag{2.70}$$

Algorithm 2: Non-maximum Suppression (NMS)

```
   Input  : predictions $\hat{b} \in \mathcal{B}_s$ after score thresholding with $\hat{s}$ as
            corresponding score vector, $IoU$ threshold $\tau_{IoU}$
 1 set $\mathcal{B}_{\text{NMS}} = \emptyset$
 2 while $\mathcal{B}_s \neq \emptyset$ do
 3     $m = \text{argmax } \hat{s}$
 4     $\mathcal{M} = \hat{b}^{(m)}$
 5     $\mathcal{B}_{\text{NMS}} = \mathcal{B}_{\text{NMS}} \cup \{\hat{b}^{(m)}\}$
 6     for $\hat{b}^i \in \mathcal{B}_s$ do
 7         if $IoU(\mathcal{M}, \hat{b}^i) \geq \tau_{IoU}$ then
 8             $\mathcal{B}_s = \mathcal{B}_s \setminus \{\hat{b}^i\}$
 9         end
10     end
11 end
   Output: predictions $\hat{b} \in \mathcal{B}_{\text{NMS}}$ after NMS
```

see fig. 2.13 (center right). If the score threshold s_ϵ is too large (here 0.8), predictions representing an object are suppressed. If the score threshold s_ϵ is too small (here 0.3), predictions representing background are not suppressed. The choice of this hyperparameter plays a crucial role and should be chosen depending on the underlying task. If false predictions are not critical and all objects should be found, a small score threshold is recommended. If false predictions should be prevented and overlooked objects are comparatively acceptable, a larger score threshold should be selected. After score thresholding, there are still several similarly located boxes representing the same object, all of which have a high score. However, since only a single prediction is supposed to represent exactly one object, score thresholding is typically followed by the *non-maximum suppression* (NMS), see fig. 2.13. Let $\mathcal{B}_{\text{NMS}} = \emptyset$. Iteratively, the NMS selects the prediction $\hat{b}^i \in \mathcal{B}_s$ on the image with the highest score ($\mathcal{B}_{\text{NMS}} = \mathcal{B}_{\text{NMS}} \cup \{\hat{b}^i\}$) and removes it from $\mathcal{B}_s$. Then, all remaining predictions that have a significantly high IoU with the selected prediction (typically between 0.5 and 0.7) are removed from $\mathcal{B}_s$. This is repeated until $\mathcal{B}_s = \emptyset$ and only the set of predictions after score thresholding and NMS $\mathcal{B}_{\text{NMS}}$ remains. Note, that the calculated IoUs are class-dependent, i.e., the IoU of two predictions with different assigned classes is always zero. For a detailed algorithm of the NMS, see alg. 2 and for a visualization, see fig. 2.13 (right).

Architectures

Object detectors generally fall into two categories: *one-stage* and *two-stage* object detectors, where the detection architecture consists of one or two stages, see

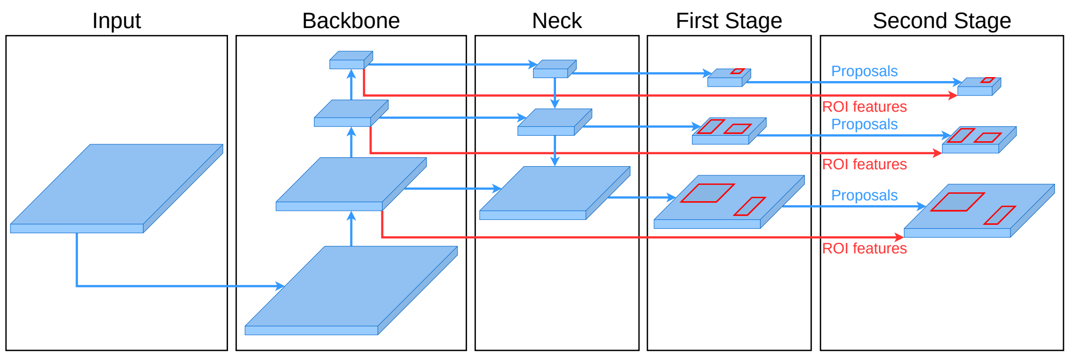


Figure 2.14: Object detector architecture with an input image, backbone, feature pyramid network as neck and an object detector head. One-stage object detectors end after the first stage and two-stage object detectors after the second stage.

fig. 2.14. In general, the backbone extracts image-wise features and the neck connects the backbone with the detection architecture. In the following, we will discuss the YoloV3 [39] architecture representative of one-stage object detectors and the Faster R-CNN [123] representative of two-stage detectors, both with a feature pyramid network [95] as a neck. YoloV3 uses the Darknet-53 [121] as a backbone and the Faster R-CNN uses ResNets [63] with various depths. Further backbones are for instance ResNeSt [187], as well as Swin transformer [100], and further architectures are RetinaNet [96] (one-stage) and Cascade R-CNN [14] (two-stage).

Each architecture has different approaches to learning and generating predictions. In the following, we introduce the training and inference of YoloV3 and Faster R-CNN in detail to better understand the different approaches. The resulting hyperparameters will affect the performance of the object detector, e.g., which labeled bounding boxes are difficult or impossible to detect due to their size or aspect ratio.

YoloV3 architecture In order to generate predictions, whether in training or inference, a number of anchors are distributed to all feature maps of the neck from fig. 2.14. The larger a feature map is, that still smaller objects should be detected on it and vice versa. The respective feature map is divided into grid cells and predefined anchors are placed in these cells. For instance, three anchors with aspect ratios $2:1$, $1:1$ and $1:2$ are placed in each cell to detect objects of any shape. See fig. 2.15 for a visualization of the grid cells and the cell-wise distributed anchors. Note, that a smaller feature map is divided into fewer cells compared to feature maps of higher resolution. The sizes and ratios of the anchors can also be determined in a pre-processing step with a cluster algorithm, e.g., k-means [58], using the localizations of the annotations used for training.

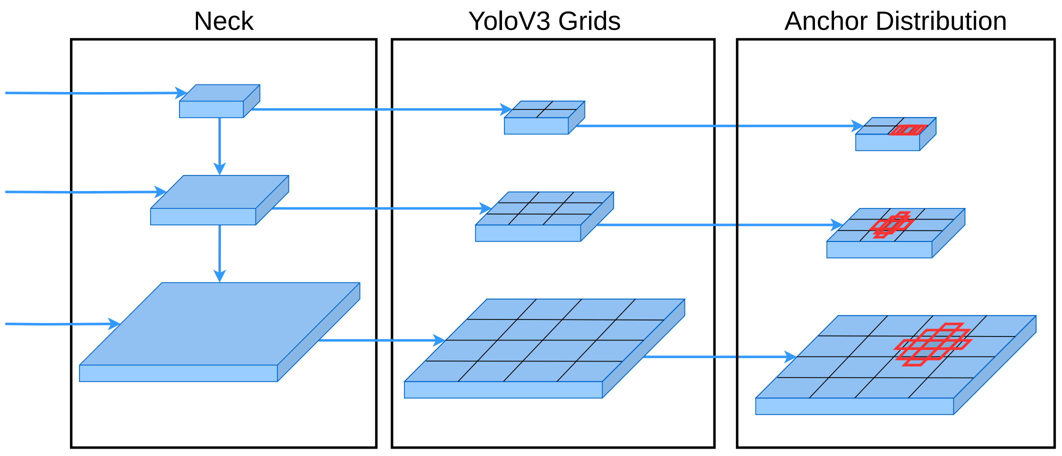


Figure 2.15: Cell splitting of feature maps and anchor distribution for the YoloV3 architecture.

After inferring the anchors, and independently of the underlying feature maps, all resulting predictions form the set $\mathcal{B}$, which are either assigned to the ground truth boxes in the training process or form the basis for score thresholding and the NMS for the inference. The architecture is called Yolo, or "you only look once", since the predictions consist of the localization, an objectness score and class probabilities. Thus, all three tasks are trained or predicted simultaneously. Let $\mathcal{B}_a$ be the set of anchors analogous to (2.58) and strictly following [39], only one anchor is assigned to a ground truth box, i.e., only the predictions that have the highest IoU with a ground truth box are positive. All other anchors that have an IoU above the IoU threshold will be ignored during training. However, in current implementations, learning still occurs from all positively assigned anchors. The localization loss $\mathcal{L}_{\text{reg}}$ is a sum of squared errors:

$$\begin{aligned}\mathcal{L}_{\text{reg}} = \sum_{i=1}^{N_0} s_a^i[(t_x^i - \hat{t}_x^i)^2 + (t_y^i - \hat{t}_y^i)^2 + \\ (t_w^i - \hat{t}_w^i)^2 + (t_h^i - \hat{t}_h^i)^2].\end{aligned} \tag{2.71}$$

For classification, YoloV3 applies a sigmoid activation, as the set of classes can contain overlapping classes, e.g., dog and animal. The resulting classification loss $\mathcal{L}_{\text{cls}}$ is the sum of the binary cross-entropy loss (see (2.64)) for every class:

$$\mathcal{L}_{\text{cls}} = \sum_{i=1}^{N_0} \sum_{c=1}^{C} p_c^i \cdot \log(\hat{p}_c^i) + (1 - p_c^i) \cdot \log(\hat{p}_c^i), \tag{2.72}$$

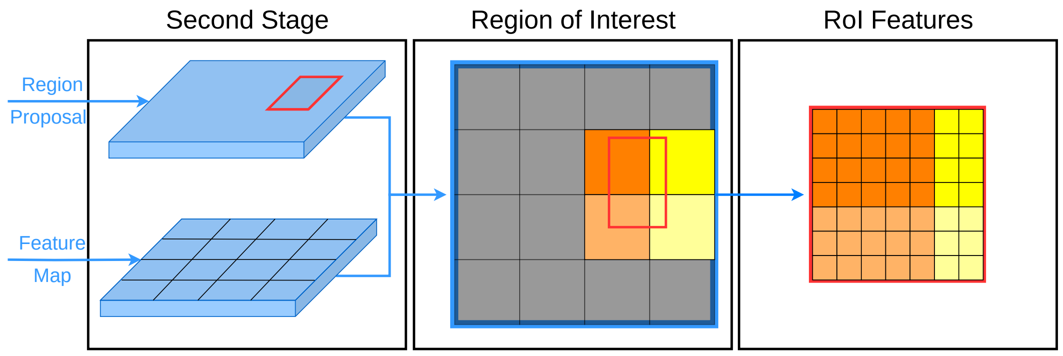


Figure 2.16: The second stage of the Faster R-CNN. Left: the proposals from the first stage and the corresponding feature map from the neck are the input for the second stage. Center: the localization-dependent feature extraction is based on the localization of the proposals. Right: the features of the overlapping cells are unpooled to a fixed resolution (here 7×7).

as well as for the objectness loss $\mathcal{L}_{\text{obj}}$:

$$\mathcal{L}_{\text{obj}} = \sum_{i=1}^{N_0} s_a^i \cdot \hat{s}^i + (1 - s_a^i) \cdot \hat{s}^i. \tag{2.73}$$

Faster R-CNN architecture Faster R-CNN consists of two stages, the region proposal network (first stage) and the region of interest (second stage). The first stage is similar to the concept of YoloV3, except that only localization and objectness are learned. Moreover, not only anchors with different aspect ratios are distributed over the cells, but also anchors of different sizes, e.g., three aspect ratios and three scales result in nine anchors per cell. The first stage regression loss $\mathcal{L}_{\text{reg},1}$ is the smooth-L_1 loss from (2.66) and the objectness loss $\mathcal{L}_{\text{obj}}$ is the binary cross-entropy from (2.64). In the second stage, Faster R-CNN learns a bounding box refinement and a classification based on the proposals from the first stage after score thresholding and NMS ($\hat{b}^i \in \mathcal{B}_{\text{NMS}}$, see (2.70) and alg. 2). For every single proposal from the first stage $\hat{b}^i \in \mathcal{B}_{\text{NMS}}$, a region of interest (RoI) pooling layer extracts a fixed-sized crop from the feature map, see fig. 2.16. After squeezing the cropped feature map to a vector, a fully-connected network ends in two sibling output layers. These are responsible for the bounding box refinement and the classification. Proposals from the first stage can have a high objectness score and will be guided to the second stage, although they have no assigned ground truth box. For bounding box refinement, the second stage regression loss $\mathcal{L}_{\text{reg},2}$ is the smooth-L_1 loss from the offset of the proposal from the first stage and the assigned ground truth box. If the proposal from the first stage has no assigned ground truth box, the second stage regression loss $\mathcal{L}_{\text{reg},2}$ is equal to zero. Since

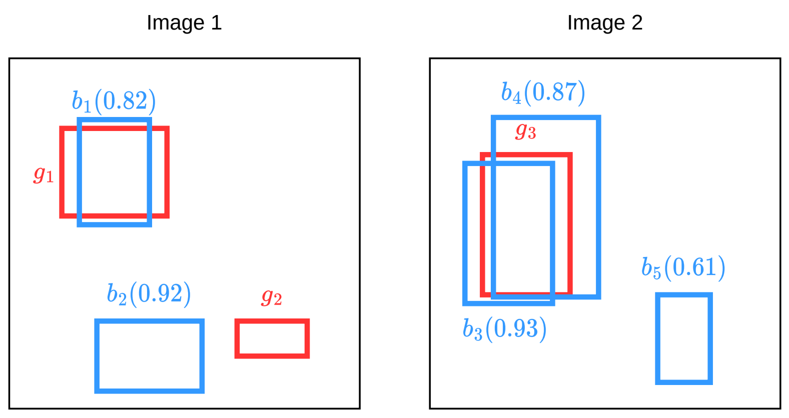


Figure 2.17: The dataset for prediction and ground truth assignment consists of two images and in total of three ground truth boxes (red) and five predictions (blue) with associated scores in brackets.

the highest class probability is the final score of the prediction, "background" is added as class $C+1$ to the set of classes in order to push the negative anchors towards a high background probability. Let $\hat{b}^i \in \mathcal{B}_{\text{NMS}}$, $i = 1, \dots, N_1$, be the region proposals from the first stage. Then, the classification loss $\mathcal{L}_{\text{cls}}$ is defined as the cross-entropy loss with $C+1$ classes:

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{N_1} \sum_{c=1}^{C+1} p_c^i \cdot \log(\hat{p}_c^i), \tag{2.74}$$

where p_{C+1}^i is equal to one if the anchor is negative and equal to zero otherwise.

Evaluation Metrics

The most commonly used evaluation metric for the performance of an object detector on a dataset is the mean average precision. Before the mean average precision is defined, further metrics are introduced.

Prediction and Ground Truth Assignment The assignment of prediction and ground truth is necessary to quantify performance. More precisely, with the assignment of prediction to ground truth, predictions are classified into correct (true positive) and incorrect (false positive), as well as a distinction is made between detected and non-detected (false negative) ground truth boxes. A schematic illustration of the assignment of predictions and ground truth is shown in fig. 2.17. The red boxes (g_1, g_2, g_3) are ground truth objects and the blue boxes $(b_1, \dots, b_5)$ are

Image Id	Prediction	Score	max IoU	Assigned GT	TP/FP	Precision	Recall
2	b_3	0.93	0.7	g_3	TP	1.00	0.33
1	b_2	0.92	0.0	–	FP	0.50	0.33
2	b_4	0.87	0.5/0.0	g_3/–	FP	0.33	0.33
1	b_1	0.82	0.65	g_1	TP	0.50	0.67
2	b_5	0.61	0.0	–	FP	0.40	0.67

Table 2.1: The predictions are sorted in image-independent descending order by the score, assigned to the ground truth boxes and classified as true positive (TP) or false positive (FP). The precision and recall are accumulated from top to bottom.

the predictions with corresponding scores in brackets. In order to assign prediction to ground truth, the predictions are sorted in image-independent descending order by the score, see table 2.1 (columns Image ID - Score). Then, in descending order, the IoU between the current prediction and all ground truth boxes of the associated image is determined. If the maximum IoU between prediction and ground truth is greater than or equal to a threshold δ (in the following $\delta = 0.5$), then the prediction is assigned to the associated ground truth box and the latter is suppressed for subsequent IoU calculations. If the prediction is assigned to a ground truth box, it is classified as a *true positive* (TP), otherwise as a *false positive* (FP). In table 2.1 (columns Image ID - TP/FP), b_3 and b_1 are TPs, and b_2, b_4, as well as b_5 are FPs. If b_3 would not exist or had a smaller score than b_4, then b_4 would have been a TP since b_4 has an $IoU = 0.5$ with g_3.

Precision and Recall *Precision* and *recall* are two performance metrics which are applied, e.g., in object detection. The precision is also called *positive predictive value* and indicates the ratio between TPs and all predictions. The recall is also called *sensitivity* and specifies the ratio between assigned ground truth boxes and all ground truth boxes. In the following, we assume a set of predictions $\mathcal{B} = \{\hat{b}^i,\, i = 1, \dots, N\}$ sorted in descending order by score and a set of ground truth boxes $\mathcal{Y} = \{b^i,\, i = 1, \dots, G\}$. In order to determine precision and recall, the following three quantities are required:

- *True positives* (TPs): set of predictions, each of which is assigned to a different ground truth box, i.e.

$$\begin{aligned} \text{TPs} = \{\hat{b}^i \in \mathcal{B} \mid \\ &\exists \text{ non-assigned } b^j \in \mathcal{Y} \text{ with } IoU(\hat{b}^i, b^j) \geq \delta, \\ &\nexists \text{ non-assigned } b^k \in \mathcal{Y} \text{ with } IoU(\hat{b}^k, b^j) \geq IoU(\hat{b}^i, b^j), \\ &i = 1, \dots, N\}, \end{aligned} \tag{2.75}$$

where non-assigned refers those ground truth boxes that were not assigned to a prediction with a higher score, i.e., to $b^1, \dots, b^{i-1}$.

- *False positives* (FPs): set of predictions that are not assigned to a ground truth box, i.e.

$$\begin{aligned} \text{FPs} = \{&\hat{b}^i \in \mathcal{B} \mid \nexists \text{ non-assigned } b^j \in \mathcal{Y} \text{ with } IoU(\hat{b}^i, b^j) \geq \delta, \\ &i = 1, \dots, N\}. \end{aligned} \tag{2.76}$$

- *False negatives* (FNs): set of ground truth boxes that were not assigned to any prediction, i.e.

$$\begin{aligned} \text{FNs} = \{&b^j \in \mathcal{Y} \mid \nexists \text{ non-assigned } \hat{b}^i \text{ with } IoU(\hat{b}^i, b^j) \geq \delta, \\ &j = 1, \dots, G\}. \end{aligned} \tag{2.77}$$

FNs are overlooked objects, such as g_2 in fig. 2.17. The precision is defined as:

$$\text{Precision}_{s_\epsilon} = \frac{|\text{TPs}|}{|\text{TPs}| + |\text{FPs}|} \tag{2.78}$$

and the recall is defined as:

$$\text{Recall}_{s_\epsilon} = \frac{|\text{TPs}|}{|\text{TPs}| + |\text{FNs}|}. \tag{2.79}$$

Note, that precision and recall are both dependent of the score threshold s_ϵ, and are accumulated for all predictions with a score greater than or equal s_ϵ, see table 2.1.

Mean Average Precision The *precision recall curve* is shown in fig. 2.18. The x-markers represent the threshold-dependent precision and recall values from table 2.1 and the blue graph visualizes the connection between them. The *average precision* (AP) is defined as a modified area under the precision recall curve. More precisely, with $prec(rec)$ as the maximal precision for recall rec, rec_i as the recall from the prediction with the i-th highest score ($rec_0 := 0$, $rec_{n+1} := 1$) and a number of n underlying predictions, the AP is defined as:

$$AP = \sum_{i=0}^{n} (rec_{i+1} - rec_i) \max_{rec \geq rec_{i+1}} prec(rec), \tag{2.80}$$

where $\max_{rec \geq rec_{n+1}} prec(rec) := 0$. In fig. 2.18, the gray area visualizes the area under the modified precision recall curve with $AP = (0.33 - 0) \cdot 1 + (0.67 - 0.33) \cdot 0.5 = 0.5$. The AP is calculated for each class individually, considering only the

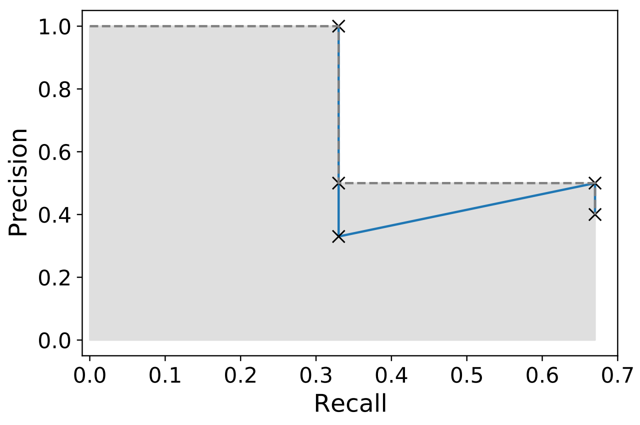


Figure 2.18: The precision recall curve (blue) for table 2.1 with corresponding interpolation and area under curve (gray).

class-associated predictions and ground truth boxes, i.e., AP_i is the AP for class i, $i = 1, \dots, C$. The *mean average precision* (mAP) is defined as the class-wise mean of the APs:

$$mAP = \frac{1}{C} \sum_{i=1}^{C} AP_i. \tag{2.81}$$

The higher the $mAP \in [0, 1]$, the better the object detector performs on the given dataset. Sometimes the mAP is written as $mAP_{0.5}$ in order to show the underlying IoU threshold. There exist different modifications of the mAP, i.e., the 11-point evaluation [38] where the sum from (2.80) is evaluated only at eleven predefined points $[0, 0.1, \dots, 1]$ or the $mAP_{0.5:0.05:0.95}$ [97] where the mAP is averaged over ten different IoU thresholds $[0.5, 0.55, \dots, 0.95]$. In the following, the mAP definition from (2.81) is used in the following chapters.

2.4 Uncertainty Quantification

CNNs are usually applied for the task of object detection due to their strong performance. However, these statistical models are error-prone, and it is important to detect and understand the errors produced. Therefore, uncertainty quantification of object detectors is of highest interest, especially in safety-relevant tasks, such as perception in autonomous driving or medical diagnosis. In general, two different types of uncertainty are distinguished: *aleatoric* and *epistemic* uncer-

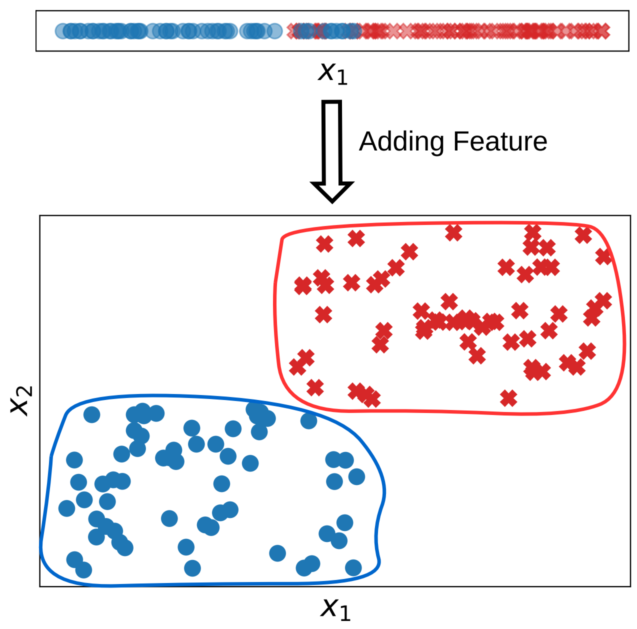


Figure 2.19: Aleatoric uncertainty for a binary classification. Top: two classes (blue and red) are overlapping which causes aleatoric uncertainty. Bottom: by adding a new feature dimension, the two classes become separable and the aleatoric uncertainty vanishes.

tainty, even if these are usually not clearly separable in applications. In practice, object detectors are often not well-calibrated and therefore not reliable. More precisely, object detectors are often over-confident, i.e., weak or even false predictions still receive high scores. Therefore, uncertainty quantification is of highest interest for the task of object detection. This subsection is mainly based on [72].

Aleatoric Uncertainty Aleatoric, or also called *statistical*, uncertainty can be described as the irreducible part of the uncertainty. If the data generating process is fixed, aleatoric uncertainty cannot be reduced through additional training data. For instance, predicting a dice result with a six-sided dice will not be more certain by adding more information about previous dice rolls. Due to the given statistical component of the task, there is an irreducible, i.e., aleatoric uncertainty. Otherwise, if the data generating process is not fixed, then aleatoric uncertainty can be reduced, see fig. 2.19. By adding a second feature dimension, the aleatoric uncertainty vanishes. However, adding feature dimensions is not feasible in most cases, and even if it were, all previously annotated data would have to be extended with the new feature, which is usually associated with a very large effort. In object detection, aleatoric uncertainty can be quantified by the prediction-wise objectness score, as well as dispersion measures applied to the predicted class probabilities,

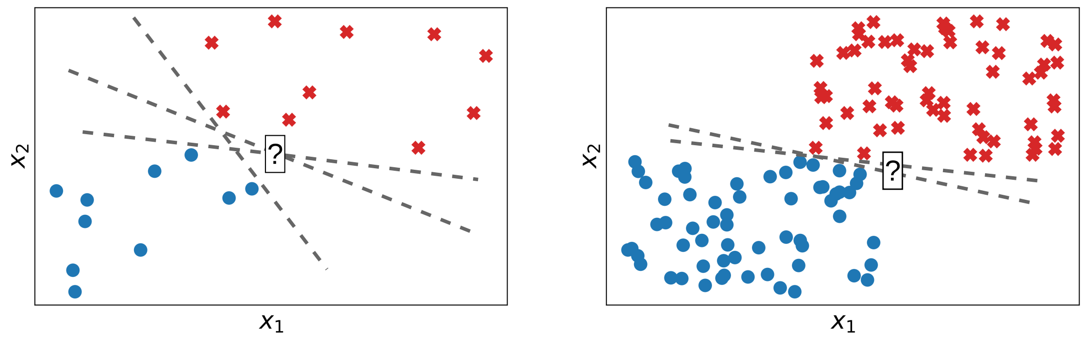


Figure 2.20: Epistemic uncertainty for a binary classification with perfect separating class boundaries in gray. Left: the model can learn very different class boundaries. As the underlying data distribution is unknown, every gray class boundary is equally good. Right: the variety of perfect separating class boundaries is reduced due to additional training data.

e.g., the classification entropy or probability margin [72].

Epistemic Uncertainty Epistemic, or also called *systematic*, uncertainty represents the reducible part of the uncertainty. Epistemic uncertainty can be reduced by adding more data to the underlying training set, see fig. 2.20. With less training data (left), the model can learn various class separations while the training error remains zero. On the other hand (right), the class boundary sharpens and the epistemic uncertainty decreases.

In the following, we will introduce the uncertainty quantification methods that are important for the present thesis. All these methods were developed for the task of image classification and then adapted to object detection.

Entropy The classification entropy [148] has its origin in information theory and indicates the mean information of a distribution. For the task of image classification, the entropy indicates the uncertainty of a prediction $\hat{p}^i = \hat{p}^i_1, \dots, \hat{p}^i_C$ and is defined as:

$$\mathrm{H}(\hat{p}^i) = -\sum_{c=1}^{C} \hat{p}^i_c \cdot \log(\hat{p}^i_c). \tag{2.82}$$

The entropy is maximal if $\hat{p}^i_c = \frac{1}{C} \forall c = 1, \dots, C$ and minimal if $\hat{p}^i = \vec{e}_c$, $c \in \{1, \dots, C\}$, see fig. 2.21 (left). In this three-class classification problem, the entropy is maximal for $\hat{p}^i_1 = \hat{p}^i_2 = \hat{p}^i_3 = \frac{1}{3}$ and minimal in each of the three corners. The entropy is always greater or equal to zero, but the maximum entropy increases with the number of classes. Dividing by the maximum entropy ($\log(\frac{1}{C})$)

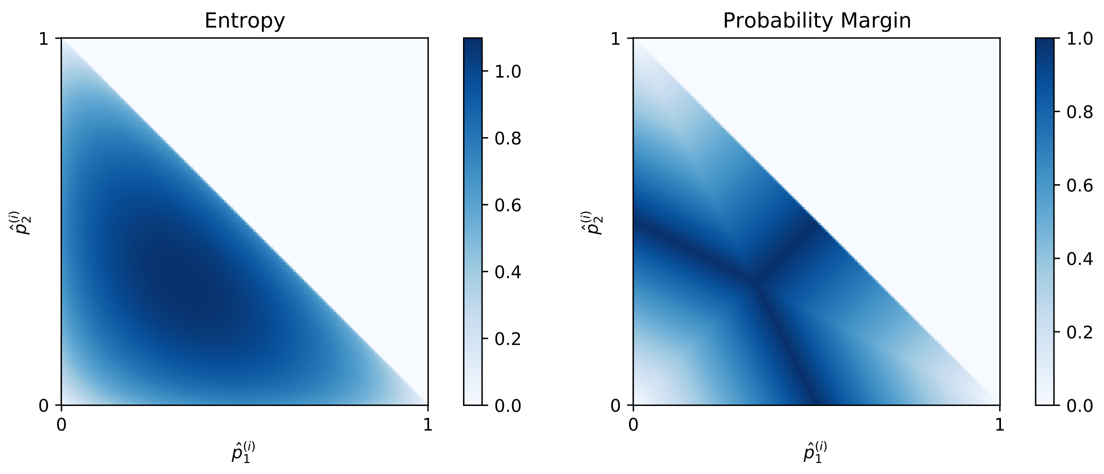


Figure 2.21: Entropy (left) and probability margin (right) of a class probability for three classes, with $p_3^i = 1 - (p_1^i + p_2^i)$. Dark blue represents high uncertainty and white represents low or no uncertainty.

results in a normalized version of the entropy ($\in [0, 1]$). Oftentimes, entropy does not represent image-wise uncertainty, but box-wise uncertainty in object detection. Each prediction also contains class probabilities, such that the entropy of each prediction can be computed without adjustment. In general, if the training dataset contains sufficient data, the entropy quantifies aleatoric uncertainty [72].

Probability Margin Analogous to entropy, the probability margin is a measure of image-wise predictive uncertainty in image classification. For a prediction $\hat{p}^i = \hat{p}_1^i, \dots, \hat{p}_C^i$, the probability margin is defined as:

$$\mathrm{PM}(\hat{p}^i) = 1 - (\hat{p}^i_{c_{\max}} - \max_{c \neq c_{\max}} \hat{p}^i_c), \tag{2.83}$$

with $c_{\max} = \operatorname*{argmax}_{c=1,\dots,C} \hat{p}^i_c$. By definition, $\mathrm{PM}(\hat{p}^i) \in [0, 1]$. The probability margin is minimal if $\hat{p}^i = \vec{e}_c$, $c \in \{1, \dots, C\}$ and maximal if $\hat{p}^i_{c_{\max}} = \max_{c \neq c_{\max}} \hat{p}^i_c$. The maximum is not unique, since only the highest and second highest class probabilities have to be identical, see fig. 2.21 (right). The adaptation for object detection is straightforward and is done analogously to entropy as a box-wise uncertainty.

Ensembles Another method for uncertainty quantification is the ensemble approach [88]. Here, several outputs are generated for one input by different models. The joint prediction is derived from averaging the outputs of several class probabilities and the standard deviations yield uncertainty predictions. Let $\hat{p}^{i,j}$ be the predicted class probabilities of the j-th output ($j = 1, \dots, J$). Then, the joint prediction $\overline{\hat{p}}^i$ and the corresponding class-wise uncertainty $\sigma_{\overline{\hat{p}}^i_c}, c = 1, \dots, C$ are given by:

$$\overline{\hat{p}}^i = \frac{1}{J} \sum_{j=1}^{J} \hat{p}^{i,j} \quad \text{and} \quad \sigma_{\overline{\hat{p}}^i_c} = \sqrt{\frac{\sum_{j=1}^{J} |\overline{\hat{p}}^i_c - \hat{p}^{i,j}_c|}{J}}. \tag{2.84}$$

After a model is trained and the weights are fixed, the model is deterministic, i.e., the model generates the identical output for the same input. Therefore, the uncertainties from (2.84) are always zero. However, training a model is not deterministic, due to, e.g., the random initialization of SGD, the randomly chosen mini batch, etc. Thus, if J models are trained independently based on a given dataset, the same input will receive J different predictions during inference. The higher the standard deviations, the less the different initialized models agree for the input, resulting in an uncertain prediction. Unlike entropy and the probability margin, which determine only classification uncertainties in object detection, the ensemble approach also considers localization uncertainties. The ensemble uncertainty D_{E} of the entire prediction can be aggregated over the individual uncertainties, e.g., the maximum:

$$D_{\mathrm{E}} = \max_{\varsigma \in \{\overline{\hat{x}}^i, \overline{\hat{y}}^i, \overline{\hat{w}}^i, \overline{\hat{h}}^i, \overline{\hat{s}}^i, \overline{\hat{p}}^i_1, \dots, \overline{\hat{p}}^i_C\}} \sigma(\varsigma). \tag{2.85}$$

Note, that this uncertainty is stemming from the weights and therefore epistemic.

Monte-Carlo Dropout Ensemble approaches are computationally very intensive, since for J different outputs, J different models have to be trained. Thus, ensembles are often replaced by Monte-Carlo dropout [44] to save computational effort. Instead of training several models, only one model is trained, where in at least one layer Monte-Carlo dropout is applied. Therefore, a predefined percentage of neurons are randomly turned off, and applying Monte-Carlo dropout during inference leads to J different predictions. Thus, the uncertainties from (2.84) are non-zero, and the prediction-wise Monte-Carlo uncertainty D_{MC} can be determined analogously to (2.85). Furthermore, Monte-Carlo dropout uncertainty is an approximation of the model uncertainty of Bayesian networks and therefore quantifies epistemic uncertainty, which is also theoretically underpinned [44].

CHAPTER 3 APPLICATIONS OF UNCERTAINTY QUANTIFICATION METHODS IN OBJECT DETECTION

Uncertainty quantification (UQ) plays an important role for the task of object detection. Due to the implementation of most object detectors, class probabilities and objectness scores cannot become exactly zero, since the respective loss function is not defined at this point, see (2.64), and (2.69). In practice, a very small value ($\sim 10^{-16}$) is added to each class probability and the objectness score, thus for each prediction the uncertainty can never be exactly zero. For the detection of false predictions and erroneous data, which is used for training and evaluating, we introduce uncertainty-based methods that quantify uncertainty, as well as apply uncertainty methods for the tasks of active learning and label error detection.

In the following, we summarize the basic idea of every subsequent chapter and reference the passages from Chapter 2 for a better understanding of the prerequisites and methodology. Furthermore, we show the most important results and provide an overview of the following chapters and explain their interplay.

3.1 MetaDetect

MetaDetect (Chapter 4) yields prediction-wise uncertainty estimations. Therefore, we utilize models based on box-wise uncertainty metrics with the aim to predict either TP vs. FP ($IoU \geq 0.5$ vs. $IoU < 0.5$) for meta classification or to predict the IoU directly for meta regression. These so-called meta models are for instance logistic/linear regression, random forest, gradient boosting and

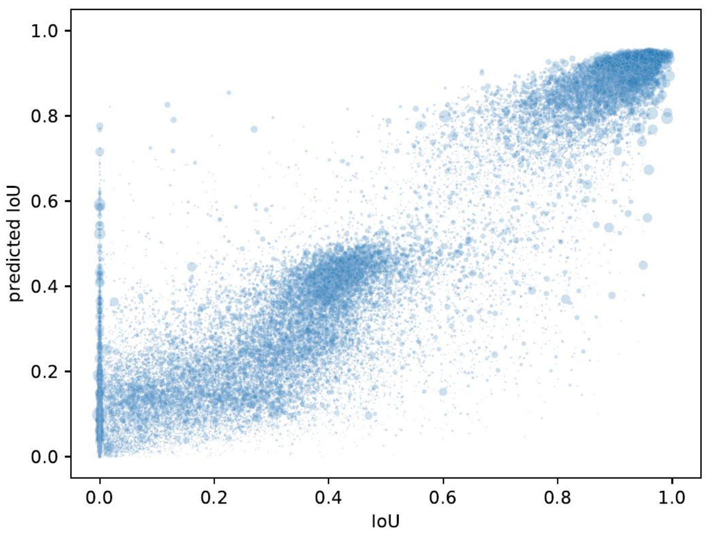


Figure 3.1: Box-wise scatter plot of true IoU and predicted IoU values for the KITTI dataset, the YoloV3 network and a score threshold $t = 0.01$. The predicted IoU values are generated with gradient boosting.

shallow neural networks. Here, the box-wise uncertainty metrics are based only on the prediction itself ($\mathcal{B}_{\text{NMS}}$; alg.2) and the associated candidate boxes that are suppressed during NMS ($\mathcal{B}_s$; (2.70)). These uncertainty metrics are for instance:

- the prediction itself, i.e., the localization, the score and the predicted class probabilities,
- dimensions of the prediction, e.g., the circumference and the area,
- the number of suppressed candidate boxes during NMS,
- as well as minimum, maximum, mean and standard deviation of the localization, scores and class probabilities of all candidate boxes.

Once the meta model has been trained on a labeled validation set, a predicted IoU (2.59), i.e., a predicted quality is obtained for each prediction, especially for unlabeled images. In our experiments, we show that MetaDetect provides more reliable uncertainty and quality estimates compared to the box-wise entropy (2.82) and the prediction-wise objectness score of the object detector. The latter is obvious, since the score itself is one of the metrics of MetaDetect. In general, the more information, i.e., metrics, available to the meta model, the better it should perform (apart from overfitting). Figure 3.1 shows a scatter plot of the true IoU and the predicted IoU obtained by a meta regression model, where a single point represents a prediction. The meta model would be perfect if each

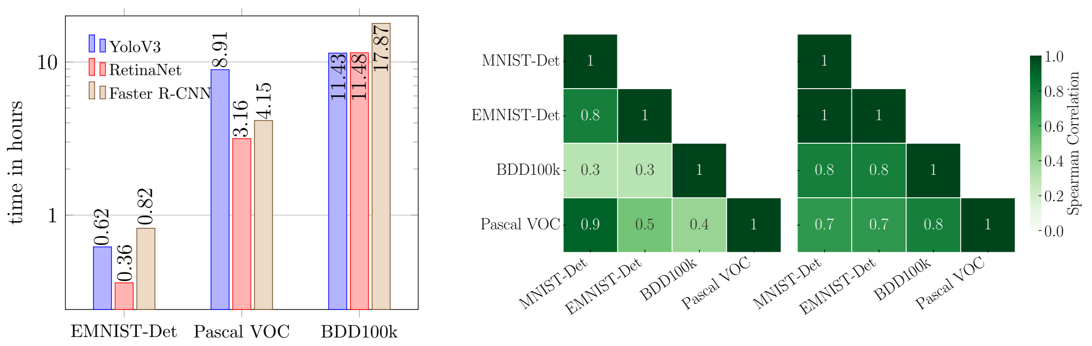


Figure 3.2: Left: Utilized time for one AL step (training until convergence + evaluating the query) for investigated settings in hours. Ranking correlations between area under mAP curve (AUC) values for the YOLOv3 detector (image-wise center and box-wise right).

point was placed on the line between the origin and (1,1), which would mean that the IoU for each prediction could be predicted perfectly. Although this is not the case, the points still seem to be concentrated along this line.

3.2 Active Learning Sandbox

In general, the research in active learning (AL) focuses on the development of new query strategies. In general, query strategies determine which previously unlabeled images should be labeled to extend the current training dataset. There are no restrictions on how the query is determined, e.g., based on random selection or based on UQ methods. When developing a new query strategy, an AL setup has to be chosen including object detectors, datasets, training parameters, the amount of initially labeled and queried images and the number of AL steps T. Since we have experienced ourselves that the development on common setups is very time-consuming, we rather focus on creating a universal setup for fast development and fair comparability of AL methods (Chapter 5), than on developing a new query strategy based on MetaDetect. With simpler tasks to be learned, which are realized by self-generated datasets, and object detectors with comparatively few learnable parameters, we achieve a reduction for the duration of an AL experiment of up to 32, see fig. 3.2 (left). Furthermore, the results generalize well when comparing different baseline methods (such as the random query, as well as uncertainty-based queries, such as the sum of box-wise entropy (2.82), probability margin (2.83), or Monte-Carlo dropout (2.85)) for different datasets, see fig. 3.2 (right). The correlation of the ranking of methods during the AL cycle is very similar across datasets, thus adding strong value to the development of new methods with our sandbox.

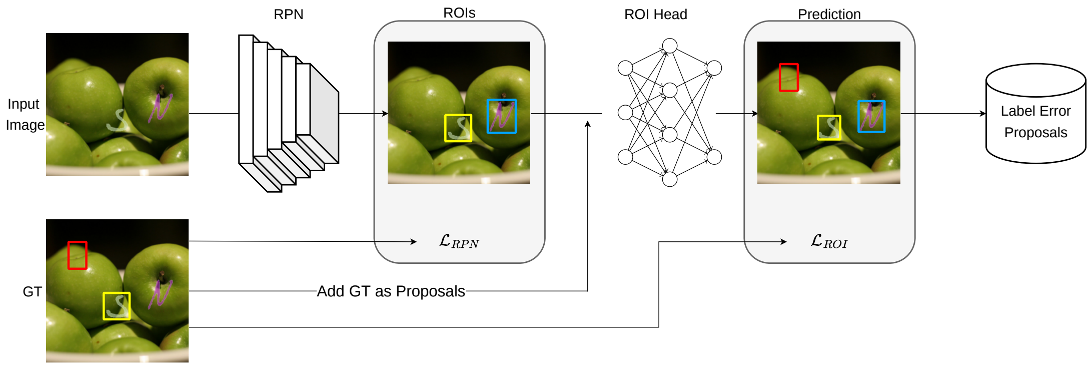


Figure 3.3: Visualization of our instance-wise loss method for detecting label errors. The red label indicates a *spawn*, the blue one a *drop* and the yellow one a correct label.

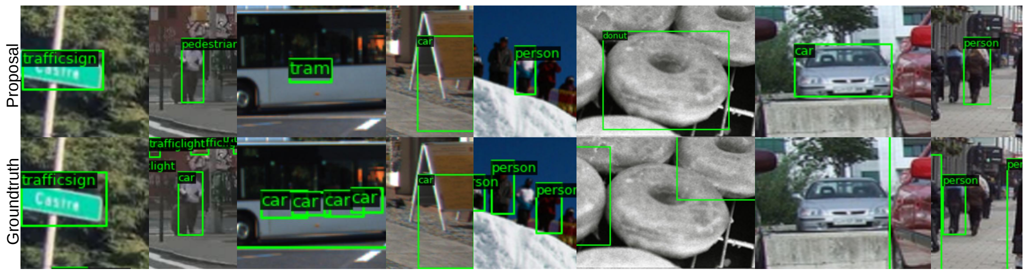


Figure 3.4: Visualization of detected label errors in real test datasets. The top row of images depicts the label error proposals and the bottom row the corresponding labels from the dataset. The image pairs belong from left to right in steps of two to BDD, KITTI, COCO and VOC.

3.3 Identifying Label Errors

During the development of MetaDetect and the AL sandbox, we often spotted images with erroneous labels. In fig. 3.1, several predictions exist with true $IoU = 0$, but with high predicted IoU. Upon closer inspection of these predictions, their evaluation was based on missing or misclassified labels. The idea of identifying label errors efficiently as an application of MetaDetect works only for missing labels and labels with an incorrect class assignment, where only false positives are reviewed in descending order by the predicted quality estimation. Since MetaDetect is prediction-based, all predictions covering a randomly spawned label are suppressed during score thresholding due to their small prediction quality score. Furthermore, MetaDetect can identify inaccurate located labels only randomly, since prediction quality estimates are determined independently of the labels. In general, clean labels of the test dataset are necessary to ensure fair method evaluation on test data, whether pure UQ, uncertainty-based query development in AL, as well as for the task of label error detection.

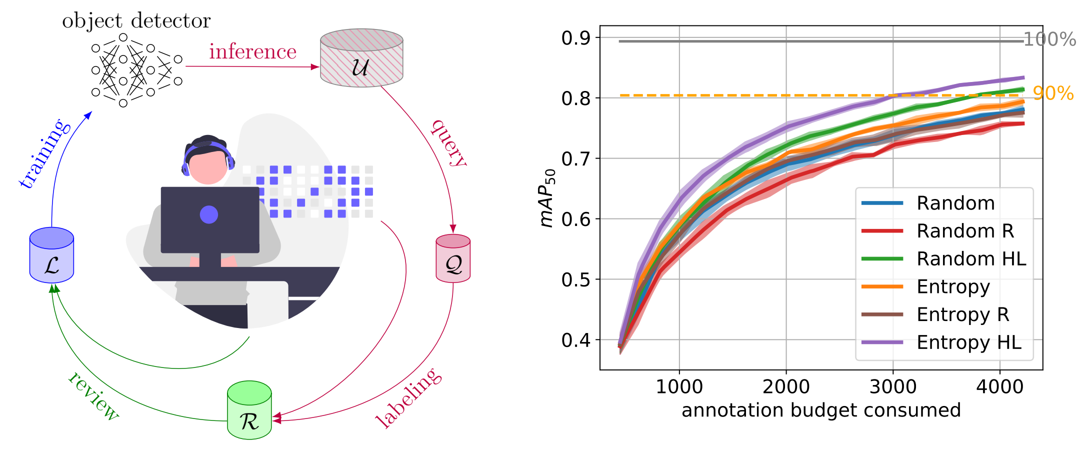


Figure 3.5: Left: Our active learning cycle consists of training on labeled data $\mathcal{L}$, querying and labeling informative data points $\mathcal{Q}$ out of a pool of unlabeled data $\mathcal{U}$ by an oracle and a review of acquired data $\mathcal{R} = \mathcal{L} \cup \mathcal{Q}$. Right: EMNIST-Det AL curves for Faster R-CNN, where flips and misses are simultaneously present in the training data.

For our label error benchmark, four different types of label errors are simulated:

- *drops:* missing labels,
- *flips:* correct localization but wrong classification,
- *shifts:* correct classification but inaccurate localization and
- *spawns:* labels that actually represent background.

In order to identify all four types of label errors efficiently, we developed an instance-wise loss method (see Chapter 6; fig. 3.3; (2.57)) and compared its performance to a random baseline, the box-wise objectness score, the box-wise classification entropy (2.82), as well as to a method from related work inspired by the box-wise probability margin, see [70] and (2.83). For the instance-wise loss, the drops are identified by a high first stage classification loss, the flips by a high second stage regression loss, the shifts by a high first and second stage regression loss and the spawns by a high first and second stage classification loss. Our method outperforms all baselines in the benchmark setting and is also capable of identifying real label errors on commonly used test datasets in object detection, see fig. 3.4.

3.4 Active Learning with Noisy Oracle

In the previous AL experiments (Section 3.2) we did not consider label errors, although we have identified a huge amount of real label errors on commonly used

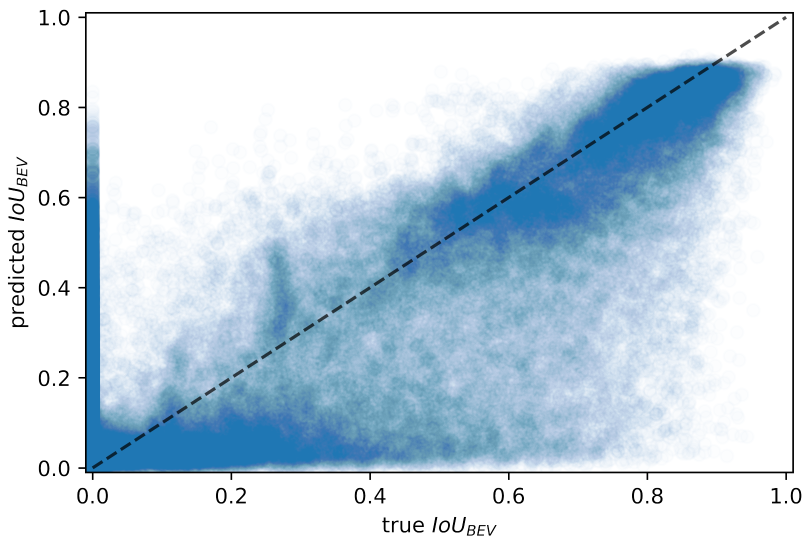


Figure 3.6: Box-wise scatter plot of true IoU_{BEV} and predicted IoU_{BEV} values for LMD on CenterPoint, nuScenes and score threshold 0.1. The predictions are based on a GB regressor.

test datasets in Section 3.3. In general, related work assume a perfect oracle, i.e., the (human) labeler does not make any errors during label acquisition, which does not hold in practice. In order to give the AL setup from Section 3.2 a more practical orientation, we will incorporate label errors into the AL cycle in the following (Chapter 7), see fig. 3.5 (left), which are efficiently identified by the instance-wise loss from Section 3.3. We evaluate all experiments in terms of annotation budget, which is split into the amount of queried bounding box labels and the amount of reviewed labels. We show that the strong performing highest loss (HL) review improves the test performance in terms of mAP (2.81) significantly compared to the same query method without or with the poor performing random (R) review, see fig. 3.5 (right). In general, the uncertainty-based query methods select the most uncertain images and the highest loss review proposes labels where we are most certain to have made an error.

3.5 LidarMetaDetect

LidarMetaDetect (LMD; Chapter 8) transfers the idea of MetaDetect (Section 3.1) to 3D bounding boxes with Lidar point cloud data. Lidar is a form of 3D laser scanning and generates sparsely distributed points in 3D space. Due to the sparsely occupied annotations, the object detection task is significantly more difficult compared to 2D and therefore UQ is of particular interest. Analogous to MetaDetect, the uncertainty metrics used for LMD are for instance:

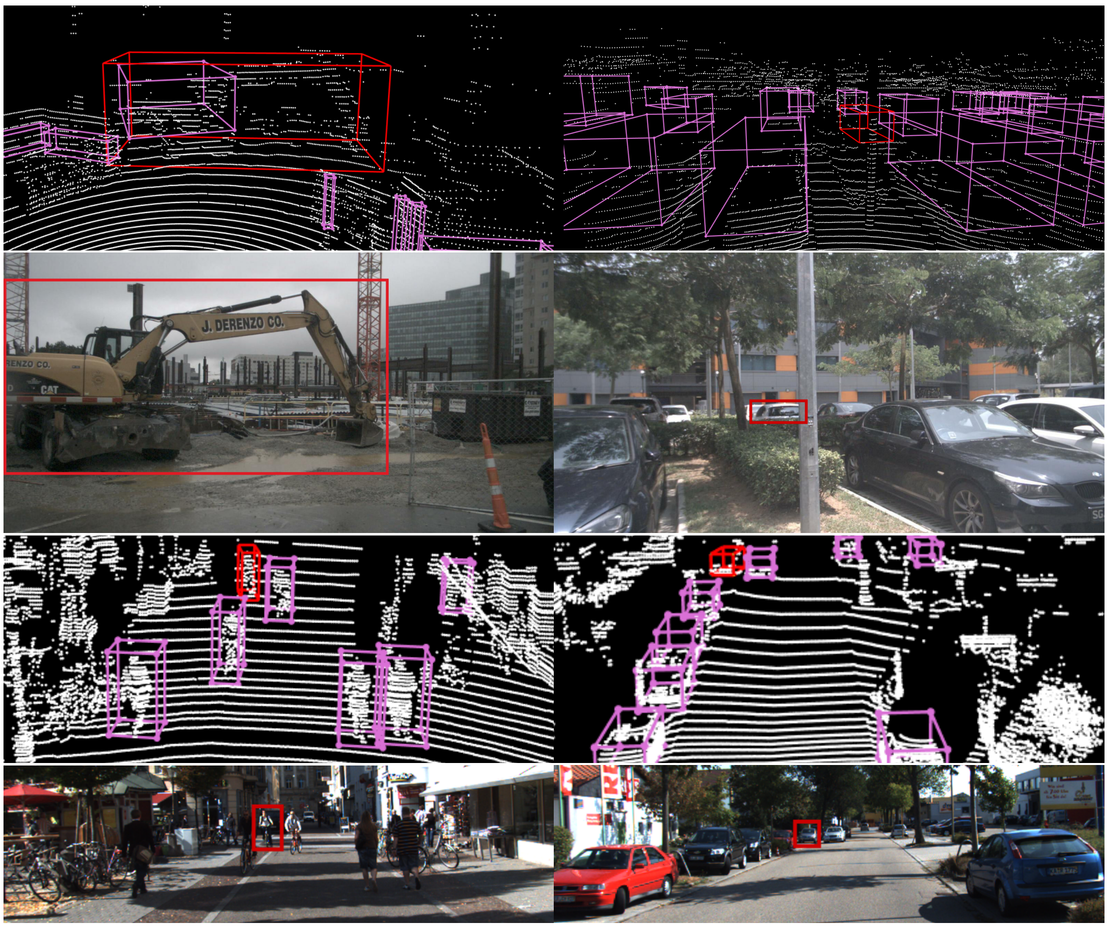


Figure 3.7: Proposed annotation errors in nuScenes (top two) and KITTI (bottom two). Top images show point clouds with annotations in purple and the proposal in red. Camera images aid the evaluation.

- the prediction itself, i.e., the geometry, the score and the predicted class probabilities,
- dimensions of the prediction, e.g., the surface area and the volume,
- the number of Lidar points within the prediction,
- minimum, maximum, mean and standard deviation over all assigned reflectance values,
- the number of suppressed candidate boxes during NMS,
- as well as minimum, maximum, mean and standard deviation of the localization, scores, class probabilities and reflectance statistics of all candidate boxes.

Figure 3.6 shows a scatter plot of the true IoU_{BEV} (which is an IoU for 3D bounding boxes) and the predicted IoU_{BEV} obtained by a meta regression model. The points concentrate along the line from (0,0) to (1,1) and therefore provide reliable uncertainty estimations in terms of IoU_{BEV}. As an application of LMD, we show results in terms of confidence calibration and label error detection. The latter focuses in particular on missing and misclassified labels, which can be identified more effectively with LMD compared to the objectness score of the object detector or a random baseline. Figure 3.7 shows examples of detected label errors in the KITTI [46] test dataset, a state-of-the-art dataset for 3D bounding boxes with Lidar point clouds. Note, that all identified label errors are real label errors and no simulated ones.

CHAPTER 4

METADETECT: UNCERTAINTY QUANTIFICATION AND PREDICTION QUALITY ESTIMATES FOR OBJECT DETECTION

The presented contents in the following chapter are taken almost word-by-word from [141].

4.1 Introduction

In recent years, deep neural networks have surpassed other approaches in many tasks with unstructured data. However, their in-transparent nature poses many questions and problems. In particular in safety-critical applications, the reliability of deep neural network predictions is of highest interest as neural networks tend to provide high confidences even when they fail [52]. In order to detect these and therefore make neural networks more interpretable, meaningful uncertainties are required [103]. A broad survey on uncertainty in machine learning can be found in [72]. A very important and also reducible type of uncertainty is the model uncertainty resulting from the fact that the ideal parameters are unknown and have to be estimated from data [3, 36, 44, 103]. Bayesian models consider these uncertainties [103]. However, as Bayesian models for deep learning are nowadays infeasible, many frameworks based on variational approximations have been proposed [3, 36, 44]. For instance, Monte-Carlo (MC) dropout [44] is used as a feasible approximation to Bayesian inference.

In classification tasks, a natural approach to detect incorrect predictions / false positives is to introduce a dispersion measure on the softmax probabilities, such as the entropy or one minus the highest softmax probability and then threshold on this [66, 94]. As in the works [17, 102, 130, 131, 133], that extend [66] to semantic segmentation, we refer to this task as *meta classification*. A good meta classification performance requires well-adjusted predictive uncertainty measures. Variational approximations of Bayesian learning are one approach to this. However, also feasible ones like [44] still require numerous inferences of one and the same input in order to estimate predictive uncertainty. In object detection, the network's objectness score is subject to thresholding which in a natural sense can be considered built in meta classification as part of state-of-the-art object detection pipelines [39, 123].

Confidence calibration has been proposed to account for the miscalibration of the objectness score [53, 118]. However, most approaches (e.g. histogram-based ones [86]) calibrate the score in a statistical sense. This is not sufficient to account for the correctness or quality of a single predicted bounding box. Therefore, uncertainty estimates for object detection networks seem more promising.

In [107] a baseline for meta classification in object detection is presented. The localization variables are determined using the candidate boxes that are present after the score thresholding and before the non-maximum suppression (NMS). The mean values of the associated candidate boxes represent the predicted bounding box and the added variances represent the localization uncertainty. In combination with the class uncertainty, which is generated from the entropy of the class probabilities in a softmax output, an uncertainty measure is obtained for each box. A threshold is used to decide whether the box is accepted or rejected as a prediction. To generate the uncertainty not only from the output, in [57, 64, 83, 90] the loss is changed in such a way that the uncertainties of the localization variables are learned and displayed in their own newly introduced neurons. In [57, 83] MC dropout is used for the localization and variance calculation of the predicted bounding boxes. The uncertainty introduced in the loss combined with the localization variance is applied to the individual anchor boxes/priors. In [57] the NMS is replaced by a Bayesian inference, in particular the assignment is done with a different cluster method. The performance with respect to prediction accuracy for all the listed performance metrics (mAP, PDQ Score, mGMUE, mCMUE) increases compared to other state-of-the-art methods introduced in [41, 42, 83, 90, 107, 108]. In [108], classification uncertainty is extracted from object detection networks via MC dropout.

For semantic segmentation, a number of works that estimate the segmentation quality have been introduced [17, 71, 102, 115, 130, 133, 134]. These works make use of the richness of information of the segmentation networks output, as there is a probability distribution available in each pixel of a given segment. In object detection tasks, this richness is not present.

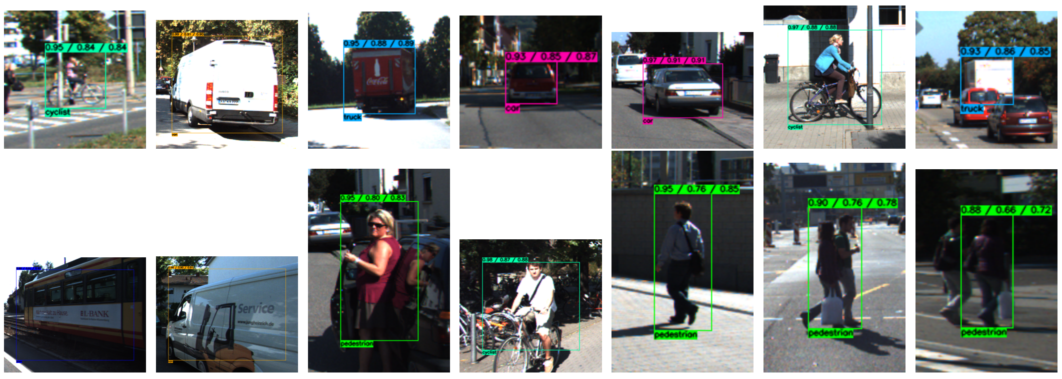


Figure 4.1: Examples of predicted bounding boxes with objectness score / true IoU / predicted IoU. The predictions, and therefore the data to train and evaluate gradient boosting, are made with the YoloV3 network, the KITTI dataset and a score threshold $t = 0.01$.

Our Contribution

In this chapter, we extend the false positive detection baseline of [66] from classification to object detection. Comparable to the works in semantic segmentation [17, 102, 130, 133] we consider two (meta) tasks: Discrimination between true positive ($IoU \geq 0.5$) and false positive ($IoU < 0.5$) boxes which is termed *meta classification* and regression with IoU values which is termed *meta regression*. To approach these tasks, we introduce a post-processing framework that in general can be added to any object detection network. Only by means of the network's output, our framework trains two models, i.e., one for each meta task. More precisely, we construct a set of handcrafted and interpretable metrics that may reveal the network's uncertainty about a prediction. Amongst others, this includes the number of candidate boxes before NMS that overlap with the given predicted box to a chosen extent, the score, the class probability distribution, the boxes size and aspect ratio and many others. If desired, these metrics can be extended by MC dropout statistics. In general any object-wise metric that seems to be helpful can be added. From this we obtain a structured dataset where the rows amount to the predicted objects in a given number of images and the columns amount to the constructed metrics. From this set of metrics we learn both meta tasks with different classical machine learning models, i.e., linear ones, shallow neural networks and gradient boosting. Figure 4.1 illustrates the effect of meta regression. In the depicted cases, the bounding boxes are predicted with an overconfident score, whereas the true IoU is significantly lower and the predicted IoU is close to the true one.

In our numerical experiments, we study the correlation of our constructed metrics with the IoU of prediction and ground truth, we study different sets of metrics and compare those to a score baseline. For both meta tasks, we significantly outperform the baseline. This is observed consistently over different datasets (KITTI [46], Pascal VOC [38] and COCO [97]) and different object detection networks (YoloV3 [39] and Faster R-CNN [123]). We do not observe an improvement when performing MC dropout with a modified YoloV3 network and including the obtained uncertainties into our set of metrics. This strengthens the statement, that our approach is reliable.

Related Work

In this section, we clarify the differences between related works and ours. The idea behind this work is in spirit similar to the approaches for semantic segmentation [17, 71, 102, 115, 130, 133, 134], however the nature of the output provided by segmentation networks and object detection networks is so different, such that the resulting uncertainty metrics are also clearly different.

Our approach is solely based on post-processing. One can plug in MC dropout quantities if the network architecture allows, however, this is not mandatory. On the other hand, references [57, 64, 83, 90] incorporate the uncertainty quantification into the original network and change their loss functions and also the ultimate layer. This requires additional training and does not aim at quantifying the uncertainty of the original network. In addition, as opposed to the other works we provide a simple and modular statistical benchmark suite that can be used for any object detection network in combination with any object-wise uncertainty quantification method to obtain performance indicators for the latter.

In [107] box-wise uncertainty information is calculated by means of associated candidate boxes. Whether a prediction is accepted or rejected is decided by simply thresholding on the resulting measure. Beyond that, neither meta regression, meta classification nor a general statistical evaluation of uncertainty measures is performed. Instead, they use their uncertainty measure to improve the NMS. In [118] a confidence calibration method is introduced, that trains a shallow Bayesian neural network on the confidence calibration task. The parameter uncertainties of the Bayesian network are then used to estimate the object detector's uncertainty, given the localization variables and the specific confidence level. However, unlike in our method, neither meta classification was performed, nor predictive uncertainty for a given input was calculated. In [108] the authors extract classification, localization and spatial uncertainty under MC dropout and turn this into an overall performance improvement. In comparison to our work, no meta classification or regression is performed, and the proposed concept requires dropout inference.

4.2 The Generic Object Detection Pipeline

In this section we briefly review the concept of object detection and its commonly used components, as they will be of interest for our uncertainty quantification method and experiments with those.

A typical performance measure, that indicates to which extent a prediction is in accordance to the ground truth, is the *IoU*, see (2.59).

Downstream of a typical object detection pipeline, the *candidate boxes* $\mathcal{B}$ are filtered by discarding all boxes $\hat{b}^i$ whose corresponding estimated score values s^i remain below a chosen threshold s_ϵ, see (2.70). That is, we define *filtered candidate boxes*

$$\mathcal{B}_s = \{\hat{b}^i \in \mathcal{B} : \hat{s}^i \geq s_\epsilon\} . \tag{4.1}$$

Typically, this is followed by the NMS, see alg. 2. Afterwards, $\mathcal{B}_{\text{NMS}}$ represents the set of predicted boxes.

4.3 Uncertainty Metrics for Object Detection

In this section we construct uncertainty metrics for every $\hat{b}^i \in \mathcal{B}_{\text{NMS}}$. We do so in two stages, first by introducing the general metrics that can be obtained from the object detection pipeline as is. Second, we extend this by additional metrics that can be computed when using MC dropout.

We consider a predicted bounding box $\hat{b}^i \in \mathcal{B}_{\text{NMS}}$ and its corresponding filtered candidate boxes $\hat{b}^j \in \mathcal{B}_s$, that were discarded by the NMS. The number of corresponding candidate boxes $\hat{b}^j \in \mathcal{B}_s$ filtered by the NMS intuitively gives rise to the likelihood of observing a true positive. The more candidate boxes $\hat{b}^j$ belong to $\hat{b}^i$, the more likely it seems that $\hat{b}^i$ is a true positive. We denote by N^i the number of candidate boxes $\hat{b}^j$ belonging to $\hat{b}^i$ but suppressed by NMS. We increment this number by 1 and also count in $\hat{b}^i$.

For a given image x we have the set of predicted bounding boxes $\mathcal{B}_{\text{NMS}}$ and the ground truth $\mathcal{Y}$. As we want to calculate values that represent the quality of the neural network's prediction, we first have to define uncertainty metrics for the predicted bounding boxes in $\mathcal{B}_{\text{NMS}}$. For each $\hat{b}^i \in \mathcal{B}_{\text{NMS}}$, we define the following quantities:

- the number of candidate boxes $N^i \geq 1$ that belong to $\hat{b}^i$ ($\hat{b}^i$ belongs to itself),
- the predicted box $\hat{b}^i$, i.e., the values of the tuple $(\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i, \hat{s}^i, \hat{p}^i_1, \dots, \hat{p}^i_C) \in \mathbb{R}^{5+C}$,

- the size $d = (\hat{w}^i \cdot \hat{h}^i)$ and the circumference $g = 2 \cdot \hat{w}^i + 2 \cdot \hat{h}^i$,

- IoU_{pb}: the IoU of $\hat{b}^i$ and the box with the second highest score that was suppressed by $\hat{b}^i$. This value is zero if there are no boxes corresponding to $\hat{b}^i$ suppressed by the NMS ($N^i = 1$),

- the minimum, maximum, arithmetic mean and standard deviation for all $(\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i, \hat{s}^i)$, size d and circumference g from $\hat{b}^i$ and all the filtered candidate boxes that were discarded from $\hat{b}^i$ in the NMS,

- the minimum, maximum, arithmetic mean and standard deviation for the IoU of $\hat{b}^i$ and all the candidate boxes corresponding to $\hat{b}^i$ that suppressed in the NMS,

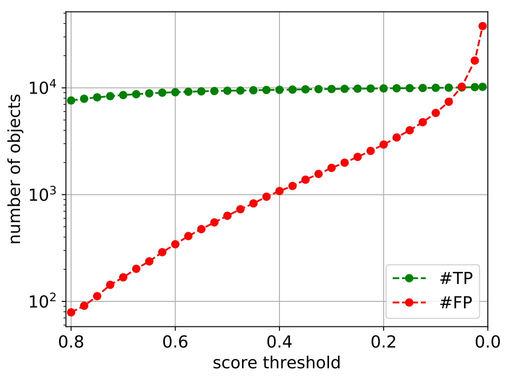
- relative sizes $rd = d/g$, $rd_{\min} = d/g_{\min}$, $rd_{\max} = d/g_{\max}$, $rd_{mean} = d/g_{mean}$ and $rd_{std} = d/g_{std}$,

- the IoU of $\hat{b}^i$ and the most overlapping ground truth box, this is not an input to a meta model but serves as the ground truth provided to the respective loss function.

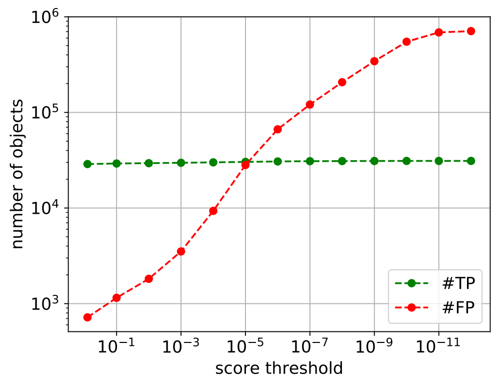
Altogether, this results in $46 + C$ uncertainty metrics.

We now elaborate on how to calculate uncertainty metrics for every $\hat{b}^i \in \mathcal{B}_{\text{NMS}}$ when using Monte-Carlo dropout. To this end, we consider the bounding box $\hat{b}^i \in \mathcal{B}_{\text{NMS}}$ that was predicted without dropout, and then we observe under dropout J times the output of the same anchor box that produced $\hat{b}^i$ and denote them by $\hat{b}^i_1, \dots, \hat{b}^i_J$. For these $J + 1$ boxes we calculate the minimum, the maximum, the arithmetic mean and the standard deviation for the localization variables and the objectness score over $\hat{b}^i, \hat{b}^i_1, \dots, \hat{b}^i_J$. This is done for every $\hat{b}^i \in \mathcal{B}_{\text{NMS}}$ and results in 20 additional dropout uncertainty metrics. This means $66 + C$ uncertainty metrics in total. Executing this procedure for all available test images we end up with a structured dataset. The number of predicted objects constitutes to the number of rows and the columns are given by the registered metrics. After defining a training / test splitting of this dataset, we learn meta classification ($IoU \geq 0.5$ vs. $IoU < 0.5$) and meta regression (quantitative IoU prediction) from the training part of this data.

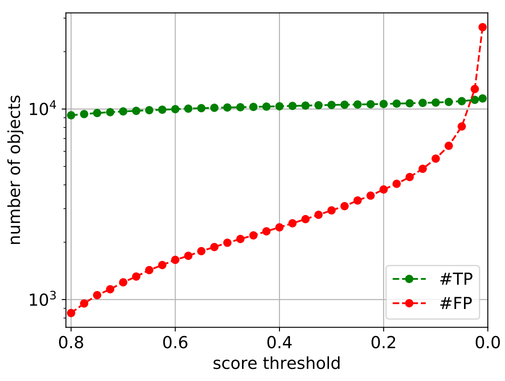
All these presented metrics, except for the true IoU, can be computed without the knowledge of the ground truth. Our aim is to analyze to which extent they are suited for the tasks of meta classification and meta regression.



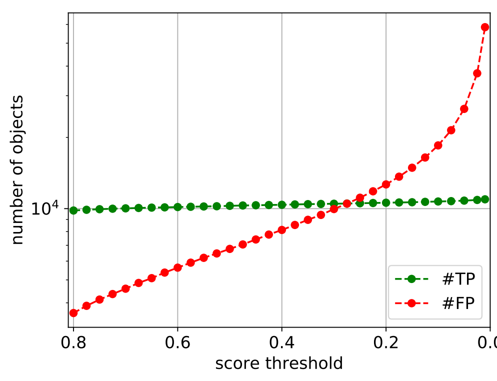
(a) KITTI + YoloV3



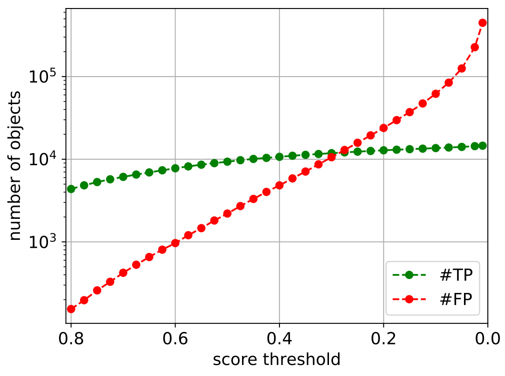
(b) KITTI + Faster R-CNN



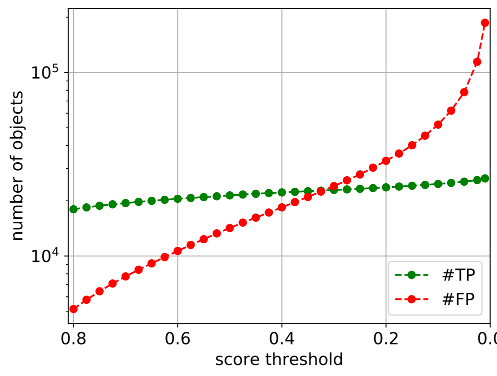
(c) Pascal VOC + YoloV3



(d) Pascal VOC + Faster R-CNN



(e) COCO + YoloV3



(f) COCO + Faster R-CNN

Figure 4.2: Comparison of the number of true positives (TP) and false positives (FP) for different score thresholds, the YoloV3 and Faster R-CNN network and the KITTI, Pascal VOC and COCO datasets.

4.4 Numerical Experiments: Pascal VOC, KITTI and COCO

In this section we investigate the properties of the metrics defined in the previous sections for the example of object detection for three different datasets and two different networks. We deploy the YoloV3 network [39] for which we use a reference implementation in Tensorflow [1] as well as weights self-trained on the KITTI dataset [46], the Pascal VOC2007 dataset [38] and the COCO dataset [97]. For KITTI, we split the labelled training images as for the test images are no labels available. The images 0-5480 are used to train our network and the last 2000 (images 5481-7480) are used to evaluate our method. For Pascal VOC, we used the training images from the years 2007 to 2011 to train our network, and we evaluate our method on the 4952 Pascal VOC2007 test images. For COCO, we train on all COCO2014 training images and evaluate our method on 2500 randomly selected test images from the COCO2014 test images. For Faster R-CNN [123], we use a reference implementation in Tensorflow as well as pre-trained weights for all three datasets. When evaluating our method on the KITTI dataset we use all available labelled images. The Faster R-CNN is trained exclusively for the two classes "person" and "car". Eventually, the Faster R-CNN might have overfitted the training data. Indeed, we observe high prediction accuracy. For Pascal VOC and COCO, we use the same images as for our tests with the YoloV3 network.

For more information about the default training and test parameters that we do not deviate from, we refer to the publicly available source codes and to [39, 121, 123] for a detailed explanation of the respective parameters.

We evaluate our methods for meta classification and regression on 33 different score thresholds, starting at 0.01, continuing with thresholds equal to $k/40$ until reaching 0.8 ($k = 1, \dots, 32$). Over the course of thresholds, the highest mean average precision (mAP) values obtained by YoloV3 are 91.99% on KITTI, 86.9% on Pascal VOC and 61.13% on COCO. For a given class, the average precision (AP) is the area under precision recall curve. Mean average precision is the mean of the AP values for all considered classes. For all three datasets the mAP is equal to the Pascal VOC metric [38], which determines true and false positives by means of an IoU threshold equal to 0.5 (AP@.5), which is consistent with our definition of meta classification. For Faster R-CNN, we evaluate our method also on these 33 different score thresholds, with one exception which is the KITTI dataset. The classes "person" and "car" are often predicted with extremely confident scores (very close to 1 or 0) such that the predictions differ only marginally at thresholds near 0.8 and 0.1. Therefore, thresholds of $10^{-1}, \dots, 10^{-12}$ were chosen for the KITTI dataset. This leads to highest obtained mAP values for Faster R-CNN which are 89.33% on KITTI, 80.31% on Pascal VOC and 55.29% on COCO.

$\hat{s}^i_{std}$	0.8370	$\hat{s}^i_{mean,MC}$	0.8239
$\hat{s}^i$	0.8231	$\hat{s}^i_{mean}$	0.8147
N^i	0.7149	$IoU^i_{pb,std}$	0.6529
$IoU^i_{pb,high}$	0.6021	$IoU^i_{pb,mean}$	0.5019
$\hat{y}^i_{high}$	0.4174	$\hat{y}^i$	0.4054
d/g^i_{high}	0.4049	$\hat{y}^i_{mean,MC}$	0.4044
$\hat{x}^i_{std}$	0.4029	$\hat{y}^i_{std}$	0.4013
d/g^i_{std}	0.3995	g^i_{std}	0.3904
$\hat{y}^i_{mean}$	0.3876	g^i_{high}	0.3821

Table 4.1: Strongest Pearson correlation coefficients for some constructed box-wise metrics for the KITTI dataset, the YoloV3 network and a score threshold $t = 0.01$.

Figure 4.2 depicts the number of true positives and false positives for the YoloV3 and Faster R-CNN network and the KITTI, Pascal VOC and COCO datasets. It is intuitively clear that as the score threshold decreases the number of TPs and FPs increases. The sum of TPs and FPs equals the number of all predicted bounding boxes and therefore constitutes the number of examples for training and evaluation of our meta models. The order of magnitude of the number of predictions is between 10^3 and 10^6.

In what follows, all results (if not stated otherwise) were computed from 10 repeated runs where training and validation sets (both of the same size) were resampled. We give mean results as well as standard deviations over the obtained results in brackets.

Correlation of Box-Wise Metrics with the IoU

Table 4.1 contains the Pearson correlation coefficients of the box-wise metrics with the IoU of prediction and ground truth for the KITTI test images and a score threshold $t = 0.01$ for the YoloV3 network. The score metrics seem to have strong correlations with the IoU, which is expected as it is supposed to discriminate true positives from false positives. The endings $high, mean, std$ represent the maximum, the arithmetic mean and the standard deviation, respectively, of the corresponding filtered candidate boxes for a given metric. The ending $mean, MC$ represents the arithmetic mean of the corresponding dropout predictions $\hat{b}^i, \hat{b}^i_1, \ldots, \hat{b}^i_J$ for the respective metric. Note that, although the 4 score related metrics (s^i_{std}, $s^i_{mean,MC}$, s^i, s^i_{mean}) show the highest correlation with the $IoU \geq 0.8$, these metrics may be very similar. Indeed, the correlation coefficients between these four metrics range from 0.81 to 0.99. Four additional metrics (N^i, $IoU^i_{pb_{std}}$, $IoU^i_{pb,high}$, $IoU^i_{pb,mean}$) also show decent correlations $0.5 \leq \rho \leq 0.8$, all other metrics only show a minor correlation. However, they may still contribute to a diverse set of metrics.

Meta Regression IoU

Score threshold s_ϵ	Method	Score baseline	MetaDetect	MetaDetect+dropout
0.5	LR	0.4188(±0.0080)	0.4447(±0.0059)	0.4477(±0.0065)
	GB	0.3966(±0.0069)	0.4485(±0.0104)	0.4478(±0.0102)
	NN	0.3448(±0.0096)	0.4435(±0.0097)	0.4436(±0.0114)
0.3	LR	0.5837(±0.0097)	0.6131(±0.0094)	0.6116(±0.0098)
	GB	0.5679(±0.0099)	0.6206(±0.0095)	0.6216(±0.0084)
	NN	0.5704(±0.0088)	0.6078(±0.0094)	0.6166(±0.0096)
0.1	LR	0.7133(±0.0032)	0.7594(±0.0032)	0.7591(±0.0033)
	GB	0.7138(±0.0037)	0.7766(±0.0025)	0.7780(±0.0025)
	NN	0.7120(±0.0029)	0.7568(±0.0042)	0.7635(±0.0060)

Table 4.2: Comparison of R^2 values for the score baseline and all available metrics (with and without dropout) for KITTI and YoloV3. We used linear regression (LR), gradient boosting (GB) and shallow neural nets (NN) for the task of meta regression.

Comparison of Different Meta Classifiers and Regressors

For meta classification (classification of $IoU \geq 0.5$ vs. $IoU < 0.5$), we compare results in terms of classification accuracy and in terms of the area under curve corresponding to the receiver operator characteristic curve ($AUROC$, see [30]). The receiver operator characteristic curve is obtained by varying the decision threshold of the classification output for deciding whether $IoU \geq 0.5$ or $IoU < 0.5$. For the task of meta regression, we state resulting standard deviations σ of the linear regression fit's residual as well as R^2 values. Throughout this section, we consider the following combinations of inputs: Score / score baseline refers to a meta regressor or classifier that was only trained by means of the score metric s^i which is a sensible baseline. MetaDetect refers to a meta regressor or classifier trained with $46+C$ metrics that include all metrics except for those that correspond to MC dropout. MetaDetect+dropout refers to all available $66+C$ metrics including dropout. MetaDetect+dropout is only available for YoloV3, since for this network we inserted a dropout layer with dropout rate 0.5 in all three detection branches right before the final convolutional layer and retrained the network.

For the task of meta regression and meta classification several models (meta models) can be used. The meta models we consider are linear regression (LR), gradient boosting (GB) and a shallow neural network (NN) with two hidden layers. In Table 4.2, we state R^2-values for meta regression with the KITTI dataset and the YoloV3 network for three different score thresholds s_ϵ. We be observed that gradient boosting outperforms linear regression and the shallow neural network consistently for all three score thresholds. This is in accordance to the results depicted by Figure 4.4. Gradient boosting outperforms the linear regression and the shallow neural network as a meta regression model for all 33 different score thresholds. In all our tests, we observed the same behavior for

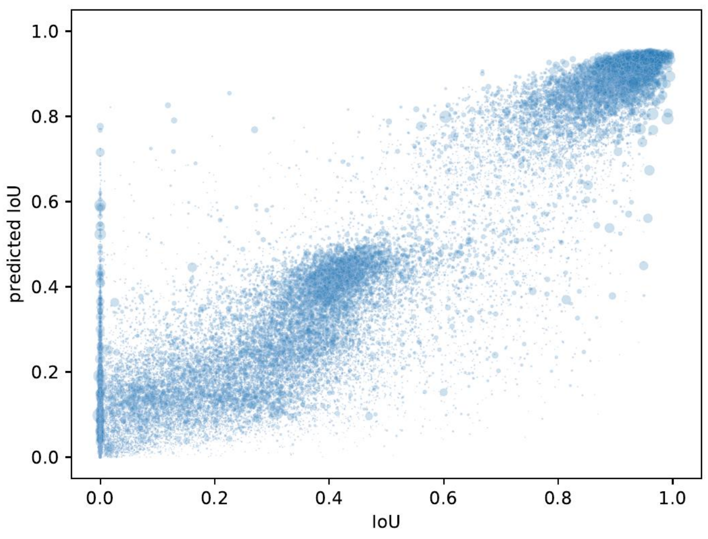


Figure 4.3: Box-wise scatter plot of true IoU and predicted IoU values for the KITTI dataset, the YoloV3 network and a score threshold $t = 0.01$. The predicted IoU values are generated with gradient boosting.

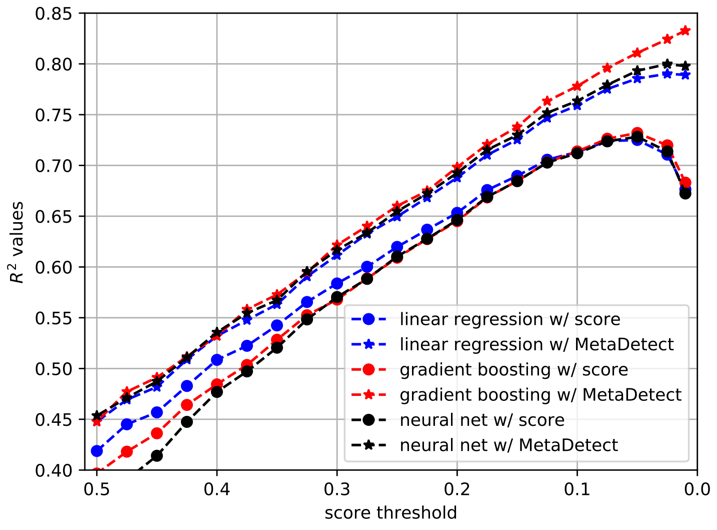


Figure 4.4: R^2 values for the task of meta regression for the score baseline and all uncertainty measures for the KITTI dataset, the YoloV3 network and different score values. The R^2 values are calculated with linear regression, gradient boosting and a shallow neural net.

			R^2-values for Meta Regression				
Network	Dataset	s_ϵ	Score baseline	Score & Loc.	MetaDetect	MetaDetect+do	dropout
YoloV3	KITTI	0.5	0.3966(±0.0069)	0.4300(±0.0101)	0.4485(±0.0104)	0.4478(±0.0102)	0.4315(±0.0127)
		0.3	0.5679(±0.0099)	0.6036(±0.0079)	0.6206(±0.0095)	0.6216(±0.0084)	0.6029(±0.0102)
		0.1	0.7138(±0.0037)	0.7517(±0.0028)	0.7766(±0.0025)	0.7780(±0.0025)	0.7527(±0.0030)
	VOC	0.5	0.3468(±0.0094)	0.3637(±0.0084)	0.3853(±0.0113)	0.3851(±0.0098)	0.3631(±0.0097)
		0.3	0.4234(±0.0081)	0.4445(±0.0067)	0.4700(±0.0078)	0.4672(±0.0089)	0.4435(±0.0081)
		0.1	0.5384(±0.0051)	0.5577(±0.0047)	0.5907(±0.0057)	0.5879(±0.0065)	0.5579(±0.0045)
	COCO	0.5	0.2317(±0.0089)	0.2570(±0.0095)	0.2787(±0.0077)	0.2774(±0.0084)	0.2486(±0.0097)
		0.3	0.4428(±0.0058)	0.4755(±0.0058)	0.4978(±0.0070)	0.4960(±0.0062)	0.4710(±0.0064)
		0.1	0.4931(±0.0028)	0.5459(±0.0035)	0.5914(±0.0030)	0.5911(±0.0028)	0.5457(±0.0033)
FRCNN	KITTI	10^{-1}	0.3703(±0.0136)	0.3921(±0.0145)	0.4101(±0.0153)		
		10^{-6}	0.8930(±0.0017)	0.9056(±0.0013)	0.9178(±0.0014)		
		10^{-12}	0.7927(±0.0020)	0.8471(±0.0017)	0.8819(±0.0013)		
	VOC	0.5	0.4844(±0.0106)	0.5120(±0.0116)	0.5430(±0.0121)		
		0.3	0.5682(±0.0084)	0.5911(±0.0091)	0.6182(±0.0089)		
		0.1	0.6289(±0.0054)	0.6601(±0.0069)	0.6836(±0.0074)		
	COCO	0.5	0.4178(±0.0046)	0.4250(±0.0049)	0.4330(±0.0052)		
		0.3	0.4730(±0.0076)	0.4799(±0.0077)	0.4890(±0.0068)		
		0.1	0.5438(±0.0056)	0.5558(±0.0050)	0.5651(±0.0041)		

Table 4.3: Comparison of R^2 values for the score baseline, score + localization, dropout (do) and all available metrics (with and without dropout) for KITTI, Pascal VOC and COCO. Especially, for Faster R-CNN there are no dropout metrics available. We used gradient boosting (GB) for the task of meta regression.

meta regression and meta classification for all three datasets. For this reason, only meta regression and meta classification results with gradient boosting are presented in the following. These findings indicate that there is a significant portion of mutual information contained in our metrics. The increase in R^2 when going from a single score metric to all our $46 + C$ metrics is ranging from 1.52 to 9.83 percent points (pp). On average the gain is about 4.86 pp. When further extending the considered metrics to $66 + C$ by the MC dropout metrics, we do neither observe a significant nor consistent increase in R^2 values.

Comparisons for Different Datasets, Networks and Sets of Metrics

Due to our observations in the previous paragraph, we fix all our meta classifiers and regressors to gradient boosting. Table 4.3 presents results for meta regression in terms of regression R^2 for all three datasets and both networks with three score thresholds each. Obviously, the tendencies indicated by the previously studied Table 4.2 are confirmed by Table 4.3 for the different datasets and networks. In all cases MetaDetect provides distinct increases in comparison to the score baseline.

Meta Classification $IoU \geq 0.5$ vs. $IoU < 0.5$

			Accuracies				
Network	Dataset	s_ϵ	Score baseline	Score & Loc.	MetaDetect	MetaDetect+do	dropout
YoloV3	KITTI	0.5	0.9411($\pm$0.0023)	0.9417($\pm$0.0021)	0.9421($\pm$0.0017)	0.9422($\pm$0.0023)	0.9415($\pm$0.0022)
		0.3	0.9090($\pm$0.0021)	0.9094($\pm$0.0020)	0.9129($\pm$0.0032)	0.9134($\pm$0.0025)	0.9092($\pm$0.0030)
		0.1	0.9063($\pm$0.0020)	0.9072($\pm$0.0022)	0.9090($\pm$0.0030)	0.9103($\pm$0.0027)	0.9058($\pm$0.0022)
	Pascal VOC	0.5	0.8562($\pm$0.0034)	0.8579($\pm$0.0030)	0.8597($\pm$0.0046)	0.8607($\pm$0.0039)	0.8559($\pm$0.0026)
		0.3	0.8456($\pm$0.0037)	0.8478($\pm$0.0027)	0.8485($\pm$0.0030)	0.8483($\pm$0.0035)	0.8456($\pm$0.0030)
		0.1	0.8496($\pm$0.0036)	0.8513($\pm$0.0029)	0.8541($\pm$0.0030)	0.8535($\pm$0.0032)	0.8505($\pm$0.0034)
	COCO	0.5	0.8106($\pm$0.0020)	0.8170($\pm$0.0019)	0.8223($\pm$0.0022)	0.8224($\pm$0.0027)	0.8181($\pm$0.0029)
		0.3	0.7828($\pm$0.0025)	0.7888($\pm$0.0026)	0.7949($\pm$0.0023)	0.7958($\pm$0.0020)	0.7901($\pm$0.0037)
		0.1	0.9006($\pm$0.0006)	0.9026($\pm$0.0007)	0.9044($\pm$0.0007)	0.9045($\pm$0.0008)	0.9028($\pm$0.0007)
Faster R-CNN	KITTI	10^{-1}	0.9637($\pm$0.0009)	0.9646($\pm$0.0012)	0.9655($\pm$0.0013)		
		10^{-6}	0.9731($\pm$0.0008)	0.9737($\pm$0.0006)	0.9741($\pm$0.0005)		
		10^{-12}	0.9949($\pm$0.0001)	0.9954($\pm$0.0001)	0.9961($\pm$0.0002)		
	Pascal VOC	0.5	0.8202($\pm$0.0051)	0.8270($\pm$0.0048)	0.8300($\pm$0.0052)		
		0.3	0.8364($\pm$0.0048)	0.8401($\pm$0.0055)	0.8433($\pm$0.0054)		
		0.1	0.8689($\pm$0.0053)	0.8711($\pm$0.0047)	0.8750($\pm$0.0058)		
	COCO	0.5	0.7679($\pm$0.0022)	0.7692($\pm$0.0021)	0.7705($\pm$0.0020)		
		0.3	0.7897($\pm$0.0028)	0.7907($\pm$0.0025)	0.7915($\pm$0.0032)		
		0.1	0.8495($\pm$0.0023)	0.8498($\pm$0.0027)	0.8501($\pm$0.0026)		
			AUROCs				
Network	Dataset	s_ϵ	Score baseline	Score & Localization	MetaDetect	MetaDetect+do	dropout
YoloV3	KITTI	0.5	0.9059($\pm$0.0069)	0.9073($\pm$0.0061)	0.9270($\pm$0.0035)	0.9272($\pm$0.0038)	0.9253($\pm$0.0052)
		0.3	0.9377($\pm$0.0028)	0.9432($\pm$0.0023)	0.9484($\pm$0.0027)	0.9489($\pm$0.0024)	0.9469($\pm$0.0028)
		0.1	0.9577($\pm$0.0018)	0.9626($\pm$0.0015)	0.9665($\pm$0.0011)	0.9666($\pm$0.0012)	0.9631($\pm$0.0011)
	Pascal VOC	0.5	0.8670($\pm$0.0058)	0.8708($\pm$0.0065)	0.8723($\pm$0.0065)	0.8732($\pm$0.0062)	0.8686($\pm$0.0053)
		0.3	0.8835($\pm$0.0044)	0.8885($\pm$0.0042)	0.8898($\pm$0.0039)	0.8907($\pm$0.0041)	0.8867($\pm$0.0044)
		0.1	0.9136($\pm$0.0018)	0.9172($\pm$0.0019)	0.9193($\pm$0.0018)	0.9199($\pm$0.0021)	0.9161($\pm$0.0020)
	COCO	0.5	0.7918($\pm$0.0049)	0.8068($\pm$0.0056)	0.8153($\pm$0.0035)	0.8160($\pm$0.0041)	0.8036($\pm$0.0046)
		0.3	0.8567($\pm$0.0025)	0.8671($\pm$0.0023)	0.8748($\pm$0.0025)	0.8751($\pm$0.0022)	0.8695($\pm$0.0022)
		0.1	0.8937($\pm$0.0019)	0.9097($\pm$0.0012)	0.9235($\pm$0.0008)	0.9239($\pm$0.0011)	0.9182($\pm$0.0012)
Faster R-CNN	KITTI	10^{-1}	0.9692($\pm$0.0025)	0.9737($\pm$0.0029)	0.9777($\pm$0.0033)		
		10^{-6}	0.9860($\pm$0.0002)	0.9899($\pm$0.0005)	0.9942($\pm$0.0004)		
		10^{-12}	0.9959($\pm$0.0002)	0.9970($\pm$0.0001)	0.9984($\pm$0.0002)		
	Pascal VOC	0.5	0.8854($\pm$0.0027)	0.8901($\pm$0.0031)	0.8985($\pm$0.0032)		
		0.3	0.9083($\pm$0.0030)	0.9113($\pm$0.0029)	0.9169($\pm$0.0033)		
		0.1	0.9284($\pm$0.0025)	0.9315($\pm$0.0030)	0.9356($\pm$0.0029)		
	COCO	0.5	0.8444($\pm$0.0021)	0.8468($\pm$0.0020)	0.8490($\pm$0.0026)		
		0.3	0.8617($\pm$0.0023)	0.8635($\pm$0.0027)	0.8654($\pm$0.0029)		
		0.1	0.8914($\pm$0.0015)	0.8932($\pm$0.0019)	0.8951($\pm$0.0018)		

Table 4.4: Comparison of meta classification accuracies and $AUROC$ values for the score baseline, the score + localization variables, dropout (do) and all available metrics (with and without dropout) for the KITTI, Pascal VOC and COCO datasets. For Faster R-CNN there are no dropout metrics available. We used gradient boosting (GB) for meta classification.

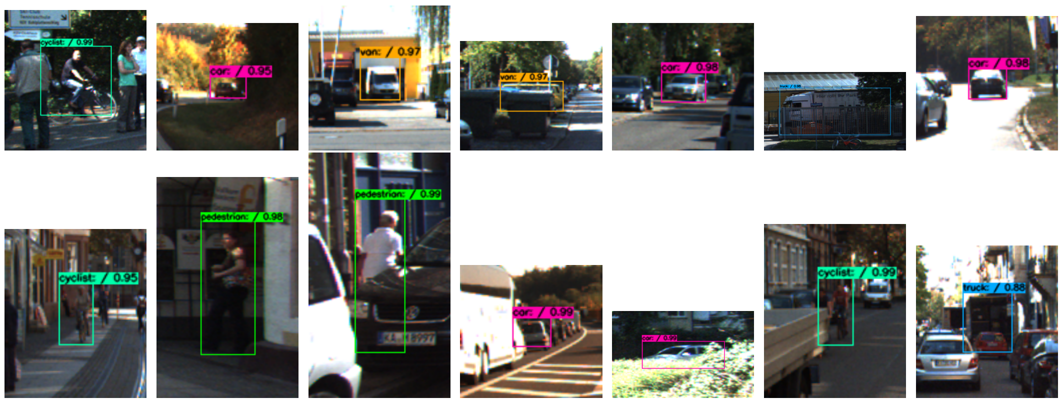


Figure 4.5: Examples of predicted bounding boxes with a true $IoU = 0$ but with high meta classification probabilities. These examples predicted by the YoloV3 network are counted as FPs as there is no corresponding ground truth available. Our prediction quality estimates signal with very high IoU values, that an object is present. The predictions, and therefore the data to train and evaluate gradient boosting, are obtained by the YoloV3 network applied to the KITTI dataset.

For the COCO dataset, YoloV3 network and score threshold 0.1 we even observe an increase of 9.83 percent points. Analogously to our findings in the previous paragraph, MetaDetect+dropout is not able to further improve. In our tests we also observed, although the dropout rate is set to 0.5, that the variation introduced by dropout inference seems to be rather limited. We performed 10 forward passes under dropout. In general, a more stochastic behavior of the inference, which could be promoted by additional dropout layers of an analogous batch normalization approach, could lead to an improvement. Here, we refer to the modular nature of our framework MetaDetect which allows for the incorporation of any uncertainty metric.

Figure 4.3 shows a scatter plot of the true IoU of prediction and ground truth and the IoU estimated by MetaDetect. The scattered points are well concentrated along the diagonal axis corresponding to the identity $\mathrm{id} : a \mapsto a$. This signals, that we obtain a well calibrated IoU estimate. The limited deviation from the identity shows, that this estimate gives rise to predictive uncertainty and an object-wise quality estimate. For example images, we refer to Figure 4.1

Considering the task of meta classification which amounts to false positive detection, we also achieve clear improvements when considering all our metrics instead of only considering the score s^i. Note that a naive meta classification baseline is random guessing. This can be implemented by randomly sampling real numbers from the uniform distribution over the unit interval and discriminating according to a chosen threshold. The corresponding $AUROC$ value is 0.5 and the best classification accuracy is given by the frequency of the dominant class (which depends on the score threshold in the object detection pipeline).

Results are summarized in Table 4.4 in terms of classification accuracy and $AUROC$. Our findings in these tables are similar to those for meta regression, the advantage of MetaDetect over the score baseline seems to be slightly more pronounced in terms of $AUROC$ than in terms of accuracy. In terms of accuracy, we observe an average increase of 0.43 pp, while in terms of $AUROC$, we obtain an average increase of 1.05 pp. The highest increase overall is obtained for the YoloV3 network and the COCO dataset with a score threshold $t = 0.5$ with 1.17 pp in terms of classification accuracy and for the YoloV3 network and the COCO dataset with a score threshold $t = 0.1$ with 2.98 pp in terms of $AUROC$.

Figure 4.5 shows examples of predicted bounding boxes that have a true $IoU = 0$. Thus, according to the ground truth they are FPs, but with high meta classification probabilities. Indeed, these boxes each represent an object of the correct class. Hence, more reliable meta classifiers may also help to identify labeling errors.

Comparison with State-of-the-Art-Methods

Related works calculate uncertainty for object detection by means of the localization variables [86] and/or MC dropout [83, 108, 118] or learn the uncertainty already in training with an adapted loss function [90]. In [57] the standard detection inference and NMS is reformulated in a Bayesian sense and in [64] different approaches of merging candidate boxes to gain performance are evaluated. Our approach, MetaDetect, also quantifies uncertainty from the softmax probabilities, the localization and MC dropout and therefore needs only the predictions that remain before the NMS. For this reason, MetaDetect can be applied to any object detection network and it, in contrast to [57, 64, 90], does not require any changes to the network's architecture or the loss function. MetaDetect can also handle other merging approaches (different from NMS) as long as there exist candidate boxes for the final predictions.

None of the works mentioned in this paragraph states meta classification / regression results. However, a viable approach to compare them with our work is to compare the corresponding uncertainty quantification (UQ) methods with respect to meta classification and regression performance.

Table 4.3 and Table 4.4 illustrate that MetaDetect outperforms score & localization (which corresponds to the UQ used in [86]) and Monte-Carlo dropoutout (corresponding to [83, 108, 118]). Furthermore, we have seen that adding the dropout metrics to the set of metrics of MetaDetect does not yield any significant improvement of R^2 / $AUROC$ values. Considering R^2 / $AUROC$ values averaged over score thresholds s_ϵ, networks and datasets, MetaDetect outperforms score &

localization by 2.24 / 0.49 pp, respectively, as well as Monte-Carlo dropoutout by 2.70 / 0.37 pp, respectively.

Note that, due to the generic nature of MetaDetect, adding more metrics as inputs will increase the meta classification / regression performance as long as we do not encounter overfitting. Those metrics can stem from the network's output but also from inside the network or any uncertainty quantification method of interest. Also approaches that modify the loss function and/or the network's training can be considered in this framework.

4.5 Conclusion and Outlook

In this chapter, we extended the notion of [66] from classification to object detection, considering the tasks of meta classification and meta regression introduced in [102, 130, 133] for semantic segmentation. We introduced a generic framework for quality estimation and false positive detection that can be extended by any uncertainty measure for object detection. To demonstrate the ability of our framework to perform these tasks more efficiently than state-of-the-art baselines, we introduced a variety of different uncertainty metrics and also considered widely used MC dropout [44] uncertainty. To study the individual metric's performance we compared correlation coefficients of the different metrics with each other and found that some of our metrics may well contribute to the overall meta regression and classification performance. This is confirmed by further numerical experiments where we compared different meta regressors and classifiers, i.e., logistic/linear regression, gradient boosting and shallow neural networks. We found that gradient boosting yields the best performance. Furthermore, with a thorough study of meta classification and regression performance over three different datasets and two different object detection networks, our method consistently outperforms the score baseline by a significant margin. In terms of meta classification we improve over the results of the score baseline by up to 1.17 pp classification accuracy and 2.98 pp $AUROC$. For meta regression obtain an improvement up to 9.83 pp in R^2.

We plan to incorporate these improved quality estimates into an active learning pipeline as well as performing an evaluation of the applicability of MetaDetect to data quality estimation. We believe that many labeling errors can be detected by means of a well-calibrated quality estimate.

Our source code is publicly available at *GitHub*. The application of MetaDetect to images of video sequences can be found on *YouTube*.

CHAPTER 5

TOWARDS RAPID PROTOTYPING AND COMPARABILITY IN ACTIVE LEARNING FOR DEEP OBJECT DETECTION

The presented contents in the following chapter are taken almost word-by-word from [126].

5.1 Introduction

Deep learning requires huge amounts of data, typically annotated by vasts amounts of human labor [11, 93, 184]. 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) [145] 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 post-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 [93, 160], which comprises several AL experiments of 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 [10, 24, 31, 37, 59, 116, 135, 140, 152, 179, 183]. However, these works did so in highly inhomogeneous settings which makes a comparison

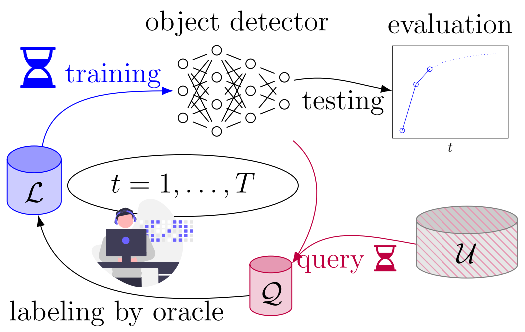


Figure 5.1: The generic pool-based AL cycle consisting of training on labeled data $\mathcal{L}$, querying informative data points $\mathcal{Q}$ out of a pool of unlabeled data $\mathcal{U}$ and annotation by a (human) oracle. In practice, training compute time is orders of magnitude larger than evaluating the AL strategy itself or the query step.

difficult. 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 [7, 8, 180, 181]. These observations demand for a development environment that enables rapid prototyping, cutting down the huge computational efforts of AL in OD and fostering comparability and transparency of different AL methods.

Our Contribution In this chapter, we propose a development environment that drastically cuts down the computational cost of developing AL methods. To this end, we construct (a) two datasets that generalize MNIST [91] and EMNIST [26] to the setting of OD making use of background images from MS-COCO [97] and (b) a selection of suitable small-scale models. We conduct experiments showing that results on our datasets generalize to a similar degree to complex real-world datasets (Pascal VOC [38] and BDD100k [182]), as they generalize among each other. We also demonstrate a reduction of computational effort of AL experiments by factors of up to 32. Further, a nuanced evaluation protocol is introduced to prevent wrong conclusions from misleading evidence encountered in experiments. We summarize our contributions as follows:

- We propose a sandbox environment with two datasets, three network architectures, several AL baselines and an evaluation protocol. This allows for broad, detailed and transparent comparisons at lowered computational effort.
- We analyze the generalization ability of our sandbox in terms of AL rank correlations. We find similar performance progressions indicating that results obtained by our sandbox generalize well to Pascal VOC and BDD100k, i.e., to the same extent as results generalize between 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. This involves configurations and hyperparameters (see `GitHub`).

5.2 Related Work

Numerous methods of AL have been developed in the classification setting [145] and largely fall into the categories of uncertainty-based and diversity-based query strategies. While uncertainty methods make use of the current model's prediction, diversity methods exploit the annotated dataset together with the current model and seek representative coverage of the data generating distribution. Due to increased complexity in annotations in OD, AL plays a large role in OD which has been addressed by some authors. Yoo and Kweon [179] present a task-agnostic method based on a loss estimation module. Brust et al. [10] estimate prediction-wise uncertainty by the probability margin and aggregate to image uncertainty in different ways. Roy et al. [135] follow a similar idea using classification entropy. Moreover, a white-box approach similar to query-by-committee is introduced. Haussmann et al. [59] utilize ensembles to estimate classification uncertainty via mutual information while Schmidt et al. [140] use combinations of localization and classification uncertainty. But in particular, as training a variety of detector heads in each step is very costly, ensemble query methods tend to be approximated by Monte-Carlo dropout [44, 45]. Other works investigate special AL-adapted OD architectures or loss functions [24, 183]. In this paper we compare uncertainty-based methods with each other that are exclusively based on fully supervised training of non-adapted object detectors [10, 24, 59, 135]. The preceding literature is difficult to compare since datasets, models, frameworks 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. In our work, we compare some above-mentioned methods to each other with equivalent configurations for frequently used datasets and architectures. Comparative investigations of this kind has escaped previous research in the field.

5.3 A Sandbox Environment with Datasets, Models and Evaluation Metrics

In this section we describe the objective of AL and our sandbox environment. The main setting we propose consists of two semi-synthetic OD datasets and

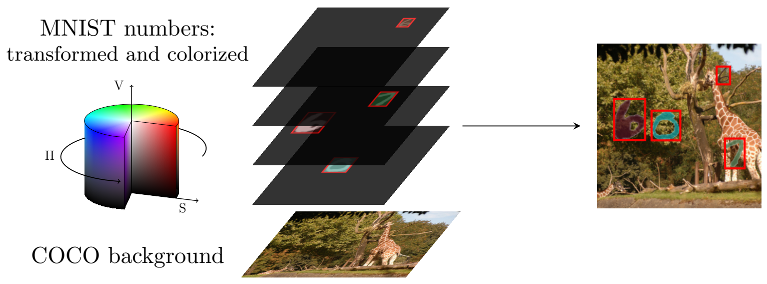


Figure 5.2: Generation scheme of semi-synthetic OD data from MNIST digits on a non-trivial background image from MS-COCO.

down-scaled versions of standard OD models leaving the detection mechanism unchanged. Additionally, we introduce evaluations capturing different aspects of the observed AL curve.

Active Learning The term active learning refers to a setup (cf. fig. 5.1) where only a limited amount of fully annotated data $\mathcal{L}$ is available together with a task-specific model. In addition, there is a pool (or a stream, however, we focus on pool-based AL) of unlabeled data $\mathcal{U}$ from which the model queries those samples $\mathcal{Q}$ which are most informative. Afterwards, $\mathcal{Q}$ is annotated by an oracle (usually a human worker), added to $\mathcal{L}$ and the model is fine-tuned or fitted from scratch again. Success of the query strategy is measured by observing an increase in test performance after training on $\mathcal{L} \cup \mathcal{Q}$. Evaluation of the current model performance before each query step leads to graphs like the ones in fig. 5.3. Querying can take diverse algorithmic forms, see Section 5.2 or [145].

Datasets We construct an OD problem by building a synthetic overlay to images from the real-world MS-COCO dataset (fig. 5.2), which constitutes the data of our sandbox. COCO images with deleted annotations provide a realistic, feature-rich background on which foreground objects are spawned to be recognized. We utilize two sets of foreground categories: MNIST digits and EMNIST letters. We apply randomized coloration (uniform $(h, s, v) \sim U([0.0, 1.0] \times [0.05, 1.0] \times [0.1, 1.0])$) and opacity ($\alpha \sim U([0.5, 0.9])$) to foreground instances such that trivial edge detection becomes unfeasible. 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 are obtained from the original transformed gray scale versions and the category label are inherited. Compared to simple OD datasets such as SVHN [112], the geometric variety in our datasets is more similar to those of large OD benchmarks such as Pascal VOC or MS-COCO. The reduction in the dataset complexity allows for high performance even for small architectures and leads to quickly converging training and low inference times. In the following we

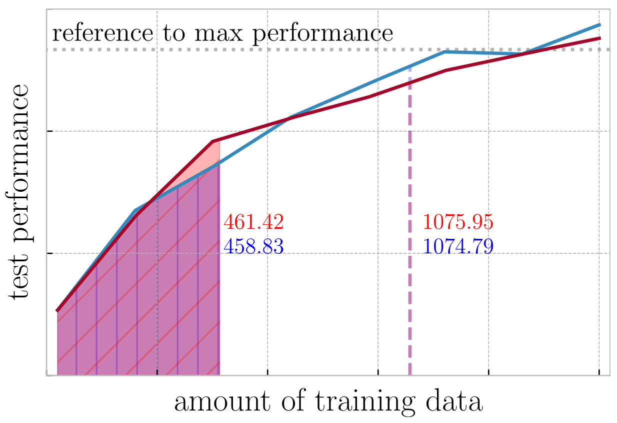


Figure 5.3: Area under AL curve (AUC) metric at different stages of an AL curve for two different query strategies (averaged, taken from experiments in fig. 5.5).

Dataset	c_x	c_y	w	h	# categories
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

Table 5.1: 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.

use the terms “MNIST-Det” and “EMNIST-Det” to refer to the two datasets.

Dataset Variability When comparing to existing OD datasets, our sandbox datasets MNIST-Det and EMNIST-Det resemble in variability the common benchmarks like Pascal VOC, MS COCO, KITTI or BDD100k. 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 c_x, c_y as well as the bounding box extent w and h, which we have collected in table 5.1. 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.

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

Table 5.2: Exemplary OD architectures with backbone configurations employed in the experiments and associated number of parameters.

		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 5.3: 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.

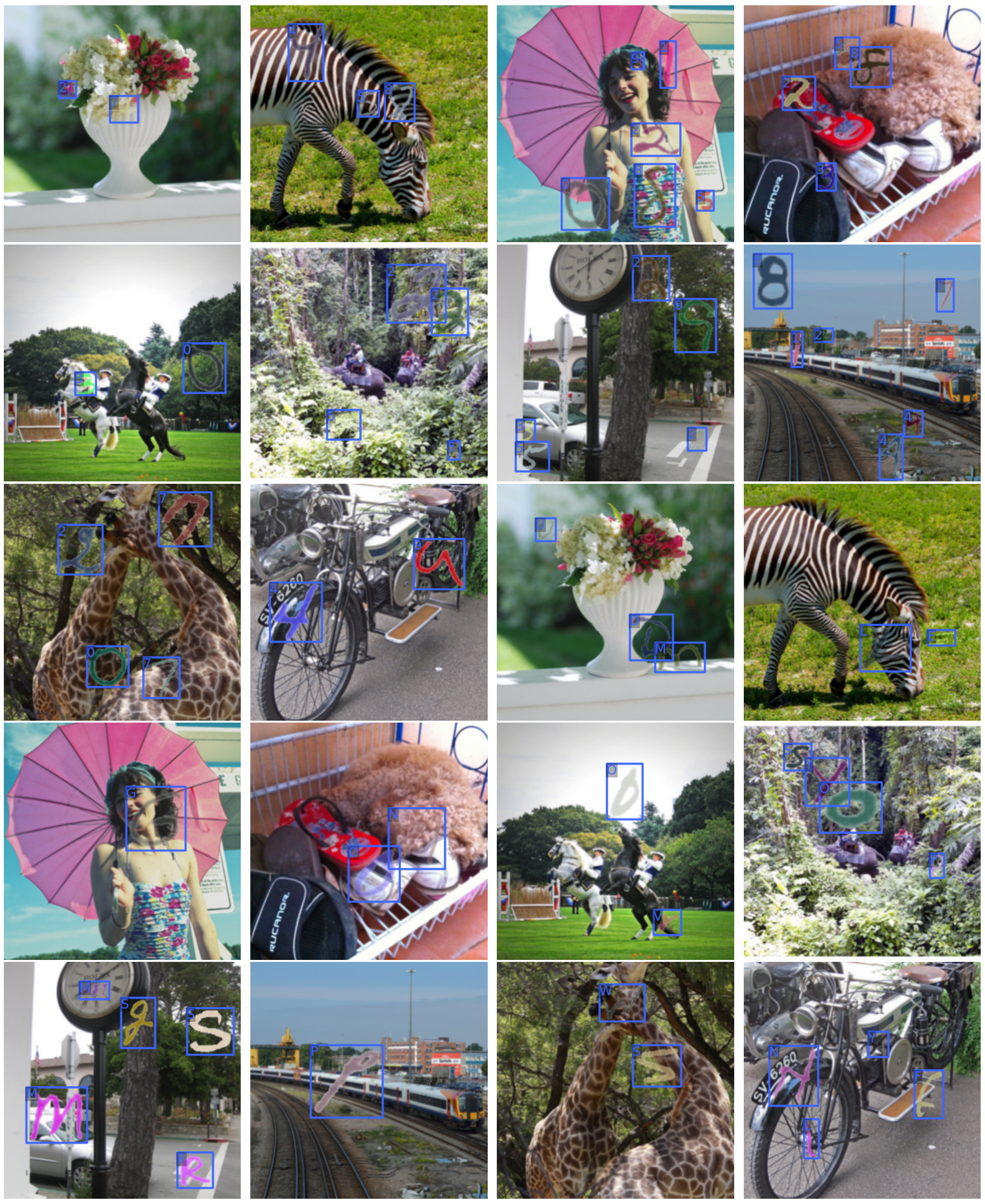


Figure 5.4: Dataset samples from MNIST-Det (top three rows) and EMNIST-Det (bottom three rows) including annotations.

Models Modern OD architectures utilize several conceptually different mechanisms to solve the detection task. 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 underlying detection mechanism is, 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 architectures with reduced network depth while keeping the detection head unchanged. Table 5.2 shows the choices for a YOLOv3 [39], RetinaNet [96] and Faster R-CNN [123] setup, which we have adapted together with the resulting number of parameters.

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 5.3 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. The parameter count is reduced by up to a factor of around six leading to a significant decrease in training and inference time.

For all architectures, we insert dropout layers between the two last layers (convolutional layers at all stages for YOLOv3/RetinaNet and fully connected for Faster R-CNN). For all experiments involving sampling, i.e., Monte-Carlo dropout and mutual information experiments, we use dropout rates of 0.5 and perform 10 forward passes.

Active Learning Methods in Object Detection The often used uncertainty-based query strategies from image classification, such as entropy, probability margin, Monte-Carlo dropout, and mutual information, determine instance-specific but not image-wise uncertainty estimates. However, the query strategies investigated here involve image-wise selection for annotation. It is, therefore, useful to introduce an aggregation step like in [10] to obtain image-wise query scores.

For a given image x, a neural network predicts a fixed number N_0 of bounding boxes $\hat{b}^i = \{\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i, \hat{s}^i, \hat{p}_1^i, \dots, \hat{p}_C\}$, $i = 1, \dots, N_0$, where $\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i$ represent the localization, $\hat{s}^i$ the objectness score (or analog) and $\hat{p}_1^i, \dots, \hat{p}_C^i$ the class probabilities for the C classes, see (2.63). Only the set of boxes post-non-maximum-suppression (NMS) and score thresholding are used to determine prediction uncertainties, see (2.70) and alg. 2. The choice of threshold parameters for NMS significantly influences the queries, since they decide surviving predictions.

Given a prediction $\hat{b}^i$ we compute its classification entropy $H(\hat{b}^i)$ (2.82) and the squared probability margin score, here also denoted as $PM(\hat{b}^i)$ (2.83). Furthermore, $c_{\max}$ denotes the class with highest probability. We implement dropout layers in order to draw Monte-Carlo dropout samples at inference time, where activations of the same anchor box $\hat{b}^{i,1}, \dots, \hat{b}^{i,J}$ are sampled J times. The final predictions and corresponding standard deviations are determined by (2.84), and the dropout uncertainty by (2.85). Note that for all these methods, uncertainty is only considered in the foreground instances.

Therefore, 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 many instances while averaging is strongly biased by the thresholds (e.g., large amounts of false positives could be filtered by a higher threshold). In the presented experiments, we use summation whereas further image-aggregation methods like averaging or taking the maximum are also addressed. Additionally, random acquisition acts as a completely uninformed baseline for us. 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 far less prominent in the literature, we focus on the more broadly established uncertainty baselines.

Evaluation 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 mAP_{50} [38] for which there is a maximum value $mAP_{50}^{\max}$ known when training on all available data. Some percentage, $\mathrm{x} \cdot 10^{-1} \cdot mAP_{50}^{\max}$ needs to be reached with as few data points as possible. Collecting performance over amount of queried data gives rise to curves such as in the top right of fig. 5.1 which we call *AL curves* in the following.

“Amount of training data” usually translates to the number of images which acts as a hyperparameter and is fixed for each method. Considering that each bounding box needs to be labeled and there tends to be high variance in the number of boxes per image in most datasets, it is not clear whether to measure annotated data in terms of images or boxes. Therefore, we stress that the scaling of the t-axis is particularly important in OD. Both views, counting images or boxes, can be argued for. Therefore, we evaluate the performance of each result not only based on images, but also transform the t-axis to the number of annotated boxes. By interpolation between query points and averaging over seeds of the same experiment, we obtain image- or box-wise standard deviations of the performance.

In light of the complexity of the AL problem, we adopt the area under the AL curve (AUC). It constitutes a more robust metric compared to horizontal or vertical cross-sections through the learning curves. Figure 5.3 shows two AL curves on the right and corresponding AUC at two distinct points t_1 and t_2. Note that in practice, $mAP_{50}^{\max}$ is not a quantity that is known. Therefore, the AL experiment may be evaluated at any given vertical section of t training data points. Knowing $mAP_{50}^{\max}$ (or the $0.9 \cdot mAP_{50}^{\max}$-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 blue by just looking at the current mAP_{50}. However, the AUC still favors red, since it takes the complete AL curve into account. This is in line with our qualitative judgement of the curves when regarded up to t_2. We use AUC for calculating rank correlations in Section 5.4. The raw results of the AUC metric are shown in the supp. material.

5.4 Experiments

In this section, we present results of experiments with our sandbox environment as well as established datasets, namely Pascal VOC and BDD100k, in the following abbreviated as VOC and BDD. We do so by presenting AL curves, summarizing benchmark results and discussing our observations for different evaluation metrics. We then show 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 when using the sandbox instead of VOC or BDD.

Implementation We implemented our pipeline in the open source MMDetection [19] toolbox. In our experiments for VOC, $\mathcal{U}$ initially consists of “2007 train” + “2012 trainval” and we evaluate performance on the “2007 test”-split. 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 $\mathcal{U}$ 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

	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 5.4: Maximum mAP_{50} values achieved by the models in table 5.2 on the respective datasets (standard-size detectors on VOC and BDD; sandbox-size on (E)MNIST-Det). The entire available training data is used.

		Query					Training		
		$\lvert\mathcal{U}_{init}\rvert$	$\lvert\mathcal{Q}\rvert$	ϵ_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

Table 5.5: Overview of important AL hyperparameters for querying data and model training for all datasets and architectures.

20,000 train images, 500 validation images and 2,000 test images. For reference, we collect in table 5.4 the achieved performance of the respective models for each dataset which determines the 90% mark investigated in our experiments.

AL Parameters Table 5.5 gives an overview of chosen hyperparameters for the AL cycle for all datasets and architectures. Thereby, $|\mathcal{U}_{init}|$ stands for the number of initially annotated images, $|\mathcal{Q}|$ for the size of the selected query, ϵ_s for the score threshold for query inference (thereby, 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, $|\mathcal{U}_{init}|$, $|\mathcal{Q}|$ 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 $|\mathcal{U}_{init}|$ and $|\mathcal{Q}|$ 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

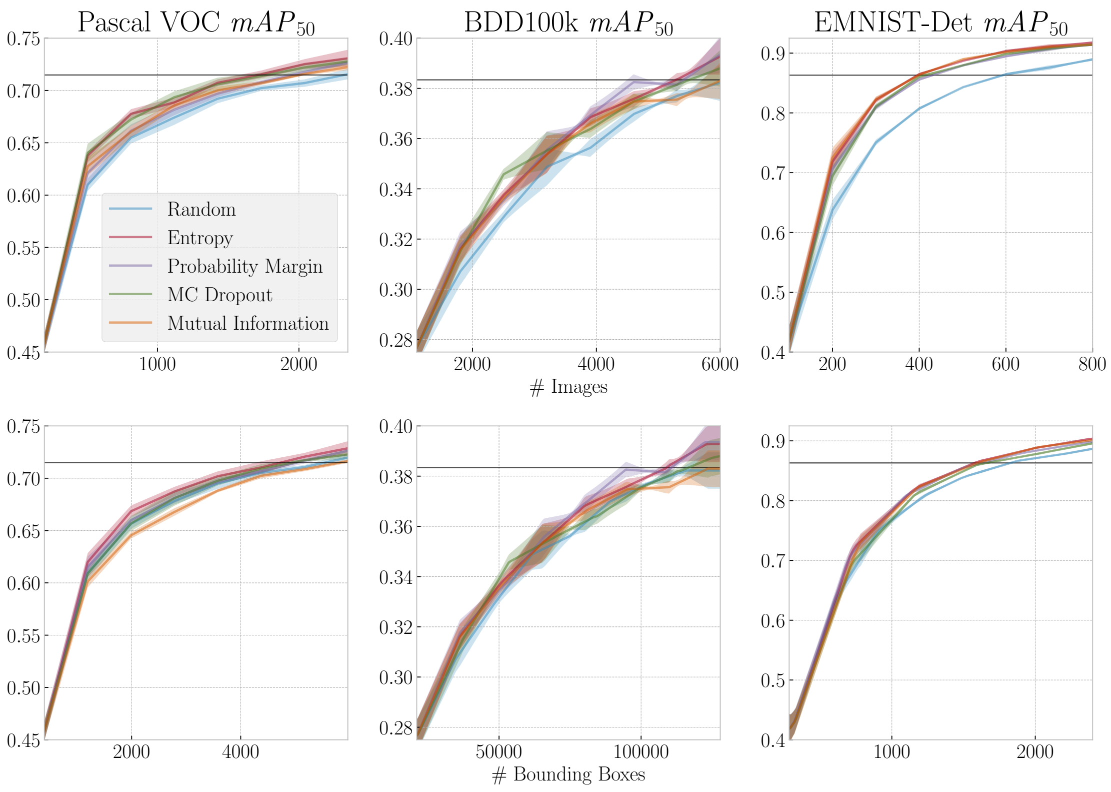


Figure 5.5: Comparison of YOLOv3 AL curves on three different datasets.

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 [10]) for computing instance-wise uncertainty scores.

Benchmark Results We first investigate differences in AL results with respect to the datasets where we fix the detector. This comparison uses the YOLOv3 detector on Pascal VOC, BDD100k and our EMNIST-Det dataset. We use the five query baselines described in Section 5.3. We obtain AL curves averaged over four random seeds and evaluated in terms of queried images as well as in terms of queried boxes, respectively. fig. 5.5 shows the AL curves with shaded regions indicating point-wise standard deviations obtained by four averaged runs each. The top column shows performance according to queried images while the second column shows the same curves but according to queried boxes. 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 separation vanishes or is far less clear. For EMNIST-Det, the

		# 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 (sum)	**288.6**	**687.7**	**1961.2**	**2866.5**	1292.0	2708.6	**5421.6**	64939.7
	Prob. Margin (sum)	310.8	733.9	2087.3	2901.5	1277.5	2721.5	5445.6	64794.9
	MC Dropout (sum)	293.3	749.6	2745.3	3027.7	1317.4	2926.4	7047.9	62395.5
	Mutual Inf. (sum)	289.6	719.0	2881.9	3124.9	**1248.0**	**2677.4**	7389.0	**61712.9**

Table 5.6: Amount of queried images and bounding boxes necessary to cross the 90% performance mark during AL. Lower values are better. Bold numbers indicate the lowest amount of data per experiment and underlined numbers are the second lowest.

difference between the random and the uncertainty-based queries decreases substantially, such that only a small difference in box-wise evaluation is visible. For VOC and BDD, the random baseline falls roughly somewhere in-between the uncertainty baselines in box-wise evaluation. This indicates that greedy acquisition with highest sum of uncertainty tends to prioritize images with a large amount of ground truth boxes. Obtaining many training signals improves detection performance in these cases, while giving rise to a higher annotation cost in the bottom panels. From this observation, we conclude that comparing AL curves based only on the number images gives an incomplete impression of performance and annotation costs. Additionally, instance-wise evaluation should be considered. We attribute the smoother curve progression in EMNIST-Det and VOC compared with BDD to the fact that BDD is a far more complicated detection problem with many small objects. However, the AL curve fluctuations on BDD tend to average out as we consider the AUC metric. This becomes clear in light of the results in the following section, where we study generalization across datasets.

In table 5.6 we show additional results. For each detector to reach $0.9 \cdot mAP_{50}^{\max}$, the table shows the number of images required, resp. the number of boxes per method. We see the rankings often favor the entropy baseline, however, the overall rankings are rather unstable throughout the table. Note in particular, that for (the arguably hardest detection problem) BDD, random beats the mutual information for YOLOv3. In the analog setting for Faster R-CNN the image-wise margin of the mutual information merely becomes slim. This observation also holds for box-wise evaluation and is more pronounced. In six cases, random beats some informed method. For RetinaNet, we 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. These results yield further evidence that the consideration of only a single

		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 5.7: 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.

evaluation metric for active learning performance is insufficient.

We conclude that in order to assess the viability of a method, AL curves should be viewed from both angles: performance over number of images and over number of boxes queried.

Further Benchmark Results 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

		# 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**

Table 5.8: Area under AL curve results per query method for maximal amount of data (images/bounding boxes) selected; higher values are better. Bold numbers indicate the highest AUC per experiment and underlined numbers are the second highest.

on the rank correlations, that even in a rather late evaluation (when some fixed reference mark in performance has already been crossed), the AUC 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.

Further Image-Aggregation Methods Figure 5.6 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 Monte-Carlo 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.

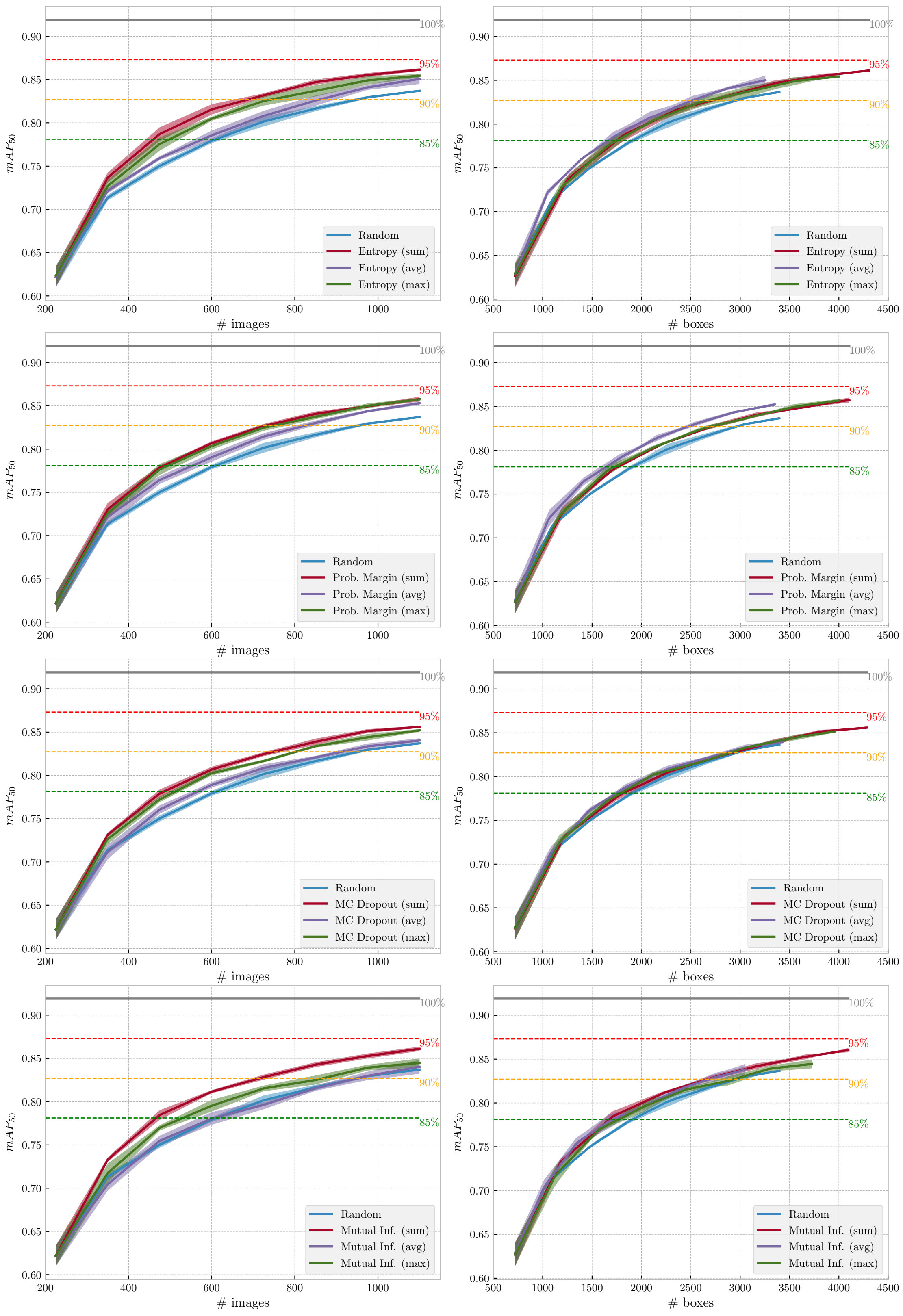


Figure 5.6: Ablation study on the aggregation method for RetinaNet on the EMNIST-Det dataset.

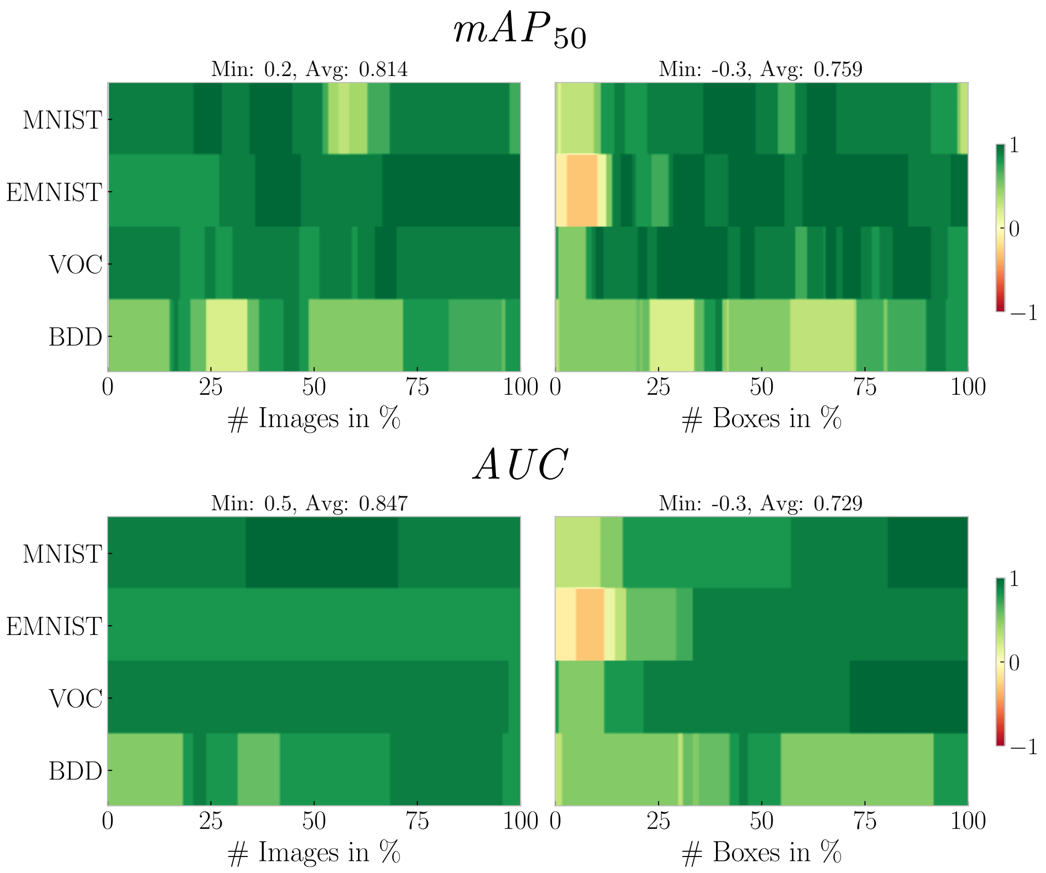


Figure 5.7: Intensity diagrams of rank correlations between the mAP_{50}, resp. cumulative AUC and the final rankings obtained at the $0.9 \cdot mAP_{50}^{\max}$-mark.

Using our sandbox environment makes extensive ablation studies of hyperparameters possible within a few days.

Generalization of Sandbox Results Instead of evaluating the pure performance at each AL step we have proposed computing the corresponding AUC as a more robust metric of AL performance. With respect to the final method ranking at $mAP_{50}^{\max}$, 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 5.7 shows intensity diagrams representing the rank correlations both, in terms of image-wise and box-wise evaluation. The t-axes are normalized to the maximum number of images, resp. bounding boxes queried, color indicates the Spearman correlation of the rankings. In fig. 5.7 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

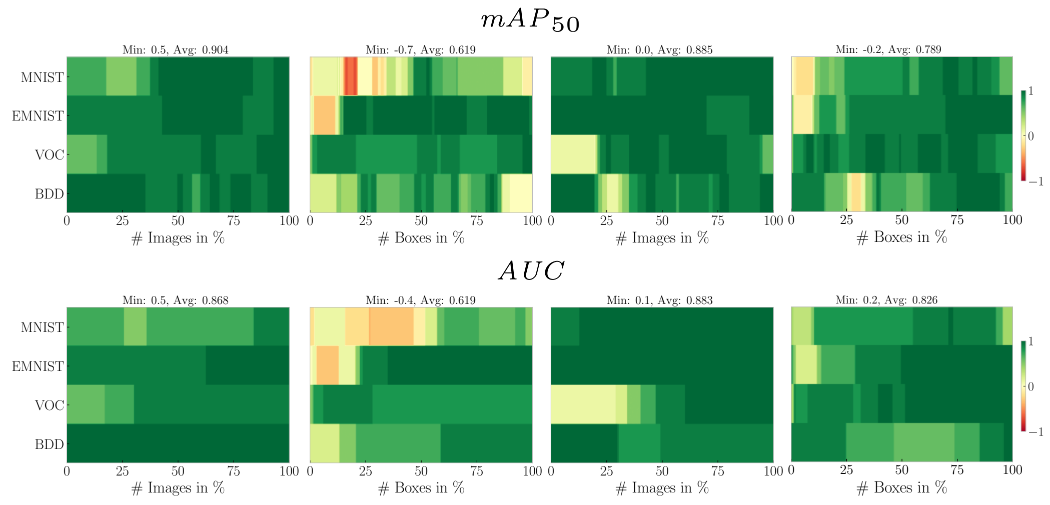


Figure 5.8: Curves of rank correlations between the cumulative AUC and the final rankings at the 90% max performance mark for the RetinaNet (left half) and the Faster R-CNN (right half) detector.

to be larger for AUC than mAP_{50}. Ranking correlations for mAP_{50} and AUC for RetinaNet and Faster R-CNN, see fig. 5.8, tend to show similar behavior as for the YOLOv3 detector. The AUC fluctuates less then 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. Note that the final ranking of either method does not need to be perfectly correlated with the $mAP_{50}^{\max}$-ranking for two reasons. Firstly, the latter does not take into consideration early performance gains and secondly, the $mAP_{50}^{\max}$-ranking is a horizontal section through the curves while mAP_{50} and AUC are vertical sections. We conclude that AUC tends to be highly correlated with the $mAP_{50}^{\max}$-ranking and is more stable with respect to t than mAP_{50}.

Next, we study comparability of AL experiments between the sandbox setting and full-complexity problems (VOC and BDD). To this end, we consider the cross-dataset correlations of the AUC score when fixing the detection architecture. Figure 5.9 shows correlation matrices for image- and instance-wise evaluation on the left for the YOLOv3 detector. VOC-BDD correlations tend to be similar to EMNIST-Det-VOC and EMNIST-Det-BDD correlations in image-wise evaluation. However, when correcting for variance in instance-count per image in box-wise evaluation on the right, we find correlations are generally high. In particular, results for BDD and VOC are roughly equally correlated with results

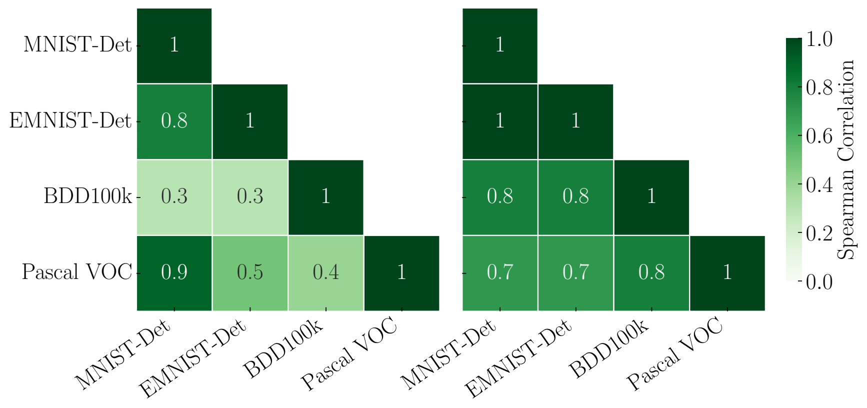


Figure 5.9: Ranking correlations between AUC values for the YOLOv3 detector (image-wise left and box-wise right).

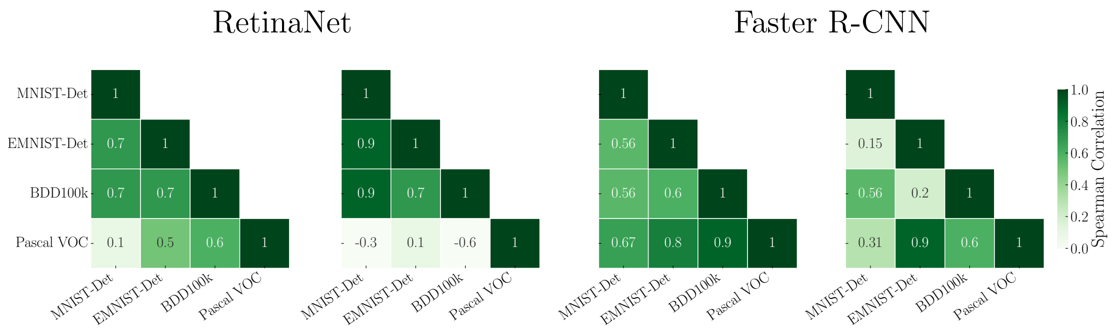


Figure 5.10: Ranking correlations between AUC values for the RetinaNet and Faster R-CNN detector, left: image-wise; right: instance-wise evaluation.

on any other dataset. Figure 5.10 shows correlation matrices for the RetinaNet and Faster R-CNN. The image-wise comparison 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 many missing labels when looking at the VOC data (cf. fig. 5.11). 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. We conclude that comparing methods in the simplified setting yields a similar amount of information about relative performances of AL as the full-complexity setting.

Compute Time AL for advanced image perception tasks 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 several random seeds to be executed. Figure 5.12 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. The time-axis is scaled logarithmically, so the experiments on EMNIST-Det are always faster by at least half an order of magnitude. Training of YOLOv3 on VOC does not start from COCO-pretrained weights (like YOLOv3+BDD) since the two datasets VOC and COCO are highly similar. In this case, we opt for an ImageNet [137]-pretrained backbone like for the other detectors. Overall, we 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 query strategies can be evaluated within half a day on a single Nvidia Tesla V100-SXM2-16GB.

5.5 Conclusion

In this cahpter, we investigated the possibility of simplifying the active learning setting in object detection to accelerate development and evaluation. We found that for a given detector, active learning results, in particular on instance level, generalize well between different datasets, including (E)MNIST-Det. Particularly, we find a representative degree of result comparability between our sandbox datasets and full-complexity active learning. In our evaluation, we included a more direct measurement of annotation effort in counting the number of boxes in addition to queried images. Meanwhile, we can save more than an order of magnitude in total compute time by the down-scaling of the detector and reducing the dataset complexity. Our environment allows for consistent benchmarking of active learning methods in a unified framework, thereby improving transparency. We hope that the present sandbox environment, findings and configurations along with the implementation will lead to further and accelerated progress in the field of active learning for object detection.

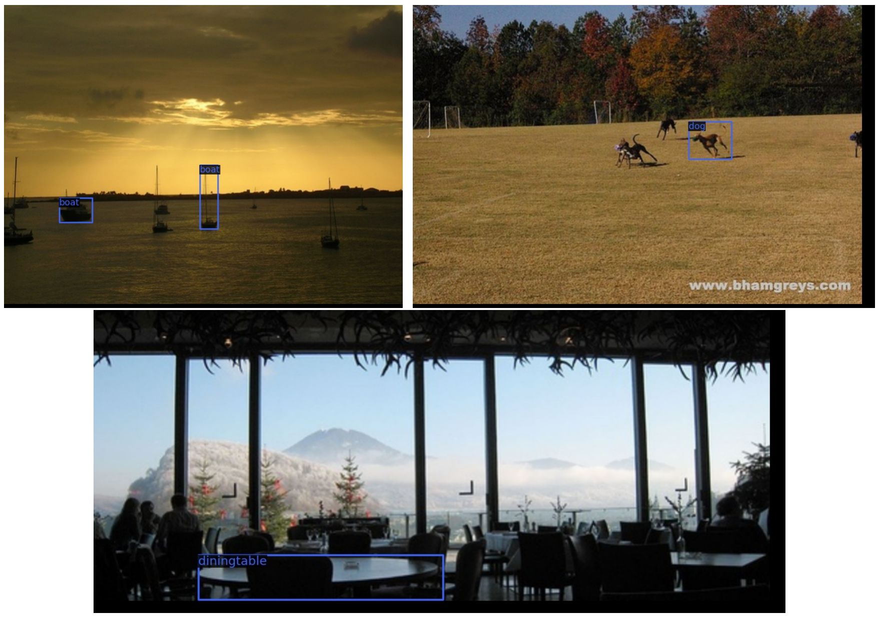


Figure 5.11: Annotation examples (full annotation!) from the Pascal VOC [38] test dataset.

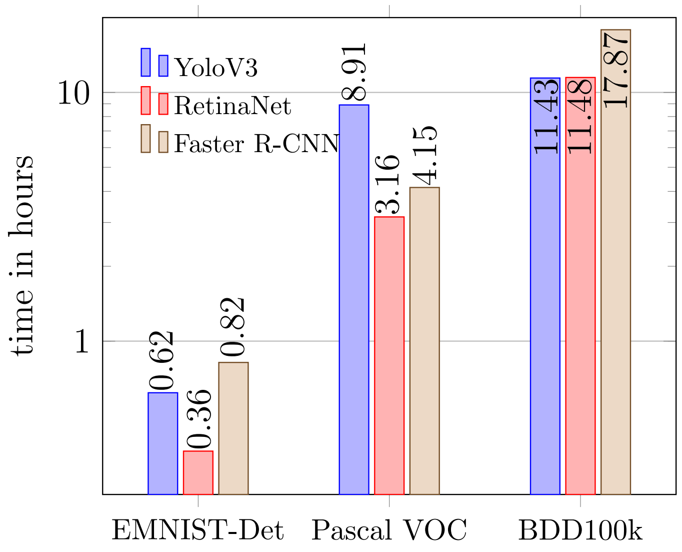


Figure 5.12: Utilized time for one AL step (training until convergence + evaluating the query) for investigated settings in hours.

CHAPTER 6

IDENTIFYING LABEL ERRORS IN OBJECT DETECTION DATASETS BY LOSS INSPECTION

The presented contents in the following chapter are taken almost word-by-word from [142].

6.1 Introduction

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, 73, 75, 79, 87]. Deep neural networks (DNNs) are particularly data hungry [154]. 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].

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 labelers 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 [33, 113, 132].

The extent to which noisy labels affect the model performance is studied by [12, 167]. Wu et al. [167] observe 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, Buttner et al. [12] shows that inaccurate labels in terms of annotation size in training data yields to significant decrease of test performance. Other methods model label uncertainty [108, 127] or improve robustness w.r.t. noisy labels [16, 43, 92, 186].

Up to now, automated detection of label errors has received less attention. There exist some works on image classification datasets [113, 114, 159], one work on semantic segmentation datasets [132] and some works for object detection [70, 82]. Label errors may affect generalization performance, which makes their detection desirable [114]. Furthermore, there is business 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 [182] and EMNIST-Det (Chapter 5) dataset. The latter is a semi-synthetic dataset consisting of EMNIST letters [26] pasted into COCO images [97] 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 [182], MS-COCO [97], Pascal VOC [38] and KITTI [46], and also for a proprietary dataset on car part detection by the company ControlExpert that our method detects 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 to reproduce of our results, see *GitHub*.

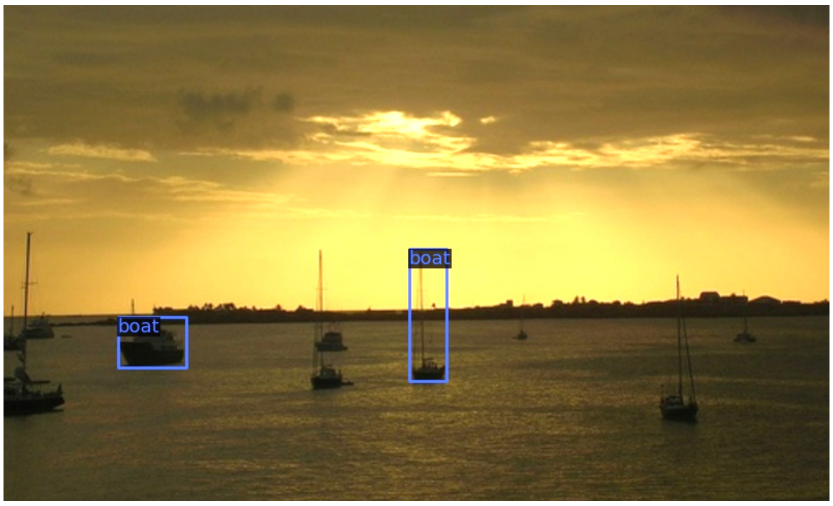


Figure 6.1: Example image from the Pascal VOC 2007 test dataset with two labeled boats marked by the blue boxes and multiple unlabeled boats.

6.2 Related Work

The influence of noisy labels in the 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 [114] and this also applies to object detection. Figure 6.1 shows an image from the Pascal VOC 2007 (VOC) 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 [50, 56, 67, 76, 113, 122, 165, 170, 185]. Moreover, the task of label error detection has been tackled in [21, 114] and theoretically underpinned in [113]. Chen et al. [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 [114] followed by the introduction of the task of label error detection [113]. 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 (Amazon Reviews), resulting in improved model performance by re-training on cleaned training data. This line of works has been recently extended to the task of multi-label classification in [159], where a single object is shown per image but may carry multiple labels.

For object detection, Wu et al. [167], as well as Xu et al. [169] study how noisy training labels affect the model performance, observing that the model is reasonably robust when dropping labels. To counter label errors in object detection, methods that model label uncertainty [108, 127] or more robust object detectors have been developed [12, 43, 77, 92, 169, 186]. Buttner et al. [12] 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. [82] 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. [70] 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 et al. [132] 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 (noisy) label and the corresponding uncertainty estimate.

Our work 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. The label error detection methods simulate a) different types of label errors and detect these with the help of a tracking algorithm [82] for images derived from video sequences or b) class-flips and identify these via a probability differential (PD) [70], where, however, the focus is on training. 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.

6.3 Label Error Detection

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.

Label Error Benchmark

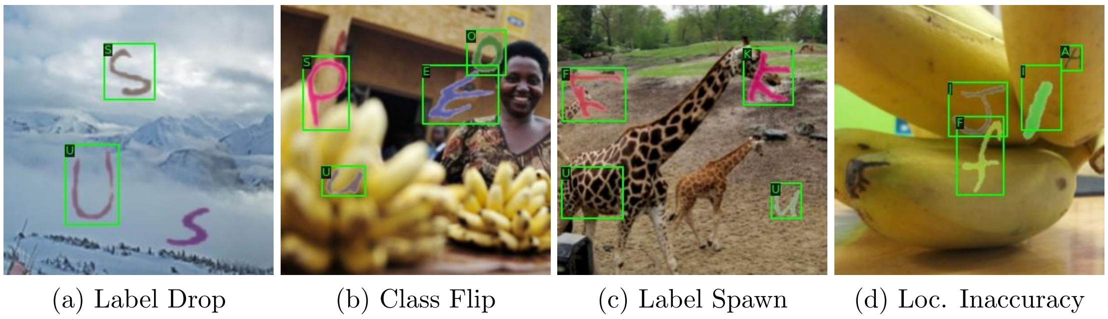
In the following, we distinguish between the label error benchmark and the detection of real label errors, where in the former, label errors are simulated (and therefore known) and the performance of the five different methods are evaluated automatically. In the latter case, label errors are not simulated but real and therefore automated evaluation is impossible as the real label errors are unknown. To enable a reliable evaluation, only datasets containing almost no real label errors are used for the benchmark. We observe that commonly used datasets in object detection, such as MS-COCO, Pascal VOC or KITTI, contain significant amounts of label errors, thus they are not suitable for the benchmark. Nevertheless, to demonstrate the performance of our instance-wise loss method on these datasets, a moderate sample size of 200 label error proposals are manually reviewed and counted for each dataset.

Datasets

For our benchmark, we use the semi-synthetic EMNIST-Det dataset and BDD100k, 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 data is split into equally-sized test and validation sets, each consisting of 882 images.

Simulated 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 is equipped with a set of G labels, see (2.55) and (2.56). Let $\mathcal{I} = \{1, \dots, G\}$ be the set of indices of all boxes $b^i \in \mathcal{Y}$, $i = 1, \dots, G$. We now describe all types of label errors applied to $\mathcal{Y}$, 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.



(a) Label Drop (b) Class Flip (c) Label Spawn (d) Loc. Inaccuracy

Figure 6.2: Examples of the different types of simulated label errors. The images are from the EMNIST-Det test dataset (Chapter 5).

Drops For dropping labels, we randomly choose a subset $\mathcal{I}_d$ of $\mathcal{I}$ with cardinality $|\mathcal{I}_d| = \lfloor \frac{\gamma}{4} \cdot G \rfloor$. We drop all labels $\mathcal{Y}_d = \{b^i : i \in \mathcal{I}_d\}$ and denote $\mathcal{I}_{\setminus d} = \mathcal{I} \setminus \mathcal{I}_d$. Analogously, $\mathcal{Y}_{\setminus d} = \mathcal{Y} \setminus \mathcal{Y}_d$.

Flips For flipping class labels, we randomly choose a subset $\mathcal{I}_f$ of $\mathcal{I}_{\setminus d}$ with cardinality $|\mathcal{I}_f| = \lfloor \frac{\gamma}{4} \cdot G \rfloor$ and copy $\tilde{\mathcal{Y}}_f = \mathcal{Y}_f = \{b^i : i \in \mathcal{I}_f\}$. Then, we randomly flip the class of every label in $\tilde{\mathcal{Y}}_f$ to a different label. We denote $\mathcal{I}_{\setminus f} = \mathcal{I}_{\setminus d} \setminus \mathcal{I}_f$ and $\mathcal{Y}_{\setminus f} = (\mathcal{Y}_{\setminus d} \setminus \mathcal{Y}_f) \cup \tilde{\mathcal{Y}}_f$.

Shifts To insert *shifts*, we change the localization of labels. We randomly choose a subset $\mathcal{I}_{sh}$ of $\mathcal{I}_{\setminus f}$ with cardinality $|\mathcal{I}_{sh}| = \lfloor \frac{\gamma}{4} \cdot G \rfloor$ and copy $\tilde{\mathcal{Y}}_{sh} = \mathcal{Y}_{sh} = \{b^i : i \in \mathcal{I}_{sh}\}$. For the shift of a box $\tilde{b}^i \in \tilde{\mathcal{Y}}_{sh}$, the new values $\tilde{y}$, $\tilde{h}$ are determined analogously to $\tilde{x} = \mathcal{N}(x, 0.15 \cdot w)$ and $\tilde{w} = \mathcal{N}(w, 0.15 \cdot w)$ drawn from a normal distribution with itself as the expected value and $0.15 \cdot w$ 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 $b^i \in \mathcal{Y}_{sh}$ and $\tilde{b}^i \in \tilde{\mathcal{Y}}_{sh}$ is in the interval of $[0.4, 0.7]$, $\forall i = 1, ..., \lfloor \frac{\gamma}{4} \cdot G \rfloor$. We denote $\mathcal{I}_{\setminus sh} = \mathcal{I}_{\setminus f} \setminus \mathcal{I}_{sh}$ and $\mathcal{Y}_{\setminus sh} = (\mathcal{Y}_{\setminus f} \setminus \mathcal{Y}_{sh}) \cup \tilde{\mathcal{Y}}_{sh}$.

Spawns For spawning labels, we randomly choose a subset $\mathcal{I}_{sp}$ of $\mathcal{I}_{\setminus sh}$ with cardinality $|\mathcal{I}_{sp}| = \lfloor \frac{\gamma}{4} \cdot G \rfloor$ and copy $\tilde{\mathcal{Y}}_{sp} = \mathcal{Y}_{sp} = \{b^i : i \in \mathcal{I}_{sp}\}$. Then, we assign every label $\tilde{b}^i \in \tilde{\mathcal{Y}}_{sp}$ 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 or outside of an 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 $\tilde{\mathcal{Y}} = \mathcal{Y}_{\setminus sh} \cup \tilde{\mathcal{Y}}_{sp}$.

One example per label error type is shown in fig. 6.2. Note that the number of labels G is unchanged as the number of *drops* and *spawns* is the same.

Baseline Methods

The four baselines that we compare our instance-wise loss method with are based on a) inspecting the labels without the use of deep learning, b) the box-wise detection score c) the classification entropy (2.82) of the two-stage object detectors and d) the probability differential from [70].

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 $x \in \mathcal{X}$, a neural network predicts a fixed number N_0 of bounding boxes for the first stage $\mathcal{B}_a$, see (2.63). Then, we add the boxes 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 $\mathcal{Y}$ is assigned with a detection score of $\hat{s}_0 = 1$. After adding the labels to $\mathcal{B}_a$, only those N_1 boxes that remain after class-independent non-maximum suppression (NMS) and score thresholding on $\hat{s}_0$ with $s_\epsilon \geq 0$, get into the second stage $\mathcal{B} = \{(\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i, \hat{s}_0^i) : i = 1, \ldots, N_1\}$. After box refinement and classification as well as NMS on the detection score $\hat{s}_2^i$, N_2 label error proposals remain. Here, $\hat{s}_2^i$ is the detection score of the detection head and unlike $\hat{s}_0^i$, $\hat{s}_2^i$ represents not only the presence of an object, but also takes the class probabilities of the predicted object into account. The remaining N_2 label error proposals $\mathcal{B}_{\text{NMS}} = \{(\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i, \hat{s}_2^i, \hat{p}_1^i, \ldots, \hat{p}_C^i) : i = 1, \ldots, N_2\}$, are defined by the localization $(\hat{x}^i, \hat{y}^i, \hat{w}^i, \hat{h}^i)$, the detection score $\hat{s}_2^i$ and the class probabilities $\hat{p}_1^i, \ldots, \hat{p}_C^i$. The predicted class is given by $\hat{c}^i = \text{argmax}_{k=1,\ldots,C}\, \hat{p}_k^i$. Score thresholding is omitted here, or the score τ used for this is equal to 0, since $\tau > 10^{-4}$ 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 $x \in \mathcal{X}$ as described above, we get label error proposals for the whole dataset by $\bigcup_{x \in \mathcal{X}} \mathcal{B}_{\text{NMS}}$.

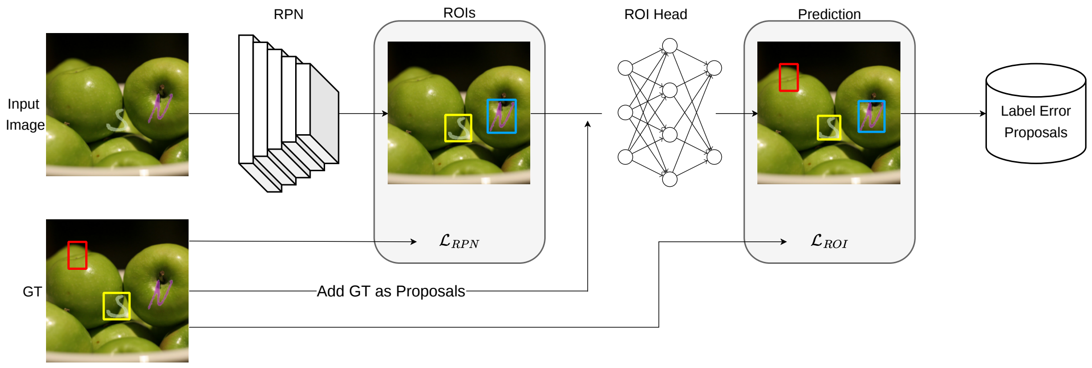


Figure 6.3: Visualization of our instance-wise loss method for detecting label errors. The red label indicates a *spawn*, the blue one a *drop* and the yellow one a correct label.

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 (2.82) rather than the detection score.

Probability Differential Baseline For the PD baseline from [70], we do not add the boxes of the labels as proposals. Furthermore, score thresholding and NMS is not applied, such that every box $\hat{b} \in \mathcal{B}_a$ also remains in $\mathcal{B}_{\text{NMS}}$. After assigning every label with sufficiently overlapping predictions, the probability differential (PD) for every label $b \in \mathcal{Y}$ with class c and the m assigned predictions $\hat{b}^i$ $(i = 1, \dots, m)$ is defined as:

$$PD(b) = \frac{\sum_{i=1}^{m} \left(1 + \max_{k \in \{1,\dots,C\} \setminus \{c\}} \hat{p}_k^i - \hat{p}_c^i\right)}{2m}. \tag{6.1}$$

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, since the probability differential is defined only for existing ground truth boxes and the *drops* are not part of the noisy labels.

Instance-wise Loss Method

Our method to detect the introduced label error types 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 $\hat{b}_0 \in \mathcal{B}_a$ is assigned with a region proposal loss ($\mathcal{L}_{RPN}$), which is

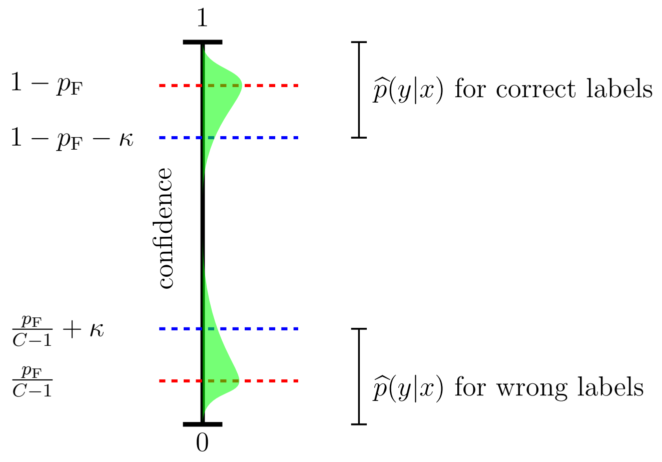

1
$1 - p_{\mathrm{F}}$
$1 - p_{\mathrm{F}} - \kappa$
confidence
$\frac{p_{\mathrm{F}}}{C-1} + \kappa$
$\frac{p_{\mathrm{F}}}{C-1}$
0
$\widehat{p}(y|x)$ for correct labels
$\widehat{p}(y|x)$ for wrong labels

Figure 6.4: 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.

the sum of a classification (binary cross-entropy; (2.64)) and regression (smooth-L_1; (2.67)) loss for the labels and the prediction itself. The computation of the loss is identical to the one in training. 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 $\hat{b}_1 \in \mathcal{B}$ is assigned with a region of interest loss ($\mathcal{L}_{ROI}$), which is the sum of a classification (cross-entropy; (2.69)) 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. 6.3. 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.

Theoretical Justification Our goal is to show that the flip of a test label is statistically captured by the cross-entropy loss evaluated at a deep neural network's

(DNN[1]) prediction on a test sample x and the corresponding label y.

The rough intuition for this statement is that a probably approximate correct learner [147] (PAC-learner) $\widehat{p}$ has probabilistic bounds for having predictive distribution close to the data-generating distribution p. Therefore, sufficient data sampling and empirical loss minimization will lead to statistical concentration of confidences $\widehat{p}$ around p. If p 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 $(x, y) \sim p$ following some noisy data generating distribution p, where $x \sim P_x$ follows a marginal distribution P_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 (x, y), x follows the same marginal distribution $x \sim P_x$. Our proof builds on the existence of a true labeling function $f : x \mapsto y$ and the assumption that the data distribution p introduces stochastic flips of labels that occur with a fixed uniform rate $p_\mathrm{F} \in [0, 1)$. This flip probability p_F is uniformly distributed over all $C - 1$ incorrect classes which are not $f(x)$. Furthermore, p_F leads to the following constraints on p when conditioned to x:

$$p(f(x)|x) := 1 - p_\mathrm{F}, \qquad p(y|x) := p_\mathrm{F}/(C-1) \quad \forall y \neq f(x). \tag{6.2}$$

A statistical model $\widehat{p} = \widehat{p}(y|x)$ PAC-learns classification on samples of the (noisy) data generating distribution $p = p(y|x)$. In the present treatment, we assume PAC-learning with respect to the Kullback-Leibler (KL) divergence

$$D_\mathrm{KL}(p(\cdot|x)\|\widehat{p}(\cdot|x)) = -\int \log\left(\frac{\mathrm{d}\widehat{p}(y|x)}{\mathrm{d}p(y|x)}\right) p(y|x)\,\mathrm{d}y. \tag{6.3}$$

In the following our goal is to show probabilistic statements about the cross-entropy loss

$$\ell_\mathrm{CE}(\widehat{p}(x)\|y) := -\sum_{c=1}^{C} y_c \cdot \log(\widehat{p}_c(x)) \tag{6.4}$$

on test data pairs (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 for the proof. This assumption can be justified via

[1]Technically, it is not required that the model is a DNN as long as PAC-learnability is fulfilled.

the error decomposition of empirical risk minimization for the KL divergence over the hypothesis space $\mathscr{H}$ with training data $\{(x_1, y_1), \dots, (x_n, y_n)\}$:

$$\begin{aligned} \mathsf{D}(p\|\widehat{p}) :=& \mathbb{E}_{x\sim p_x}[D_{\mathrm{KL}}(p(\cdot|x)\|\widehat{p}(\cdot|x))] \\ \leq& \inf_{h\in\mathscr{H}} \mathsf{D}(p\|h) + \left(\frac{1}{n}\sum_{j=1}^{n} \ell_{\mathrm{CE}}(\widehat{p}(x_j)\|y_j) - \inf_{h\in\mathscr{H}} \ell_{\mathrm{CE}}(h(x_j)\|y_j)\right) \\ &+ 2\cdot\sup_{h\in\mathscr{H}}\left|\mathsf{D}(p\|h) - \frac{1}{n}\sum_{j=1}^{n}\ell_{\mathrm{CE}}(h(x_j)\|y_j) - H(p(\cdot|x))\right| < \varepsilon \end{aligned} \tag{6.5}$$

where $H = -\sum_{c=1}^{C} p(c|x)\cdot\log(p(c|x))$ is the entropy of the data generating distribution[2]. The first term is the model misspecification 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 w.r.t. 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 $\mathsf{D}(p\|h)$ in the true distribution. The third term can be controlled by application of concentration inequalities and chaining under certain assumptions (see [161]) which is why the sum of the three terms can be made smaller than some fixed $\varepsilon > 0$ given sufficient amount of data.

Proposition 1 (Statistical separation of the cross-entropy loss). *Let training and testing labels be given under a stochastic flip in $p(\cdot|x)$ with probability p_{F} as above, let the label distribution $p(\cdot|x)$ be PAC-learnable by the hypothesis space of $\widehat{p}(\cdot|x)$ w.r.t. 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*

$$\ell_{\mathrm{CE}}(\widehat{p}(x)\|f(x)) < -\log(1-p_{\mathrm{F}}-\kappa) < -\log(\kappa + \tfrac{p_{\mathrm{F}}}{C-1}) < \ell_{\mathrm{CE}}(\widehat{p}(x)\|\widetilde{y}) \tag{6.6}$$

for any incorrect label $\widetilde{y} \neq f(x)$ with probability $1-\delta$ over chosen training data and with probability $1-\frac{2\varepsilon}{\kappa^2}$ over the choice of x.

Proof. We aim at bounding $\max_{y=1,\dots,C} |p(y|x) - \widehat{p}(y|x)|$ by the total variation distance. PAC-learnability asserts that given enough data, the $\widehat{p}$-distributions illustrated in Fig. 6.4 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

$$\mathbb{E}_{x\sim p_x}[D_{\mathrm{KL}}(p(\cdot|x)\|\widehat{p}(\cdot|x))] < \varepsilon \tag{6.7}$$

[2]Together with the cross-entropy ℓ_{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 $\max_{y=1,\dots,C} |p(y|x) - \widehat{p}(y|x)|$ exceeding κ via the total variation distance. We have

$$\begin{aligned} P_x(\|p(\cdot|x) - \widehat{p}(\cdot|x)\|_{\mathrm{TV}} \geq \kappa) &\leq P_x(\sqrt{2D_{\mathrm{KL}}(p(\cdot|x)\|\widehat{p}(\cdot|x))} \geq \kappa) \\ &\leq P_x\left(D_{\mathrm{KL}}(p(\cdot|x)\|\widehat{p}(\cdot|x)) \geq \frac{\kappa^2}{2}\right) \\ &\leq \frac{2}{\kappa^2}\mathbb{E}_{x\sim p_x}[D_{\mathrm{KL}}(p(\cdot|x)\|\widehat{p}(\cdot|x))] < \frac{2\varepsilon}{\kappa^2} \end{aligned} \tag{6.8}$$

with probability $1-\delta$ over the choice of training data. Here, the first inequality is the application of Pinsker's inequality and the third due to the Markov inequality.

Assume that we are given a correct label y for x, then with probability $1-\delta$ over training data and with probability $1-\frac{2\varepsilon}{\kappa^2}$ over sampling x, we have that

$$|p(y|x) - \widehat{p}(y|x)| = |(1-p_{\mathrm{F}}) - \widehat{p}(y|x)| \leq \max_y |p(y|x) - \widehat{p}(y|x)| \tag{6.9}$$

$$\leq \|p(\cdot|x) - \widehat{p}(\cdot|x)\|_{\mathrm{TV}} < \kappa. \tag{6.10}$$

This implies $\widehat{p}(y|x) > 1 - p_{\mathrm{F}} - \kappa$ and therefore, by monotony of the logarithm $\ell_{\mathrm{CE}}(\widehat{p}(y|x)\|y) < -\log(1-p_{\mathrm{F}}-\kappa)$. Similarly, if y is any incorrect label, we have the probabilistic statement

$$|p(y|x) - \widehat{p}(y|x)| = \left|\frac{p_{\mathrm{F}}}{C-1} - \widehat{p}(y|x)\right| \leq \max_y |p(y|x) - \widehat{p}(y|x)| < \kappa, \tag{6.11}$$

i.e., $\widehat{p}(y|x) < \kappa + \frac{p_{\mathrm{F}}}{C-1}$ and we have $\ell_{\mathrm{CE}}(\widehat{p}(x)\|y) > -\log\left(\kappa + \frac{p_{\mathrm{F}}}{C-1}\right)$. Finally, we obtain separability of losses with true versus false labels in probability if

$$1 - p_{\mathrm{F}} - \kappa > \kappa + \frac{p_{\mathrm{F}}}{C-1} \iff p_{\mathrm{F}} < \frac{C-1}{C}(1-2\kappa). \tag{6.12}$$

□

Evaluation Metrics

Ignoring that a few 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 $\mathcal{Y}$ 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*

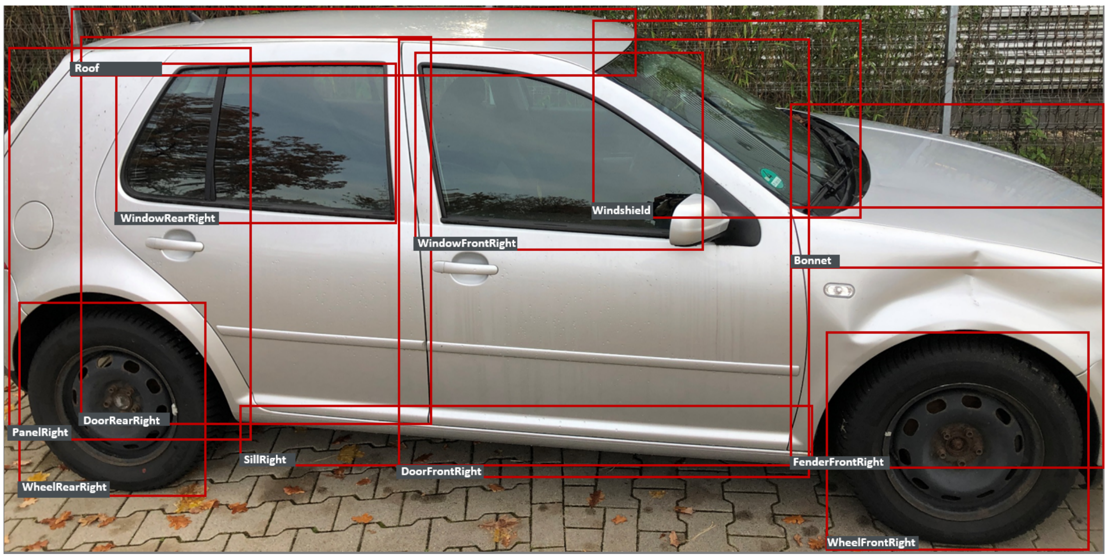


Figure 6.5: Example image from the CE test data with labels and a missing “MirrorRight”.

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, the area under the receiver operator characteristic curve ($AUROC$, see [30]) and F_1 values, which is the harmonic mean of precision and recall (see [32]), is 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.

Detection of Real Label Errors

For commonly used datasets we proceed as follows. We consider for each dataset 200 proposals of our method with 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 since 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.

Dataset	Backbone	mAP_{50}	$mAP_{50}^{(*)}$
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	

Table 6.1: Validation of object detection performance on our datasets. $^{(*)}$ indicates learning with simulated label errors ($\gamma = 0.2$).

6.4 Numerical Results

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 from ControlExpert (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. For the latter, we show how many real label errors we can detect among the top-200 proposals for each real-world dataset. For an exemplary image of the CE dataset, see fig. 6.5. 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”.

Implementation Details

We implemented our benchmark and methods in the open source MMDetection toolbox ([19]). Our models are based on a Swin-T transformer and a ResNeSt101 backbone, both with a CascadeRoIHead as the object detection head, with a total number of trainable parameters of approximately 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 $s_\epsilon = 0.25$, the value for score thresholding after the second stage $\tau = 0$ and the IoU-value $\alpha = 0.3$ from which a proposal for a 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/original labels and the

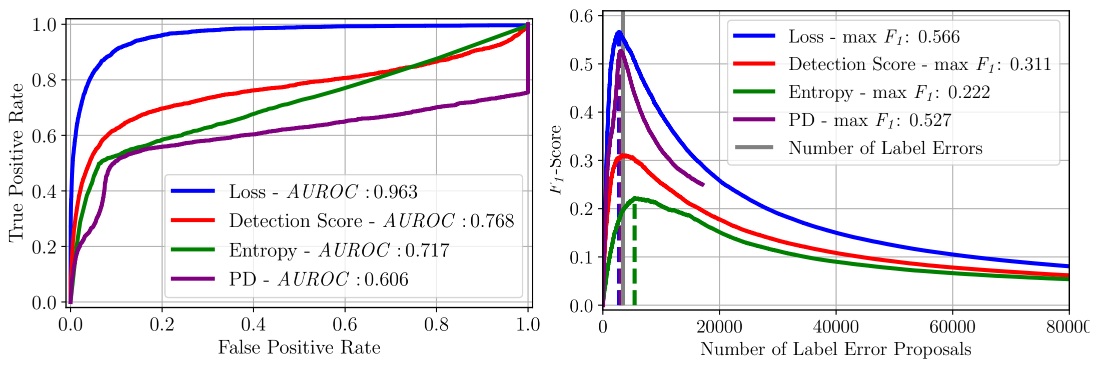
Dataset	Batch Size	Image Resolution	# Training Iterations	Learning Rate
EMNIST-Det	24	300×300	24,000/48,000$^{(*)}$	0.02
BDD	4	1333×800	150,000/250,000$^{(*)}$	0.01
KITTI	6	1000×600	70,000	0.01
COCO	12	1000×600	250,000	0.02
VOC	6	1000×600	70,000	0.02
CE	4	1000×600	200,000	0.01

Table 6.2: Training hyperparameters for the Swin-T and the ResNeSt101 ($^{(*)}$) backbone.

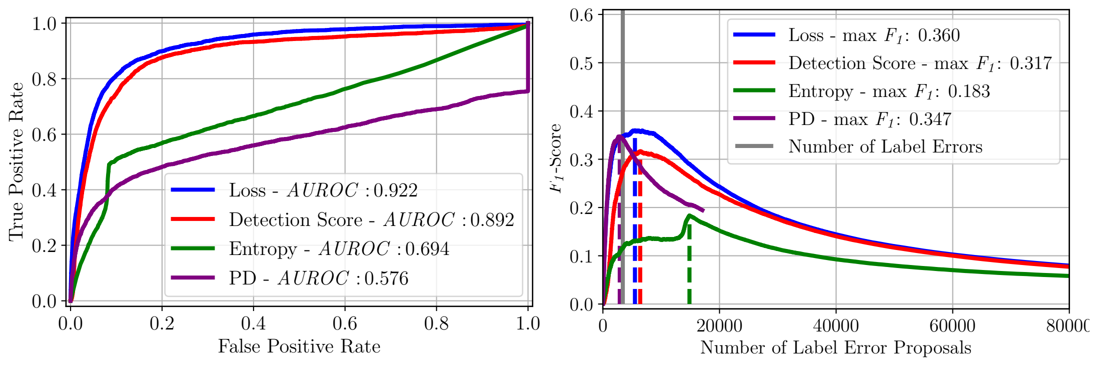
Swin-T backbone. The performance results obtained have all been evaluated on unmodified test datasets.

The dataset-dependent hyperparameters for training are stated in table 6.2. 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 24,000-250,000. 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 are published with the code on *GitHub*.

Datasets For the detection of real label errors we use the same split for BDD as introduced in Section 6.3 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. COCO is trained on the train split and label errors are predicted on the validation split from 2017. For KITTI we use a scene-wise split, resulting in 5 scenes ($S = \{2, 8, 10, 13, 17\}$) and 1,402 images for evaluation as well as 16 scenes ($\{0, 1, \ldots, 20\} \setminus S$) 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.



(a) Original training data



(b) Noisy training data with $\gamma = 0.2$

Figure 6.6: The two left plots in (a) show evaluations based on the predictions of a model trained on original training data and the two right ones in (b) based on noisy training data with $\gamma = 0.2$. The number of considered label error proposals depends on threshold τ.

Benchmark Results for Simulated Label Errors

Table 6.1 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 from [114]. 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 6.6 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 based on noisy training data. The ranking of the methods is not identical everywhere:

			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**	<u>73.24</u>	71.49	59.67	**95.54**	<u>64.74</u>	49.58	62.32
EMNIST-Det	Swin-T	Noisy	**99.40**	<u>82.44</u>	77.32	62.26	**93.43**	<u>62.37</u>	45.25	62.24
EMNIST-Det	ResNeSt101	Original	**99.84**	<u>88.45</u>	86.70	60.59	**94.31**	<u>62.56</u>	38.81	60.82
EMNIST-Det	ResNeSt101	Noisy	**99.87**	<u>93.11</u>	86.40	61.82	**90.74**	<u>59.50</u>	34.53	59.01
BDD	Swin-T	Original	**96.30**	<u>76.82</u>	71.73	60.59	**56.59**	31.14	22.21	<u>52.66</u>
BDD	Swin-T	Noisy	**92.16**	<u>89.21</u>	69.42	57.58	**35.97**	31.68	18.33	<u>34.72</u>
BDD	ResNeSt101	Original	**95.79**	<u>87.47</u>	83.58	60.31	**54.62**	31.99	20.37	<u>47.16</u>
BDD	ResNeSt101	Noisy	**92.97**	<u>90.76</u>	78.18	56.79	**27.85**	25.65	18.10	<u>27.74</u>

Table 6.3: 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.

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 inferior compared to the loss. Because *AUROC* considers rates and (max) F_1 considers absolute values and the number of label error proposals varies widely (PD = number of labels G, here 17,064; others $>$ 80,000), the methods behave very differently with respect to *AUROC* and max F_1. However, our 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 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 6.3. When comparing the results for the different backbones with each other the *AUROC* for the loss and PD seems to remain similar, whereas the *AUROC* for detection score/entropy increases by 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 superior 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 more effectively, 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 increases by 0.85 pp 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 outperforms the detection score and both outperform 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 *AUROC* values for the detection score and entropy are superior with the ResNeSt101 backbone, but inferior in terms of max F_1 and the detection score performs superior in terms of *AUROC* based on noisy training data, but inferior in terms of max F_1. For

				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**

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.

PD, the *AUROC* seems to be rather stable comparing the two backbones, but the max F_1 is superior for Swin-T compared to ResNeSt101.

Benchmark Results for Individual Simulated Label Error Types

In the above-analyzed 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 $\tilde{\mathcal{Y}}$. For the *spawns*, we must consider both true positives and false positives according to $\tilde{\mathcal{Y}}$, 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 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 inferior 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 $< 14\%$. 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.

Benchmark Results for Different Noise Intensity in 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.

			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**	76.82	71.73	60.59	**56.59**	31.14	22.21	52.66
0.05	12,454	50.4	**93.44**	88.09	71.76	59.16	**43.36**	30.78	18.54	42.98
0.1	12,454	50.2	**93.21**	89.05	70.98	58.56	**39.36**	30.79	18.53	37.92
0.2	12,454	50.3	**92.61**	89.21	69.42	57.58	**35.97**	31.68	18.33	34.72

Table 6.5: Validation of object detection performance and label error detection experiments for different noise 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

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.

On the one hand, the *AUROC*/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 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 6.3.

Benchmark Results for Different Amounts 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

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

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*.

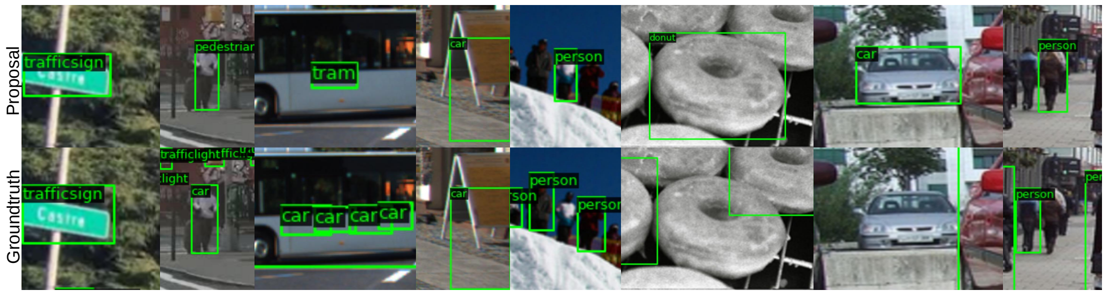


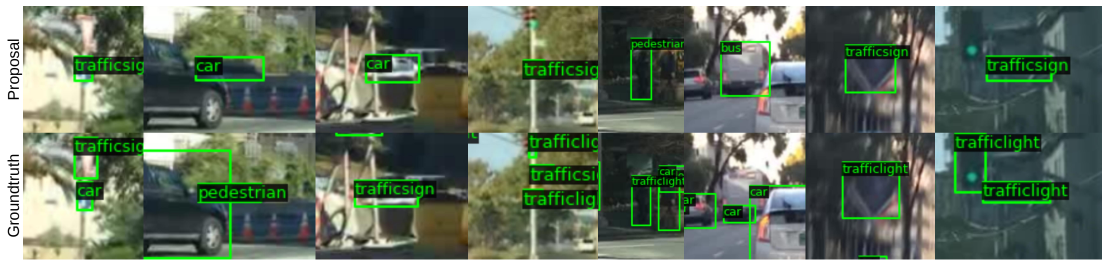
Figure 6.7: Visualization of detected label errors in real test datasets. The top row of images depicts the label error proposals and the bottom row the corresponding labels from the dataset. The image pairs belong from left to right in steps of two to BDD, KITTI, COCO and VOC.

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.

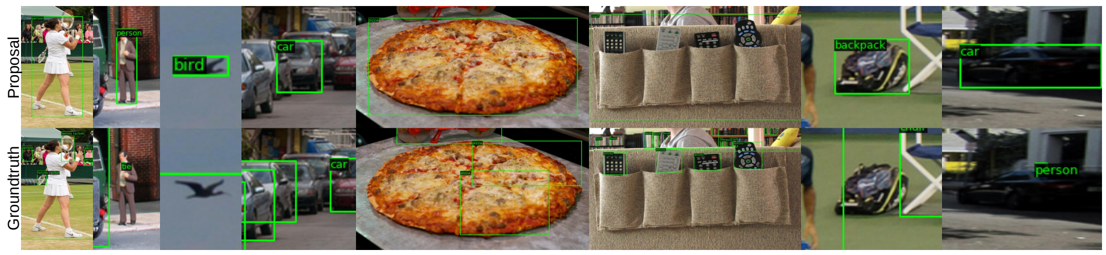
The $AUROC$ values increase as the number of images increases with $\gamma = 0$. With $\gamma = 0.2$, the values for loss and entropy decrease as the number of images increases. The max F_1 values decrease independently of γ with increasing number of images for loss and entropy, whereas the values increase 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 $AUROC$/max F_1 evaluations are based on the test data with $\gamma = 0.2$ and thus on the identical label basis as for table 6.3.

Evaluation for Real Label Errors

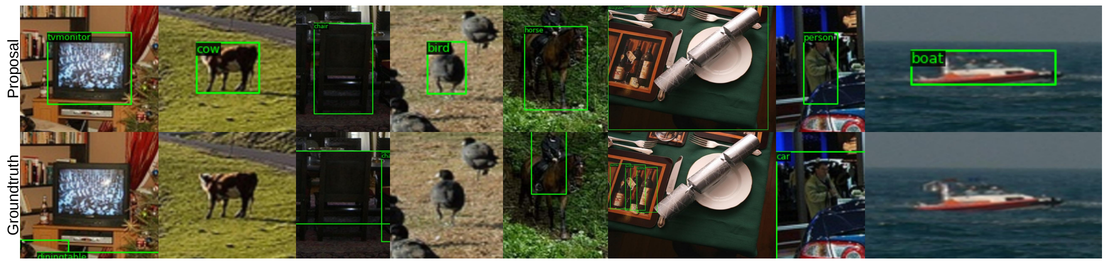
We now aim at detecting real instead of simulated label errors. The considered real-world datasets apart from BDD (VOC, COCO, KITTI, CE) are more similar in complexity to BDD than to EMNIST-Det. For BDD we observed in the benchmark results that the loss method for the Swin-T backbone seems to be more stable according to label errors in the training data, as especially the max F_1 values for the loss and noisy labels are superior 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 6.3 that the loss method outperforms the detection score, entropy and PD in each presented experiment, hence we detect label errors using only the loss method in the following. 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. 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 precisions 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. 6.7. The first proposal detects a *shift*, the



(a) BDD



(b) COCO



(c) VOC

Figure 6.8: Visualization of further detected real label errors in test datasets for BDD (top), COCO (center) and VOC (below).

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. 6.8. The top row shows examples for BDD, where all found label errors are *flips*, except for the third proposal from the right. These proposals 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”, the two small labels “pizza” are also count as *spawns* resulting in three label errors for this single proposal.

6.5 Conclusion

In this chapter, we 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 fewer parameters are more robust to label errors in training sets while models with more parameters suffer more. We make our code for benchmark, evaluation and method publicly available on *GitHub*.

CHAPTER 7

DEEP ACTIVE LEARNING WITH NOISY ORACLE IN OBJECT DETECTION

The presented contents in the following chapter are taken almost word-by-word from [143].

7.1 Introduction

In the previous decade, deep learning has revolutionized computer vision models across many different tasks like supervised object detection [15, 39, 123]. Object detection has various potential real-world applications, many of which have not yet been developed in a sense that public datasets are rarely or just not available. When such a new field is to be developed, there are many practical challenges during dataset curation and creation. Oftentimes, data can be recorded with, e.g., cameras in large amount at acceptable cost, while acquiring corresponding labels might be comparatively costly and might require expert knowledge. Active learning aims at maintaining model performance while reducing the amount of training data by leveraging data informativeness for the label acquisition. The model is utilized in turn to find the data, in our case from a large pool of unlabeled data, for which new labels would improve the model performance most efficiently, see [10, 145] and Chapter 5. The goal is to request as few annotations with human labor as possible and to obtain a well-performing model that makes accurate predictions. When developing and simulating active learning models in a laboratory setup, one typically assumes an oracle that provides correct labels for queried data points [10]. However, in practice, such an oracle does not exist

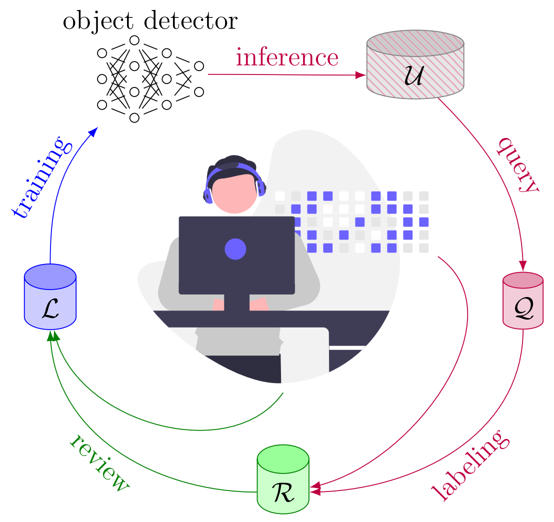


Figure 7.1: Our active learning cycle consists of training on labeled data $\mathcal{L}$, querying and labeling informative data points $\mathcal{Q}$ out of a pool of unlabeled data $\mathcal{U}$ by an oracle and a review of acquired data $\mathcal{R} = \mathcal{L} \cup \mathcal{Q}$.

and any person that labels data produces errors with some frequency [173]. Especially in complex domains such as medical applications where expert opinions are required for the annotation process, there exists variability between different oracles [139]. Some works have considered active learning with noisy oracles in image classification [55, 171, 180, 181]. In the present work, we consider active learning with a noisy oracle, to the best of our knowledge for the first time, in object detection. More precisely, we utilize recent findings on label errors in Chapter 6 to simulate two types of predominantly occurring label errors in object detection oracles. On the one hand we treat missed bounding box labels which do not appear in the ground truth at all. On the other hand, we consider bounding box labels with incorrect class assignment which are likely to induce undesired training feedback. We do so for the EMNIST-Det dataset (Chapter 5) which is an extension of EMNIST [26] to the object detection and instance segmentation setting. We complement this with the BDD100k dataset [182] which has mostly clean bounding box annotations of variable size. Both datasets have high quality bounding box labels such that we can simulate label errors without greater influence of naturally occurring label noise. We introduce independent and identically distributed errors into the labels which have been queried at some point in the data-acquisition process. We simulate a label reviewer as a human in the loop who has access to the label error detection module introduced in Chapter 6, which is integrated into the active learning cycle, see fig. 7.1. We compare different sources for label error proposals which are to be considered after data acquisition. Furthermore, we use different methods to generate label error proposals for the reviewer. The efficiency of the proposal method controls the frequency of justified review cases, i.e., the efficiency of the budget utilization for label reviewing. The review oracle is assumed to contain smaller amounts of noise since labels do not have to be generated from scratch. Instead, only individual proposals have to be reviewed.

In our experiments, we observe that a label error detection method applied to active learning with a noisy oracle clearly outperforms active learning with manual (uninformed / random) label review and active learning without any label review. We compare different query strategies with and without review in terms of performance as a function of annotation budget (split into labeling and reviewing cost). Improvement of performance is observed consistently for random queries as well as for an uncertainty query based on the entropy of the object detector's softmax output. Furthermore, our findings are consistent over two datasets, i.e., an artificial one and a real world one, as well as across two different object detectors. The success of our method seems to be due to a strong performance of the label error detection method.

Our contribution can be summarized as follows:

- We contribute the first method that performs partially automated label review and active label selection for object detection.
- We provide an environment for performing rapid prototyping of methods for active learning with noisy oracles.
- Our method outperforms manual and review-free active learning for different queries, datasets and underlying object detectors.

Our method can be used with humans in the loop for labeling and label review to maximize model performance at minimal annotation budget, thus aiding data acquisition pipelines with partial automation.

7.2 Related Work

Our contribution is located at the intersection of two disciplines which both aim at reducing the tiresome workload of repetitive image annotation by human workers. Active learning aims at reducing the overall amount of annotations given while the goal of label reviewing is to control or improve the quality of present annotations.

For an overview of related work for label error detection in object detection, see Section 6.2, and for active learning in object detection, see Section 5.2.

Active Learning with Noisy Oracle in Classification The intersection between active learning and training under label noise has been addressed in the context of classification tasks before. While Kim [80] used an active query mechanism for cleaning up labels, the proposed training algorithm itself is not active. Younesian et al. [180] consider noisy binary and categorical oracles by assigning different label costs to both in an online, stream-based active learning setting. Yan et al. [171] treat the query complexity of noisy oracles with a reject option in a theoretical manner. Gupta et al. [55] consider batch-based active learning with noisy oracles under the introduction of the QActor framework by Younesian et al. [181] which has a label cleaning module in its active learning cycle is most related to our approach. One of the proposed quality models chooses examples to clean via the cross-entropy loss which are then re-labeled by the oracle.

7.3 Active Learning with Noisy Oracle

In this section, we describe the task of active learning in object detection as well as the addition of a review module to the generic active learning cycle. While in the active learning setting, new labels are queried on the basis of an informativeness measure, the presence of erroneous or misleading oracle responses can counteract the benefit of the informed data selection. In order to account for new data containing incorrect labels, we introduce a review module that generates proposals for label errors to review and to potentially correct.

Active Learning with Review in Object Detection

Most of the commonly used datasets in object detection, e.g., MS-COCO [97] and Pascal VOC [38], are also the most commonly used datasets in active learning and contain label errors [132], see Chapter 6. This means that active learning methods developed on these datasets are also evaluated based on noisy labels. To consider label errors during active learning experiments, we introduce a review module. The adapted active learning cycle is visualized in fig. 7.1 where the annotation budget is divided into the query and the review with parameter $\lambda \in [0, 1]$. After obtaining labels for the queried images $\mathcal{Q}$ by an oracle, we introduce a review step where an oracle reviews the active labels $\mathcal{R} = \mathcal{L} \cup \mathcal{Q}$. Note, that acquisition and review of data are two independent modules.

Queries In the following, we investigate two different query strategies: random selection and selection based on the entropy of the softmax output of the object detector, see Section 5.3. For the former, images are randomly chosen from $\mathcal{U}$. For

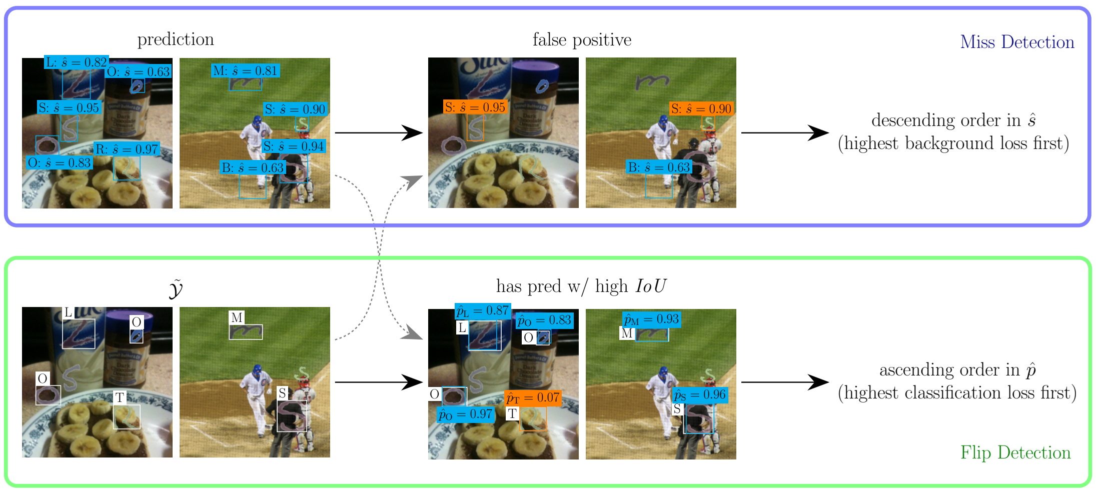


Figure 7.2: Schematic illustration of the label error detection mechanism using highest loss for missed labels (top, blue) and label flips (bottom, green). The detection of misses considers false positive predictions of the object detector while the detection of flips considers ground truth boxes, each of them matching at least one of the predictions in localization.

the latter, images are selected based on the predictive classification uncertainty according to the current model, where we use the sum of instance-wise entropies as aggregation method. Note, that both queries are independent of the label errors, since the random selection is independent of predictions and labels, as well as the image-wise query score for the entropy method is determined based on the predictions only. Note, that query algorithms are based on unlabeled data and oracle noise does not directly impact the selection of images. However, oracle noise does influence model training.

Review Module In order to account for noisy oracles, we allow for incorrect annotations given in response to an active learning query. To counter-act the negative influence of noisy annotations, we introduce a review module into the active learning cycle in which proposals for label reviews are given and part of the annotation budget is used to clean up some annotation errors. In the following, we introduce the detection of two different kinds of label errors: missing labels (*misses*) and labels with incorrect class assignments (*flips*).

For one active learning cycle, we allow for the consumption of a fixed annotation budget $\mathcal{C}$. This budget $\mathcal{C}$ is split up into a fraction $\mathcal{C}_\mathcal{Q} = (1-\lambda)\mathcal{C}$ used for querying new annotations and $\mathcal{C}_\mathcal{R} = \lambda\mathcal{C}$ used for reviewing data.

After the query, $\mathcal{Q}$ is automatically labeled and, together with $\mathcal{L}$, forms the set of active images for the next cycle. Before the next training cycle starts, we regard $\mathcal{R} = \mathcal{L} \cup \mathcal{Q}$ as the set of annotations which are potentially reviewed.

We adapt the instance-wise loss method from Chapter 6 to a post-processing label error detection method, where the detection of *misses* and *flips* are two independent tasks. When both types of label errors are simultaneously present, we use a parameter $\alpha \in [0,1]$ to distribute the review budget $\mathcal{C}_{\mathcal{R}}$ in reviewing *misses* ($\alpha\mathcal{C}_{\mathcal{R}}$) and *flips* ($(1-\alpha)\mathcal{C}_{\mathcal{R}}$). In the following, we introduce two different review functions: a random review and the highest loss based review.

An illustration of our label error detection method can be found in fig. 7.2. We consider the set of all predictions on images from $\mathcal{R}$ and the corresponding noisy labels $\tilde{\mathcal{Y}}$. To detect *misses*, we select those predictions that are identified as false positive predictions according to the noisy ground truth $\tilde{\mathcal{Y}}$. To get an order for the review, we sort the false positives in descending order by the objectness score $\hat{s}$ for the highest loss based review and in random order for the random review. Large values of $\hat{s}$ on false positives, where objectness is supposed to be small, amounts to a large objectness loss.

For the *flips*, every label from $\tilde{\mathcal{Y}}$ is assigned to the most overlapping prediction if the IoU of the two boxes is greater than of equal to IoU_ϵ. Then, the cross-entropy loss of the possibly incorrect label and the predicted probability distribution is used as a review score for every given label. In case there is no sufficiently overlapping prediction, the respective label is not considered for review. Note, that for given label class $\tilde{c}$, the cross-entropy loss of the assigned prediction $\hat{b}$ is

$$\mathrm{CE}(\hat{b}|\tilde{c}) = -\sum_{c=1}^{C} \delta_{c\tilde{c}} \log\left(\hat{p}_c\right) = -\log\left(\hat{p}_{\tilde{c}}\right), \tag{7.1}$$

where δ_{ij} is the Kronecker symbol, i.e., $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise. That is, the label with assigned prediction that has the lowest corresponding class probability $\hat{p}_{\tilde{c}}$ generates the highest loss. In case of the random review method, we randomly select assigned labels for review by uniformly sampling over all labels.

7.4 Numerical Experiments

In this section, we present our active learning setup with automatically labeling and reviewing data as well as new active learning hyperparameters. Afterwards, we show results for both query functions with and without review for two different datasets and object detectors in terms of mAP. We also measure the performance of the review proposal mechanism in terms of precision.

Experimental Setup

For our active learning setup, automated labeling and reviewing is desirable. Therefore, we simulate label errors for all training images of the underlying datasets. We do not include evaluations of the active learning experiments on the widely used MS-COCO or Pascal VOC datasets. For an automated review procedure, the frequently occurring label errors in both datasets would lead to strongly biased results. Evaluations on either dataset would require manual annotation review after each active learning cycle for several repetitions of the same experiment. This manual review after each cycle is necessary in practice, however, out of scope for an experimental evaluation of the proposed method.

Datasets and Models We make use of the EMNIST-Det dataset (Chapter 5) with 20,000 training images and 2,000 test images as well as BDD100k [182] (BDD), where we filter the training and validation split, such that we only use daytime images with clear weather conditions, resulting in 12,454 training images. Furthermore, the validation set is split into equally-sized test and validation sets, each consisting of 882 images. Since EMNIST-Det is a simpler task to learn compared to BDD, we apply a RetinaNet [96] and a Faster R-CNN [123] with a ResNet-18 [63] backbone for EMNIST-Det, as well as a Faster R-CNN with a ResNet-101 backbone for BDD. Note, that this setup was introduced and used in Chapter 5 and Chapter 6.

Based on clean training data, the test performance for EMNIST-Det in terms of mAP is 91.2% for Faster R-CNN and 90.9% for RetinaNet. For Faster R-CNN, the test performance decreases to 90.2% with simulated *misses* in the training data and to 89.2% with simulated *flips*. Simulating *misses* and *flips* simultaneously yields a test performance of 89.4% for Faster R-CNN and 89.3% for RetinaNet. For BDD and Faster R-CNN, a test performance of 50.0% is obtained for unmodified training data and 48.9% for training data including *misses* and *drops*.

Simulation of Label Errors For the simulation of *misses* and *flips*, we follow Chapter 6. Since only two types of label errors are present, we simulate *misses* and *flips* with cardinality $\frac{\gamma}{2} \cdot G$ each, where G is the amount of clean labels and γ is the relative frequency of label errors.

We denote the training set including label errors by

$$\tilde{\mathcal{Y}} = \{(x^i, y^i, w^i, h^i, \tilde{c}^i)\}, \tag{7.2}$$

where $\tilde{c}^i$ represents the potentially flipped class for each label b^i.

		Active Learning				Review				Training	
		$\|\mathcal{U}_{init}\|$	$\mathcal{C}$	s_ϵ	T	γ_l	γ_r	IoU_ϵ	α	batch size	training iters
Faster R-CNN	EMNIST-Det	150	200	0.7	20	0.2	0.05	0.3	0.5	4	25,000
RetinaNet	EMNIST-Det	150	200	0.5	20	0.2	0.05	0.3	0.5	4	38,000
Faster R-CNN	BDD100k	625	10,000	0.7	7	0.2	0.05	0.3	0.5	4	170,000

Table 7.1: Overview of important training, review and active learning hyperparameters for all datasets and networks.

Note, that label errors are not simulated on the test dataset to ensure an unbiased evaluation of test performance.

Automated Review of Label Errors Since the oracle is noisy with error frequency γ_r, the review is also error-prone, i.e., *misses* are detected with probability $1 - \gamma_r$ and still overlooked with probability γ_r. The *flips*, whether the label error proposal was a false alarm or not, are corrected with probability $1 - \gamma_r$ and randomly misclassified with probability γ_r.

Implementation Details We implemented our active learning methods in the open source MMDetection toolbox [19]. For the label error simulation, we choose the relative frequency of label errors $\gamma_l = 0.2$, the relative frequency of label errors during review $\gamma_r = 0.05$, the value for score-thresholding $s_\epsilon = 0.7$ for Faster R-CNN and $s_\epsilon = 0.5$ for RetinaNet as well as the IoU-value $IoU_\epsilon = 0.3$ that assigns predictions with labels. We choose $\gamma_r < \gamma_l$, assuming that the oracle is more engaged in viewing and evaluating single boxes during the review compared to labeling from scratch, with all boxes having to be located and classified on a new image. As hyperparameters for the active learning cycle, the initially labeled set consists of 150 randomly picked images EMNIST-Det and of 625 randomly picked images for BDD. The budget for a single active learning step $\mathcal{C}$ is 200 for EMNIST-Det and $\mathcal{C} = 10{,}000$ for BDD. Labeling a single box has cost 1, as does reviewing a label error proposal, whether *miss* or *flip* and also whether a label error was identified or not. If *misses* and *flips* are simultaneously present in the experiment, we set the ratio between reviewing *misses* and *flips* $\alpha = 0.5$. Finally, the number of active learning steps for EMNIST-Det is $T = 20$ and for BDD $T = 7$. For an overview of training, review and active learning hyperparameters, see table 7.1.

Results

In the following, we show active learning results for EMNIST-Det and BDD. Therefore, we compare six different methods, the random and entropy query,

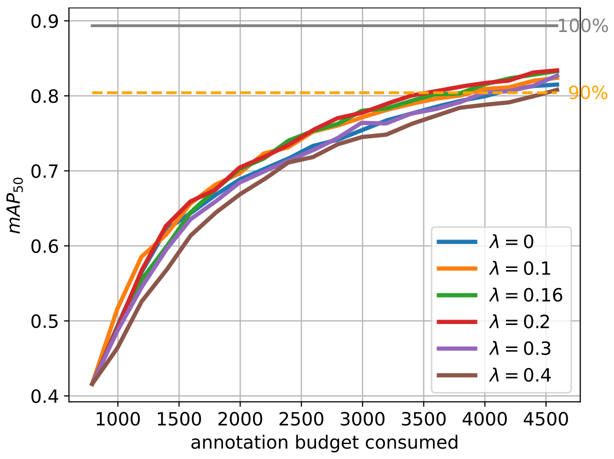
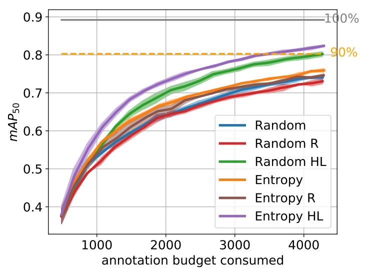


Figure 7.3: EMNIST-Det ablation study of the ratio between labeled and reviewed bounding boxes for Faster R-CNN where both label error types are present. We compare the random query without review with random query and highest loss review (RHL) with the chosen ration λ in parentheses.

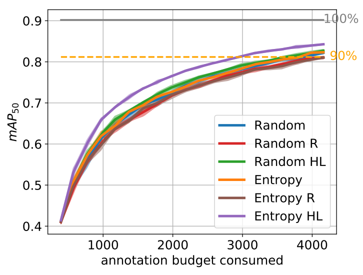
both without review, as well with random review or review by highest loss. Furthermore, we present performance results for both review methods in terms of precision over the whole active learning course.

Ablation for the Ratio of Queried and Reviewed Bounding Boxes For the methods with review, the fraction λ of the amount of new data queries and the amount of bounding box reviews plays a significant role. Therefore, we repeat the same experiment for EMNIST-Det and Faster R-CNN with different values for λ, see fig. 7.3. These results are based on training data with simulated *flips* and *misses*. The gray and yellow lines indicate the 100% and 90% reference performance mark of the model trained with the entire (noisy) dataset. The random query with highest loss review (RHL) is visually almost identical for $\lambda = 0.1$, $\lambda = 0.16$ and $\lambda = 0.2$. All these three methods outperform the uninformed random query. The random query without review performs similar to the random query with highest loss review with $\lambda = 0.3$ and outperforms the informed random query for $\lambda = 0.4$, i.e., at about $\lambda = 0.3$ is the break-even-point, at which it is no longer worthwhile to review more bounding boxes instead of labeling new ones. We hypothesize that this tipping point is strongly dependent on our chosen setup with a relative frequency of label errors of $\gamma_l = 0.2$. Since the red curve seems to be most favorable, we set the fraction between queries and reviews to $\lambda = 0.2$ in all the following experiments.

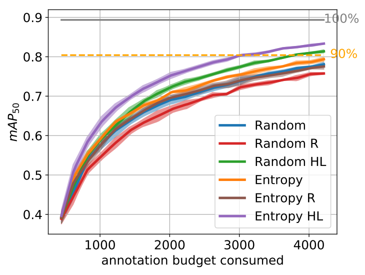
Active Learning with Different Label Error Types We first investigate active learning curves for EMNIST-Det and Faster R-CNN. We consider active learning



(a) Class Flips



(b) Label Misses



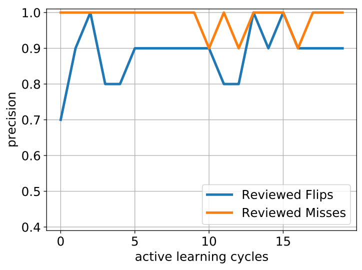
(c) Flips + Misses

Figure 7.4: EMNIST-Det active learning curves, where only flips are present (left), only misses (center) as well as where flips and misses are simultaneously present (right).

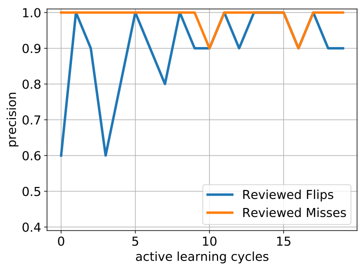
curves where a) we simulated only *misses*, b) only *flips*, and c) both label error types occur equally often in the training dataset, each with noise rate $\gamma_l = 0.1$.We compare both query strategies, random and entropy, without review, with highest loss (HL) review and with random (R) review, respectively. The obtained active learning curves are averaged over four random initializations and evaluated in terms of the total annotation budget consumed. Note, that for the active learning methods without review the total amount of annotation budget is equal to the amount of (possibly incorrect) labeled bounding boxes. For those methods incorporating a review step the total amount of annotation budget represents the sum of labeled and reviewed bounding boxes. Figure 7.4 shows active learning curves in terms of test performance with point-wise standard deviations. For all three active learning curves, we observe that the entropy method outperforms random at every point. Furthermore, the queries without review perform superior to the respective query with random review. Entropy HL and random HL, i.e., both methods with highest loss review clearly outperform the strategies without review and with random review. We conclude that the success of reviewing queries strongly depends on the performance of the review methods and that random review is too expensive in terms of annotation budget. From this we conclude that it is more worthwhile to acquire new (noisy) labels than to randomly review the active labels, at least for the given amount of noise we studied.

All in all, the distance between the active learning curves of the six methods is significantly larger for label *flips* as compared to *misses*. Moreover, the maximum performance with simulated label errors is also inferior for the *flips* compared to the *misses*. We hypothesize that the reason for this is the sub-sampling from the negatively associated anchors [123] during training. This mechanism leads to only partial learning from the *misses*, whereas an incorrect foreground class induced from *flips* has a negative impact on every gradient step.

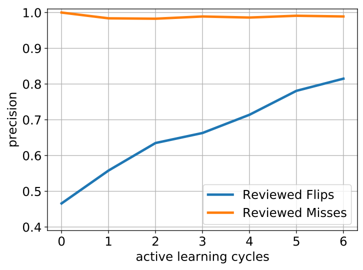
The significant difference of the active learning curves of the respective queries with random review and the highest loss review can be attributed to the high



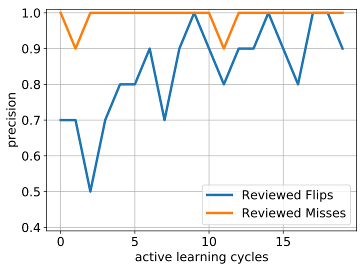
(a) FRCNN + EMNIST



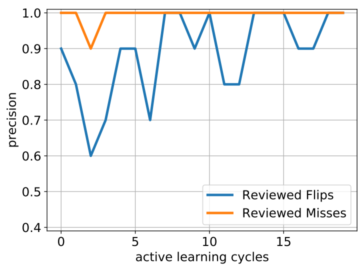
(b) RetinaNet + EMNIST



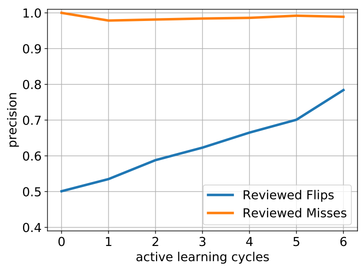
(c) FRCNN + BDD



(d) FRCNN + EMNIST



(e) RetinaNet + EMNIST

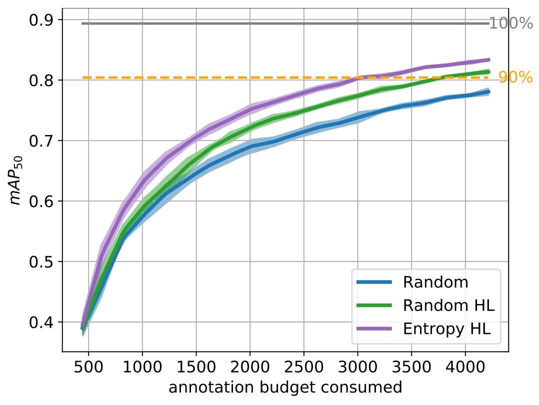


(f) FRCNN + BDD

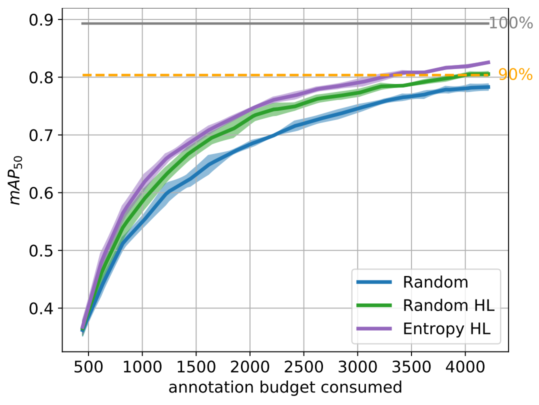
Figure 7.5: Review quality results for random highest loss (top) and entropy highest loss (bottom). Misses and flips are simultaneously present in all experiments.

precision of the highest loss review. The random review has an expected precision of γ_l. Figure 7.5 shows the precision for the highest loss review applied after random query in (a) and after entropy query in (d) across the span of all active learning cycles. In both plots, (a) and (d), *flips* and *misses* are simultaneously present, i.e., both plots correspond to the respective method from fig. 7.4 (c). The blue lines visualize the precision for the review identifying a *flip* and the orange lines analogously for the *misses*. Here, the precision for detecting *flips* is always above 50% and tends to improve as the active learning experiment progresses, whereas the precision for the detection of *misses* is even consistently above 90%. In general, *flips* are more difficult to detect compared to *misses* due to the different construction of the detection methods of either label error type, recall fig. 7.2.

Comparing Faster R-CNN with RetinaNet on EMNIST-Det For the following results, we compare only the random query without review with the random and entropy query, both with highest loss review. Figure 7.6 shows active learning curves for these methods for Faster R-CNN in (a) and for RetinaNet in (b). Note, that in both cases both label error types are present. Moreover, (a) is a trimmed version of fig. 7.4 (c) to make it more convenient to compare the results from both detectors visually. We observe that the curves for Faster R-CNN and RetinaNet



(a) Faster R-CNN



(b) RetinaNet

Figure 7.6: Comparison of EMNIST-Det active learning curves for Faster R-CNN (left) and RetinaNet (right) where both label error types are present.

Network	Method	$mAP_{@2000}$	$mAP_{@4000}$
Faster R-CNN	Random	68.97(±1.09)	77.41(±0.28)
	Random HL	72.13(±0.64)	80.92(±0.23)
	Entropy HL	75.13(±0.79)	82.87(±0.29)
RetinaNet	Random	68.38(±0.52)	78.04(±0.51)
	Random HL	72.94(±1.06)	80.40(±0.34)
	Entropy HL	74.39(±0.28)	81.86(±0.27)

Table 7.2: Mean average precision values in % with standard deviations in parentheses for 2000 and 4000 queried and reviewed annotations for Faster R-CNN (top) and RetinaNet (bottom) for EMNIST-Det. Note, that in every experiment both label error types are present; the upper half represents fig. 7.6 (a) and the bottom half fig. 7.6 (b).

look very similar over the entire active learning course. All curves start at just below 40% mAP and the respective methods end at similar test performances. The ranking of the methods is always the same: entropy HL outperforms random HL and random without review. Also, random HL outperforms random without review. For RetinaNet, random HL seems to be closer to entropy HL as compared to Faster R-CNN.

These observations are also supported by table 7.2, wherein we stated the mAP values with standard deviations in parentheses. We compare performance for the total annotation budget consumed equal to 2000 and 4000 from the active learning curves shown in fig. 7.6. In particular, for entropy with highest loss review, the $mAP_{@2000}$ for Faster R-CNN is 0.74 percent points (pp) higher and even 1.01 $mAP_{@4000}$ pp higher. For Faster R-CNN, the difference between entropy HL and random HL is 3 pp for $mAP_{@2000}$ and 1.95 for $mAP_{@4000}$. For RetinaNet, the difference is only 1.45 pp for $mAP_{@2000}$ and 1.46 for $mAP_{@4000}$.

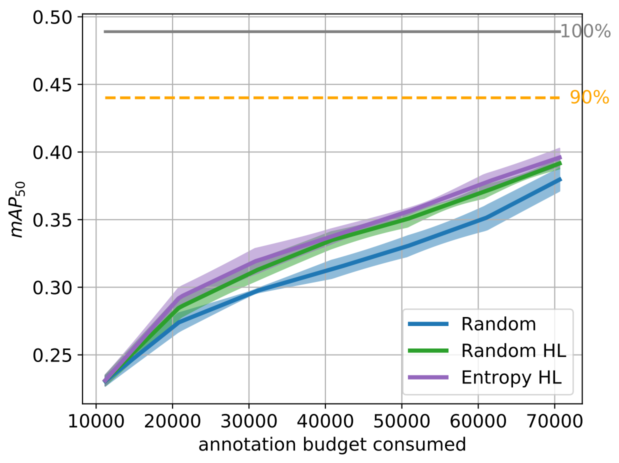


Figure 7.7: BDD active learning curves for Faster R-CNN where both label error types are present.

Comparing the quality of the highest loss review, the results for RetinaNet are highly correlated to the results of Faster R-CNN, see fig. 7.5. For RetinaNet, the precision for the highest loss review in combination with random query is visualized in (b) and with the entropy query in (e). Here, the precision for detecting the *misses* is at or above 90%. The precision for the detection of *flips* is always greater or equal to 60% and from active learning cycle 7 onward even always above 80%. We observe that the precision of the highest loss review increases while the experiments progresses, i.e., object detectors trained with more data generate better label error proposals. We hypothesize that with more data, overfitting can be more effectively prevented and that the object detectors will generalize better, thus label errors in the active labels will not be as significant when sufficient data is available.

Results for BDD with Faster R-CNN Figure 7.7 shows active learning curves for the random query without review, as well as for the random and entropy query both with highest loss review. Comparable to the results for EMNIST-Det, the ranking of the methods is identical over the entire active learning course. The uninformed random query is inferior to both informed queries and entropy HL is superior to random HL. Note, that the distance between the two queries with review is marginal. In contrast, there is a significant difference between either one and the random query without review.

The review quality of the highest loss review for the random query is shown in fig. 7.5 (c) and for the entropy query in (f). Again, the *misses* are detected at all times with a precision of nearly 100%. Starting at just under 50%, the precision for identifying *flips* increases steadily over the active learning course ending at close to 80%. We conclude that involving a label review in the active learning cycle is also highly beneficial in the more complex BDD real world dataset. Analogous to the results for EMNIST-Det, the highest loss review becomes more precise as the experiments progress and the number of active labels increases.

7.5 Conclusion

In this chapter, we considered label errors in active learning cycles for object detection for the first time, where we assumed a noisy oracle during the annotation process. We realized this assumption by simulating two types of label errors for the training data of datasets which are reasonably free of intrinsic label errors. These types of label errors are missing bounding box labels as well as bounding box labels with an incorrect class assignment. We introduce a review module to the active learning cycle, that takes as input the currently labeled images and the corresponding predictions of the most recently trained object detector. Furthermore, we detect both types of label errors by a random review method and a method based on the highest loss of the model's predictions and the corresponding noisy labels. We observe that the incorporation of random review leads to an even worse test performance compared to the corresponding query without review. Nevertheless, we show that the combination of query strategies, like random selection or instance-wise entropy, with an accurate review yields a significant performance increase. For both query strategies, the improvement obtained by including the highest loss review persists during the whole active learning course for different dataset-network-combinations. We make our code for reproducing results and further development publicly available at `GitHub`.

CHAPTER 8

LMD: LIGHT-WEIGHT PREDICTION QUALITY ESTIMATION FOR OBJECT DETECTION IN LIDAR POINT CLOUDS

The presented contents in the following chapter are taken almost word-by-word from [128].

8.1 Introduction

In recent years, deep learning has achieved great advances in the field of 3D object detection on Lidar data [89, 172, 174, 178]. Deep neural network (DNN) architectures for this task are well-developed, however, there is little work in the area of uncertainty quantification (UQ) for such models [18, 105, 106, 119, 175]. UQ is crucial for deployment of DNN-based object detection in the real world, since DNNs as statistical models statistically make erroneous predictions. Downstream 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 [18, 41, 42, 105, 106]. 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 dropout [18] or deep ensembles [175]. In such approaches, model sampling leads to a significant increase in inference time. Inspired by lines of research from Chapter 4 and [125] in the field of 2D object detection on camera

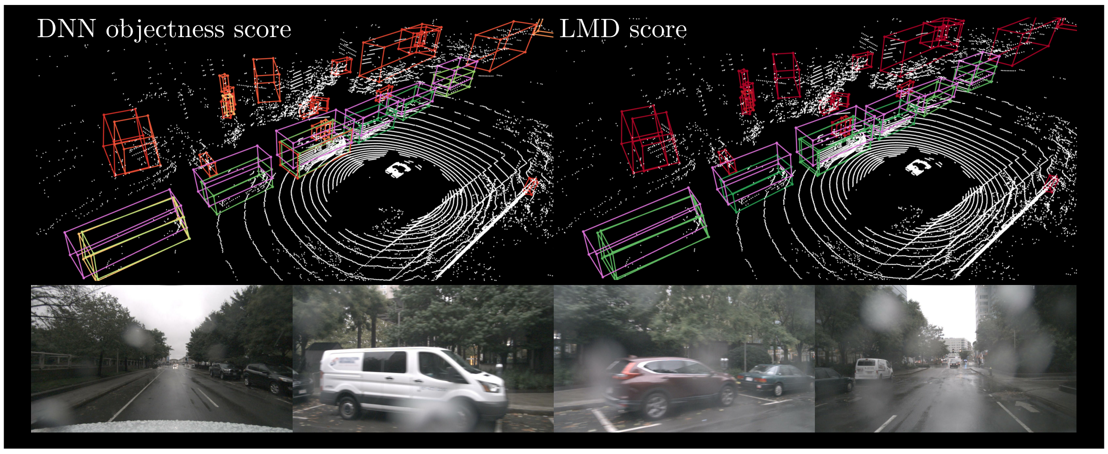


Figure 8.1: Prediction of a Lidar point cloud object detector with the native objectness score (*left*) and LMD meta classifier scores (*right*) and corresponding camera images below. Detections based on the objectness score are highly threshold-dependent and may lead to false positive detections. Detections based on LMD scores are more reliable and separate true from false predictions in a sharper way.

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. being a false positive; (2) meta regression, which estimates the localization quality of a prediction compared with the ground truth. Note that, outside of the context of UQ for DNNs, the terms meta classification and meta regression refer to different concepts, see [98] and [151], 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 [13] as well as a propriety dataset from Aptiv, 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 the label error detection capabilities of our method by reviewing 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.

We make our code publicly available on *GitHub*.

8.2 Related Work

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 Monte-Carlo dropout or deep ensembles. The authors of [105] estimate aleatoric uncertainty by a two-dimensional discretization scheme over the Lidar range and introducing a variance-weighted regression loss for a multi-modal distributional prediction in order to improve detection performance. In [106], aleatoric uncertainty is estimated by adding scale regression variables to the network output, modeling Laplace-distributed residuals under a label noise assumption via a Kullback-Leibler divergence loss. Heteroscedastic aleatoric uncertainty is estimated in [42] 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 Monte-Carlo dropout total variance for the epistemic component. Chen et al. [18] 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 Monte-Carlo dropout. Yang et al. [175] perform UQ for 3D object detection on Lidar and extend the multi-input multi-output model

MIMO [60] 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 Chapter 4 and [125] related with research on UQ in semantic segmentation [130, 131]. In Chapter 4, we utilize the pre-NMS anchor statistics in a post-processing approach to obtain box-wise confidence and IoU-estimates. In [125], instance-wise gradient scores are used 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 Monte-Carlo 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.

8.3 Proposed Method

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 $f(\cdot)$ which maps point clouds $\boldsymbol{X}$ to a list of N bounding boxes

$$f(\boldsymbol{X}) = \left\{\widehat{b}^1, \dots, \widehat{b}^N\right\}. \tag{8.1}$$

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 $\widehat{b}^j(\boldsymbol{X}) = (\widehat{x}^j, \widehat{y}^j, \widehat{z}^j, \widehat{\ell}^j, \widehat{w}^j, \widehat{h}^j, \widehat{\theta}^j, \widehat{s}^j, \widehat{\pi}_1^j, \dots, \widehat{\pi}_C^j)$. Here, $\widehat{x}^j, \widehat{y}^j, \widehat{z}^j, \widehat{\ell}^j, \widehat{w}^j, \widehat{h}^j, \widehat{\theta}^j$ 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 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 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

$$\mathrm{NMS}[f(\boldsymbol{X})] = \left\{\widehat{b}^i : i \in I_{\mathrm{NMS}}\right\}, \tag{8.2}$$

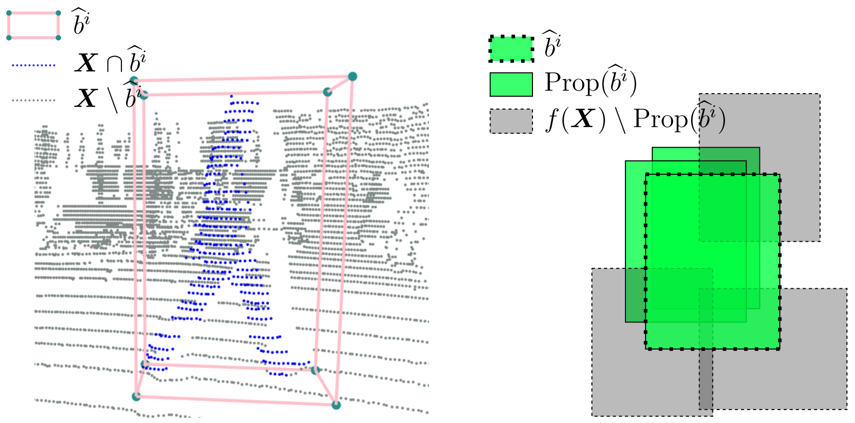

$\widehat{b}^i$
$\boldsymbol{X} \cap \widehat{b}^i$
$\boldsymbol{X} \setminus \widehat{b}^i$
$\widehat{b}^i$
$\mathrm{Prop}(\widehat{b}^i)$
$f(\boldsymbol{X}) \setminus \mathrm{Prop}(\widehat{b}^i)$

Figure 8.2: 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.

where we let $I_{\mathrm{NMS}} \subset \{1, \ldots, N\}$ denote the post-NMS index set indicating survivor boxes.

LMD Features From this information we generate geometrical and statistical features for each $\widehat{b}^i \in \mathrm{NMS}[f(\boldsymbol{X})]$ for the purpose of UQ. In addition to the bounding box features

$$\widehat{\phi}^i := \{\widehat{x}^i, \widehat{y}^i, \widehat{z}^i, \widehat{\ell}^i, \widehat{w}^i, \widehat{h}^i, \widehat{\theta}^i, \widehat{s}^i, \widehat{\kappa}^i\} \tag{8.3}$$

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* $\varPhi^i = P^i/|\boldsymbol{X}|$ in $\widehat{b}^i$, see fig. 8.2 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 $f(\boldsymbol{X})$ that are NMS-filtered by $\widehat{b}^i$, i.e., the pre-image

$$\mathrm{Prop}(\widehat{b}^i) := \mathrm{NMS}^{-1}[\{\widehat{b}^i\}]. \tag{8.4}$$

These are characterized by having a significant three-dimensional IoU_{3D} with $\widehat{b}^i$, see fig. 8.2 on the right.

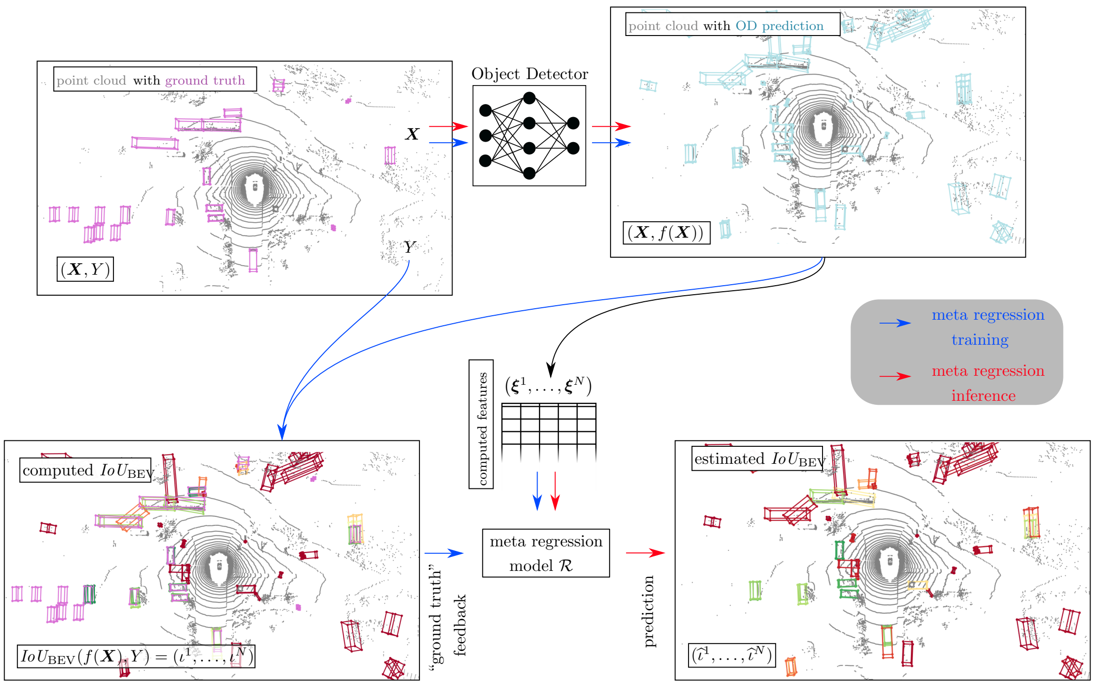

point cloud with ground truth
$(\boldsymbol{X}, Y)$
$\boldsymbol{X}$
Y
Object Detector
point cloud with OD prediction
$(\boldsymbol{X}, f(\boldsymbol{X}))$
meta regression training
meta regression inference
computed features
$(\xi^1, \dots, \xi^N)$
computed IoU_{BEV}
$IoU_{\mathrm{BEV}}(f(\boldsymbol{X}), Y) = (\iota^1, \dots, \iota^N)$
"ground truth" feedback
meta regression model $\mathcal{R}$
prediction
estimated IoU_{BEV}
$(\widehat{\iota}^1, \dots, \widehat{\iota}^N)$

Figure 8.3: 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 Y. Meta classification follows the same scheme with binary training targets $\tau^i = 1_{\{\iota^i > 0.5\}}$.

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

$$m^i \in \widehat{\phi}^i \cup \{V^i, A^i, F^i, P^i, \Phi^i, \rho^i_{\mathrm{max}}, \rho^i_{\mathrm{mean}}, \rho^i_{\mathrm{std}}\}, \tag{8.5}$$

as well, as the IoU_{3D} and bird-eye intersection over union IoU_{BEV} values between $\widehat{b}^i$ and all proposals $\mathrm{Prop}(\widehat{b}^i)$. Overall, this amounts to a vector $\boldsymbol{\xi}^i(\boldsymbol{X})$ of length $n = 90$ consisting of co-variables (features) on which post-processing models are fit in order to predict the IoU_{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 $IoU_{\mathrm{BEV}} \geq 0.5$, otherwise we declare it FP.

Post-Processing On an annotated hold-out dataset $\mathcal{D}_{\mathrm{val}}$ (consisting of point cloud-annotation tuples $(\boldsymbol{X}, Y)$), we compute a structured dataset

$$\mathsf{X} = (\boldsymbol{\xi}^1, \dots, \boldsymbol{\xi}^{N_{\mathrm{val}}}) \in \mathbb{R}^{n \times N_{\mathrm{val}}}$$

consisting of feature vectors for each of the N_{val} predicted boxes over all of $\mathcal{D}_{\text{val}}$. The illustration of our method in fig. 8.3 shows this scheme for one particular Lidar frame $(\boldsymbol{X}, Y)$ and the respective prediction on it. Further, we compute $\iota^i := IoU_{\text{BEV}}(\widehat{b}^i(\boldsymbol{X}), Y)$ between prediction and ground truth form $\mathcal{D}_{\text{val}}$ as target variables $\mathsf{Y} = (\iota^1, \ldots, \iota^{N_{\text{val}}}) \in \mathbb{R}^{N_{\text{val}}}$. We then fit a light-weight *(meta-) regression model* $\mathcal{R} : \boldsymbol{\xi}^i \mapsto \mathsf{Y}_i$ on (X, Y) which acts as post-processing module of the detector in order to produce IoU_{BEV}-estimates $\widehat{\iota}^i := \mathcal{R}(\boldsymbol{\xi}^i)$ for each detection $\widehat{b}^i$. Similarly, we fit a binary *(meta-) classification model* $\mathcal{C}$ obtaining the binary targets $1_{\{\mathsf{Y}>0.5\}}$ which allows us to generate alternative confidence estimates $\widehat{\tau}^i := \mathcal{C}(\boldsymbol{\xi}^i) \in [0, 1]$ 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 $\boldsymbol{\xi}^i$ and, therefore, can change the obtained confidence ranking per frame and influence detection performance as opposed to simple re-calibration methods [53, 110].

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 IoU_{BEV} and $\mathcal{C}$, we frequently find that $\mathcal{C}$ is to be trusted more than the computed IoU_{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.

8.4 Numerical Results

In this section we study meta classification and meta regression performance for two benchmark datasets as well as a proprietary dataset by Aptiv. The meta classification results are presented in terms of accuracy and area under receiver operator characteristics curve ($AUROC$ [30]) 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 detect annotation errors on both benchmark datasets using LMD.

Implementation Details We implemented our method in the MMDetection3D toolbox [109]. For our experiments, we consider the PointPillars [89] and CenterPoint [178] architectures. The mean average precision ($mAP@IoU_{0.5}$) for KITTI based on IoU_{BEV} is 69.0 for CenterPoint and 68.8 for PointPillars. On KITTI, the $mAP@IoU_{0.5}$ based on IoU_{3D} is 64.2 for CenterPoint and 68.8 for PointPillars and for Aptiv, the $mAP@IoU_{0.5}$ based on IoU_{3D} is 39.5 for CenterPoint and 43.7

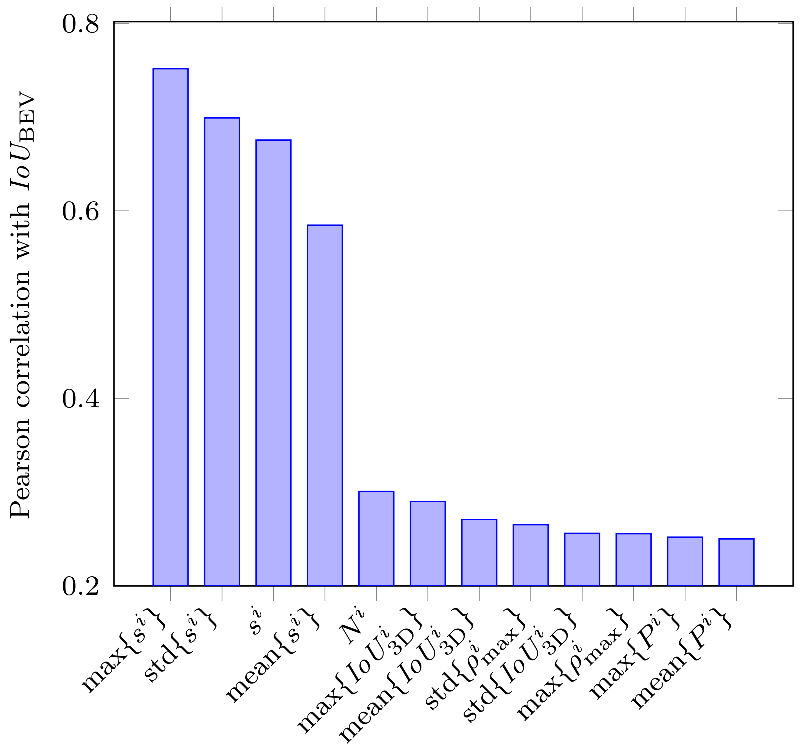


Figure 8.4: Strongest correlation coefficients for constructed box-wise features and IoU_{BEV} for the CenterPoint architecture on the nuScenes test dataset and a score threshold $\tau = 0.1$.

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. 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 Aptiv dataset consists of 50 sequences, split into $27, 14, 9$ sequences with about 145K, 75K, 65K cuboid annotations for training, validation and 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 (anonymous). 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.

$\max\{s^i\}$	0.8056	s^i	0.8050
$\mathrm{std}\{s^i\}$	0.7456	$\mathrm{mean}\{s^i\}$	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
$\mathrm{mean}\{IoU^i_{\mathrm{BEV}}\}$	0.5083	$\min\{IoU^i_{\mathrm{3D}}\}$	0.4984
$\max\{IoU^i_{\mathrm{3D}}\}$	0.4974	$\max\{P^i\}$	0.4593

$\max\{s^i\}$	0.8493	s^i	0.8490
$\mathrm{mean}\{s^i\}$	0.8309	$\mathrm{std}\{s^i\}$	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

Table 8.1: Strongest correlation coefficients for constructed box-wise features and IoU_{BEV} for the CenterPoint (left) and PointPillars (right) architecture on the KITTI test dataset and a score threshold $\tau = 0.1$.

$\max\{s^i\}$	0.7516	$\mathrm{std}\{s^i\}$	0.6991
s^i	0.6755	$\mathrm{mean}\{s^i\}$	0.5847
N^i	0.3007	$\max\{IoU^i_{\mathrm{3D}}\}$	0.2900
$\mathrm{mean}\{IoU^i_{\mathrm{3D}}\}$	0.2707	$\mathrm{std}\{\rho^i_{\max}\}$	0.2652
$\mathrm{std}\{IoU^i_{\mathrm{3D}}\}$	0.2560	$\max\{\rho^i_{\max}\}$	0.2556
$\max\{P^i\}$	0.2519	$\mathrm{mean}\{P^i\}$	0.2500

$\max\{s^i\}$	0.7554	$\mathrm{std}\{s^i\}$	0.7344
$\mathrm{mean}\{s^i\}$	0.7161	N^i	0.6629
s^i	0.6244	$\mathrm{std}\{w^i\}$	0.4535
$\mathrm{std}\{F^i\}$	0.4228	$\mathrm{std}\{y^i\}$	0.3879
$\mathrm{std}\{\ell^i\}$	0.3794	$\mathrm{std}\{IoU^i_{\mathrm{3D}}\}$	0.3689
$\mathrm{std}\{IoU^i_{\mathrm{BEV}}\}$	0.3581	$\mathrm{std}\{P^i\}$	0.3433

Table 8.2: Strongest correlation coefficients for constructed box-wise features and IoU_{BEV} for the CenterPoint (left) and PointPillars (right) architecture on the nuScenes test dataset and a score threshold $\tau = 0.1$.

Correlation of Box-wise Features with the IoU_{BEV} Figure 8.4 shows the Pearson correlation coefficients of the constructed box-wise dispersion measures with the IoU_{BEV} of prediction and ground truth for CenterPoint on the nuScenes test dataset. The score features have strong correlations (> 0.5) with the IoU_{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 IoU_{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.

Table 8.1, table 8.2 and table 8.3 show the Pearson correlation coefficients of the constructed box-wise dispersion measures with the IoU_{BEV} of prediction and ground truth for the KITTI, nuScenes and Aptiv test datasets. Comparable to the results from fig. 8.4, the score features have strong correlations (> 0.5) with the IoU_{BEV}, independently of the underlying dataset or architecture. Especially for the PointPillars architecture, the number of proposal boxes N^i has a correlations > 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 ℓ, h and w) and the maximal reflectance value of points within the box ($\rho^i_{\max}$) seem to be reasonably correlated with the IoU_{BEV}. Although the other features have rather smaller correlations with the IoU_{BEV}, they may still contribute to a more informative set of features for meta classification and regression.

$\max\{s^i\}$	0.7649	s^i	0.7358	$\max\{s^i\}$	0.7596	$\mathrm{mean}\{s^i\}$	0.7577
$\mathrm{std}\{s^i\}$	0.6598	$\mathrm{mean}\{s^i\}$	0.5913	$\mathrm{std}\{s^i\}$	0.7570	s^i	0.7108
$\max\{IoU^i_{\mathrm{3D}}\}$	0.5656	$\mathrm{mean}\{IoU^i_{\mathrm{3D}}\}$	0.5596	N^i	0.5226	$\mathrm{std}\{\ell^i\}$	0.5225
$\max\{IoU^i_{\mathrm{BEV}}\}$	0.5573	$\mathrm{mean}\{IoU^i_{\mathrm{BEV}}\}$	0.5555	$\mathrm{std}\{F^i\}$	0.4488	$\mathrm{std}\{h^i\}$	0.3998
ℓ^i	0.3736	$\min\{\ell^i\}$	0.3731	$\mathrm{std}\{IoU^i_{\mathrm{3D}}\}$	0.3879	$\mathrm{std}\{\rho^i_{\max}\}$	0.3802
$\mathrm{mean}\{\ell^i\}$	0.3724	$\max\{\ell^i\}$	0.3694	$\mathrm{std}\{z^i\}$	0.3727	$\max\{IoU^i_{\mathrm{3D}}\}$	0.3443

Table 8.3: Strongest correlation coefficients for constructed box-wise features and IoU_{BEV} for the CenterPoint (left) and PointPillars (right) architecture on the Aptiv test dataset and a score threshold $\tau = 0.1$.

	Meta Classification								Meta Regression			
	Accuracies				*AUROC*s				R^2			
Method	LogReg	RF	GB	MLP	LogReg	RF	GB	MLP	RR	RF	GB	MLP
Score	**0.8777**	0.8524	0.8772	$\underline{0.8773}$	**0.8644**	0.8617	0.8623	$\underline{0.8640}$	0.4641	0.4675	$\underline{0.4733}$	**0.4751**
Box Features	0.8877	$\underline{0.9049}$	**0.9203**	0.8975	0.9056	$\underline{0.9454}$	**0.9529**	0.9293	0.5292	$\underline{0.6681}$	**0.6792**	0.6249
LMD	0.9118	0.9166	**0.9297**	$\underline{0.9200}$	0.9450	$\underline{0.9581}$	**0.9628**	0.9530	0.6451	$\underline{0.7242}$	**0.7296**	0.7122

Table 8.4: 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 0.1; higher values are better. Bold numbers indicate the highest performance and underlined numbers represent the second highest (row-wise). Models used are Logistic Regression (LogReg), Ridge Regression (RR), Random Forest (RF), Gradient Boosting (GB) and a Multi Layer Perceptron (MLP).

Comparison of Different Meta Classifiers and Regressors Different models can serve as post-processing modules for meta classification ($\mathcal{C}$) and meta regression ($\mathcal{R}$, see Section 8.3). For meta classification, the meta models under consideration are logistic regression (LogReg), random forest (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 meta models are trained on the box-wise features $\boldsymbol{\xi}^i$ 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 meta models, whereas in the score baseline only the score of the prediction $\widehat{s}^i$ is used to fit the meta model. For the bounding box features baseline, the box features of the prediction $\widehat{\phi}^i$ are used, in which the score $\widehat{s}^i$ is also included. Table 8.4 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 meta models 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. For 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

		Meta Classification						Meta Regression		
		Accuracies			$AUROC$s			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**

Table 8.5: 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 IoU_{BEV} threshold 0.5, score threshold 0.1 and GB as meta model. Higher values are better. Bold numbers indicate the highest performance and underlined numbers represent the second highest (row-wise).

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 meta model is made unlikely by choosing appropriate hyperparameters, the performance of the meta model typically benefits from adding more features, since the available information and number of parameters for fitting are increased. Overall, GB outperforms all other meta models, 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.

Comparison for Different Datasets and Networks Table 8.5 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.

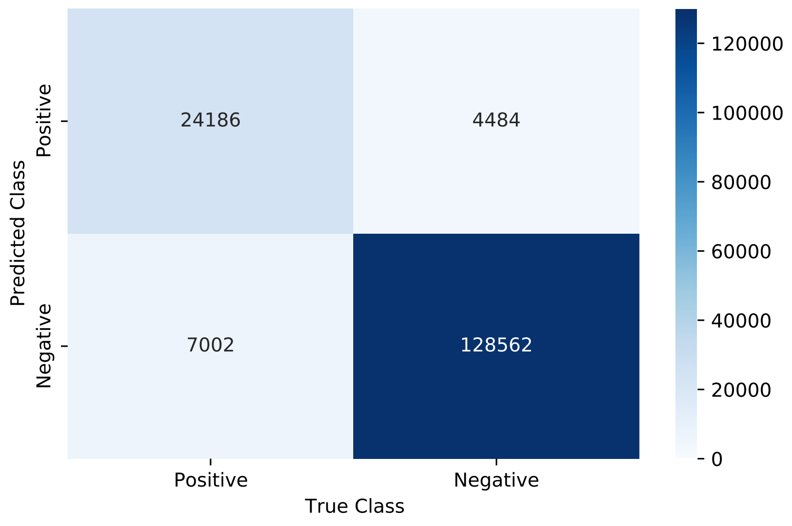


Figure 8.5: Confusion matrix of a GB classifier for LMD on CenterPoint, nuScenes and score threshold 0.1.

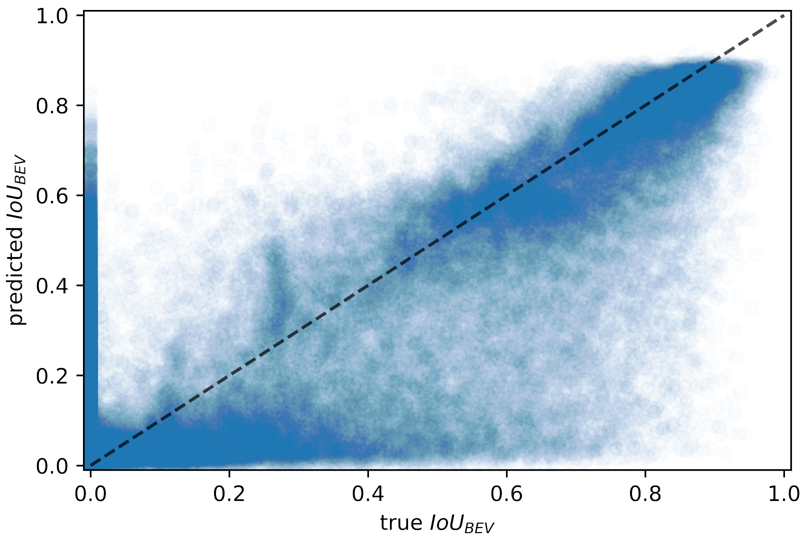


Figure 8.6: Box-wise scatter plot of true IoU_{BEV} and predicted IoU_{BEV} values for LMD on CenterPoint, nuScenes and score threshold 0.1. The predictions are based on a GB regressor.

			Accuracies				*AUROC*s			
Dataset	Network	Method	LogReg	RF	GB	MLP	LogReg	RF	GB	MLP
KITTI	PointPillars	Score	**0.8955**	0.8848	0.8921	$\underline{0.8940}$	**0.9566**	0.9497	0.9530	**0.9566**
		Box Features	**0.8961**	$\underline{0.8958}$	0.8931	0.8846	**0.9564**	$\underline{0.9548}$	0.9537	0.9482
		LMD	0.9000	**0.9028**	$\underline{0.9004}$	0.8844	0.9589	**0.9621**	$\underline{0.9592}$	0.9479
	CenterPoint	Score	**0.8726**	0.8651	0.8688	$\underline{0.8725}$	**0.9322**	0.9242	0.9274	**0.9322**
		Box Features	**0.8727**	$\underline{0.8719}$	0.8691	0.8608	0.9244	**0.9387**	$\underline{0.9343}$	0.9271
		LMD	$\underline{0.8818}$	**0.8847**	0.8806	0.8700	0.9421	**0.9522**	$\underline{0.9466}$	0.9362
nuScenes	PointPillars	Score	$\underline{0.8402}$	0.8106	0.8398	**0.8403**	**0.8151**	0.8130	0.8129	$\underline{0.8150}$
		Box Features	0.8409	0.8613	**0.8708**	$\underline{0.8653}$	0.8499	**0.9018**	$\underline{0.9002}$	0.8957
		LMD	0.8875	0.8842	**0.8915**	$\underline{0.8908}$	0.9208	$\underline{0.9257}$	**0.9280**	0.9252
	CenterPoint	Score	**0.8777**	0.8524	0.8772	$\underline{0.8773}$	**0.8644**	0.8617	0.8623	$\underline{0.8640}$
		Box Features	0.8877	$\underline{0.9049}$	**0.9203**	0.8975	0.9056	$\underline{0.9454}$	**0.9529**	0.9293
		LMD	0.9118	0.9166	**0.9297**	$\underline{0.9200}$	0.9450	$\underline{0.9581}$	**0.9628**	0.9530
Aptiv	PointPillars	Score	**0.7956**	0.7929	0.7939	$\underline{0.7954}$	**0.8582**	0.8555	0.8558	$\underline{0.8580}$
		Box Features	0.8184	**0.8508**	$\underline{0.8489}$	0.8472	0.8933	**0.9289**	$\underline{0.9274}$	0.9255
		LMD	0.8506	$\underline{0.8599}$	**0.8615**	0.8515	0.9273	$\underline{0.9382}$	**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**	$\underline{0.8440}$	0.8439	0.9029	**0.9155**	$\underline{0.9134}$	0.9085
		LMD	0.8478	**0.8559**	$\underline{0.8548}$	0.8529	0.9187	**0.9294**	$\underline{0.9275}$	0.9227

Table 8.6: 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 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).

For CenterPoint and nuScenes, the confusion matrix fig. 8.5 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 detectors 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 8.6 shows a scatter plot of the true IoU_{BEV} of prediction and ground truth and the IoU_{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 IoU_{BEV}-estimates and, therefore, object-wise quality estimates.

Extended Comparison of Different Meta Classifiers Table 8.6 presents meta classification accuracy and $AUROC$ for different meta classification models for the KITTI, nuScenes and Aptiv test datasets. For the score baseline and besides of 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

			R^2			
Dataset	Network	Method	RR	RF	GB	MLP
KITTI	PointPillars	Score	0.6901	0.6726	$\underline{0.7108}$	**0.7146**
		Box Features	0.6973	$\underline{0.7044}$	**0.7131**	0.6819
		LMD	0.7151	**0.7301**	$\underline{0.7287}$	0.6837
	CenterPoint	Score	0.6220	0.5877	$\underline{0.6235}$	**0.6273**
		Box Features	0.6260	$\underline{0.6446}$	**0.6472**	0.6274
		LMD	0.6631	**0.6863**	$\underline{0.6840}$	0.6538
nuScenes	PointPillars	Score	0.3903	0.4006	**0.4055**	$\underline{0.4054}$
		Box Features	0.4187	$\underline{0.5586}$	**0.5593**	0.5356
		LMD	0.6105	$\underline{0.6346}$	**0.6413**	0.6244
	CenterPoint	Score	0.4641	0.4675	$\underline{0.4733}$	**0.4751**
		Box Features	0.5292	$\underline{0.6681}$	**0.6792**	0.6249
		LMD	0.6451	$\underline{0.7242}$	**0.7296**	0.7122
Aptiv	PointPillars	Score	0.5005	0.5013	$\underline{0.5096}$	**0.5106**
		Box Features	0.5469	$\underline{0.6484}$	**0.6568**	0.6482
		LMD	0.6401	$\underline{0.6830}$	**0.6924**	0.6614
	CenterPoint	Score	$\underline{0.5458}$	0.5329	0.5456	**0.5469**
		Box Features	0.5749	$\underline{0.6200}$	**0.6286**	0.6136
		LMD	0.6210	$\underline{0.6541}$	**0.6591**	0.6332

Table 8.7: Comparison of meta regression R^2 for the score baseline, bounding box features and LMD for all available network-dataset combinations using an 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).

classification $AUROC$, the linear model outperforms all other meta models of at most 0.8 pp. For the bounding box features and LMD, random forest or gradient boosting are in most cases the best meta models 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.

Extended Comparison of Different Meta Regressors Table 8.7 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 superior meta models (compared to random forest and ridge regression) in terms of meta regression R^2 for the score baseline. Comparable to the results of table 8.4 of the main paper and the results for meta classification (table 8.6), the bounding box features outperform the score baseline and LMD outperforms the bounding box features.

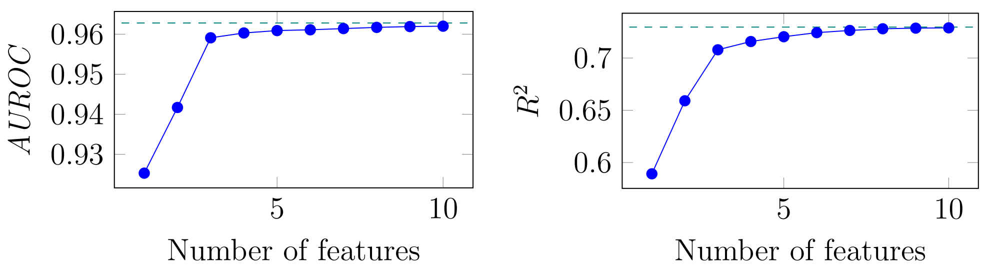


Figure 8.7: Feature selection using a greedy heuristic for CenterPoint, nuScenes and score threshold 0.1. The left figure contains meta classification $AUROC$ and the right one contains meta regression R^2. The dashed line shows the performance when incorporating all features (LMD).

Meta Classification												
		Number of Features										
Dataset	Network	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

Table 8.8: Feature selection for meta classification $AUROC$ using a greedy heuristic for all network-dataset combinations, score threshold 0.1 and a GB classifier. The right-most column shows the the performance when incorporating all features (LMD).

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 8.7 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 meta models 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.

Extended Feature Selection for Meta Classification and Meta Regression Table 8.8 shows feature selection results in terms of meta classification $AUROC$ and table 8.9 shows feature selection results in terms of meta regression R^2. For

Meta Regression												
		Number of Features										
Dataset	Network	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

Table 8.9: Feature selection for meta regression R^2 using a greedy heuristic for all network-dataset combinations, score threshold 0.1 and a GB regressor. The right-most column shows the the performance when incorporating all features (LMD).

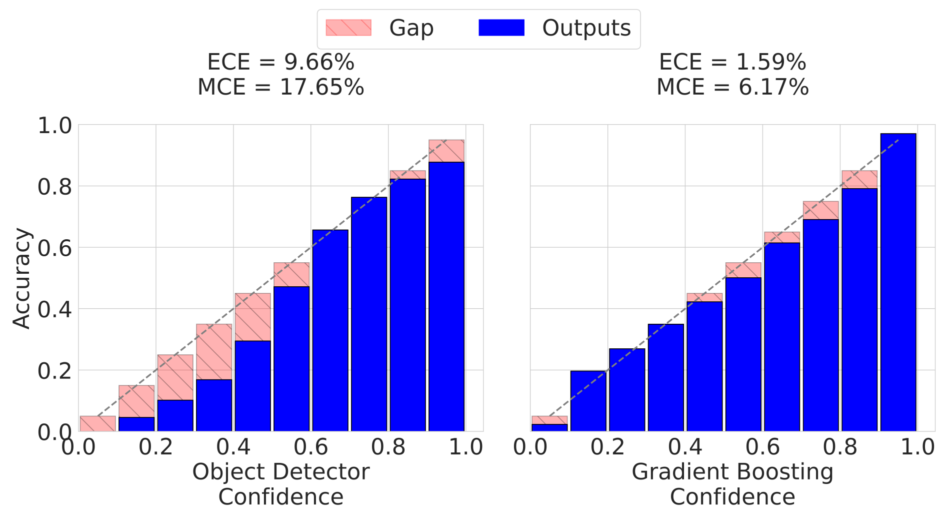


Figure 8.8: Reliability plots of the score (left) and GB classifier for LMD (right) with calibration errors (ECE, MCE) for CenterPoint, nuScenes test dataset, score threshold 0.1 and IoU_{BEV} threshold 0.5.

both tasks, meta classification and regression, a few features are sufficient to reach roughly the same performance as when using all features (LMD). Meta models 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.

Confidence Calibration The score and the meta classifier confidences are divided into 10 confidence bins to evaluate their calibration errors. Figure 8.8 shows exemplary reliability plots for the object detector score and LMD based on a GB

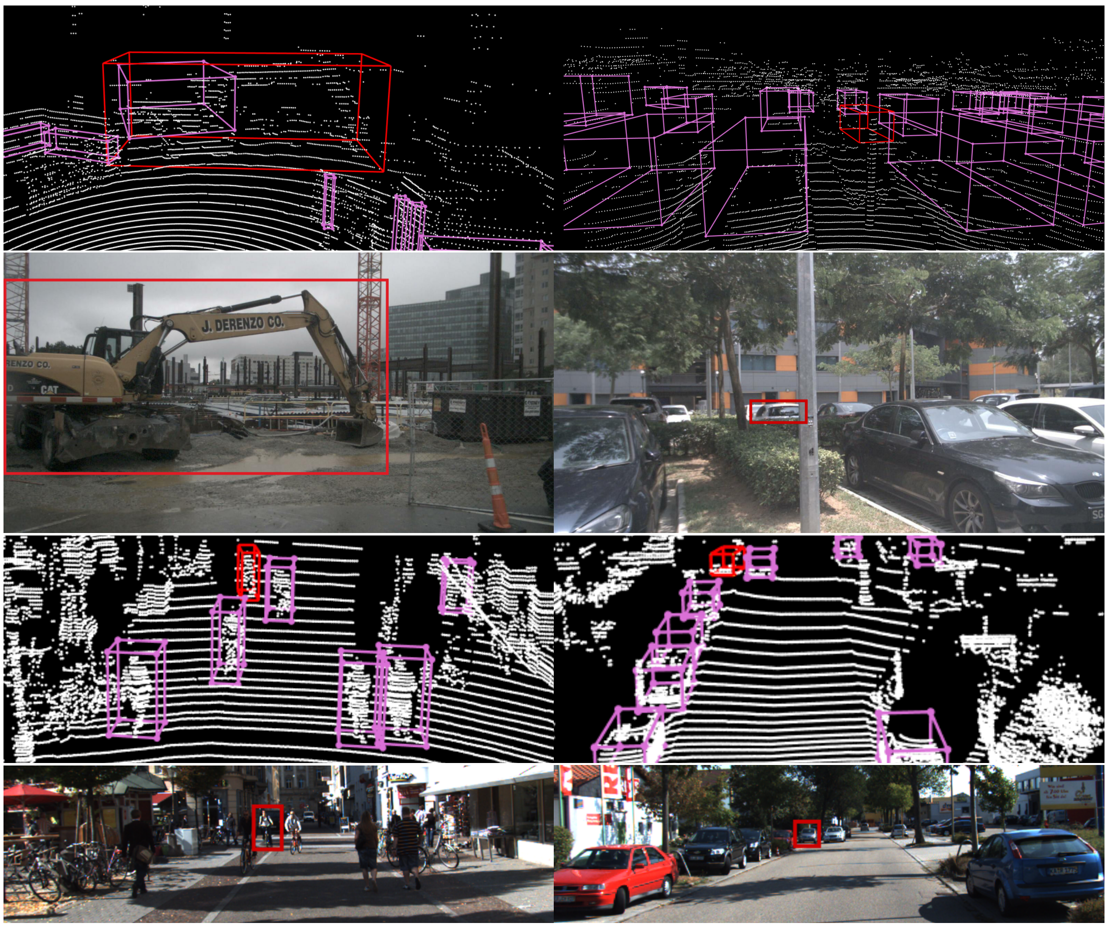


Figure 8.9: Proposed annotation errors in nuScenes (top two) and KITTI (bottom two). Top images show point clouds with annotations in purple and the proposal in red. Camera images aid the evaluation.

classifier with corresponding expected (ECE [110]) and maximum calibration error (MCE [110]). 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 reliability of the confidence assignment.

Annotation Error Detection as an Application of Meta Classification The task of annotations error detection with LMD is inspired by fig. 8.6. There are a number of predictions with $IoU_{\text{BEV}} = 0$ but with high predicted IoU_{BEV}. After looking at these FPs it has been noticed that the prediction itself is, in fact, correct and the corresponding ground truth is not. More precisely, incorrect ground truth

KITTI Annotation Error Analysis (RF)				
Network	Classes	Random	Smart	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**

Table 8.10: 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 an IoU_{BEV} threshold of 0.5.

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 $IoU_{\mathrm{BEV}} < 0.5$ with the ground truth, are sorted by the predicted IoU_{BEV} in a descending order across all images. Then, the first 100 predictions, i.e., the top 100 FPs with highest predicted IoU_{BEV}, are manually reviewed, see fig. 8.9 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 is a significant number of annotation errors in the nuScenes test dataset and that these can be found at far smaller effort with LMD than with the score.

Extended Annotation Error Detection as an Application of Meta Classification Table 8.10 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

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**

Table 8.11: 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 an IoU_{BEV} threshold of 0.5.

and CenterPoint, which indicates that there might be significant number of annotation errors in the KITTI test dataset. This is already confirmed by the smart 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 8.11 shows annotation error detection results for the nuScenes test dataset. We detect only 6 annotation errors using the smart 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 smart score baseline and LMD vanishes. With the smart score baseline we detect 27 and with LMD 28 annotation errors. This observation is in agreement with table 8.6, where CenterPoint achieves superior $AUROC$ values compared to PointPillars on the nuScenes test dataset.

Supplementing the samples of our annotation error proposal method, we show additional proposals for the nuScenes test dataset in fig. 8.10 and respectively for the KITTI test dataset in fig. 8.11.

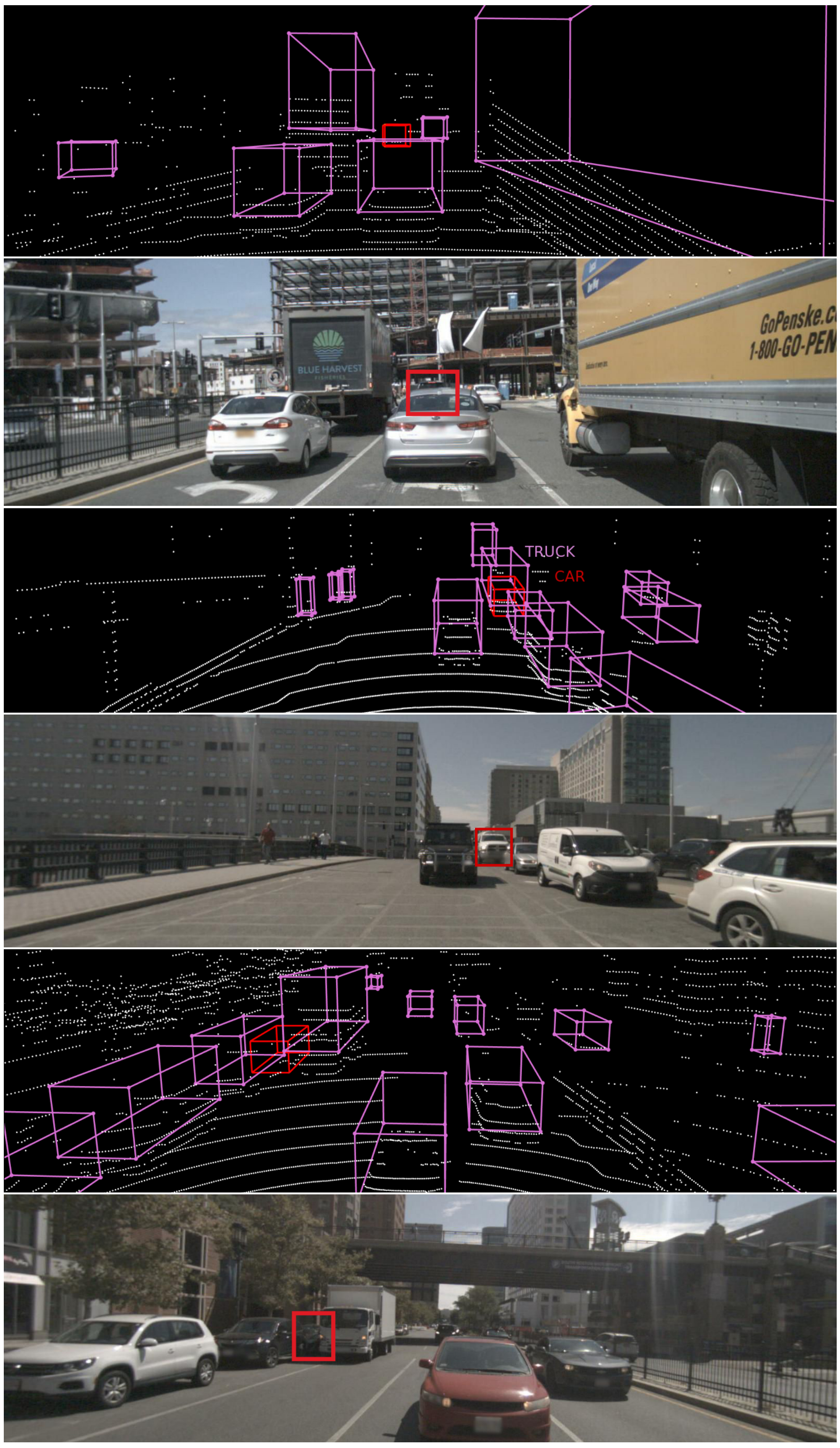


Figure 8.10: Additional annotation error proposals on the nuScenes test dataset.

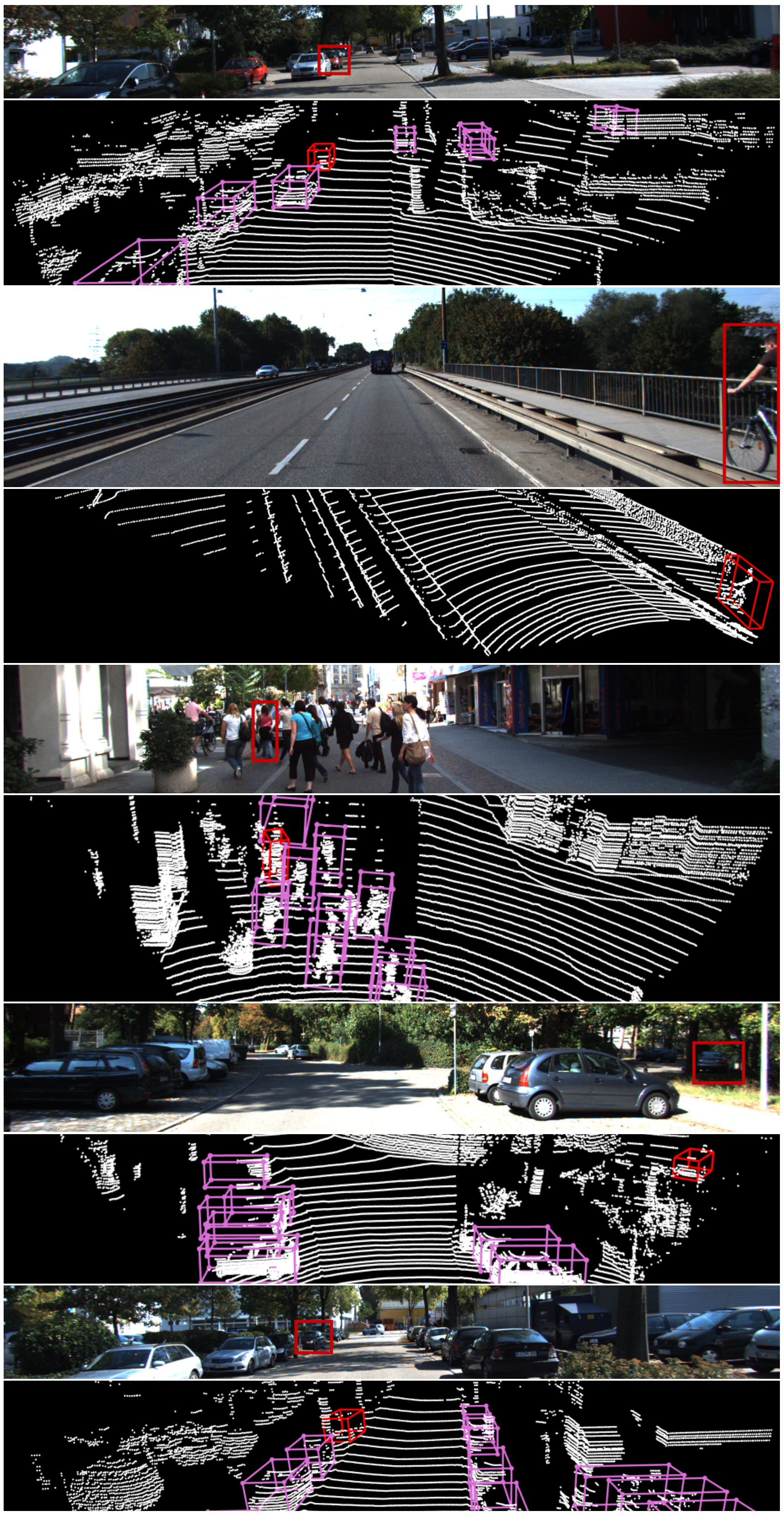


Figure 8.11: Additional annotation error proposals on the KITTI test dataset.

8.5 Conclusion

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 IoU_{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 IoU_{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. We also hope that our investigations will spark future research in the domains of light-weight uncertainty estimation and annotation error detection for large-scale datasets.

CHAPTER 9 CONCLUSION AND OUTLOOK

In this thesis, we have introduced new uncertainty quantification methods for 2D (Chapter 4) and 3D (Chapter 8) object detection. Both methods are post-processing approaches and are thus independent of the underlying object detection architecture. Furthermore, we have studied different uncertainty quantification methods applied to active learning (Chapter 5), label error detection (Chapter 6), and active learning with noisy oracle (Chapter 7).

The following paragraphs consist of the connections between the individual methods, as well as open questions and further areas of application for the future.

Uncertainty Quantification Methods in Object Detection In Chapter 4, we presented MetaDetect, a post-processing and output-based uncertainty quantification method for 2D object detection. MetaDetect provides prediction quality estimates based on meta models and uncertainty metrics, that depend only on the predictions itself and the corresponding discarded candidate boxes during non-maximum suppression. These prediction quality estimates are obtained by the prediction of either a meta regression or a meta classification model, and provide more reliable confidence scores compared to the objectness score of the object detector, as well as the confidence scores obtained by Monte-Carlo dropout uncertainty. LidarMetaDetect (LMD) is introduced in Chapter 8, which is the application of MetaDetect to 3D object detection with Lidar point cloud data. LMD is also a post-processing and output-based uncertainty quantification method, where the set of uncertainty metrics is extended by prediction-wise point cloud information, e.g., the amount of Lidar points that are located in the predicted bounding box. Apart from the higher reliability of the quality estimates obtained by LMD, these estimates are well-calibrated in contrast to the objectness score of the object detector. Moreover, we used LMD uncertainties to identify real label errors on commonly used test datasets in 3D object detection efficiently, resulting in a fairer evaluation of methods after correcting these errors. The results of both, MetaDetect and LMD, agree, that tree-based models (random forests, gradient boosting) outperform linear models and shallow neural networks in terms of meta classification accuracy and $AUROC$, as well as in meta regression R^2. Since we have shown that a small amount of metrics (about 5-10) is sufficient to obtain comparable performances for meta classification and regression compared to using all available metrics (LMD), and since the metrics used can be calculated and provided directly in the forward pass for each prediction, both MetaDetect and LMD are suitable for real-time applications. Once a meta model has been trained, a quality estimate could be obtained for each prediction without significant additional effort, which is particularly interesting for safety-critical applications, such as automated driving. On the other hand, the set of uncertainty metrics can be easily extended to increase meta classification and regression performance, which has been already shown in [125], where MetaDetect is extended with gradient-based and architecture-dependent metrics.

Application of Uncertainty Quantification Methods in Active Learning for Object Detection Developing new active learning methods is very costly and time-consuming, since a single active learning experiment consists of several model trainings. When uncertainty-based methods are chosen as query, a number of hyperparameters have to be studied, since their choice can significantly affect the performance, e.g., the score threshold that decides which predictions are used for uncertainty determination. In Chapter 5, we have introduced a sandbox environment for faster development of active learning methods. Therefore, we presented datasets, which consist of COCO images as background and labeled MNIST letters or EMNIST digits as ground truth. With these datasets, several implemented uncertainty-based active learning methods, and down-scaled versions of state-of-the-art object detectors, we obtained comparable rankings of the methods compared to state-of-the-art datasets and architectures. Since the datasets are self-created, they can be adapted to other datasets as desired, e.g., special class imbalances or different sizes and ratios of labels can be simulated. Furthermore, other active learning methods, e.g., density-based methods, can be implemented and compared with the existing uncertainty-based and random methods.

Uncertainty Quantification Methods for Label Error Detection in Object Detection In Chapter 8, we applied a meta classifier to sort false positive predictions by the obtained confidence in descending order. In general, two types of label errors could be detected efficiently: missing labels, and labels with correct localization but an incorrect class affiliation. Note, that some very poorly localized labels could also be identified, since the corresponding prediction was marked as false positive. This pipeline for label error detection can also be applied analogously for MetaDetect, but poorly localized labels are detected rarely and randomly spawned labels are not detected at all. In order to identify all four types of label errors efficiently, we have introduced a label error detection method for 2D object detection based on instance-wise loss in Chapter 6. We used a two-stage object detector to accumulate the instance-wise first and second stage classification and regression loss to obtain scores for the individual label error proposals. Moreover, each label was added as a proposal for the second stage to provide at least one label error proposal for every labeled bounding box, which is vital for detecting spawned labels. We used simulated label errors to show that models have higher test performance when trained on clean data, and that it may be worth cleaning the current data rather than putting the same effort into acquiring new images and labels. Furthermore, both methods, LMD and the instance-wise loss, identified a huge amount of real label errors on commonly used test datasets in object detection, like Pascal VOC, COCO, BDD100k, and KITTI. We have concluded this work by demonstrating that our methods can also be applied in industry to remove label errors from training and test datasets. In addition, the instance-wise loss method could be extended to effectively identify all four types of label errors in 3D object detection as well.

Uncertainty Quantification Methods in Active Learning with Noisy Oracle
Since we have introduced an active learning sandbox that enables fast development, and since we have found a huge amount of real label errors on commonly used test datasets in object detection, we combined both topics and incorporate label errors into the generic active learning cycle. Therefore, we have introduced a review module that allows to re-label incorrect labels, and which is independent of the chosen query strategy. We have applied the instance-wise loss method of Chapter 6 to efficiently detect missing labels, as well as correctly localized labels with an incorrect class affiliation. We have observed, that an efficient review leads to a significant performance increase compared to a query without or with an inefficient (random) review. For all experiments shown, we have simulated label errors to automatically review and re-label them if necessary. This simulation entails further hyperparameters to be studied, such as the simulated noise for the training data, i.e. how good is the (human) labeler, the noise during the review, i.e. how good is the (human) reviewer, how much effort is invested in the acquisition of new labels relative to the review of current labels, etc. All these ablations can be studied comparatively fast with the active learning sandbox, and the possible customization of datasets, architectures, and hyperparameters, allowing the entire setup to be tailored to the specific setup of individual industrial applications.

BIBLIOGRAPHY

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-Scale Machine Learning*, 2016.

[2] J. Aldrich, *RA Fisher and the Making of Maximum Likelihood*, Statistical Science, 12 (1997), pp. 162–176.

[3] H. Attias, *A Variational Baysian Framework for Graphical Models*, in International Conference on Neural Information Processing Systems (NeurIPS), 1999, pp. 209–215.

[4] R. E. Bellman, *Dynamic Programming*, Princeton University Press, 2010.

[5] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, *YOLACT: Real-Time Instance Segmentation*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2019, pp. 9157–9166.

[6] L. Bottou, *Online Algorithms and Stochastic Approximations*, Online Learning and Neural Networks, (1998).

[7] M.-R. Bouguelia, Y. Belaïd, and A. Belaïd, *Identifying and Mitigating Labelling Errors in Active Learning*, in International Conference on Pattern Recognition Applications and Methods (ICPRAM), Springer, 2015, pp. 35–51.

[8] M.-R. Bouguelia, S. Nowaczyk, K. 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.

[9] S. C. Brenner, *The Mathematical Theory of Finite Element Methods*, Springer, 2008.

[10] C.-A. Brust, C. Käding, and J. Denzler, *Active Learning for Deep Object Detection*, International Conference on Computer Vision Theory and Applications (VISAPP), (2019).

[11] 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.

[12] 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.

[13] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2020, pp. 11618–11628.

[14] Z. Cai and N. Vasconcelos, *Cascade R-CNN: High Quality Object Detection and Instance Segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 43 (2019), pp. 1483–1498.

[15] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, *End-To-End Object Detection with Transformers*, in European Conference on Computer Vision (ECCV), Springer, 2020, pp. 213–229.

[16] S. Chadwick and P. Newman, *Training Object Detectors with Noisy Data*, in IEEE Intelligent Vehicles Symposium (IV), IEEE, 2019, pp. 1319–1325.

[17] R. Chan, M. Rottmann, F. Hüger, P. Schlicht, and H. Gottschalk, *Controlled False Negative Reduction of Minority Classes in Semantic Segmentation*, in International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–8.

[18] H. Chen, Y. Huang, W. Tian, Z. Gao, and L. Xiong, *MonoRUn: Monocular 3D Object Detection by Reconstruction and Uncertainty Propagation*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2021, pp. 10374–10383.

[19] 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).

[20] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*, in European Conference on Computer Vision (ECCV), Springer, 2018, pp. 801–818.

[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 (ICML), Proceedings of Machine Learning Research (PMLR), 2019, pp. 1062–1070.

[22] P.-C. Chen, B.-H. Kung, and J.-C. Chen, *Class-Aware Robust Adversarial Training for Object Detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2021, pp. 10420–10429.

[23] S.-T. Chen, C. Cornelius, J. Martin, and D. H. Chau, *ShapeShifter: Robust Physical Adversarial Attack on Faster R-CNN Object Detector*, in European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Springer, 2019, pp. 52–68.

[24] J. Choi, I. Elezi, H.-J. Lee, C. Farabet, and J. M. Alvarez, *Active Learning for Deep Object Detection via Probabilistic Modeling*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2021, pp. 10264–10273.

[25] F. Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 1251–1258.

[26] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, *EMNIST: Extending MNIST to Handwritten Letters*, in International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 2921–2926.

[27] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 3213–3223.

[28] G. Cybenko, *Approximation by Superpositions of a Sigmoidal Function*, Mathematics of Control, Signals and Systems, 2 (1989), pp. 303–314.

[29] C. Darken, J. Chang, and J. Moody, *Learning Rate Schedules for Faster Stochastic Gradient Search*, in Neural Networks for Signal Processing, vol. 2, Citeseer, 1992, pp. 3–12.

[30] J. Davis and M. Goadrich, *The Relationship between Precision-Recall and ROC Curves*, in International Conference on Machine Learning (ICML), Proceedings of Machine Learning Research (PMLR), 2006, pp. 233–240.

[31] 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 British Machine Vision Conference (BMVC), K. Sidorov and Y. Hicks, eds., BMVA Press, 2019, pp. 177.1–177.13.

[32] L. R. Dice, *Measures of the Amount of Ecologic Association between Species*, Ecology, 26 (1945), pp. 297–302.

[33] M. Dickinson and D. Meurers, *Detecting Errors in Part-Of-Speech Annotation*, in European Chapter of the Association for Computational Linguistics (EACL), Association for Computational Linguistics, 2003.

[34] Z. Dong, P. Wei, and L. Lin, *Adversarially-Aware Robust Object Detector*, in European Conference on Computer Vision (ECCV), Springer, 2022, pp. 297–313.

[35] 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 International Conference on Learning Representations (ICLR), OpenReview.net, 2020.

[36] D. Duvenaud, D. Maclaurin, and R. Adams, *Early Stopping as Non-parametric Variational Inference*, in Artificial Intelligence and Statistics, Proceedings of Machine Learning Research (PMLR), 2016, pp. 1070–1077.

[37] I. Elezi, Z. Yu, A. Anandkumar, L. Leal-Taixé, and J. M. Alvarez, *Not All Labels Are Equal: Rationalizing the Labeling Costs for Training Object Detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2022, pp. 14492–14501.

[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 (IJCV), 88 (2010), pp. 303–338.

[39] A. Farhadi and J. Redmon, *Yolov3: An Incremental Improvement*, in Computer Vision and Pattern Recognition, vol. 1804, Springer, 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*, vol. 22, IEEE, 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 IEEE Intelligent Transportation Systems Conference (ITSC), IEEE, 2018, pp. 3266–3273.

[42] D. Feng, L. Rosenbaum, F. Timm, and K. Dietmayer, *Leveraging Heteroscedastic Aleatoric Uncertainties for Robust Real-Time LiDAR 3D Object Detection*, in IEEE Intelligent Vehicles Symposium (IV), IEEE, 2019, pp. 1280–1287.

[43] 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 Intelligent Transportation Systems Conference (ITSC), (2021).

[44] Y. Gal and Z. Ghahramani, *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*, in International Conference on Machine Learning (ICML), Proceedings of Machine Learning Research (Proceedings of Machine Learning Research (PMLR)), 2016, pp. 1050–1059.

[45] Y. Gal, R. Islam, and Z. Ghahramani, *Deep Bayesian Active Learning with Image Data*, in International Conference on Machine Learning (ICML), Proceedings of Machine Learning Research (PMLR), 2017, pp. 1183–1192.

[46] A. Geiger, P. Lenz, and R. Urtasun, *Are We Ready for Autonomous Driving? the KITTI Vision Benchmark Suite*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2012.

[47] S. Geman, E. Bienenstock, and R. Doursat, *Neural Networks and the Bias/Variance Dilemma*, Neural Computation, 4 (1992), pp. 1–58.

[48] R. Girshick, *Fast R-CNN*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2015, pp. 1440–1448.

[49] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2014, pp. 580–587.

[50] J. Goldberger and E. Ben-Reuven, *Training Deep Neural-Networks using a Noise Adaptation Layer*, in International Conference on Machine Learning (ICML), Proceedings of Machine Learning Research (PMLR), 2017.

[51] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[52] I. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and Harnessing Adversarial Examples*, in International Conference on Learning Representations (ICLR), OpenReview.net, 2015.

[53] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, *On Calibration of Modern Neural Networks*, in International Conference on Machine Learning (ICML), Proceedings of Machine Learning Research (PMLR), 2017, pp. 1321–1330.

[54] A. Gupta, S. Narayan, K. Joseph, S. Khan, F. S. Khan, and M. Shah, *OW-DETR: Open-World Detection Transformer*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2022, pp. 9235–9244.

[55] G. Gupta, A. K. Sahu, and W.-Y. Lin, *Noisy Batch Active Learning with Deterministic Annealing*, arXiv preprint arXiv:1909.12473, (2019).

[56] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. W. Tsang, and M. Sugiyama, *Co-Teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels*, in International Conference on Neural Information Processing Systems (NeurIPS), 2018, pp. 8536–8546.

[57] A. Harakeh, M. Smart, and S. L. Waslander, *BayesOD: A Bayesian Approach for Uncertainty Estimation in Deep Object Detectors*, in IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 87–93.

[58] J. A. Hartigan and M. A. Wong, *Algorithm AS 136: A K-Means Clustering Algorithm*, Journal of the Royal Statistical Society. Series C (Applied Statistics), 28 (1979), pp. 100–108.

[59] 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 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2020, pp. 1430–1435.

[60] 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 (ICLR), OpenReview.net, 2020.

[61] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask R-CNN*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 2961–2969.

[62] K. He and J. Sun, *Convolutional Neural Networks at Constrained Time Cost*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 5353–5360.

[63] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 770–778.

[64] Y. He, X. Zhang, M. Savvides, and K. Kitani, *Softer-NMS: Rethinking Bounding Box Regression for Accurate Object Detection*, arXiv preprint arXiv:1809.08545, 2 (2018), pp. 69–80.

[65] D. O. Hebb, *The First Stage of Perception: Growth of the Assembly*, The Organization of Behavior, 4 (1949), pp. 78–60.

[66] D. Hendrycks and K. Gimpel, *A Baseline for Detecting Misclassified and Out-Of-Distribution Examples in Neural Networks*, in International Conference on Learning Representations (ICLR), OpenReview.net, 2016.

[67] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, *Using Trusted Data to Train Deep Networks on Labels Corrupted by Severe Noise*, in International Conference on Neural Information Processing Systems (NeurIPS), 2018, pp. 10477–10486.

[68] J. J. Hopfield and D. W. Tank, *"Neural" Computation of Decisions in Optimization Problems*, Biological Cybernetics, 52 (1985), pp. 141–152.

[69] K. Hornik, M. Stinchcombe, and H. White, *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, 2 (1989), pp. 359–366.

[70] 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 (GRSL), 19 (2022), pp. 1–5.

[71] C. Huang, Q. Wu, and F. Meng, *QualityNet: Segmentation Quality Evaluation with Deep Convolutional Networks*, in IEEE International Conference on Visual Communications and Image Processing (VCIP), IEEE, 2016, pp. 1–4.

[72] 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.

[73] R. Hussain and S. Zeadally, *Autonomous Cars: Research Results, Issues, and Future Challenges*, IEEE Communications Surveys & Tutorials, 21 (2018), pp. 1275–1313.

[74] P. Jaccard, *The Distribution of the Flora in the Alpine Zone*, 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 Workshops (ML4HW), Proceedings of Machine Learning Research (PMLR), 2020, pp. 171–183.

[76] 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 (ICML), Proceedings of Machine Learning Research (PMLR), 2018, pp. 2304–2313.

[77] 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 International Conference on Management of Data (SIGMOD), 2022, pp. 496–505.

[78] J. Kang and J. Gwak, *Ensemble of Instance Segmentation Models for Polyp Segmentation in Colonoscopy Images*, IEEE Access, 7 (2019), pp. 26440–26447.

[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] K. I. Kim, *Active Label Correction using Robust Parameter Update and Entropy Propagation*, in European Conference on Computer Vision (ECCV), Springer, 2022, pp. 1–16.

[81] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, International Conference on Learning Representations, (2015).

[82] A. Koksal, K. G. Ince, and A. Alatan, *Effect of Annotation Errors on Drone Detection with YOLOv3*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, 2020, pp. 1030–1031.

[83] F. Kraus and K. Dietmayer, *Uncertainty Estimation in One-Stage Object Detection*, in IEEE Intelligent Transportation Systems Conference (ITSC), IEEE, 2019, pp. 53–60.

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, in International Conference on Neural Information Processing Systems (NeurIPS), 2012, pp. 1097–1105.

[85] S. Kullback and R. A. Leibler, *On Information and Sufficiency*, The Annals of Mathematical Statistics, 22 (1951), pp. 79–86.

[86] F. Kuppers, J. Kronenberger, A. Shantia, and A. Haselhoff, *Multivariate Confidence Calibration for Object Detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPR), IEEE, 2020, pp. 326–327.

[87] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, *A Survey of Deep Learning Applications to Autonomous Vehicle Control*, vol. 22, IEEE, 2020, pp. 712–733.

[88] B. Lakshminarayanan, A. Pritzel, and C. Blundell, *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*, in International Conference on Neural Information Processing Systems (NeurIPS), 2017, pp. 6405–6416.

[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 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2019, pp. 12689–12697.

[90] M. T. Le, F. Diehl, T. Brunner, and A. Knol, *Uncertainty Estimation for Deep Neural Object Detectors in Safety-Critical Applications*, in IEEE Intelligent Transportation Systems Conference (ITSC), IEEE, 2018, pp. 3873–3878.

[91] 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.

[92] H. Li, Z. Wu, C. Zhu, C. Xiong, R. Socher, and L. S. Davis, *Learning from Noisy Anchors for One-Stage Object Detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2020, pp. 10588–10597.

[93] M. Li and I. K. Sethi, *Confidence-Based Active Learning*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 28 (2006), pp. 1251–1261.

[94] S. Liang, Y. Li, and R. Srikant, *Principled Detection of Out-Of-Distribution Examples in Neural Networks*, arXiv preprint arXiv:1706.02690, (2017).

[95] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, *Feature Pyramid Networks for Object Detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 2117–2125.

[96] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal Loss for Dense Object Detection*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 2980–2988.

[97] 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 (ECCV), Springer, 2014, pp. 740–755.

[98] 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, Springer, 2003, pp. 217–231.

[99] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*, in European Conference on Computer Vision (ECCV), Springer, 2016, pp. 21–37.

[100] 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 IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2021, pp. 10012–10022.

[101] J. Lu, H. Sibai, and E. Fabry, *Adversarial Examples that Fool Detectors*, arXiv preprint arXiv:1712.02494, (2017).

[102] K. Maag, M. Rottmann, and H. Gottschalk, *Time-Dynamic Estimates of the Reliability of Deep Semantic Segmentation Networks*, in IEEE International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2020, pp. 502–509.

[103] D. J. MacKay, *A Practical Bayesian Framework for Backpropagation Networks*, Neural Computation, 4 (1992), pp. 448–472.

[104] 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.

[105] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2019, pp. 12669–12678.

[106] G. P. Meyer and N. Thakurdesai, *Learning an Uncertainty-Aware Object Detector for Autonomous Driving*, in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 10521–10527.

[107] D. Miller, F. Dayoub, M. Milford, and N. Sünderhauf, *Evaluating Merging Strategies for Sampling-Based Uncertainty Techniques in Object Detection*, in IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 2348–2354.

[108] D. Miller, L. Nicholson, F. Dayoub, and N. Sünderhauf, *Dropout Sampling for Robust Object Detection in Open-Set Conditions*, in IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 3243–3249.

[109] MMDetection3D Contributors, *OpenMMLab's Next-Generation Platform for General 3D Object Detection*, 2020. https://github.com/open-mmlab/mmdetection3d.

[110] M. P. Naeini, G. F. Cooper, and M. Hauskrecht, *Obtaining Well Calibrated Probabilities using Bayesian Binning*, in AAAI Conference on Artificial Intelligence (AAAI), 2015.

[111] NATIONAL TRANSPORTATION SAFETY BOARD, *Collision between Vehicle controlled by Developmental Automated Driving System and Pedestrian*, Highway Accident Report, (2019).

[112] Y. NETZER, T. WANG, A. COATES, A. BISSACCO, B. WU, AND A. Y. NG, *Reading Digits in Natural Images with Unsupervised Feature Learning*, International Conference on Neural Information Processing Systems (NeurIPS) Workshop on Deep Learning and Unsupervised Feature Learning, (2011).

[113] C. NORTHCUTT, L. JIANG, AND I. CHUANG, *Confident Learning: Estimating Uncertainty in Dataset Labels*, Journal of Artificial Intelligence Research (JAIR), 70 (2021), pp. 1373–1411.

[114] C. G. NORTHCUTT, A. ATHALYE, AND J. MUELLER, *Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks*, International Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track, (2021).

[115] O. OZDEMIR, B. WOODWARD, AND A. A. BERLIN, *Propagating Uncertainty in Multi-Stage Bayesian Convolutional Neural Networks with Application to Pulmonary Nodule Detection*, in International Conference on Neural Information Processing Systems (NeurIPS) Workshop on Bayesian Deep Learning, 2017.

[116] D. P. PAPADOPOULOS, J. R. UIJLINGS, F. KELLER, AND V. FERRARI, *Training Object Class Detectors with Click Supervision*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 6374–6383.

[117] P. PETERSEN AND F. VOIGTLAENDER, *Equivalence of Approximation by Convolutional Neural Networks and Fully-Connected Networks*, Proceedings of the American Mathematical Society, 148 (2020), pp. 1567–1581.

[118] B. T. PHAN, R. SALAY, K. CZARNECKI, V. ABDELZAD, T. DENOUDEN, AND S. VERNEKAR, *Calibrating Uncertainties in Object Localization Task*, in International Conference on Neural Information Processing Systems (NeurIPS) Workshop on Bayesian Deep Learning, 2018.

[119] M. PITROPOV, C. HUANG, V. ABDELZAD, K. CZARNECKI, AND S. WASLANDER, *LiDAR-MIMO: Efficient Uncertainty Estimation for LiDAR-Based 3D Object Detection*, in IEEE Intelligent Vehicles Symposium (IV), 2022, pp. 813–820.

[120] N. QIAN, *On the Momentum Term in Gradient Descent Learning Algorithms*, Neural Networks, 12 (1999), pp. 145–151.

[121] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 779–788.

[122] S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, *Training Deep Neural Networks on Noisy Labels with Bootstrapping*, in International Conference on Learning Representations (ICLR) Workshop, OpenReview.net, 2015.

[123] 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 (PAMI), 39 (2017), pp. 1137–1149.

[124] S. M. Riad, *The Deconvolution Problem: An Overview*, Proceedings of the IEEE, 74 (1986), pp. 82–85.

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

[126] 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).

[127] 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, 2022, pp. 251–275.

[128] T. Riedlinger, M. Schubert, S. Penquitt, J.-M. 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*, arXiv preprint arXiv:2306.07835, (2023).

[129] F. Rosenblatt, *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*, Psychological Review, 65 (1958), p. 386.

[130] 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 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–9.

[131] 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 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2020, pp. 1351–1356.

[132] M. Rottmann and M. Reese, *Automated Detection of Label Errors in Semantic Segmentation Datasets via Deep Learning and Uncertainty Quantification*, in IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), IEEE, 2023, pp. 3214–3223.

[133] M. Rottmann and M. Schubert, *Uncertainty Measures and Prediction Quality Rating for the Semantic Segmentation of Nested Multi Resolution Street Scene Images*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, 2019, pp. 0–0.

[134] A. G. Roy, S. Conjeti, N. Navab, and C. Wachinger, *Inherent Brain Segmentation Quality Control from Fully ConvNet Monte Carlo Sampling*, in International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI), Springer, 2018, pp. 664–672.

[135] S. Roy, A. Unmesh, and V. P. Namboodiri, *Deep Active Learning for Object Detection*, in British Machine Vision Conference (BMVC), BMVA Press, 2019.

[136] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Representations by Back-Propagating Errors*, Nature, 323 (1986), pp. 533–536.

[137] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), 115 (2015), pp. 211–252.

[138] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2018, pp. 4510–4520.

[139] M. P. Schilling, T. Scherr, F. R. Munke, O. Neumann, M. Schutera, R. Mikut, and M. Reischl, *Automated Annotator Variability Inspection for Biomedical Image Segmentation*, IEEE access, 10 (2022), p. 2753.

[140] S. Schmidt, Q. Rao, J. Tatsch, and A. Knoll, *Advanced Active Learning Strategies for Object Detection*, in IEEE Intelligent Vehicles Symposium (IV), IEEE, 2020, pp. 871–876.

[141] M. Schubert, K. Kahl, and M. Rottmann, *MetaDetect: Uncertainty Quantification and Prediction Quality Estimates for Object Detection*, in International Joint Conference on Neural Networks (IJCNN), IEEE, 2021, pp. 1–10.

[142] 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).

[143] M. Schubert, T. Riedlinger, K. Kahl, and M. Rottmann, *Deep Active Learning with Noisy Oracle in Object Detection*, arXiv preprint arXiv:2310.00372, (2023).

[144] S. Seo, P. H. Seo, and B. Han, *Learning for Single-Shot Confidence Calibration in Deep Neural Networks through Stochastic Inferences*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2019, pp. 9030–9038.

[145] B. Settles, *Active Learning Literature Survey*, Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[146] S. Shahriar and K. Hayawi, *Let's Have a Chat! a Conversation with ChatGPT: Technology, Applications, and Limitations*, in Artificial Intelligence and Applications (AIA), 2022.

[147] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014.

[148] C. E. Shannon, *A Mathematical Theory of Communication*, The Bell System Technical Journal, 27 (1948), pp. 379–423.

[149] K. Simonyan and A. Zisserman, *Very Deep Convolutional networks for Large-Scale Image Recognition*, in International Conference on Learning Representations (ICLR), Computational and Biological Learning Society, OpenReview.net, 2015.

[150] T. Siri, *Deep Learning for Siri's Voice: On-Device Deep Mixture Density Networks for Hybrid Unit Selection Synthesis*, Journal of Machine Learning Research (JMLR), 1 (2014).

[151] 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.

[152] A. Subramanian and A. Subramanian, *One-Click Annotation with Guided Hierarchical Object Detection*, arXiv preprint arXiv:1810.00609, (2018).

[153] F. Sultana, A. Sufian, and P. Dutta, *Advancements in Image Classification using Convolutional Neural Network*, in International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), IEEE, 2018, pp. 122–129.

[154] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, *Revisiting Unreasonable Effectiveness of Data in Deep Learning Era*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 843–852.

[155] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going Deeper with Convolutions*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 1–9.

[156] M. Telgarsky, *Representation Benefits of Deep Feedforward Networks*, arXiv preprint arXiv:1509.08101, (2015).

[157] G. Tesauro, *Practical Issues in Temporal Difference Learning*, in International Conference on Neural Information Processing Systems (NeurIPS), 1991, pp. 259–266.

[158] L. H. Thai, T. S. Hai, and N. T. Thuy, *Image Classification using Support Vector Machine and Artificial Neural Network*, International Journal of Information Technology and Computer Science (IJITCS), 4 (2012), pp. 32–38.

[159] A. Thyagarajan, E. Snorrason, C. G. Northcutt, and J. Mueller, *Identifying Incorrect Annotations in Multi-Label Classification Data*, in International Conference on Learning Representations (ICLR) Workshop on Pitfalls of Limited Data and Computation for Trustworthy ML, OpenReview.net, 2023.

[160] 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, pp. 1198–1218.

[161] R. Van Handel, *Probability in High Dimension*, tech. rep., PRINCETON UNIV NJ, 2014.

[162] V. Vapnik, *Principles of Risk Minimization for Learning Theory*, in International Conference on Neural Information Processing Systems (NeurIPS), 1991, pp. 831–838.

[163] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention Is All You Need*, in International Conference on Neural Information Processing Systems (NeurIPS), 2017, pp. 6000–6010.

[164] P. Viola and M. Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, IEEE, 2001, pp. I–I.

[165] Y. Wang, X. Ma, Z. Chen, Y. Luo, J. Yi, and J. Bailey, *Symmetric Cross Entropy for Robust Learning with Noisy Labels*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2019, pp. 322–330.

[166] B. Widrow and M. E. Hoff, *Adaptive Switching Circuits*, in IRE WESCON Convention Record, vol. 4, New York, 1960, pp. 96–104.

[167] Z. Wu, N. Bodla, B. Singh, M. Najibi, R. Chellappa, and L. S. Davis, *Soft Sampling for Robust Object Detection*, in British Machine Vision Conference (BMVC), BMVA Press, 2020.

[168] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille, *Adversarial Examples for Semantic Segmentation and Object Detection*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 1369–1378.

[169] M. Xu, Y. Bai, and B. Ghanem, *Missing Labels in Object Detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), vol. 3, IEEE, 2019, p. 5.

[170] 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*, in International Conference on Neural Information Processing Systems (NeurIPS), 2019, pp. 6225–6236.

[171] S. Yan, K. Chaudhuri, and T. Javidi, *Active Learning from Imperfect Labelers*, in International Conference on Neural Information Processing Systems (NeurIPS), 2016, pp. 2136–2144.

[172] Y. Yan, Y. Mao, and B. Li, *Second: Sparsely Embedded Convolutional Detection*, Sensors, 18 (2018), p. 3337.

[173] Y. Yan, R. Rosales, G. Fung, R. Subramanian, and J. Dy, *Learning from Multiple Annotators with Varying Expertise*, Machine Learning, 95 (2014), pp. 291–327.

[174] B. Yang, W. Luo, and R. Urtasun, *PIXOR: Real-Time 3D Object Detection from Point Clouds*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2018, pp. 7652–7660.

[175] Q. Yang, H. Chen, Z. Chen, and J. Su, *Uncertainty Estimation for Monocular 3D Object Detectors in Autonomous Driving*, in IEEE International Conference on Robotics and Automation Engineering (ICRAE), IEEE, 2021, pp. 55–59.

[176] D. Yarotsky, *Error Bounds for Approximations with Deep ReLU Networks*, Neural Networks, 94 (2017), pp. 103–114.

[177] D. Yarotsky, *Universal Approximations of Invariant Maps by Neural Networks*, Constructive Approximation, 55 (2022), pp. 407–474.

[178] T. Yin, X. Zhou, and P. Krahenbuhl, *Center-Based 3D Object Detection and Tracking*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2021, pp. 11779–11788.

[179] D. Yoo and I. S. Kweon, *Learning Loss for Active Learning*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2019, pp. 93–102.

[180] T. Younesian, D. Epema, and L. Y. Chen, *Active Learning for Noisy Data Streams using Weak and Strong Labelers*, arXiv preprint arXiv:2010.14149, (2020).

[181] T. Younesian, Z. Zhao, A. Ghiassi, R. Birke, and L. Y. Chen, *QActor: Active Learning on Noisy Labels*, in Asian Conference on Machine Learning (ACML), Proceedings of Machine Learning Research (PMLR), 2021, pp. 548–563.

[182] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2020, pp. 2636–2645.

[183] T. Yuan, F. Wan, M. Fu, J. Liu, S. Xu, X. Ji, and Q. Ye, *Multiple Instance Active Learning for Object Detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2021, pp. 5330–5339.

[184] X. Zhan, Q. Wang, K.-h. Huang, H. Xiong, D. Dou, and A. B. Chan, *A Comparative Survey of Deep Active Learning*, arXiv preprint arXiv:2203.13450, (2022).

[185] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, *mixup: Beyond Empirical Risk Minimization*, in International Conference on Learning Representations (ICLR), OpenReview.net, 2018.

[186] H. Zhang and J. Wang, *Towards Adversarially Robust Object Detection*, in IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, 2019, pp. 421–430.

[187] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2022, pp. 2736–2746.

[188] D.-X. Zhou, *Universality of Deep Convolutional Neural Networks*, Applied and Computational Harmonic Analysis, 48 (2020), pp. 787–794.

[189] Y.-T. Zhou and R. Chellappa, *Computation of Optical Flow using a Neural Network*, in IEEE International Conference on Neural Networks (ICNN), 1988, pp. 71–78.