Uncertainty Quantification and its Applications for Multimodal Semantic Segmentation

Dissertation

Bergische Universität Wuppertal
Fakultät für Mathematik und Naturwissenschaften

zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften

eingereicht von
Pascal Colling, M. Sc.

Betreut durch
Prof. Dr. Hanno Gottschalk
Dr. Matthias Rottmann

Wuppertal, 02.05.2022

The PhD thesis can be quoted as follows:

urn:nbn:de:hbz:468-20220714-104644-0
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20220714-104644-0]

DOI: 10.25926/8edd-w264
[https://doi.org/10.25926/8edd-w264]

Abstract

This thesis is about uncertainty quantification and its applications for multimodal semantic segmentation in the area of automated driving. The first main contribution is a new active learning method. The annotation of images for semantic segmentation is very costly. Active learning describes an iterative process in which data is annotated based on a selection criterion. This criterion is usually based on image predictions that are considered uncertain. The active learning method presented here does not select whole images, but only image regions that have poor estimated segmentation quality and are estimated to be labeled with low costs. The second main contribution is about a method for semantic segmentation of point clouds to detect false positive segments as well as a segment-wise prediction quality estimation. For this purpose, point cloud data is projected onto an image representation. Based on this representation, uncertainty measures are defined per pixel and aggregated at predicted segment level. The aggregated data is used to train two machine learning models, the first model for detecting false positive segments and the second for estimating the prediction quality per predicted segment. The third main contribution is the automated generation of high definition (HD) maps. HD maps are becoming increasingly important in the field of automated driving and represent a semantic segmentation of the road environment. For the automated HD map generation method, other road users are tracked. The tracks as well as the driving path of the recording vehicle are aggregated and the information for the HD map such as lanes is extracted. It is assumed that multiple recordings are taken for a given region. This results in numerous observations of the lanes by the tracked road users. By processing many road users from multiple recordings, uncertainty is reduced and reliable HD maps are generated.

Zusammenfassung

Die vorliegende Arbeit behandelt Quantifizierung von Unsicherheit und ihre Anwendungen für multimodale semantische Segmentierung. Der Fokus der Anwendungen liegt im Bereich des automatisierten Fahrens. Der erste Hauptbeitrag ist eine neue Methode des Aktiven Lernens. Die Annotation von Bildern für die semantische Segmentierung ist sehr aufwendig. Aktives Lernen beschreibt einen iterativen Prozess, bei dem Daten auf Basis eines Auswahlkriteriums annotiert werden. Dieses Kriterium basiert meist auf Bildvorhersagen, die als unsicher gelten. Die hier vorgestellte Methode des Aktiven Lernens wählt keine ganzen Bilder, sondern nur Bildregionen aus, die zu einem eine schlechte Segmentierungsqualität aufweisen und zum anderen nicht aufwendig zu annotieren sind – beide Kriteren werden geschätzt. Der zweite Hauptbeitrag entwickelt eine Methode für die semantische Segmentierung von Punktwolken für die Erkennung von falsch prädizierten Segmenten sowie eine Qualitätsschätzung, ebefalls pro prädiziertem Segment. Dazu werden die Daten der Punktwolke auf eine Bilddarstellung projiziert. Auf Basis dieser Darstellung werden Unsicherheitsmaße pro Pixel definiert und auf Ebene des prädizierten Segments aggregiert. Die aggregierten Daten werden genutzt, um zwei Modelle des Maschinellen Lernens zu trainieren, das erste Model für die Erkennung von falsch prädizierten Segmenten und das zweite zur Einschätzung der Vorhersagequalität pro prädiziertem Segment. Der dritte Hauptbeitrag ist die automatisierte Erstellung von hochauflösenden (HD) Karten. HD Karten werden im Bereich des automatisierten Fahrens immer wichtiger und stellen eine semantische Segmentierung der Straßenumgebung dar. Für die Methode der automatisierten HD Kartenerstellung werden andere Verkehrsteilnehmer detektiert und getrackt. Die getrackten Verkehrsteilnehmer sowie die Fahrspur des Aufnahmefahrzeugs werden aggregiert und die Informationen für die HD Karte wie Fahrbahnspuren werden extrahiert. Es wird angenommen, dass für eine gegebene Region mehrere Aufnahmen gemacht werden. Daraus ergibt sich eine Vielzahl an Beobachtungen der Fahrspuren durch die detektierten Verkehrsteilnehmer. Durch die Verarbeitung vieler Verkehrsteilnehmer aus mehreren Aufnahmen wird die Unsicherheit reduziert und zuverlässige HD Karten werden erstellt.

Acknowledgments

First and foremost, I would like to thank my supervisors Hanno Gottschalk and Matthias Rottmann. Your close contact and exchange with students and employees is unprecedented. I have appreciated that very much. Thank you for your guidance, support and open discussions. Thank you for giving me the opportunity to do my PhD and the freedom to choose many aspects of my research.

I want to thank Lutz Roese-Koerner, Dennis Müller and Chrisitian Nunn for enabling this scholarship program and the industry-related research opportunity with it. Thank you to the whole 'AI Core' team of Aptiv's Advanced Engineering department in Wuppertal. It is a pleasure to work with so many smart people and on topics that are highly related to current research areas and their applications.

Furthermore, I would like to thank my seatmates at the university, Kira Maag and Robin Chan as well as fellow scholars, researchers and friends Lukas Hahn, Frederik Hasecke, Martin Alsfasser and Ido Freeman at Aptiv. I very much appreciate the fruitful discussions with you and your helpful feedback in every way.

A big thank you to my friends Cedric, Daniel, Jannik, Lukas, Patrick and Sarvin who supported and motivated me during this exciting time.

Last, I would love to thank my mother for supporting me in everything in my life as well as my wonderful wife Amrei. Amrei, you had the most difficult time with me. You have supported me in hard times and stood by me when I was almost going crazy.

Thanks to Kira Maag and Tobias Riedlinger for proofreading this thesis.

Foreword

Parts of this thesis have already been published or are incorporated in the following works.

Parts of chapter 2 are published in the following work. The main contribution of this work is not made by the author of this dissertation.

- **Matthias Rottmann, Pascal Colling, Thomas-Paul Hack, Robin Chan, Fabian Hüger, Peter Schlicht, Hanno Gottschalk**
  "Prediction Error Meta Classification in Semantic Segmentation: Detection via Aggregated Dispersion Measures of Softmax Probabilities",
  International Joint Conference on Neural Networks (IJCNN), 2020. ©2020 IEEE

Parts of chapter 3 are published in the following work.

- **Pascal Colling, Lutz Roese-Koerner, Hanno Gottschalk, Matthias Rottmann**
  "MetaBox+: A new Region Based Active Learning Method for Semantic Segmentation using Priority Maps",
  International Conference on Pattern Recognition Applications and Methods (ICPRAM), 2021. ©2021 SCITEPRESS

Parts of chapter 4 are published in the following works.

- **Pascal Colling, Matthias Rottmann, Lutz Roese-Koerner, Hanno Gottschalk**
  "False Positive Detection and Prediction Quality Estimation for LiDAR Point Cloud Segmentation",
  International Conference on Tools with Artificial Intelligence (ICTAI), 2021. ©2021 IEEE
- **Pascal Colling, Matthias Rottmann, Lutz Roese-Koerner, Hanno Gottschalk**
  "Prediction Quality Meta Regression and Error Meta Classification for Segmented Lidar Point Clouds",
  International Journal on Artificial Intelligence Tools (IJAIT), 2022. (submitted)

Parts of chapter 5 are published in the following work and filled as patent applications. In addition to the patent application names, the underlying record of invention (ROI) title is given.

- **Pascal Colling, Dennis Müller, Matthias Rottmann**
  "HD Lane Map Generation Based on Trail Map Aggregation",
  Intelligent Vehicles Symposium (IV), 2022. ©2022 IEEE

- **Pascal Colling, Peet Cremer**
  "Methods and Systems for Estimating Lanes for a Vehicle",
  ROI title: Lane Map Aggregation with a low-cost sensor system,
  European Patent Application EP21200267.9, 2021.

- **Pascal Colling, Peet Cremer**
  "Methods and Systems for Estimating Lanes for a Vehicle",
  ROI title: Cross-checking for Lane Map Aggregation,
  European Patent Application EP21200261.2, 2021.

- **Pascal Colling, Dennis Müller, Lutz Roese-Koerner, Christian Nunn**
  "Method for Generating High Definition Maps, and Cloud Server and Vehicle",
  ROI title: Creation of HD Lane Maps based on Trail Map Aggregation,
  Great Britain Patent Application GB2201080.5, 2022.

- **Lutz Roese-Koerner, Dennis Müller, Pascal Colling, Christian Nunn**
  "Method of determining a point of interest and/or a road type in a map, and related cloud server and vehicle",
  ROI title: A System and a Method for Classification of Points of Interest and Road Types based on Trail Maps,
  Great Britain Patent Application GB2201078.9, 2022.

Contents

1 Introduction

In 2012, Alex Krizhevsky achieved outstanding performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVR) [36]. An achievement which brought increased attention to the (now very popular) field of deep learning (DL). The used DL model, a convolutional neural network (CNN) named AlexNet [73] had an error that was 10.8 percentage points lower than that of the runner-up. Even if the foundations and concepts of artificial intelligence (AI) and DL go back to the 1950s [87], AlexNet marks the start of a new DL era. Novel models motivated by AlexNet have been developed and the areas of application have increased and become more intensive. This also includes applications in the automotive industry. Advanced driver assistant systems (ADAS) support the driver and make driving safer. Such systems are for example adaptive cruise control, blind spot monitor, collision avoidance system, lane centering and traffic sign recognition [65]. Most of such systems are based on DL models that detect lane markings or objects such as vehicles and humans from camera images as well as object detection and distance measurement with radar data. Automotive manufacturer and suppliers are working towards automated driving (AD) in which those ADAS come together with additional future developments. Further object recognition tasks which are studied for AD are semantic segmentation of images and lidar point clouds, for instance. In summary, many recognition tasks for multiple sensors modalities are considered in AD. For the remainder of this thesis, AD is referred to ADAS and AD, for simplicity.

Although DL models achieve outstanding performance, they still tend to prediction errors. Furthermore, DL models are referred to as black boxes [9] i.e., they are often non-transparent and their predictions are not traceable by humans. This is even worse in safety relevant applications such as AD. In addition to a high accuracy of DL models, uncertainty quantification (UQ) [1, 62] is essential. UQ has an important role in decision-making and during optimization of the DL model. It is important in several aspects: using DL models in a real-time system of AD requires UQ to identify prediction errors that can occur. Furthermore, it is highly desirable to represent uncertainty in any AI-based system in a trustworthy manner. When developing or training a new DL model, UQ helps to identify specific classes, samples or scenarios that are not really learned yet. UQ helps to optimize the model training. One

particular example is active learning, where a query function is often based on uncertainties, i.e., data is queried with the highest uncertainty.

Semantic segmentation of images is the task of the pixel-wise classification. A predicted segment is then an area of neighboring pixels that belong to the same predicted class. The method presented in [24, 113] called MetaSeg, is a method to detect false positive segments and to estimate the prediction quality segment-wise. A false positive segment is a predicted segment, that does not overlap with a ground truth segment of the same class. Prediction quality estimation is the estimation of the segment-wise intersection over union (IoU)[63] between a predicted and a ground truth segment of the same class. The IoU is a commonly used evaluation metric in semantic segmentation. Based on the CNN's prediction of an image, MetaSeg computes pixel-wise uncertainty measures and aggregates them on a predicted segment level. The aggregated data is then used to train a classification and a regression model for false positive detection and prediction quality estimation, respectively. MetaSeg was one of the first works providing uncertainty quantification in semantic segmentation of images and also goes a step beyond other approaches (cf. [68, 58] for instance) by processing uncertainty measures and meta data from the prediction to detect false positive segments and to have a general segment-wise prediction quality estimation.

The possibility of false positive detection and prediction quality estimation as well as using uncertainty measures and meta data of a prediction, i.e., additional information that can be derived from a prediction, motivates the investigation in several aspects. To be more precise, this includes DL models and aspects in and for semantic segmentation in AD. This leads to the main contributions of this work as follows.

The first main contribution is using the prediction quality estimation of MetaSeg in an active learning (AL) [120] strategy for semantic segmentation of images. Since DL models learn from annotated data, in general a large amount of annotated and varying data is required to achieve high accuracy of DL models. The annotation of data is time consuming and expensive, especially for semantic segmentation of images. The heart of an AL strategy consist of a query function, that selects the data to be labeled. Query functions are mostly based on the CNN's predictions and are uncertainty-based [140, 46, 114, 54, 4]. By defining a query function based on prediction quality estimation, this goes a step further, since the prediction quality estimation includes processed uncertainty measures to identify prediction errors. Thus, the query function aims to select images that the network cannot predict correctly. Apart from this, a new and simple cost estimation approach is incorporated to be cost efficient, which is an important role in real-world applications.

Moreover, the novel AL strategy queries only image regions for annotation, to be even more cost efficient. For evaluation of the method, exhaustive experiments have been made and new evaluation metrics are introduced. This yields a novel AL strategy for achieving high accuracy with low costs by selecting image regions where the highest prediction errors occur while having low estimated annotation costs.

Another application of MetaSeg and the second main contribution is an adaptation of MetaSeg by extending and further developing it for semantic segmentation of lidar point cloud data. First models for semantic segmentation in this area go back to 2017 [108, 109, 144], while the first publicly available data set for semantic segmentation of lidar point clouds was released in 2019 [35]. Since then, models with higher accuracy have continuously been developed and published e.g., [91, 30, 151, 146] but there are only a very few works about uncertainty quantification [30]. To this end, the adaptation of MetaSeg to lidar data presents one of the first approaches for UQ in lidar point cloud segmentation. In addition, it should be emphasized that this new approach goes significantly beyond other approaches [30] by detecting false positive segments and providing a segment-wise prediction quality estimation. For this purpose, the point cloud data is projected to a 2D representation. An adaptation of MetaSeg is applied to the 2D representations, while the features of the point cloud are taken into account. Finally, the results of the false positive detection and prediction quality estimation are re-projected to the point cloud.

To enable real-time recognition of the environment in AD, other recognition tasks are used in addition to semantic segmentation. This includes object detection i.e., the task of detecting road users such as vehicles or pedestrians in lidar point clouds. However, segmentation or detection may still fail because, for example, errors or uncertainties are detected in the prediction, the sensors fail for a short time or are occluded. Regarding such scenarios, high definition (HD) maps are becoming increasingly important [5, 106]. These maps include accurate information about the road system and semantic information. By localizing the vehicle and the availability of an HD map of the current location, it is still possible to localize the vehicle in the environment. That means, even if a sensor fails or is occluded, the position of the vehicle w.r.t. the environment is known. Thus, HD maps provide a backup solution and are often referred to as an additional sensor information. Generation of HD maps is a challenging task. In order to have them available almost everywhere, an automated method based on sensor data is required, where the sensor system can be equipped in production series vehicles. Current approaches [89, 88, 59, 153] do not take this fact into account or lack of the application in real-world scenarios. Furthermore, most approaches work only in specific scenarios. A fully automated procedure for HD map creation is presented and is the third main contribution in this

thesis. For this purpose, vehicles detected and tracked by a DL model are aggregated w.r.t. to uncertainty measures that can be derived from detection and tracking. By aggregating data from multiple object detection and tracking predictions and processing this data, further possible detection errors can be filtered out. Applying a novel extraction method from the aggregated data yields HD maps including lanes.

In summary, the main contributions of this thesis are an active learning method based on prediction quality and annotation cost estimation, a method for false positive detection and prediction quality estimation in segmented point clouds as well as a procedure to create HD maps based on the predictions of DL models w.r.t. uncertainty measures.

The structure of this thesis is as follows. The fundamentals in chapter 2 give an overview of the basic material that is employed in the remainder of that thesis. That includes the sensor modalities, a review of neural networks and a presentation of DL models that are used in this thesis: semantic segmentation of images and lidar point clouds as well as object detection and tracking, also for lidar point clouds. The fundamentals conclude with a review of uncertainty quantification and MetaSeg. The first main chapter, chapter 3 is about the new active learning method for semantic segmentation of images. After reviewing the concept of AL and the annotation process of images for semantic segmentation, the related work is provided. Afterwards, the new active learning method that employs MetaSeg and includes an annotation effort estimation is given, followed by experiments and result. The adaptation and extension of MetaSeg to lidar point cloud semantic segmentation is presented in chapter 4. After the introduction and reviewing related work, the method for false positive detection and prediction quality estimation for segmented point clouds is presented. Numerical experiments with in-depth studies to several aspects of the method are provided. Chapter 5 is about the new procedure for HD lane map generation. First, the need of HD maps and state-of-the-art methods are shown. The details for the procedure to create HD maps based on detected and tracked vehicles are provided, followed by numerical experiments. Moreover, an outlook to future work for a next generation HD map creation and update system is presented. The thesis ends with a conclusion and discussion of the findings in chapter 6. The main contributions of this work and the related results are summarized. Future work and how the new methods can be combined and how they improve each other, is pointed out.

2 Fundamentals

This chapter gives an overview of basic material that is employed in the remainder of this thesis. The fundamentals start with section 2.1, a presentation of sensor and data types that are commonly used in AD. These are camera, lidar, radar and HD maps. Afterwards, in section 2.2, the concept of neural networks will be explained. Thereafter, in section 2.3, the recognition tasks to be solved by means of neural networks are presented: semantic segmentation of camera and lidar data as well as lidar-based object detection and tracking. Section 2.4 is about uncertainty quantification and presents MetaSeg, a method to detect prediction errors and to estimate the prediction quality of semantic segmentation of images.

2.1 Sensor and Data Types

In AD, multiple sensor and data types are of interest, such as camera, lidar and radar, which will be explained in the following. A survey about sensor modalities is for instance given in [142]. Furthermore, the environment can also be represented in HD maps, which are basically a top-down view of a location (birds-eye-view) containing information about specific classes given in a global coordinate system, accurate up to a few centimeters.

Camera. Cameras are widely known and are one of the most commonly used types of sensors in AD. An image captures the semantic environment, similar to how humans see it. They have some disadvantages under bad weather conditions like rain or fog as well as at night. Their widespread application in AD is also supported by their comparably low price and ease of processing. A colored image is a three dimensional matrix $w \times h \times 3$, where w and h define the image width and height, respectively, as well as the intensity of elementary colors red, green and blue, represented in the name red-green-blue (RGB) image. The higher the resolution of a camera, the more details can be captured in an image. Other types of cameras might record gray-scale, red-pixel or depth images. In this thesis, only RGB images

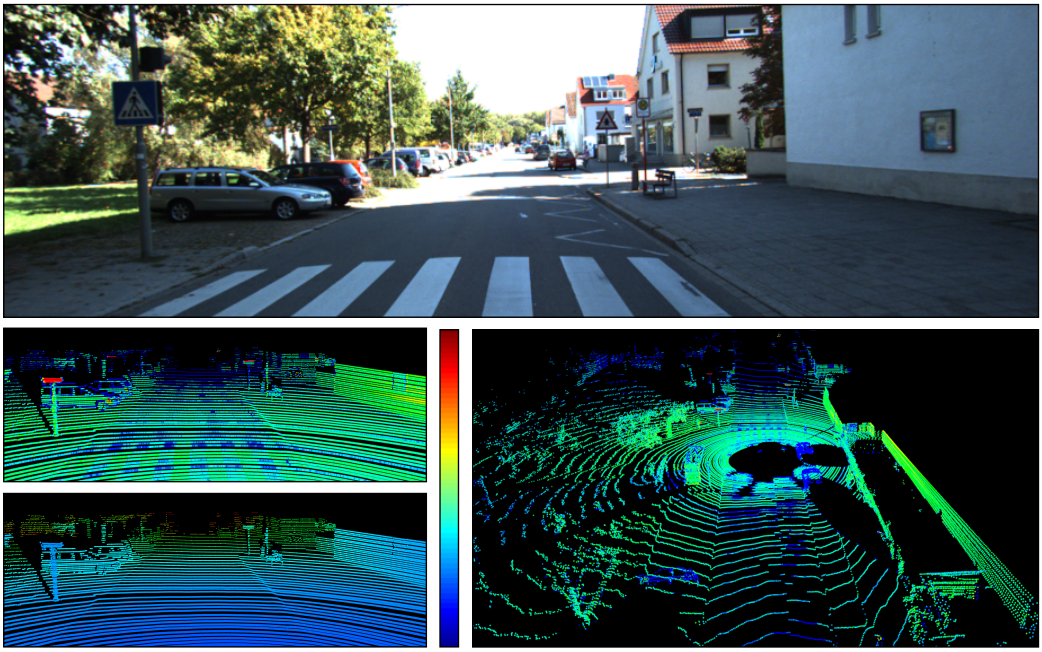


Figure 2.1: The top part shows a camera image while the lower part shows multiple views of the corresponding lidar point cloud: on the left site, the ego perspective is shown, where in the upper part the intensity is visualized and in the lower part the range as euclidean distance to the ego vehicle. In both figures, the values are normalized to 1 w.r.t. to their highest values. Higher values are represented in red and lower ones in blue. The bottom right part shows the whole point cloud to this example, colored by the intensity values.

are considered. An example of an image from the ego perspective from the vehicle is shown in figure 2.1 (top).

Lidar. Lidar, short for *light detection and ranging* is a method that uses light in form of a pulsed laser to measure range. There exist mainly two types of lidar sensors in AD: rotating and solid state lidar sensors. Here only rotating lidar sensors are considered that are characterized by the following:

- number of laser beams, also known as channels or scan lines, which are arranged vertically,
- vertical field of view (FOV), given in degree ψ: alignment of the scan lines, split into in the upper ψ_{up} and lower ψ_{down} FOV, with $\psi = \psi_{up} + \psi_{down}$,
- horizontal angular resolution ϕ in degree: angle within the rotation between time t and $t + 1$ of the laser scans,

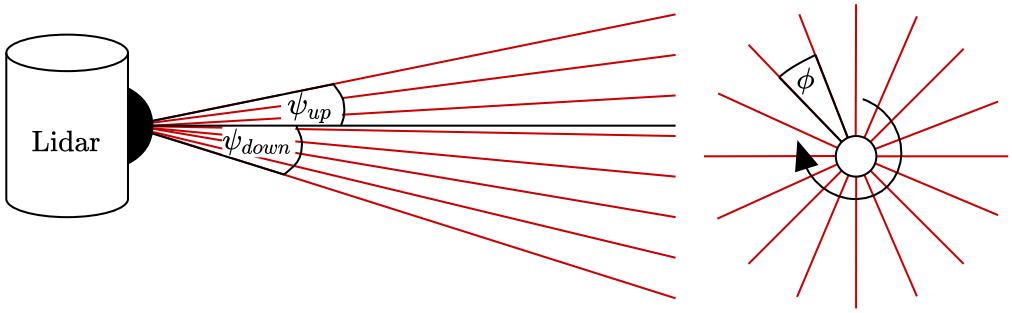


Figure 2.2: Illustration of a rotating lidar sensor. Left: the side view of a lidar sensor, consisting of 8 scan lines. Right: the top-down view to illustrate the rotation.

- frequency (Hz) : defines the number of frames per second, a frame consists of the recorded points by a $360°$ rotation.

By means of vertical and horizontal alignment of a scan line as well as the required time until the pulsed laser is reflected, the range and thus the point p_j of the scan line can be determined in Cartesian coordinates (x_j, y_j, z_j). In addition, most lidar sensors measure the intensity or remission i_j of the reflection. The range r_j of each point w.r.t. to the sensor is determined as the euclidean distance, i.e.,

$$r_j = \sqrt{x_j^2 + y_j^2 + z_j^2}\,, \tag{2.1}$$

for point p_j. Thus, each point $p_j,\ j = 1, \ldots, n$ of a point cloud $\mathbf{p}$ is given as $p_j = (x_j, y_j, z_j, i_j, r_j)$ with $x_j, y_j, z_j \in \mathbb{R}$ and $i_j, r_j \in \mathbb{R}_+$ while n is the number of points in the point cloud. An example of a point cloud is given figure 2.1 (bottom). Furthermore, an illustration of a lidar sensor is shown in figure 2.2.

The higher the number of scan lines and the smaller the angular resolution ϕ are, the higher the resolution of the lidar sensor. Similar to cameras, lidar sensors are affected by some weather conditions. Rain, wet ground or fog affect the pulsed laser beams yielding scatter points in those regions. High resolution lidar sensors are too expensive to equip them in production vehicles, as they usually cost several thousand dollars. Compared to images, point clouds are sparse but provide highly accurate range information. They also provide intensity values which indicate the material, e.g., a vehicle reflected points have higher intensity values than those of a wall.

Radar. Radar, short for *radio detection and ranging* works by emitting electromagnetic (EM) radio waves that are reflected when they hit an obstacle. Since the emitted radio waves are larger compared to the EM waves of the lidar, a radar sensor

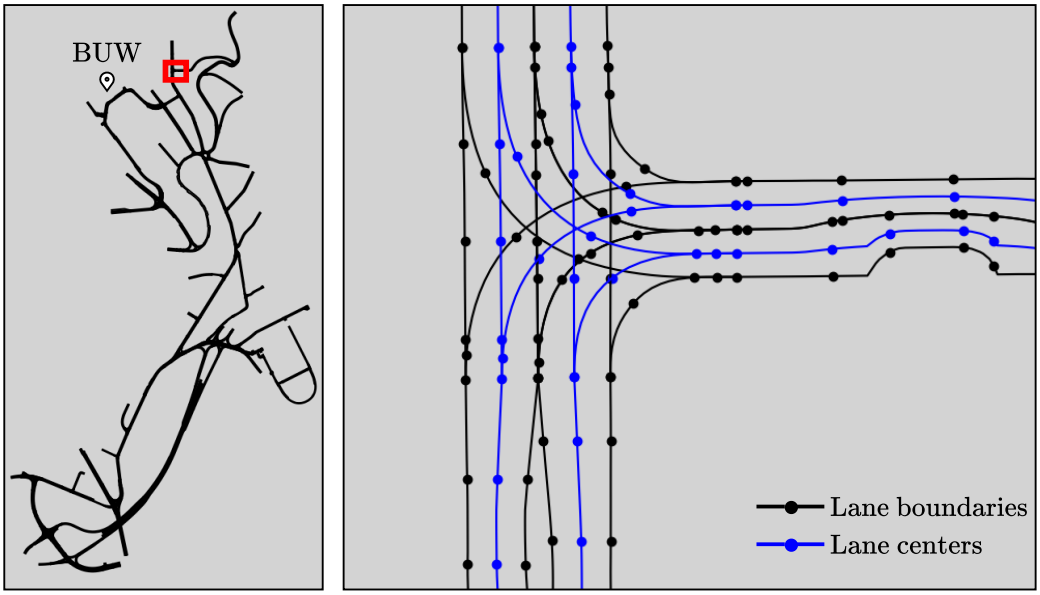


Figure 2.3: HD map containing information about lanes in terms of lane boundaries and lane centers. The left panel shows an HD map near and around the University of Wuppertal (BUW). The right panel shows the highlighted sub-region (red) from the left panel.

works even in bad weather e.g., rain or fog and by night. Advantages are to measure the velocity of objects by the Doppler effect and to achieve longer detection ranges than other sensors. Moreover, a radar sensor is cheap, similar to camera sensors. More information is given in [37, 132].

High Definition Maps. Compared to common maps e.g., standard definition (SD) maps or maps from Google (Google Maps), HD maps are highly accurate and precise at a centimeter level [5, 106]. HD maps contain features such as the road shape and lanes, lane markings, traffic signs and other important features. These features in general are given and available in a structured data format. An example of an HD map including lane information is shown in figure 2.3.

2.2 Artificial Neural Networks

Artificial neural networks or neural networks (NN) are machine learning (ML) models, while in turn ML is a sub-field of artificial intelligence (AI). AI is the theory and development of computer systems able to perform tasks in general requiring human intelligence. ML basically describes an algorithm that improves through

experience and by the use of data. Thus, also neural networks do so and require annotated data to learn from. NN will be considered in more detail in the following. It is assumed that the reader is familiar with the main concept of ML. Additional material is for instance given in [121] which is also the work this section follows. It attempts to explain the core concepts and the elements used in this work.

Artificial neural networks are inspired by neural networks in the brain. Simplified, a neural network in the brain consists of many neurons that are interconnected in a complex network. NN aim to imitate the functional patterns of a brain and to make decisions based on learned patterns from data.

2.2.1 Feedforward Neural Networks

An artificial neural network is a computational model, that can be described as a graph $G = (V, E)$ whose nodes $v \in V$ correspond to neurons and edges $e \in E$ to the connection between the neurons. Furthermore, the edges are weighted with a function $w : E \to \mathbb{R}$. Here, only *feedforward* networks are considered, which means the underlying graph does not contain cycles. In general, a neural network is organized in *layers*. That means, the set of nodes can be decomposed into a union of non-empty disjoint subsets, i.e., $V = \dot{\bigcup}_{k=0}^{K} V_k$, such that every edge in E connects some node in V_{k-1} to some node in V_k, for some $k = 1, \ldots, K$.

The signal or also named output at each neuron is modeled as a scalar function $\sigma : \mathbb{R} \to \mathbb{R}$, called *activation function*. More details about the activation function will be given later.

The first layer V_0 is the input layer, consisting of m_0 neurons. The input of the NN is $\mathbf{x} = (x_1, \ldots, x_{m_0})^T \in \mathbb{R}^{m_0}$ which is also the output of the first layer, i.e., the output of neuron $i \in \{1, \ldots, m_0\}$ is x_i. The i-th neuron in layer k is denoted by $v_{k,i}$ and its output $o_{k,i}(\mathbf{x})$. In layer k, $m_k \in \mathbb{N}$ is the number of neurons or nodes. Furthermore, every layer $k, 1 \leq k < K$ contains a *bias* node, whose output is a constant, for brevity this constant is 1. A bias node in layer k is connected to every node in layer $k + 1$.

Suppose the outputs of the neurons of a deeper layer $k-1, k > 1$ are calculated, then the outputs of the neurons in layer k are calculated as follows. For a node $v_{k,i} \in V_k$, i.e., neuron i in layer k, the input is denoted with $a_{k,i}(\mathbf{x})$ when the network is fed with the input vector $\mathbf{x}$. Moreover, $w_{k-1,i,j}$ is the weight of edge $(v_{k-1,j},\ v_{k,i})$, so the edge weight from neuron j to considered neuron i. If an edge $(v_{k-1,j},\ v_{k,i})$ does not exist, a phantom edge with $w_{k-1,i,j} = 0$ is added for brevity. Then,

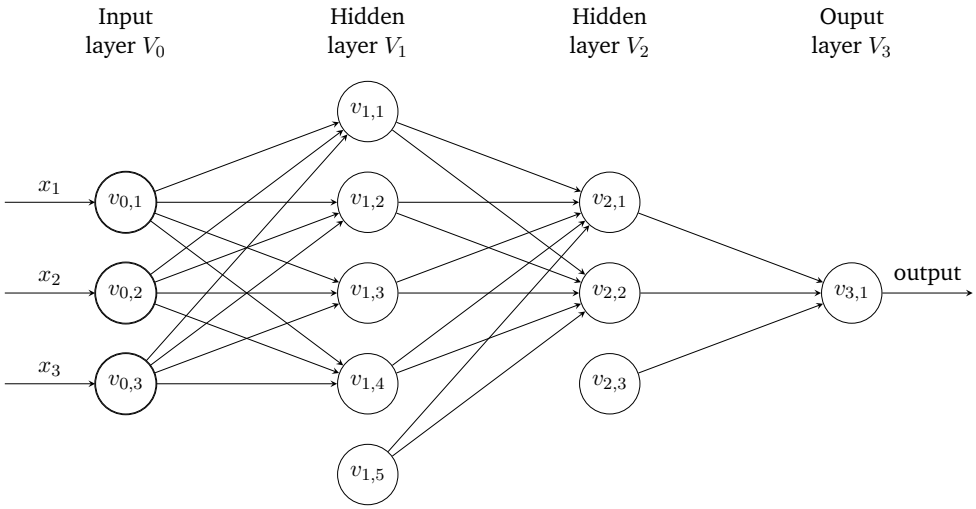


Figure 2.4: An illustration of a neural network of depth 3, size 12 and width 5.

$$a_{k,i}(\mathbf{x}) = \sum_{j=1}^{m_{k-1}} w_{k-1,i,j}\, o_{k-1,j}(\mathbf{x}) \tag{2.2}$$

and

$$o_{k,i}(\mathbf{x}) = \sigma(a_{k,i}(\mathbf{x}))\,. \tag{2.3}$$

Thus, the input $a_{k,i}$ to a neuron i in layer k (node $v_{k,i}$) is a weighted sum of the outputs $o_{k-1,j}$ of the neurons $j \in \{1, \ldots, m_{k-1}\}$ in the previous layer $k-1$ (nodes $v_{k-1,j} \in V_{k-1}$) that are connected to $v_{k,i}$. Finally, the output $o_{k,i}(\mathbf{x})$ of the node is determined by applying the activation function σ to the weighted sum. Note, the activation function can be different in every layer.

Layers $V_1, \ldots, V_{K-1}$ are called *hidden layers*. The last layer V_K is called the output layer. The output of the last layer is also the output of the neural network and defined as vector $\hat{\mathbf{y}} \in \mathbb{R}^{m_K}$. K is referred to as the number of layers in the network (excluding V_0), or the *depth* of the network. The *size* of the network is defined by the number of nodes, i.e., $|V|$. The *width* of the network is $\max_k |V_k|$. An illustration of a neural network of depth 3, size 12 and width 5 is given in figure 2.4. Note that there are neurons in both hidden layers that have no incoming edges. These neurons, the bias nodes, will output the constant values 1. In the case where every node is connected to all neurons of the next layer, then this structure is called *fully-connected feedforward network*. Note, a neural network with a high depth (i.e., a high number of hidden layers) is referred to as a deep learning model.

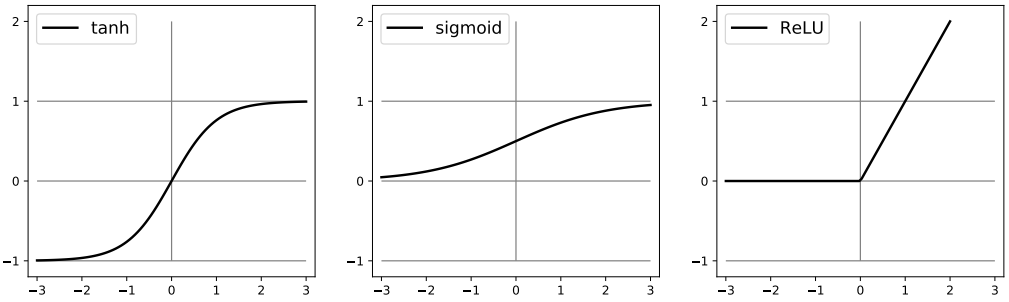


Figure 2.5: Activation functions. The x-axis corresponds to the input of an activation function and the y-axis to the output.

The structure of NN has been presented. The weights over the edges w (see equation (2.2)) can be modified. In the following, the weights are denoted by $\mathbf{w}$. In applications, a structure of a NN is assumed to be given, and the weights $\mathbf{w}$ have to be fitted to a given data set since, as already mentioned, NN learn from data. In application and when these weights have been fitted, then an input $\mathbf{x}$ is fed into the NN and forwarded through all layers and an output or prediction $\hat{\mathbf{y}}$ is made. Outstanding is how the best weights $\mathbf{w}$ are calculated and what 'best' means in this sense. This requires first some more details about the activation function and a repetition of basics in learning theory and statistics. Then, the methods for computing the weights $\mathbf{w}$, the gradient descent and backpropagation algorithm, are presented in details.

Activation Function. As presented previously, the activation or output of a neuron is based on applying the activation function (see equation (2.3)) to the weighted sum of the neuron's input (see equation (2.2)). First activation functions go back to the threshold function $\sigma_{\text{th}}(a) = \mathbb{1}_{\{a>0\}}$ and the sigmoid function $\sigma_{\text{sigmoid}}(a) = 1/(1+\exp(-a))$ as well as hyperbolic tangent $\sigma_{\tanh}(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$. For the mentioned gradient descent algorithm, the chosen activation function must be differentiable. Furthermore, the derivative should be easy to compute since the gradients are required. Both requirements are met by the *rectified linear unit* (ReLU) function [57]

$$\sigma_{\text{ReLU}}(a) = \max\{0, a\} \tag{2.4}$$

when setting the derivative for $a = 0$ to 0. The activation functions are shown in figure 2.5. The figure illustrates that the ReLU function consist of two piece-wise linear functions, which make the computation of the derivative easy.

Another activation function, that is especially often used in the last layer (for classification tasks), is the *softmax function*. Assuming, the class label $c \in \mathcal{C} = \{1, \dots, q\}$ of an input $\mathbf{x}$ has to be predicted, i.e., $\mathbf{x}$ has to be classified. Then the output of the NN consists of q neurons, while every neuron indicates the class probability to which the input $\mathbf{x}$ belongs to. The corresponding label is given by a one-hot vector $\mathbf{y} = (y_1 = 0, y_2 = 0, \dots, y_c = 1, \dots, y_q = 0)^T$. For $\mathbf{a} \in \mathbb{R}^q$ the corresponding softmax values are given by

$$\sigma_{\text{softmax}}(\mathbf{a})_j = \frac{e^{a_j}}{\sum_{c=1}^{q} e^{a_c}} \,. \tag{2.5}$$

Thus, it is $\sigma(\mathbf{a})_j \in [0, 1]$ and $\sum_{j=1}^{q} \sigma(\mathbf{a})_j = 1$, which meets the properties of a probability distribution of the prediction.

Empirical Risk Minimization and Maximum Likelihood Estimation. The main principles of learning theory, known as empirical risk minimization is presented, followed by showing the similarities to maximum likelihood estimation, which is a parameter estimation method in statistics. Finally, the transition back to NN will be made.

Given is a set of input samples $\mathcal{X}$ and a set of labels $\mathcal{C}$. The training set is $\mathcal{S} = ((\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)}))$ with m samples in $\mathcal{X} \times \mathcal{C}$. Every input sample $\mathbf{x}^{(i)}$ has a label $y^{(i)} \in \mathcal{C}$ for $i = 1, \dots, m$. Furthermore, the training set is generated by a probability distribution $\mathcal{D}$ over $\mathcal{X}$ representing the environment. Now, a learner is requested to output a prediction rule $h_{\mathcal{S}} : \mathcal{X} \to \mathcal{C}$ based on the training set $\mathcal{S}$, which is also named hypothesis or classifier. It is assumed that a correct labeling function $f^* : \mathcal{X} \to \mathcal{C}$ exists, i.e., $y^{(i)} = f^*(\mathbf{x}^{(i)})$, $i = 1, \dots, m$. The error of the prediction rule $h_{\mathcal{S}}$ is the probability of randomly picking a sample $\mathbf{x}$ generated by $\mathcal{D}$, such that $h_{\mathcal{S}}(\mathbf{x}) \neq f^*(\mathbf{x})$. More precisely,

$$L_{\mathcal{D},f^*}(h_{\mathcal{S}}) := \mathop{\mathbb{P}}_{\mathbf{x}\sim\mathcal{D}}[h_{\mathcal{S}}(\mathbf{x}) \neq f^*(\mathbf{x})] := \mathcal{D}(\{\mathbf{x} : h_{\mathcal{S}}(\mathbf{x}) \neq f^*(\mathbf{x})\}) \,. \tag{2.6}$$

The probability distribution $\mathcal{D}$ and the labeling function f^* are unknown to the learner. The task is to find a prediction rule based on the training data $\mathcal{S}$ that minimizes the error. The training error or *empirical error* of a learner is defined as

$$L_{\mathcal{S}}(h_{\mathcal{S}}) = \frac{|i \in \{1, \dots, m\} : h_{\mathcal{S}}(\mathbf{x}^{(i)}) \neq y^{(i)}|}{m} \,. \tag{2.7}$$

The learning paradigm in which a prediction rule $h_{\mathcal{S}}$ minimizes the empirical risk $L_{\mathcal{S}}(h_{\mathcal{S}})$ is called *empirical risk minimization* (ERM).

Minimizing the empirical risk might lead to overfitting. That means, the hypothesis is excellent on the training data but very poor on unseen data. A solution to this is to apply the ERM learning rule over a restricted search space, more precisely over a *hypothesis class* $\mathcal{H}$. Every hypothesis $h \in \mathcal{H}$ is a mapping function $h : \mathcal{X} \to \mathcal{C}$. For a given hypothesis class $\mathcal{H}$ and training samples $\mathcal{S}$, the $\text{ERM}_\mathcal{H}$ learner uses the ERM rule to choose $h \in \mathcal{H}$ with the lowest error, i.e.,

$$\text{ERM}_\mathcal{H}(\mathcal{S}) \in \operatorname*{argmin}_{h \in \mathcal{H}} L_\mathcal{S}(h)\,. \tag{2.8}$$

Since the prediction rule (or 'hypothesis') is restricted or biased to the hypothesis class, this is called inductive bias.

Maximum likelihood estimation is a method of estimating the parameters of a probability distribution, based on observed data from a training set $\mathcal{S}$ [2]. This is achieved by maximizing a likelihood function so that the observed data from $\mathcal{S}$ are most probable under the assumed statistical model. As it will be shown, the maximum likelihood estimator is an empirical risk minimizer. For given parameters θ and a sample $\mathbf{x}$ the loss is defined as

$$\ell(\theta, \mathbf{x}) = -\log(\mathcal{P}_\theta(\mathbf{x}))\,, \tag{2.9}$$

which is the negative log likelihood of $\mathbf{x}$ assuming the data is distributed according to $\mathcal{P}_\theta$. Based on this definition it follows, that minimizing the empirical risk is equivalent to maximizing the log likelihood:

$$\operatorname*{argmin}_\theta \sum_{i=1}^{m} (-\log(\mathcal{P}_\theta(\mathbf{x}^{(i)}))) = \operatorname*{argmax}_\theta \sum_{i=1}^{m} \log(\mathcal{P}_\theta(\mathbf{x}^{(i)}))\,, \tag{2.10}$$

for $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ independent and identically distributed, which also means ERM is equivalent to maximum likelihood principles.

Then, the true risk (or errors) of θ becomes

$$\mathbb{E}_\mathbf{x}[\ell(\theta, \mathbf{x})] = -\sum_\mathbf{x} \mathcal{P}(\mathbf{x}) \log(\mathcal{P}_\theta(\mathbf{x})) \tag{2.11}$$

$$= \underbrace{\sum_\mathbf{x} \mathcal{P}(\mathbf{x}) \log\left(\frac{\mathcal{P}(\mathbf{x})}{\mathcal{P}_\theta(\mathbf{x})}\right)}_{D_{\text{KL}}(\mathcal{P}(\mathbf{x}) || \mathcal{P}_\theta(\mathbf{x}))} + \underbrace{\sum_\mathbf{x} \mathcal{P}(\mathbf{x}) \log\left(\frac{1}{\mathcal{P}(\mathbf{x})}\right)}_{H(\mathcal{P}(\mathbf{x}))}\,. \tag{2.12}$$

It is assumed that the data is distributed according to distribution $\mathcal{P}(\mathbf{x})$ and not necessarily to $\mathcal{P}_\theta(\mathbf{x})$. The true risk as presented in equation (2.12) consist of the Kullback-Leibler divergence (first term) which measures the distance between two

probabilities and the (Shannon) entropy [122] (second term). For discrete variables, the Kullback-Leibler (KL) divergence is always non-negative and is equal to 0 if the two distributions are equal. It follows that the true risk is minimal when $\mathcal{P}_\theta = \mathcal{P}$

The functional behavior of a NN is denoted by $\bar{f}_{\mathbf{w}}(\mathbf{x})$, where $\mathbf{w}$ are the weights and $\mathbf{x}$ is the input. The task is to classify the input $\mathbf{x}$. In the output layer, the softmax function (equation (2.5)) is the activation function of every neuron. Thus, the prediction of $\mathbf{x}$ is given in probabilities $\hat{\mathbf{y}}^{prob} \in [0,1]^q$, where q is the number of possible classes. The true label is $\mathbf{y} \in \{0,1\}^q$ i.e., one-hot encoded. As stated above, a loss function is required to determine the loss or error between the prediction $\hat{\mathbf{y}} \in \{0,1\}^q$ and the label $\mathbf{y}$. By choosing the cross-entropy as loss function for predicting the probabilities of $\mathbf{x}$ by the NN $\bar{f}_{\mathbf{w}}$ with $\hat{\mathbf{y}}^{prob} = \bar{f}_{\mathbf{w}}(\mathbf{x})$ and true class label $\mathbf{y}$, i.e.,

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{c=1}^{q} y_c \log(\hat{y}_c^{prob}) \,, \tag{2.13}$$

which is a generalization of equation (2.11), this yields the best weights $\mathbf{w}$ in terms of an optimal parameter to maximize the likelihood of observed data under the model $\bar{f}_{\mathbf{w}}$. The softmax outputs are the probabilities and the Bayesian rule is applied to get the final prediction, i.e., applying the argmax to the probabilities

$$f_{\mathbf{w}}(\mathbf{x}) := \operatorname*{argmax}_{c \in \mathcal{C}} \hat{\mathbf{y}}^{prob} \,. \tag{2.14}$$

In short summary, under the above assumption, the NN can be seen as statistical model [119] and it is optimized by minimizing the weights $\mathbf{w}$ under the loss function and using Bayesian rule. It remains to compute the optimal weights $\mathbf{w}$ by minimizing the loss function and using the gradient descent algorithm.

Stochastic Gradient Descent and Backpropagation. The triple (V, E, σ) is the *architecture* of the neural network. The function of the neural network is $f_{\mathbf{w}} : \mathbb{R}^{m_0} \to \mathbb{R}^{m_K}$, where m_0 is the input, m_K is the output dimension and weights $\mathbf{w}$. In order to fit the neural network to a data set $\mathcal{X} \times \mathcal{Y}$ with $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y},\ \mathbf{x} \in \mathbb{R}^{m_0}, \mathbf{y} \in \mathbb{R}^{m_K}$, the weights $\mathbf{w}$ have to be determined over the edges. In practice, this is done with the stochastic gradient descent (SGD) algorithm. The loss of predicting $f_{\mathbf{w}}(\mathbf{x}) = \hat{\mathbf{y}}$ with target label $\mathbf{y}$ is denoted by $\Delta(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y})$. For the following explanation of the SGD, Δ is the squared loss, i.e.,

$$\Delta(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = \frac{1}{2}\|f_{\mathbf{w}}(\mathbf{x}) - \mathbf{y}\|_2^2 = \frac{1}{2}\|\mathbf{o}_K - \mathbf{y}\|_2^2 \,. \tag{2.15}$$

Note, the output of the NN $f_{\mathbf{w}}(\mathbf{x}) = \hat{\mathbf{y}}$ can be written as the output of the last layer $\mathbf{o}_K$. However, any differentiable loss function can be used. The training process of the weights $\mathbf{w}$ of neural networks with SGD consists of the following two steps: first, a *forward pass* by applying the function $f_{\mathbf{w}}(\mathbf{x})$ of the neural network to input data $\mathbf{x}$ and computing the loss $\Delta(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y})$ to the target $\mathbf{y}$ and second, a *backward pass*

Algorithm 1: SGD for Neural Networks

Input: Layered graph (V, E), differentiable $\sigma : \mathbb{R} \to \mathbb{R}$

Set $\mathbf{w}^{(1)} \in \mathbb{R}^{|E|}$ close to 0, learning rate $\eta > 0$ and $i = 1$
while *Stopping condition is not met* **do**
  Sample $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$
  Calculate gradient $\mathbf{v}^{(i)} = \texttt{backpropagation}(\mathbf{x}, \mathbf{y}, \mathbf{w}^{(i)}, (V, E), \sigma)$
  Update $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \mathbf{v}^{(i)}$
  Set $i = i + 1$
end

Output: Best performing weights $\mathbf{w}^*$ on a validation set

Algorithm 2: Backpropagation

Input: Example $(\mathbf{x}, \mathbf{y})$, weights $\mathbf{w}$, layered graph (V, E), activation function $\sigma : \mathbb{R} \to \mathbb{R}$

Define: Graph layers $V_0, \ldots, V_K$ where $V_k = \{v_{k,1}, \ldots, v_{k,m_k}\}$, weight $w_{k,i,j}$ of $(v_{k,j}, v_{k+1,i})$ with $w_{k,i,j} = 0$ if $(v_{k,j}, v_{k+1,i}) \notin E$

`// Forward pass`
Set $\mathbf{o}_0 = \mathbf{x}$
for $k = 1, \ldots, K$ **do**
  for $i = 1, \ldots, m_k$ **do**
    Set $a_{k,i} = \sum_{j=1}^{m_{k-1}} w_{k-1,i,j}\ o_{k-1,j}$
    Set $o_{k,i} = \sigma(a_{k,i})$
  end
end
`// Backward pass`
Set $\boldsymbol{\delta}_K = \mathbf{o}_K - \mathbf{y}$
for $k = K - 1, K - 2, \ldots, 1$ **do**
  for $j = 1, \ldots, m_k$ **do**
    Set $\delta_{k,j} = \sum_{i=1}^{m_{k+1}} w_{k,i,j}\ \delta_{k+1,i}\ \sigma'(a_{k+1,i})$
  end
end

Output: Partial derivatives $\delta_{k,i}\ \sigma'(a_{k,i})\ o_{k-1,j}\ \forall$ edges $(v_{k-1,j},\ v_{k,i}) \in E$

where the gradient $\mathbf{v}$ of the loss is *backpropagated* through the network to update the weights $\mathbf{w}$. This is usually done until the accuracy of the network on validation data saturates. Gradient descent is an iterative optimization algorithm to find a local minimum of a differentiable function. The idea of minimizing a function (here the loss function) with gradient descent is to make iteratively steps in the opposite direction of the gradient of the loss function at the current point, because this is the direction of the steepest descent. The length of the step is controlled with the learning rate η. 'Stochastic' in this sense means, that the loss for predicting one input sample $\mathbf{x}$ is computed and not for all samples of the training set as it is done in gradient descent algorithm. The SGD and the backpropagation algorithm are given in algorithm 1 and algorithm 2, respectively.

Before the calculation of the gradient $\mathbf{v}$ in the backpropagation algorithm 2 is explained, a repetition of some vector calculus and the definition of partial derivative is needed. Moreover, in the following part the sigmoid activation function is used, i.e., $\sigma(a) = 1/(1 + \exp(-a))$.

Given an arbitrary function $f : \mathbb{R}^n \to \mathbb{R}$, the partial derivative w.r.t. the i-th variable at $\mathbf{w}$ is obtained by fixing the values of all $w_j,\ j \neq i$ which yields a scalar function $g : \mathbb{R} \to \mathbb{R}$ defined by $g(a) = f((w_1, \ldots, w_{i-1}, w_i + a, w_{i+1}, \ldots, w_n))$, and then taking the derivative of g at 0. For a function with multiple outputs, $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, the *Jacobian* of $\mathbf{f}$ at $\mathbf{w} \in \mathbb{R}^n$, denoted by $J_{\mathbf{w}}(\mathbf{f})$, is the $m \times n$ matrix whose (i, j) element is the partial derivative of $f_i : \mathbb{R}^n \to \mathbb{R}$ w.r.t. its j-th variable at $\mathbf{w}$ i.e., $\frac{\partial f_i}{\partial w_j}$. For $m = 1$ the Jacobian matrix is the gradient of the function (represented as a row vector). Then it holds:

- Let $\mathbf{f}(\mathbf{w}) = A\mathbf{w}$ for $A \in \mathbb{R}^{m \times n}$, then

$$J_{\mathbf{w}}(\mathbf{f}) = A\,. \tag{2.16}$$

- For every n, the function $\boldsymbol{\sigma} : \mathbb{R}^n \to \mathbb{R}^n$ applies the sigmoid function element-wise. That is, $\boldsymbol{\alpha} = \boldsymbol{\sigma}(\boldsymbol{\theta})$, with

$$\alpha_i = \sigma(\theta_i) = \frac{1}{(1 + e^{-\theta_i})}\,, \tag{2.17}$$

  for every $i = 1, \ldots, n$. Then, the Jacobian matrix $J_{\boldsymbol{\theta}}(\boldsymbol{\sigma})$ is a diagonal matrix whose (i, i) entry is $\sigma'(\theta_i)$ i.e., the derivative function of the scalar sigmoid function

$$\sigma'(\theta_i) = \frac{1}{(1 + e^{\theta_i})(1 + e^{-\theta_i})} = \sigma(\theta_i)\sigma(-\theta_i)\,. \tag{2.18}$$

In the following the notation

$$\operatorname{diag}(\boldsymbol{\sigma}'(\boldsymbol{\theta})) = J_{\boldsymbol{\theta}}(\boldsymbol{\sigma}) \tag{2.19}$$

will be used.

- The *chain rule* is applied to determine the partial derivatives of a composition of functions and can be written in terms of the Jacobian as follows. Given two function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbf{g} : \mathbb{R}^k \to \mathbb{R}^n$, the Jacobian of the composition function $(\mathbf{f} \circ \mathbf{g}) : \mathbb{R}^k \to \mathbb{R}^m$ at $\mathbf{w}$ is

$$J_{\mathbf{w}}(\mathbf{f} \circ \mathbf{g}) = J_{g(\mathbf{w})}(\mathbf{f}) \circ J_{\mathbf{w}}(\mathbf{g})\,. \tag{2.20}$$

  Applying the chain rule to $J_{\mathbf{w}}(\boldsymbol{\sigma} \circ \mathbf{g})$ with $\mathbf{g}(\mathbf{w}) = A\mathbf{w},\ A \in \mathbb{R}^{n\times k}$ and using equation (2.16) and equation (2.19) it is

$$\begin{aligned} J_{\mathbf{w}}(\boldsymbol{\sigma} \circ \mathbf{g}) &= J_{\mathbf{g}(\mathbf{w})}(\boldsymbol{\sigma}) J_{\mathbf{w}}(\mathbf{g}) \\ &= \operatorname{diag}(\boldsymbol{\sigma}'(A\mathbf{w}))A\,. \end{aligned} \tag{2.21}$$

For the description of the backpropagation algorithm, the set of nodes V is considered to be decomposed in layers $V = \dot{\bigcup}_{k=0}^{K} V_k$. For every k, $V_k = \{v_{k,1}, \ldots, v_{k,m_k}\}$ is the set of nodes with $m_k = |V_k|$.

For every layer k, the matrix $\mathbf{W}_k \in \mathbb{R}^{m_{k+1}\times m_k}$ gives a weight to every potential edge between V_k and V_{k+1}. If the edge exists in E then $w_{k,i,j}$ is the weight, according to $\mathbf{w}$, of the edge $(v_{k,j}, v_{k+1,i})$. Otherwise, a phantom edge is added with weight 0, i.e., $w_{k,i,j} = 0$. The phantom edge does not have an effect on the partial derivative w.r.t. the existing edge since all other weights are fixed. Thus, it can be assumed, that all edges exist i.e., $E = \bigcup_k (V_k \times V_{k+1})$.

Next, the partial derivatives w.r.t. to $\mathbf{W}_{k-1}$ i.e., the weights of the edges from V_{k-1} to V_k will be discussed. The outputs $\mathbf{o}_{k-1}$ of all neurons in V_{k-1} not depending on $\mathbf{W}_{k-1}$ are fixed numbers since all other values are fixed. In addition, $\ell_k : \mathbb{R}^{m_k} \to \mathbb{R}$ is the loss function of the sub-network defined by layers $V_k, \ldots, V_K$ as a function of the outputs of the neurons in V_k. The input of the neurons of V_k can be written as

$$\mathbf{a}_k = \mathbf{W}_{k-1}\mathbf{o}_{k-1} \tag{2.22}$$

and the output of the neurons of V_k is $\mathbf{o}_k = \boldsymbol{\sigma}(\mathbf{a}_k)$. That is $o_{k,j} = \sigma(a_{k,j})$ for every $j = 1, \dots, m_k$. Then the loss is obtained as a function g_k of $\mathbf{W}_{k-1}$, written as

$$\begin{aligned} g_k(\mathbf{W}_{k-1}) &= \ell_k(\mathbf{o}_k) && (2.23)\\ &= \ell_k(\boldsymbol{\sigma}(\mathbf{a}_k)) && (2.24)\\ &= \ell_k(\boldsymbol{\sigma}(W_{k-1}\mathbf{o}_{k-1}))\,. && (2.25) \end{aligned}$$

$\mathbf{W}_{k-1}$ can be rewritten as vector $\mathbf{w}_{k-1} \in \mathbb{R}^{m_{k-1}m_k}$ by transposing the concatenated rows of $\mathbf{W}_{k-1}$. Furthermore the matrix $\mathbf{O}_{k-1} \in \mathbb{R}^{m_k \times (m_{k-1}m_k)}$ is defined as

$$\mathbf{O}_{k-1} = \begin{pmatrix} \mathbf{o}_{k-1}^T & 0 & \cdots & 0 \\ 0 & \mathbf{o}_{k-1}^T & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{o}_{k-1}^T \end{pmatrix}. \tag{2.26}$$

Then $\mathbf{W}_{k-1}\mathbf{o}_{k-1} = \mathbf{O}_{k-1}\mathbf{w}_{k-1}$ and equation (2.25) can also be rewritten as

$$g_k(\mathbf{w}_{k-1}) = \ell_k(\boldsymbol{\sigma}(\mathbf{O}_{k-1}\mathbf{w}_{k-1}))\,. \tag{2.27}$$

Applying the chain rule (see equation (2.21)) and using the notation from above results in

$$\begin{aligned} J_{\mathbf{w}_{k-1}}(g_k) &= J_{\boldsymbol{\sigma}(\mathbf{O}_{k-1}\mathbf{w}_{k-1})}(\ell_k)\mathrm{diag}(\boldsymbol{\sigma}'(\mathbf{O}_{k-1}\mathbf{w}_{k-1}))\mathbf{O}_{k-1} && (2.28)\\ &= J_{\boldsymbol{\sigma}(\mathbf{o}_k)}(\ell_k)\mathrm{diag}(\boldsymbol{\sigma}'(\mathbf{a}_k))\mathbf{O}_{k-1}\,. && (2.29) \end{aligned}$$

With defining $\boldsymbol{\delta}_k := J_{\mathbf{o}_k}(\ell_k)$, the equation can be written as

$$J_{\mathbf{w}_{k-1}}(g_k) = (\delta_{k,1}\sigma'(a_{k,1})\mathbf{o}_{k-1}^T, \dots, \delta_{k,m_k}\sigma'(a_{k,m_k})\mathbf{o}_{k-1}^T)\,. \tag{2.30}$$

To this end, the gradient $\boldsymbol{\delta}_k = J_{\mathbf{o}_k}(\ell_k)$ of ℓ_k at $\mathbf{o}_k$ for every k has to be calculated, which will be done in a recursive manner.

For the last layer K, the squared loss is

$$\begin{aligned} \ell_K(\mathbf{o}_K) &= \Delta(\mathbf{o}_K, \mathbf{y}) && (2.31)\\ &= \frac{1}{2}\|\mathbf{o}_K - \mathbf{y}\|_2^2\,. && (2.32) \end{aligned}$$

This results in

$$\boldsymbol{\delta}_K = J_{\mathbf{o}_K}(\ell_K) = (\mathbf{o}_K - \mathbf{y})\,. \tag{2.33}$$

For $k < K$ it holds

$$\ell_k(\mathbf{o}_k) = \ell_{k+1}(\boldsymbol{\sigma}(\mathbf{W}_k\mathbf{o}_k))\,. \tag{2.34}$$

Applying the chain rule yields the gradient

$$\begin{aligned}
\boldsymbol{\delta}_k &= J_{\mathbf{o}_k}(\ell_k) && (2.35)\\
&= J_{\boldsymbol{\sigma}(\mathbf{W}_k\mathbf{o}_k)}(\ell_{k+1})\,\mathrm{diag}(\boldsymbol{\sigma}'(\mathbf{W}_k\mathbf{o}_k))\mathbf{W}_k && (2.36)\\
&= J_{\mathbf{o}_{k+1}}(\ell_{k+1})\,\mathrm{diag}(\boldsymbol{\sigma}'(\mathbf{a}_{k+1}))\mathbf{W}_k && (2.37)\\
&= \boldsymbol{\delta}_{k+1}\,\mathrm{diag}(\boldsymbol{\sigma}'(\mathbf{a}_{k+1}))\mathbf{W}_k\,. && (2.38)
\end{aligned}$$

In summary, first the network outputs $\mathbf{a}_K$ and $\mathbf{o}_K$ are calculated by forward passing the input sample $\mathbf{x}$. Then the gradients $\{\boldsymbol{\delta}_k\}_{k=1,\dots,K}$ are calculated and backpropagated in recursive manner, see equation (2.30). Thus, it is shown that algorithm 2 indeed calculates the gradient. Then, the gradients are used to update the weights, see algorithm 1 (line 5).

Neural networks are capable to approximate almost any function $f : \mathbb{R}^n \to \mathbb{R}$. This is proven by *universal approximation theorem*. There exist several proofs, most of them refer to a specific assumption. It is proven that any continuous function on a closed and bounded subset of $\mathbb{R}^n$ can be approximated [60]. This requires at least one hidden layer, the accuracy of the approximation depends on the number of neurons in the hidden layer. The work [31] and [77] have proven a universal approximation theorem for sigmoid activation function and ReLU activation function, respectively, the last one is commonly used in state-of-the art neural networks. Apart from that, the expressive power of neural networks have yield impressive advances in practical applications.

Overfitting and Regularization. NN tend to *overfitting* [148], that means they tend to learn details and noise of the training data. NN and ML methods shall be able to generalize i.e., the ability to apply what is learned to elsewhere, i.e., to unseen data with a high accuracy. Regularization aims to prevent overfitting.

A common method to prevent overfitting is *early stopping* [107]. The training data is used to train the model. After each epoch, i.e., the NN was fed with every training sample once, the NN is evaluated on validation data. The training process stops when the accuracy of the NN saturates or starts decreasing on the validation data. Otherwise, if the training process would continue, the NN could potentially lose generalization by learning the noise and details present in the training data.

Other regularization techniques are *L^2-regularization*, also called *weight decay* [74] and *L^1-regularization* [135, 138], also named *lasso regression*. Both techniques add a penalty term to the loss function, which prevents the weights from becoming too large. For L^2-regularization that means, the new loss function $\ell : \mathbb{R}^n \to \mathbb{R}$ of a NN is

$$\bar{\ell}(\mathbf{w}) := \ell(\mathbf{w}) + \frac{\alpha}{2}||\mathbf{w}||_2^2 \,. \tag{2.39}$$

Using the squared loss function (see equation (2.15)) and replacing the derivative of the new weight update step yields:

$$\begin{aligned} \mathbf{w}^{(i+1)} &= \mathbf{w}^{(i)} - \eta(\mathbf{v}^{(i)} + \alpha\mathbf{w}^{(i)}) && (2.40)\\ &= (1 - \eta\alpha)\mathbf{w}^{(i)} - \eta\mathbf{v}^{(i)} \,, && (2.41) \end{aligned}$$

with learning rate $\eta > 0$, regularization factor $\alpha > 0$ and gradient $\mathbf{v}^{(i)}$, see algorithm 1. Adding the L^2-norm of the weights to the loss function prevents the weights from becoming too large. Besides that, the L^1-regularization includes the L^1-norm of the weights as penalty term to the loss function:

$$\begin{aligned} \bar{\ell}(\mathbf{w}) =& \ell(\mathbf{w}) + \alpha||\mathbf{w}||_1 && (2.42)\\ =& \ell(\mathbf{w}) + \alpha|\mathbf{w}| \,. && (2.43) \end{aligned}$$

While L^2-regularization prevents the weights from becoming too large, L^1-regularization aims to set weights with very little activation to 0. This can be seen as a selection of the most important nodes.

An other regularization technique is *dropout* [129]. Dropout is applied during training and randomly sets the activation, i.e., the output of a neuron to 0, in the forward but also in the backward pass. The proportion of neurons on which dropout is applied is determined by the dropout rate $\lambda \in [0, 1]$. Moreover, dropout is applied layer-wise. By dropping-out the activations of randomly picked neurons, this imitates the training of similar but different neural network architectures.

2.2.2 Convolutional Neural Networks

In the previous section, the main concepts of neural networks have been presented. Now, convolutional neural networks (CNN) will be introduced, which are commonly used in computer vision and object recognition tasks. In the previous section, it was mentioned that neural networks (under some assumptions) are capable to

approximate any function f. The work in [150] shows that (also under some assumptions) this is also possible for a CNN.

CNN [76], having their name from the convolutional layer or *convolutions* simplified, are commonly used for visual data like images or point clouds. Compared to fully-connected feedforward networks, in CNNs filters with shared weight are used in order to capture the spatial or temporal dependencies of the input data and therefore reducing the number of parameters (weights).

In the field of computer vision, a CNN can be divided into two main parts:

1. encoder, feature extractor or backbone: the first part of the network, that extracts or encodes features from the input,
2. head or remaining architecture: has to be specified for the recognition task, e.g., if an image has to be classified or specific objects like vehicles should be detected.

Since the head or the rest of the network architecture depends on the specified task, more details will be given in the section 2.3.

Typically, convolutions are used with further components, defining convolutional blocks. These are pooling and activation function, while the latter have been discussed in section 2.2.1. Remaining are convolutions and pooling.

Convolution. A convolutional operation [49, 40] of two functions $g : \mathbb{R} \to \mathbb{R}$ and $h : \mathbb{R} \to \mathbb{R}$ is defined by

$$(g * h)(t) = \int_{\mathbb{R}} g(\tau)h(t - \tau)d\tau \tag{2.44}$$

or simplified in the discrete case with $g : \mathbb{Z} \to \mathbb{R}$ and $h : \mathbb{Z} \to \mathbb{R}$

$$(g * h)(t) = \sum_{\tau \in \mathbb{Z}} g(\tau)h(t - \tau)\,. \tag{2.45}$$

A convolutional operation is denoted with $*$ and expresses how the shape of the one function, here g is modified by the other function, here h. In a convolutional operation in neural networks, the function g refers to the input and the function h as the *filter* or also called *kernel*, yielding the *feature map*.

For a two-dimensional input $\mathbf{g}$ and using a two-dimensional filter $\mathbf{h}$, the convolution is given by

$$(\mathbf{g} * \mathbf{h})_{i,j} = \sum_{\tau_1} \sum_{\tau_2} g_{\tau_1,\tau_2} \cdot h_{i-\tau_1,j-\tau_2} \quad \forall\, i, j \in \mathbb{Z}\,. \tag{2.46}$$

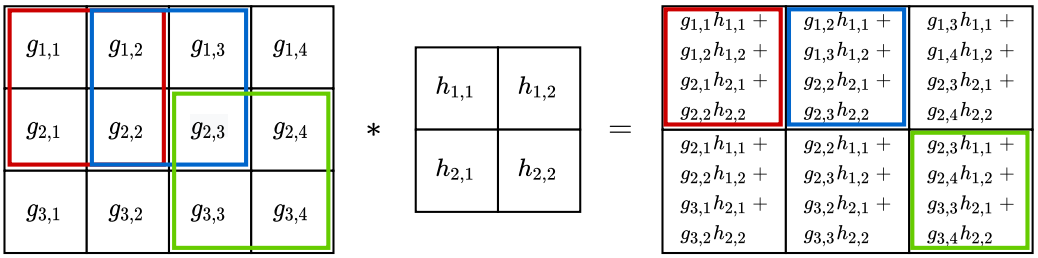


Figure 2.6: Illustration of a convolution. Input (left), filter (center) and the resulting feature map (right).

Due to the commutativity of the convolution, this can be rewritten as

$$(\mathbf{h} * \mathbf{g})_{i,j} = \sum_{\tau_1} \sum_{\tau_2} g_{i-\tau_1, j-\tau_2} \cdot h_{\tau_1,\tau_2} \quad \forall\, i,j \in \mathbb{Z}\,. \tag{2.47}$$

The commutative property arises by kernel flipping, relative to the input. This is interesting from a theoretical point of view. In applications and in most libraries of computer languages, the cross-correlation without kernel flipping is used

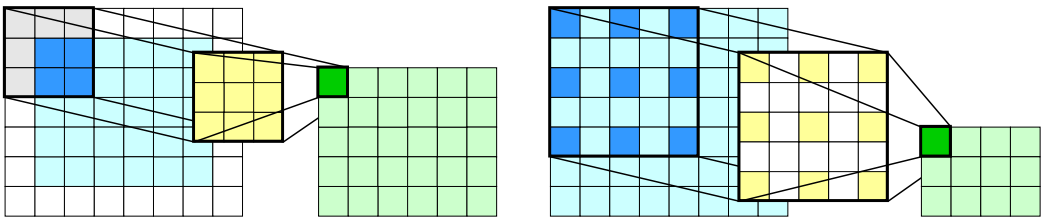
$$(\mathbf{h} * \mathbf{g})_{i,j} = \sum_{\tau_1} \sum_{\tau_2} g_{i+\tau_1, j+\tau_2} \cdot h_{\tau_1,\tau_2} \quad \forall\, i,j \in \mathbb{Z}\,. \tag{2.48}$$

An illustration of a convolution is shown in figure 2.6. Convolutions aim to extract features from the input, giving the feature map (output of a convolution) its name. In a CNN, the kernel values to be fitted are the weights $\mathbf{w}$. CNNs extract in the first layer low level features such as lines and curves. These features become more complex in deeper layers. Important parameters that are not included in the formulas above are *stride* $s \in \mathbb{N}$ and *zero padding* $p \in \mathbb{N}_0$. The stride defines the step size with which the filter $\mathbf{h}$ moves over the input $\mathbf{g}$. Zero padding adds zero values around the input, where p defines how often zero borders are added to the input. Depending on the convolution's parameters i.e., kernel size and stride, the feature map has a lower dimension than the layer input. In order to retain the dimension, the input is enlarged with zero values.

Let be $g_w \times g_h$ the input size, $h_w \times h_h$ the kernel size, stride s and zero padding p. Then the dimension of the feature map is

$$\left(\frac{g_w + h_w + 2p}{s} + 1\right) \times \left(\frac{g_h + h_h + 2p}{s} + 1\right), \tag{2.49}$$

while the values shall be chosen, such that both terms are integer values. Figure 2.6 shows an example with stride $s = 1$ and no zero padding. In figure 2.7 a) an



a) Convolution b) Atrous convolution

Figure 2.7: Illustration of different convolutional layers. The input is given in blue colors, the filter in yellow and the feature map in green. Zero padding or added zero values are white colored.

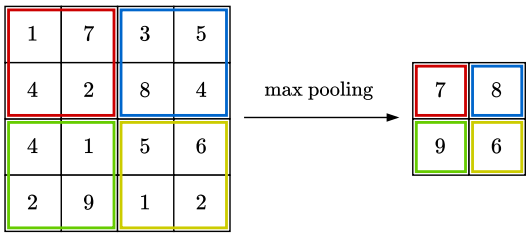


Figure 2.8: Illustration of max pooling.

example with stride $s = 1$ and zero padding $p = 1$ is shown, where the dimension of the feature map is equal to the input dimension.

A variant of convolutions are *atrous convolutions* [16] (also named dilated convolutions). They work similar to a convolution, but with an additional parameter, the atrous rate $r \in \mathbb{N}$ that introduces $r-1$ zeros between consecutive filter values. Thus, the filter size is enlarged from size $h_w \times h_h$ to $h_w+(h_w-1)(r-1) \times h_h+(h_h-1)(r-1)$. The added zeros are fixed. Thus, the computational effort does not change compared to a standard convolution. An example of an atrous convolution with $r = 2$, stride $s = 1$ (and no zero padding) is shown in figure 2.7 b).

Pooling. A CNN tries to reduce the size of the input sample and to extract more and more features. Usually, the convolution is chosen such that the input and feature map dimension are equal. In order to reduce the size and to aggregate the most important features, a pooling layer [40, 48] is applied. There are multiple versions of pooling layers, for instance maximum (max) pooling, where the maximum value of the considered area is taken or the mean pooling, where the mean is computed. Note, a pooling layer does not have learnable weights $\mathbf{w}$ as a convolution. The max pooling is shown in figure 2.8. A review of pooling methods is presented in [48].

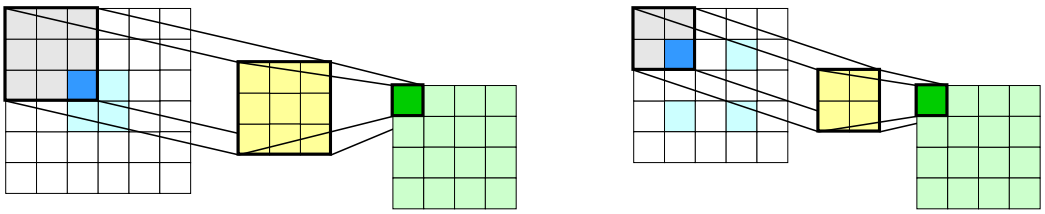


Figure 2.9: Illustration of two deconvolutions. The input is given in blue color, the filter in yellow and the output in green. Added zero values are white colored.

Deconvolution. In some applications, the dimension of a CNN's input and output shall be equal. Convolutional and pooling layers reduce the input dimension and extract features – the input is downsampled. *Deconvolutional layers* or *deconvolutions* (sometimes called transposed convolutions) [40, 99] upsample an input and increase the dimension. A deconvolution is the inverse of a convolution. Two examples of a deconvolution are depicted in figure 2.9. In the left panel, zero padding is used to increase the output dimension, while in the right panel, zero values are added between the input values. Note, in deep learning, the terms deconvolution and transposed convolution are used equally. Technically, this is not correct. Deconvolution is the inverse operation of a convolution, while a transposed convolution reconstruct the spatial dimension.

2.3 Object Recognition

In computer vision, object recognition includes for instance classification, detection, and segmentation. In classification, the input e.g., an image has to be assigned to a class out of a pre-defined set of classes, in object detection an object of interest has to detected and localized using a bounding box and in segmentation each entry of the input e.g., each pixel of an image or each point of a point cloud has to be classified. In this section the focus is on semantic segmentation of images and point clouds as well as object detection and tracking in point clouds; visual examples are shown in figure 2.10. This section reviews CNNs for the aforementioned object recognition tasks. Furthermore, the CNNs presented in the following are also the ones used in the experiments in this thesis.

2.3.1 Semantic Segmentation

Semantic segmentation is the task of assigning a class label $\hat{y} \in \mathcal{C}$ to each pixel or point z from a pre-defined label space $\mathcal{C}$. Thus, the input, i.e., in the following an

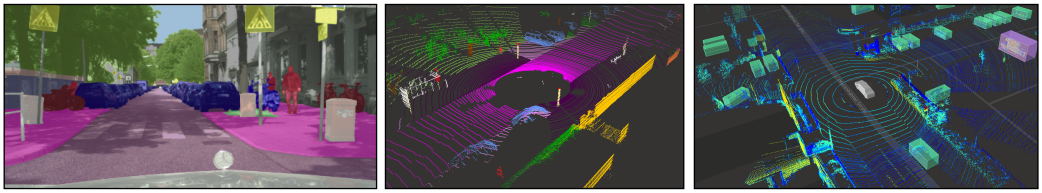


Figure 2.10: Semantic segmentation of an image (left) and a point cloud (center) as well as point cloud object detection (right).

image or a point cloud, is divided into areas of the same class. If two instances of the same class are next to each other, they are considered as one segment. Further segmentation tasks are instance and panoptic segmentation: in instance segmentation [101, 98, 78], only objects of interest are separably segmented while panoptic segmentation [94, 18] is a combination of both: it is similar to semantic segmentation but in addition, single objects are separated. In the following, the term segmentation is only used for semantic segmentation for brevity.

Image Segmentation

FCN. The first state-of-the art image segmentation network is the so-called *fully convolutional network (FCN)* [82] and goes back to 2015. The idea of this network is to employ solely locally connected layers, such as convolution, pooling and upsampling. The authors propose to adapt contemporary classification networks, i.e., a CNN, here referred to as backbones. An example is the VGG16 [128] network that is shown as backbone of the FCN in figure 2.11. The input of the FCN is an image $\mathbf{x} \in [0,1]^{w \times h \times 3}$. The ground truth is $\mathbf{y} \in C^{w \times h}$ and the task is to predict the pixel-wise class labels, i.e., $\hat{\mathbf{y}} \in C^{w \times h}$. As it can be seen in figure 2.11, the first part of the networks – the encoder – consist mainly of convolutions ('Conv') and max pooling ('Max Pooling'). The input features are extracted and aggregated and thus, the input dimension is reduced and downsampled. Since the pixel-wise classification is required, the downsampled features have to be upsampled such that the size (w, h) of the input $\mathbf{x}$ and output $\hat{\mathbf{y}}$ are the same. Upsamling layers can be learnable e.g., using deconvolutional layer or can be fixed by using bilinear interpolation, for instance. Both approaches are proposed in the work of the FCN model. The upsamling layers define here the segmentation head. The upsamling part is also referred to as decoder.

The FCN model also includes *skip connections* ('Add') in order to fuse the output of a convolutional layer from the encoder part with the corresponding upsampling layer in the decoder part by adding them up. In newer architectures, the skip connection

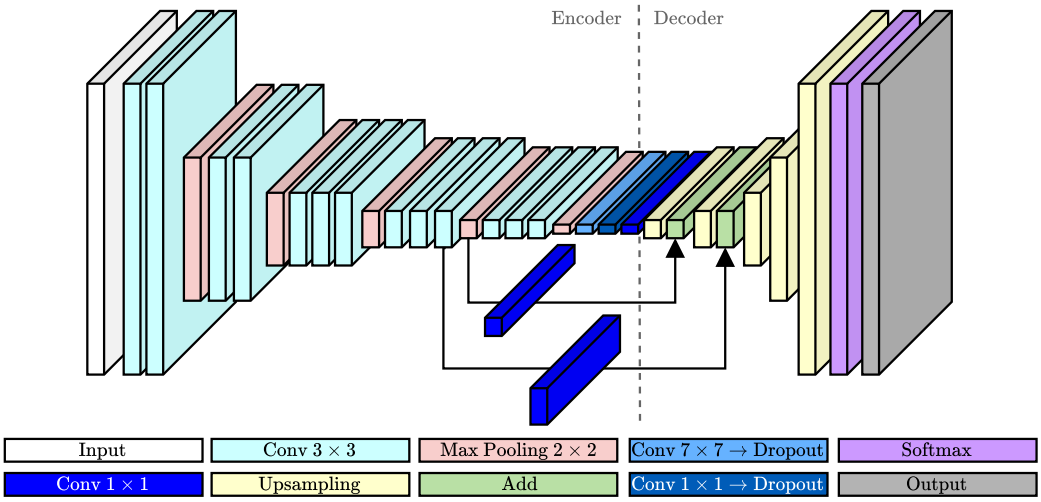


Figure 2.11: FCN architecture. The model shows an FCN architecture with VGG16 backbone. The VGG16 consist of convolutions with ReLU activation function, followed by max pooling. In total, 16 convolutions are applied. In the decoder part, the features of the decoder are summed up with the upsampling layers. In the end, a softmax layer is applied. The output is the semantic segmentation of the input image.

employ concatenation instead of adding up. Skip connections allow the upsampling layers to re-use the representation of the encoder part in order to maintain more features, which can lead to a higher accuracy. In order to prevent overfitting, to the end of the encoder part, FCN employ *dropout*. After the last upsampling layer, the feature map is of size (w, h, q). A softmax layer follows, i.e., the application of the softmax function to every feature entry $(i, j),\ i = 1, \ldots, w,\ j = 1, \ldots, h$, yielding the probabilities $\hat{\mathbf{y}}^{prob}$. For the remainder of this thesis, the output of the softmax is referred to the term *probabilities* since it represents the class-wise probabilities, provided by the CNN. If a segmentation model does not contain a softmax layer, this can always be incorporated without changing the segmentation results. The output or *prediction* $\hat{\mathbf{y}}$ of the FCN is the segmentation, i.e., applying the argmax function to the softmax values, see equation (2.14).

Deeplab. In recent years, newer architectures and techniques have followed, often building on each other. One of them is the model *Deeplab*, which was first published in 2015 [16] (v1) and has been continuously developed and improved [15, 17, 19] (v2, v3, v3+). In what follows the term Deeplab is used as a short term for the newest version v3+ for brevity. Compared to FCN, the Deeplab consists of some new and advanced components, which will be outlined next. This includes

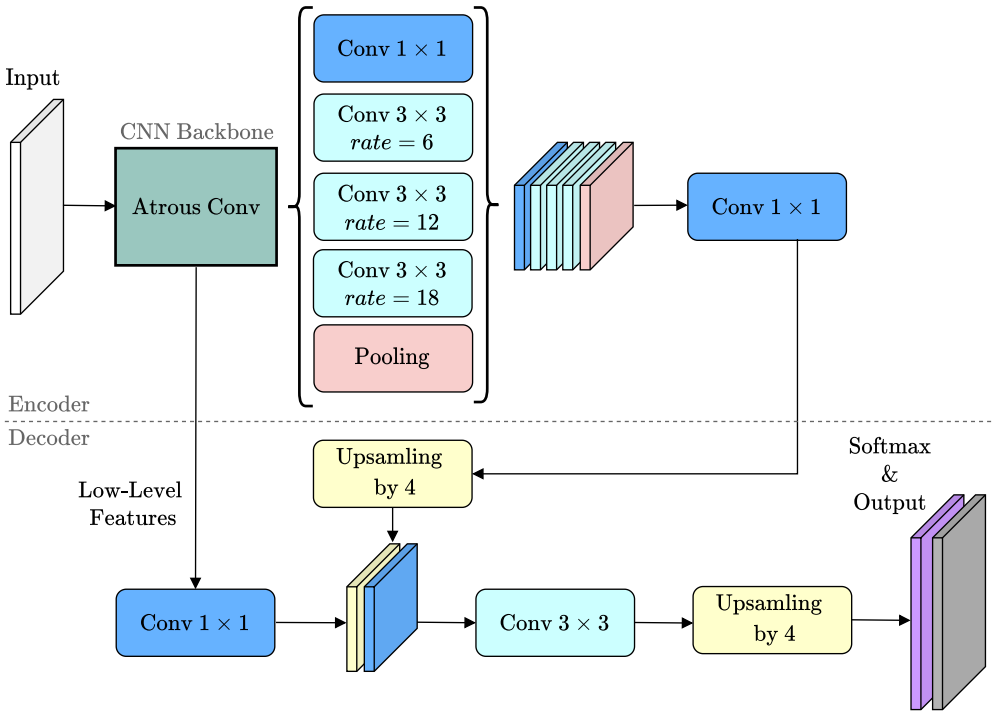


Figure 2.12: DeeplabV3+ architecture. The DeeplabV3+ model consists of a CNN backbone including atrous convolutions in which the input image is fed. The next part in the encoder is the ASPP module. In the decoder, the output of the ASPP module is concatenated with the low-level features of the backbone. In the end, a convolution and upsampling is applied.

atrous convolutions (see section 2.2.2), depthwise separable convolutions and atrous spatial pyramid pooling.

A *depthwise separable convolutions* consists of two layer operations. First, a depthwise convolution is applied, in which a single filter per depth or channel is applied. Second, to the resulting feature map of the depthwise convolution, a point-wise convolution is applied. A point-wise convolution is a 1×1 convolution while the depth of the filter is equal to the number of channels of the input. Thus, and compared to a standard convolution, the operation is separated. Furthermore, a depthwise separable convolution has fewer parameters (weights to be trained) and is therefore more computationally efficient without losing performance. Analogously, an *atrous separable convolution* is a depthwise atrous convolution, followed by a point-wise convolution.

The Deeplab(v3+) architecture, shown in figure 2.12, consists of a backbone (encoder) that uses atrous convolutions ('Atrous Conv'). Two common backbones are MobileNet [118] and Xception [23]. MobileNet requires little computing power and has a fast inference time. As the name suggests, it was developed for use on mobile

devices. Xception requires more computational power and inference time, but it achieves higher accuracy than MobileNet. The output of the backbone is followed by *atrous spatial pyramid pooling* (ASPP). ASPP is basically a module that consists of one 1×1 convolution, three atrous convolutions with different atrous rates $(6, 12, 18)$ and image pooling, finally followed by a point-wise convolution. ASPP was first introduced in [15] to take multiple scales of objects into account. In the decoder part, low level features from the backbone as well upsampled results from the ASPP module are concatenated. Finally, a convolution and upsampling layer is applied.

Lidar Point Cloud Segmentation

In lidar point cloud segmentation, the network architectures can be divided into projection-based and non-projection-based networks. In the following RangeNet++ [91], SalsaNext [30], and Cylinder3D [151] will be explained in more details. Compared to images, point clouds are sparse and given in an unstructured format. To this end, a lot of approaches pre-process the point cloud data to an image-like representation.

RangeNet++. The architecture of RangeNet++ [91] is shown in figure 2.13. As it can be seen, it has a typical encoder-decoder structure and looks similar to the FCN architecture, cf. figure 2.11. The network uses a 2D representation of the point cloud, called *range image* as input and the target i.e., the labels are also in an image representation. The projection of a point cloud to a range image representation follows two steps: a transformation from Cartesian to spherical coordinates, and then a transformation from spherical to image coordinates. For a point $p_j = (x_j, y_j, z_j)$ in Cartesian coordinate, the spherical coordinate is given as $(r_j, \theta_j, \varphi_j)$ with range r_j, polar angle θ_j and azimuth angle φ_j. The transformation from Cartesian to spherical coordinate is given by

$$r_j = \|p_j\|_2, \quad \theta_j = \arcsin\left(\frac{z_j}{r_j}\right) \quad \text{and} \quad \varphi_j = \arctan\left(\frac{y_j}{x_j}\right). \tag{2.50}$$

Based on the spherical coordinate, the range image coordinate (u_j, v_j) is given as

$$\begin{pmatrix} u_j \\ v_j \end{pmatrix} = \begin{pmatrix} \frac{1}{2}[1 - (\varphi_j \pi^{-1})]w \\ [1 - (\theta_j + \psi_{up})\psi^{-1}]h \end{pmatrix}, \tag{2.51}$$

with (w, h) the width and height of the image and $\psi = \psi_{up} + \psi_{down}$ the vertical field of view (FOV) of the lidar sensor, see also figure 2.2. The described projection has

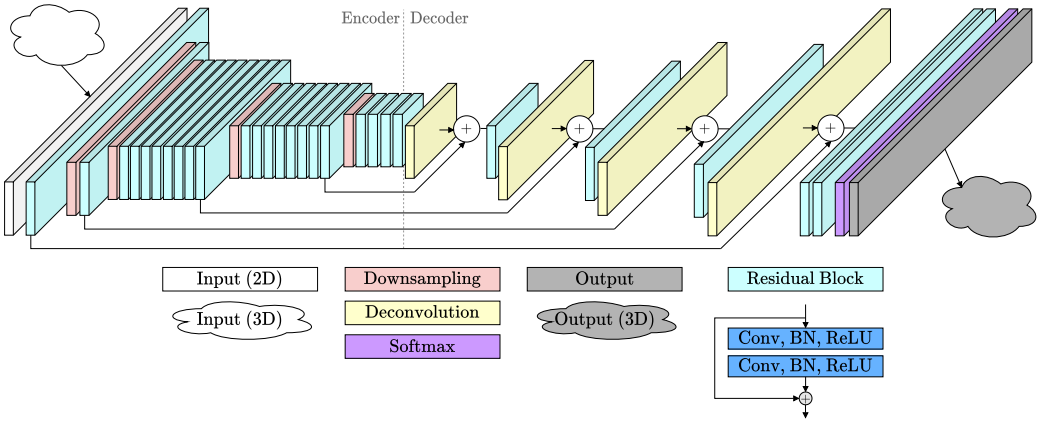
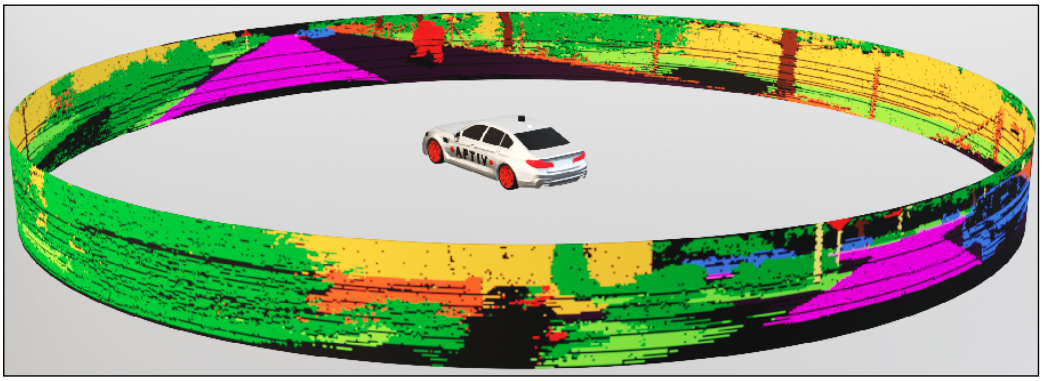


Figure 2.13: RangeNet++ architecture. The network processes the point cloud in an image-like representation. The architecture is an encoder-decoder structure. Finally, the network output is re-projected to the point cloud.

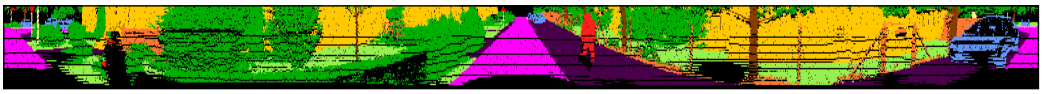
to be done for all points $p_j, j = 1, \ldots, n$. The range image of size $(w, h, 5)$ where each entry or pixel (for brevity) (u_j, v_j) is a vector $(x_j, y_j, z_j, r_j, i_j)$ consisting of the original features Cartesian coordinates, range and intensity, respectively. Figure 2.14 shows the panorama view from the lidar sensor, the range image as well as the point cloud.

As shown in figure 2.13, the architecture of RangeNet++ consists mainly of three blocks: residual blocks ('Residual Block'), downsample blocks ('Downsampling') and deconvolutions ('Deconvolution') for upsampling. In general and introduced by [56], a residual block contains one or up to a few convolutional layers with a skip connection that maps the input (identity, i.e., $id(x) = x$) from the beginning to the end – the identity is skipped over those values. Residual blocks aim to avoid the problem of vanishing gradient [6]. For very deep CNNs it can occur that gradients during backpropagation become too small. Here, a residual block consist of two convolutions, batch normalization and ReLU activation function, see ('Residual Block'). Batch normalization (BN) [8] normalizes the feature map or input between two layers. BN forces the activation functions to take a stable distribution. Therefore, the learning rate of the neural network can be increased, the training is more stable and faster. More details are also given in [111]; on which the backbone of RangeNet++ is based on. For downsampling, a convolution with a stride of 2 in the horizontal and a stride of 1 in the vertical is used. RangeNet++ has skip connection (adding up, no concatenation) between mirrored blocks. Furthermore, a residual block follows an upsampling convolution. The output of the network is a prediction of the range image of the same shape (w, h). For the segmentation of the point cloud, the prediction (on image level) is re-projected to the point cloud. In

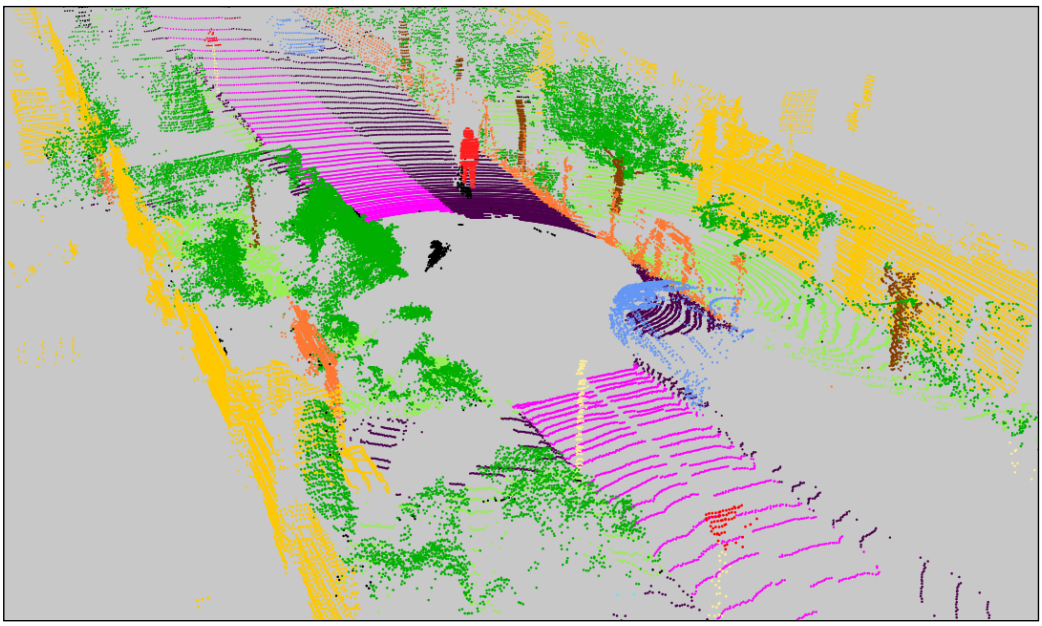
order to refine the re-projected results and to clean the point cloud from undesired discretization and inference artifacts, a k-nearest neighbor (kNN) post-processing is applied [91].



a) View from Lidar sensor



b) Range image



c) Point cloud

Figure 2.14: Spherical or panorama view from lidar sensor (a), range image (b) and the corresponding point cloud (c). The ground truth data is shown.

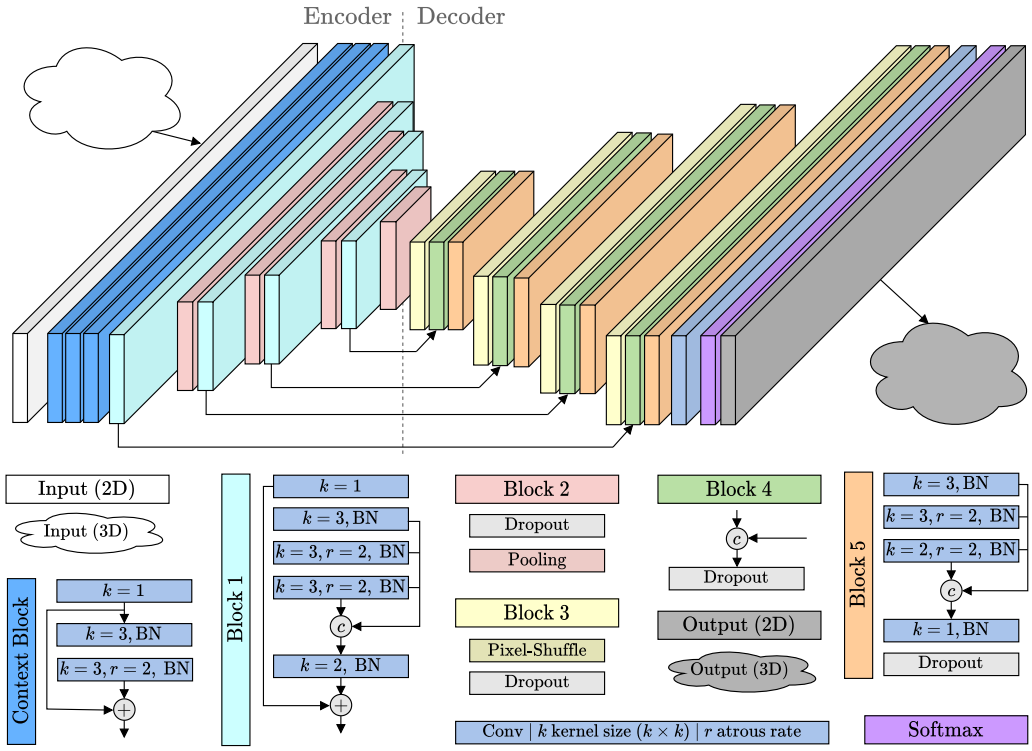


Figure 2.15: SalsaNext architecture. The network is projection-based and takes a range image as input. SalsaNext has an encoder-decoder structure whose layer compositions are defined in blocks. In the end, the network output is re-projected to the point cloud.

SalsaNext. The network SalsaNext [30] is a projection-based network with an encoder-decoder structure, as can be seen in figure 2.15. The authors define blocks, which consist mostly of layers that have already been presented. The 'Context Block' as well as 'Block 1' and 'Block 5' are residual blocks, containing convolutions and atrous convolutions, often followed by BN. 'Block 2' represents the downsamling part and includes average pooling. For upsamling, SalsaNext includes pixel-shuffle, see 'Block 3'. Pixel-shuffle [124] leverages the learned feature maps to produce the upsampled features by shuffling the pixels from the channel dimension to the spatial dimension. Similar to the previous presented networks, SalsaNext includes also skip connections between mirrored blocks. Moreover, SalsaNext employs the kNN post-processing step after re-projecting the range image prediction to the point cloud, as it is done and introduced by RangeNet++. SalsaNext provides the option of additional uncertainty quantification. More details about this will be given later in section 4.2.

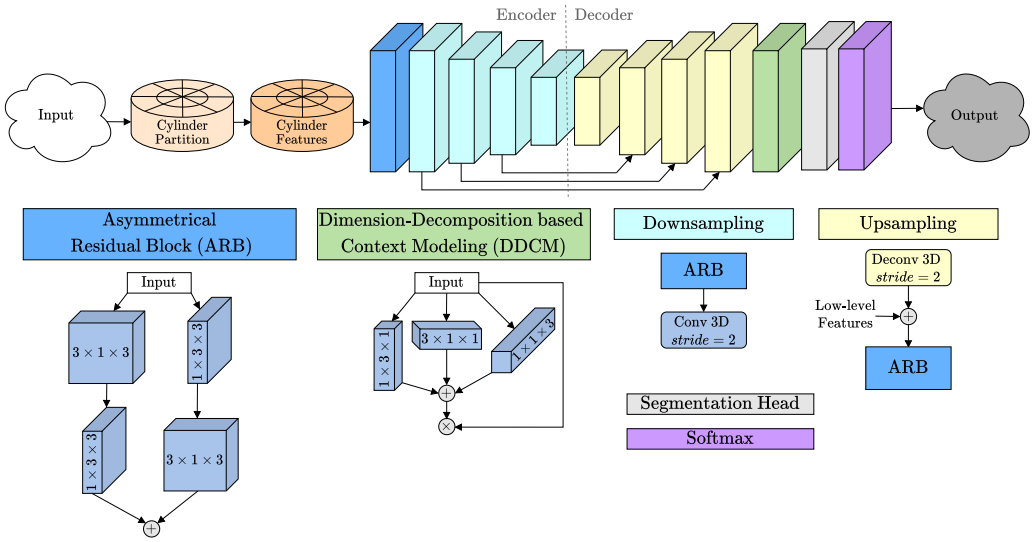


Figure 2.16: Cylinder3D architecture. The input consists of point clouds. In the first part, each point cloud is partitioned into cylindrical coordinates, followed by a cylinder feature extraction part. The remaining architecture in an encoder-decoder structure. The output is provided on point cloud level.

Cylinder3D. The last network architecture is non-projection-based. This network is called Cylinder3D [151] and its essential feature is that it processes the point cloud in a cylinder partition from which cylinder features are extracted. An illustration of Cylinder3D is shown in figure 2.16. In general, point clouds are given in Cartesian coordinate system. First, the points are transformed from Cartesian (x_j, y_j, z_j) to cylinder coordinates (ρ_j, θ_j, z_j) with

$$\rho_j = \sqrt{x_j^2 + y_j^2}, \quad \arctan\left(\frac{y_j}{x_j}\right) \quad \text{and} \quad z_j = z_j\,. \tag{2.52}$$

In the cylinder partition, the three dimension height, width and depth are split uniformly, yielding a cylinder grid representation including the projected points. A voxelization of this is fed into a multilayer perceptron (MLP)-based PointNet [108] to get cylinder features. MLP are feedforward neural networks, see also figure 2.4. PointNet is a network architecture that directly uses the point cloud as input without any pre-processing or projections. This means that the points from their cylinder partition are first transformed to a voxel partition. Then a MLP is applied to the points of each voxel to extract features, which are then concatenated to every point. The feature extraction based on voxel representation is described in more details in [152]. Apart from the cylinder partition, the remaining architecture consist of down- and upsampling blocks as depicted in figure 2.16, consisting of convolutions and deconvolutions.

2.3.2 Object Detection and Tracking

A lidar sensor provides point clouds of the environment including range information w.r.t. the mounting position as well as intensity values. Additionally, lidar data can be recorded at high frequency, usually $10\,\text{Hz}$. Thus, this sensor data is particularly well suited for 3D bounding box detection and tracking. First, 3D bounding box detection is explained using two networks, namely PointPillars [75] and Part-A^2 [123]. Second, a brief introduction to tracking methods using a Kalman-Filter [70] is given.

Detection

The task of object detection is to detect objects that belong to a pre-defined set of classes, e.g., vehicles, pedestrians or traffic signs. Here, the objects are detected by 3D bounding boxes.

PointPillars. PointPillars [75] is one of the first DL-based methods for object detection from point clouds and was published in late 2018. The network, depicted in figure 2.17 consist of three components: a pillar features net that processes the point cloud to an image-like representation, called pseudo-image. The next component is a CNN (backbone) and the last component is a detection head. As it will be shown, the pseudo-image is different to the range image. However, the idea of pre-processing the point cloud to a 2D image representation is the same. The pillar feature net which gives the network its name, converts the point cloud to a pseudo-image. First, the point cloud is divided into a grid in the $x-y$ plane – the z coordinate defines the height. Each grid cell in the $x-y$ plane represents a pillar. Each point $p_j = (x_j, y_j, z_j, i_j)$ is augmented to a $D = 9$ dimensional point,

$$p_j^D = (x_j, y_j, z_j, i_j, x_j^c, y_j^c, z_j^c, x_j^p, y_j^p)\,, \tag{2.53}$$

where the superscript c denotes the distance to the arithmetic mean of all points in the pillar and p denotes the distance of the point from the center of the pillar w.r.t. x_j and y_j. Due to the sparsity of the point cloud, most of the pillars will be empty, i.e., do not include points. This sparsity will be utilized by imposing a limit on the number of non-empty pillars P and on the number of points per pillar N to create a dense tensor of size

$$(D, P, N)\,, \tag{2.54}$$

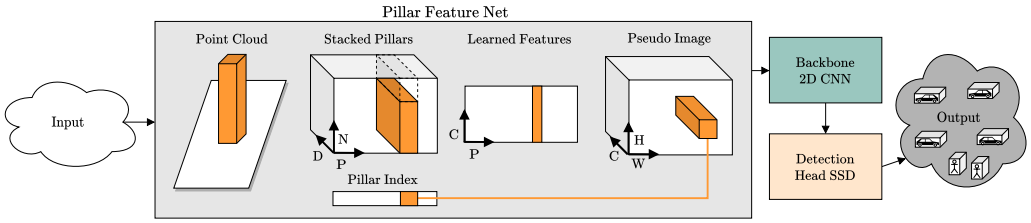


Figure 2.17: PointPillars architecture. The main part is the pillar feature net in which the point cloud is transformed to a pseudo-image, which is the input for the CNN. For detection, a SSD head is used. Finally, the bounding boxes and their scores are provided as output.

see 'Stacked Pillars'. If a pillar contains more than N points, N points are randomly sampled. That means, the tensor consist of P pillars, while each pillar has a maximum of N points and each point is of dimension D, given as in equation (2.53). Then, a simplified version of PointNet [108] is applied to every point p_j^D. That means, a linear layer, followed by BN and ReLU is applied. The output dimension by applying the simplified PointNet version is C. This yields a tensor of size

$$(C, P, N)\,. \tag{2.55}$$

Afterwards, a maximum operation is used over the channels (points per pillars) to create an output tensor of size

$$(C, P)\,, \tag{2.56}$$

that presents a feature vector of dimension C for every pillar ('Learned Features'). Finally, the encoded features in the (C, P) tensor are scattered back to its original pillar location to create the pseudo-image of size

$$(C, H, W)\,, \tag{2.57}$$

with height H and width W, see 'Pseudo Image'. Simplified, H and W depend on the rectangular region around the lidar sensor and the choice of pillar size. For example, the rasterized region is of size $80 \times 60\,\mathrm{m}^2$ and each pillar is of size $0.5 \times 0.5\,\mathrm{m}^2$, then is $H = 80/0.5 = 160$ and $W = 60/0.5 = 120$. The pillar feature net is visualized in figure 2.17. The backbone has an encoder-decoder structure and consist of convolutions and deconvolutions. The last part of PointPillars is the detection head. In PointPillars a Single Shot Detector (SSD) [81] is used as detection head. In short terms, that means, that data is rasterized in anchor boxes. Each anchor box will be assigned with a probability for covering a target, i.e., an object that has to be detected. With non-maximum suppression [96], the boxes are filtered and the size and orientation of the box are regressed.

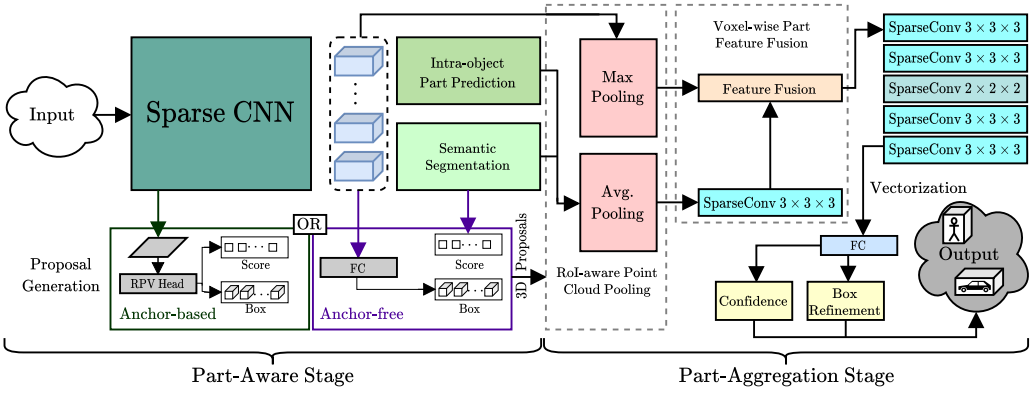


Figure 2.18: Part-A^2 architecture. A description is given in the text.

Part-A^2. Another network architecture for 3D object detection is Part-A^2 [123], which was first published mid of 2019, while the components that will be presented next are from the latest version from the beginning of 2020. Part-A^2 consists of two stages: first, a part-aware stage, in which the locations of objects of interest are predicted and 3D proposals are generated. The second stage consist of a part-aggregation stage that applies pooling and aggregates the outcome in order to refine the locations of the previous stage. The network architecture of Part-A^2 is shown in figure 2.18. First, the point cloud is fed into a sparse CNN [79] that outputs the relative location of each foreground point w.r.t. the object box that it belongs to, by the authors called intra-object parts. Furthermore, the sparse CNN outputs semantic segmentation of foreground objects. The relative locations of foreground points provide strong cues for box scoring and localization. Moreover, the network provides two options for box proposals: anchor-based and anchor-free. Anchor-based is what has been presented with the SSD above. Anchor-free means, that the center point of a location is estimated and the corresponding box in terms of dimension and orientation is regressed. More details about anchor-based and anchor-free methods are given in [80]. The intra-object proposals, the semantic segmentation as well as the 3D proposals – as outcome of the part-aware stage – are the input for the part-aggregation stage. In the next part – 'Region of Interest (ROI) point cloud pooling' – maximum and average pooling are applied to conduct box scoring and refinement. This is achieved by aggregating the learned features of all points within the same proposals given by the previous network part. Afterwards, a feature fusion is done. Thereafter, mainly sparse convolutions are applied. The outcome of the last sparse convolution is vectorized and a fully connected layer is applied. Finally, the box refinement and the confidence of the predictions are given. A detailed description is provided by [123].

Object Tracking

The task of object tracking is to identify and track objects over multiple frames of sequential data. A common method for that is using a Kalman filter.

The Kalman filter [70, 143] is a recursive estimation algorithm to provide the (estimated) state $\mathbf{x}_t \in \mathbb{R}^n$ of a linear system at discrete timestamp t. The state estimation is based on an initial estimate $\mathbf{x}_0$ and a series of measurements $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t \in \mathbb{R}^m$. Furthermore, the information of the system is described by

- $\mathbf{F} \in \mathbb{R}^{n \times n}$ state transition model, applied to the previous state $\mathbf{x}_{t-1}$,
- $\mathbf{B} \in \mathbb{R}^{n \times l}$ control input model, will be applied to a control vector $\mathbf{u} \in \mathbb{R}^l$,
- $\mathbf{H} \in \mathbb{R}^{m \times n}$ observation model, to map the true state space to the measured space,
- $\mathbf{Q} \in \mathbb{R}^{n \times n}$ covariance of the process noise,
- $\mathbf{R} \in \mathbb{R}^{m \times m}$ covariance of the measurement noise,

while it is assumed that $\mathbf{F}, \mathbf{B}, \mathbf{H}, \mathbf{Q}$ and $\mathbf{R}$ are invariant over time. If they are not to be assumed invariant, they also depend on timestamp t. The covariances $\mathbf{Q}$ and $\mathbf{R}$ are assumed to be zero-mean Gaussian. Furthermore, n is the number of states, m the number of measurements and l the number of control values. The iterative process consists of two steps: a *prediction* step in which the state estimate $\hat{\mathbf{x}}_t^-$ and error covariance $\mathbf{P}_t^-$ are predicted and an *update* step in which both are updated (corrected).

The prediction for $t > 0$ is:

$$\hat{\mathbf{x}}_t^- = \mathbf{F}\hat{\mathbf{x}}_{t-1}^+ + \mathbf{B}\mathbf{u}_{t-1}\,, \tag{2.58}$$

$$\mathbf{P}_t^- = \mathbf{F}\mathbf{P}_{t-1}^+\mathbf{F}^T + \mathbf{Q}\,. \tag{2.59}$$

The operator $\hat{}$ denotes an estimate. The superscript $-$ denotes predicted (prior) estimates and $+$ denotes updated (posterior) estimates.

Before updating both, the measurement residual $\tilde{\mathbf{y}}_t$ and the Kalman gain $\mathbf{K}_t$ have to be calculated:

$$\tilde{\mathbf{y}}_t = \mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^-\,, \tag{2.60}$$

$$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}^T(\mathbf{R} + \mathbf{H}\mathbf{P}_t^-\mathbf{H}^T)^{-1}\,. \tag{2.61}$$

The Kalman gain determines how much the difference between the previous estimate and the current measurement is included in the next estimate. In particular, the update step is given as:

$$\hat{\mathbf{x}}_t^+ = \hat{\mathbf{x}}_t^- + \mathbf{K}_t\tilde{\mathbf{y}}\,, \tag{2.62}$$

$$\mathbf{P}_t^+ = (I - \mathbf{K}_t\mathbf{H})\mathbf{P}_t^-\,. \tag{2.63}$$

In order to use the Kalman filter in object detection, a separate Kalman filter is used for every object. For example, the state $\mathbf{x}$ could consist of the center of the object, the size, the yaw rate, the velocity and the acceleration. For initialization, the above values have to be set. An example is provided in [70].

The presented Kalman filter is a well established approach in tracking. There exist some extensions, e.g., the extended Kalman filter which is used for non-linear systems. Some further approaches can be used for tracking, such as a Particle filter [22] or DL tracking methods [103, 86]. Since only the Kalman filter is used in this thesis, no more details will be given on those approaches.

2.3.3 Metrics for Evaluation

Next, the commonly used evaluation metrics for segmentation, object detection and tracking are presented.

Semantic Segmentation. In the following, the evaluation metrics will be given for image segmentation. However, all presented metrics can be evaluated with points instead of pixels, analogously. The evaluation metric for semantic segmentation is the intersection over union (IoU) [63], more precisely the mean intersection over union ($mIoU$) over all classes $c \in \mathcal{C} = \{1, \dots, q\}$. Let $\hat{y}_z \in \mathcal{C}$ be the predicted class label of pixel z and $y_z \in \mathcal{C}$ the ground truth label. Then the prediction of pixel z is a true positive if $\hat{y}_z = y_z$ and a false positive if $\hat{y}_z \neq y_z$. The true and false positives evaluate the accuracy w.r.t. the prediction. In order to evaluate the miss classified pixel w.r.t. the ground truth label, a false negative is if $y_z \neq \hat{y}_z$. For every class $c \in \mathcal{C}$, the pixel counts true positives TP_c, false positives FP_c and false negatives FN_c of the segmentation results are computed. Then the class-wise $IoU_c \in [0, 1]$ is the number of TP_c (intersection) divided (over) by the sum of TP_c, FP_c and

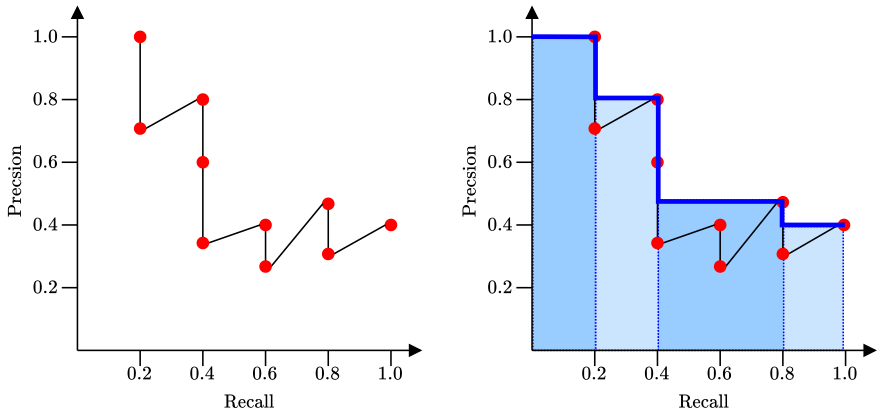


Figure 2.19: Precision-recall curve (left) and the interpolated values (right).

FN_c (union). Finally, $mIoU \in [0, 1]$ is the arithmetic mean over all class-wise IoU_c values, i.e.,

$$mIoU = \frac{1}{q} \sum_{c=1}^{q} \frac{TP_c}{TP_c + FP_c + FN_c} . \tag{2.64}$$

The higher the $mIoU$, the better the performance of the segmentation model. By taking the arithmetic mean over all class-wise IoU_c all classes are weighted equally. This is important to note since most segmentation data sets are highly imbalanced.

Object Detection. The most common evaluation metric for object detection is the (mean) average precision [102]. In object detection methods, every predicted box is associated with a confidence value between 0 and 1, generated by the detector. The final predictions depend on a threshold $conf_{th} \in [0, 1]$ which has to be set. The ground truth and predicted objects are represented by boxes. In order to evaluate the accuracy of a predicted box, the intersection over union IoU_b is computed, by dividing the intersection of the predicted and the ground truth box by their union. Then, a predicted box is a true positive if the IoU_b is above a defined threshold, usually this is 0.5, i.e., $IoU_b \geq 0.5$. Analogously, a prediction is a false positive if $IoU_b < 0.5$ and a false negative if a ground truth object is not detected, i.e, no box (with the same class) overlaps enough with the ground truth box. Then TP_b denotes the number of all true positives, FP_b of all false positives and FN_b of all false negatives. Based on these quantities, precision P and recall R are

$$P = \frac{TP_b}{TP_b + FP_b} , \tag{2.65}$$

$$R = \frac{TP_b}{TP_b + FN_b} . \tag{2.66}$$

Computing precision and recall for varying confidence thresholds $conf_{th}$, yields the *precision-recall curve*. A common evaluation metric is the area under curve, in particular the area under precision-recall curve $(AUPRC)$. In many practical cases, the precision-recall curve is a zigzag-like curve, as depicted in figure 2.19 (left). To avoid this zigzag behavior, the precision-recall curve is interpolated and the area under this interpolated precision recall-curve is the average precision (AP)

$$AP = \sum_{i=1}^{n-1} (R_{i+1} - R_i) P_{intp}(R_{i+1}) \,, \tag{2.67}$$

where

$$P_{intp}(R_{i+1}) = \max_{\tilde{R}:\tilde{R} \geq R_{i+1}} P(\tilde{R}) \,, \tag{2.68}$$

with n precision-recall values. Figure 2.19 (right) shows the interpolation, the AP corresponds to the sum of the four blue areas. Computing the average precision AP_c for every class $c \in \mathcal{C} = \{1, \ldots, q\}$ and taking the mean, yields the *mean average precision* (mAP)

$$mAP = \frac{1}{q} \sum_{c=1}^{q} AP_c \,. \tag{2.69}$$

Object Tracking. Two commonly used evaluation metrics for object tracking are multiple object tracking precision $(MOTP)$ and multiple object tracking accuracy $(MOTA)$ [7]. Let FN_t be the number of false negatives and FP_t the number of false positives at time t. Furthermore, mme_t is the number of mismatches and gt_t the number of ground truth object. Then it is

$$MOTA = 1 - \frac{\sum_{t=1}^{T} FN_t + FP_t + mme_t}{\sum_{t=1}^{T} gt_t} \,. \tag{2.70}$$

Note, false negative and false positive objects are defined as above (by means of IoU). The range of $MOTA$ is $(-\infty, 1]$; the higher the value, the better is the tracking accuracy. One can also define a correct detection by measuring the distance d^i between the center point of a predicted and a ground truth object's bounding box. Then, the distance d^i must not be greater than a given threshold for the prediction to be considered correct i.e., a true positive.

The $MOTP$ is defined as

$$MOTP = \frac{\sum_{t=1}^{T} \sum_{i=1}^{n_t} d_t^i}{\sum_{t=1}^{T} c_t} \,, \tag{2.71}$$

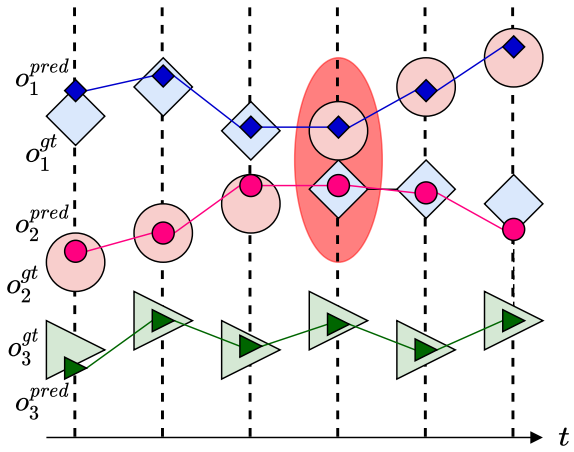


Figure 2.20: Illustration of tracking for three objects. o^{pred} is the prediction by a detector and the connection between the corresponding markers, is the tracker output. o^{gt} is the ground truth object. The mismatch between object 1 and 2 is highlighted.

where c_t is the number of matches found at time t and d_t^i the distance between predicted and ground truth box i at time t. The number of matches at time t is n_t. The lower $MOTP \in \mathbb{R}_+$, the better is the tracking precision. When using the IoU for the distance between a predicted and a ground truth object, it is $d_t^i = 1 - IoU_t^i$. Figure 2.20 illustrates tracking of three objects, including one mismatch.

2.4 Uncertainty Quantification and MetaSeg

In almost every real-world application where neural networks are used, it is not only important to have a high accuracy, it is also important to quantify uncertainty. Uncertainty quantification (UQ) has an important role in decision-making, e.g., using NN in real-world applications such as automated driving or during optimization of the model e.g., in active learning strategies where the acquisition function is often uncertainty-based. In general, a distinction is made between *epistemic* and *aleatoric* uncertainty [62]. Epistemic uncertainty is referred to the model uncertainty and is caused by the lack of knowledge. It can be reduced by adding more training data. Besides that, aleatoric uncertainty is referred to observation uncertainty. It is not reducible, since it captures noise or internal randomness of the phenomena in the training data. Suppose a model is trained to classify images of cats and dogs. With too little or unbalanced training data, the model makes most likely prediction errors and the prediction includes uncertainty. This is epistemic uncertainty that can be reduced by adding more training data such that it is balanced or includes

more varying images of cats and dogs. An example for aleatoric uncertainty is coin flipping. Even the best model is not able to predict the outcome with high accuracy, since there is always a fifty-fifty chance for head or tail. In addition to [62], the work [1] presents a detailed review about techniques, applications and challenges in UQ. In section 2.2.1 NN were introduced, and it was shown that they can be considered as statistical models, whose weights become optimal by minimizing the loss function and using the Bayesian rule. Since the hypothesis class $\mathcal{H}$ is limited by the architecture of the NN and the SGD can end up in a local and not a global optimum, prediction errors will occur. The method presented in [113, 24], called MetaSeg quantifies the uncertainty of semantically segmented images per predicted segment and will be presented in the following.

MetaSeg is a post-processing tool for semantic segmentation of images. Semantic segmentation models are evaluated using the IoU. Computing the IoU per predicted segment with the ground truth yields a segment-wise IoU. MetaSeg includes a meta classification model to detect false positive segments, i.e., predicted segments with $IoU = 0$ as well as a meta regression model to estimate the segment-wise prediction quality in terms of $IoU \in [0, 1]$. Both meta models, i.e., meta classification and regression model, are fitted on aggregated dispersion measures, here defined as meta data, based on pixel-wise probabilities. In what follows MetaSeg is explained in detail. Furthermore, since MetaSeg was used in other works [115, 13, 83], additional meta data is presented as well.

2.4.1 Dispersion Measures and Segment-wise Aggregation

A segmentation network with a pixel-wise softmax output can be seen as a statistical model. For each image pixel z, a probability distribution

$$\hat{\mathbf{y}}_z^{prob} := f_{\mathbf{w},z}(\mathbf{x}) \in [0, 1]^q \tag{2.72}$$

over the q class labels $c \in \mathcal{C} = \{1, \ldots, q\}$ is provided, with given weights $\mathbf{w}$ and image $\mathbf{x}$; the subscript for weights $\mathbf{w}$ in the left part is omitted for brevity. The predicted class of pixel z is then given by

$$\hat{y}_z = \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, \hat{\mathbf{y}}_z^{prob}\,. \tag{2.73}$$

The true class label of pixel z is $y_z \in \mathcal{C}$. Thus, $\hat{\mathbf{y}}_z^{prob}$ is the probability distribution and $\hat{y}_{z,c}^{prob}$ the probability for class $c \in \mathcal{C}$ and pixel z.

Based on $\hat{\mathbf{y}}_z^{prob}$, the dispersion measures *entropy* E_z, *probability difference* D_z and *variation ratio* V_z are defined as:

$$E_z = -\frac{1}{\log(q)} \sum_{c=1}^{q} \hat{y}_{z,c}^{prob} \log(\hat{y}_{z,c}^{prob}) \,, \tag{2.74}$$

$$D_z = 1 - \max_{c_1 \in \mathcal{C}} \hat{y}_{z,c_1}^{prob} + \max_{c_2 \in \mathcal{C} \setminus c_1} \hat{y}_{z,c_2}^{prob} \,, \tag{2.75}$$

$$V_z = 1 - \max_{c \in \mathcal{C}} \hat{y}_{z,c}^{prob} \,. \tag{2.76}$$

The set of dispersion measures for a pixel z is defined as $\mathcal{M}_z = \{E_z, D_z, V_z\}$.

Representing the above definitions and dispersion measures on image level, i.e., for all pixels z of an image $\mathbf{x}$, this is defined for the reminder of this thesis as:

- image $\mathbf{x} \in [0,1]^{w \times h \times 3}$,
- ground truth $\mathbf{y} \in \mathcal{C}^{w \times h}$,
- probabilities $\hat{\mathbf{y}}^{prob} \in [0,1]^{w \times h \times q}$,
- prediction $\hat{\mathbf{y}} \in \mathcal{C}^{w \times h}$,

and

- entropy $\mathbf{E} \in [0,1]^{w \times h}$,
- probability difference $\mathbf{D} \in [0,1]^{w \times h}$,
- variation ratio $\mathbf{V} \in [0,1]^{w \times h}$,

while $\mathbf{E}, \mathbf{D}$ and $\mathbf{V}$ are also referred to as (dispersion) heatmaps. Furthermore, it is

$$\mathcal{M} = \{\mathbf{E}, \mathbf{D}, \mathbf{V}\} \,. \tag{2.77}$$

For a given image $\mathbf{x}$ the set $\hat{\mathcal{K}}_{\mathbf{x}}$ denotes the set of connected components (segments) of the (predicted) segmentation $\hat{\mathbf{y}}$. Analogously, $\mathcal{K}_{\mathbf{x}}$ denotes the set of connected components in the ground truth $\mathbf{y}$. For each predicted segment $k \in \hat{\mathcal{K}}_{\mathbf{x}}$ the following quantities are defined:

- the interior $k_{in} \subset k$ where a pixel z is an element of k_{in} if all eight neighboring pixels are an element of k,
- the boundary $k_{bd} = k \setminus k_{in}$,
- the size $S = |k|$, $S_{in} = |k_{in}|$, $S_{bd} = |k_{bd}|$.

Based on the above quantities and the pixel-wise dispersion measures, the aggregated dispersion measures for a predicted segment k are defined as:

- the mean μ and variance υ metrics

$$\mu M_\sharp := \mu(M_\sharp) = \frac{1}{S_\sharp} \sum_{z \in k_\sharp} M_z \,, \tag{2.78}$$

$$\upsilon M_\sharp := \upsilon(M_\sharp) = \mu(M_\sharp^2) - \mu(M_\sharp)^2 \,, \tag{2.79}$$

  for $\sharp \in \{\_, in, bd\}$ and $M_z \in \mathcal{M}_z$,

- the relative sizes $\bar{S} = S/S_{bd}$, $\bar{S}_{in} = S_{in}/S_{bd}$,

- the relative mean and variance metrics

$$\tau \bar{M} := \tau(M)\bar{S} \,, \tag{2.80}$$

$$\tau \bar{M}_{in} := \tau(M)\bar{S}_{in} \,, \tag{2.81}$$

  for $\tau \in \{\mu, \upsilon\}$ and all $M \in \mathcal{M}$,

- the ratio of the neighborhood's correct predictions of each class

$$N_c = \frac{1}{|k_{nb}|} \sum_{z \in k_{bd}} \mathbb{1}_{\{c=y_z\}} \qquad \forall c \in \mathcal{C} \,, \tag{2.82}$$

  with k_{nb} the set of neighbors of segment k, i.e., $k_{nb} = \{z' \in [u \pm 1] \times [v \pm 1] \subset \{1, \dots, w\} \times \{1, \dots, h\} : (u, v) \in k, z' \notin k\}$,

- the mean class probabilities

$$P_c = \frac{1}{S} \sum_{z \in k} y_{z,c}^{prob} \qquad \forall c \in \mathcal{C} \,. \tag{2.83}$$

Furthermore, the definition of the target variables is as follows:

- IoU: let $\mathcal{K}_\mathbf{x}|_k$ be the set of all $k' \in \mathcal{K}_\mathbf{x}$ that have non-trivial intersection with k and whose class label equals the predicted class for k, then

$$IoU(k) = \frac{|k \cap K'|}{|k \cup K'|} \,, \qquad K' = \bigcup_{k' \in \mathcal{K}_\mathbf{x}|_k} k' \,, \tag{2.84}$$

- the adjusted IoU_{adj} does not count pixels in the overlapping ground truth segment that are covered by predicted segments, but in other predicted segments of the same class: let $Q = \{q \in \hat{\mathcal{K}}_{\mathbf{x}} : q \cap K' \neq \emptyset\}$, then

$$IoU_{\text{adj}}(k) = \frac{|k \cap K'|}{|k \cup (K' \setminus Q)|} \,. \tag{2.85}$$

In cases where a ground truth segment is covered by more than one predicted segment of the same class, each predicted segment would have a low IoU, while the predicted segments represent the ground truth quite well. The adjusted IoU_{adj} does not punish this situation. The adjusted IoU_{adj} is more suitable for the task of meta regression. For the meta classification it holds $IoU = 0, > 0 \Leftrightarrow IoU_{\text{adj}} = 0, > 0$.

Typically, the dispersion measures E_z, D_z, V_z are large for $z \in k_{bd}$. This motivates the separate treatment of interior and boundary measures. Furthermore a correlation between fractal segment shapes and a bad or wrong prediction can be observed, which motivates the relative sizes $\bar{S},\ \bar{S}_{in}$.

The above definitions lead to a total of $35 + 2q$ metrics: the (relative) mean and variance metrics $\tau M, \tau M_{in}, \tau M_{bd}, \tau \bar{M}, \tau \bar{M}_{in}\ \forall \tau \in \{\mu, \upsilon\},\ \forall M \in \mathcal{M}$, the (relative) size metrics $S, S_{in}, S_{bd}, \bar{S}, \bar{S}_{in}$ as well as $N_c, P_c\ \forall c \in \mathcal{C}$.

Except for the segment-wise IoU and IoU_{adj} values, all quantities defined above can be computed without the knowledge of the ground truth. The task of *meta classification* is to predict whether a predicted segment has an IoU greater or equal to 0. A segment with a $IoU = 0$ is called a false positive segment. Thus, meta classification is in this work also referred to false positive detection. Besides that, the task of *meta regression* is to predict the value of the $IoU \in [0, 1]$. The higher the segment-wise IoU the more does a predicted segment overlap with its ground truth segment.

The classification and regression models that are chosen in [113] are logistic and linear regression, respectively. However, any classification and regression model can be used, e.g., neural networks or tree based ensemble methods [42]. As shown in [83], gradient boosting leads to the best results. *Gradient boosting* [43, 44, 20] is an ensemble method consisting of weak learners for classification and regression tasks. Typically, and assumed for the following, the weak learners are decision trees. Taking the decision trees as an ensemble makes gradient boosting to a strong prediction model. Note, gradient boosting is one of the most powerful methods for classification and regression tasks of structured data [126].

Algorithm 3: Gradient boosting with decision trees

Input: Data $\{\mathbf{x}_i, y_i\}_{i=1,\dots,n}$, differentiable loss function $L(y, g(\mathbf{x}))$, number estimators M, maximum tree depth d, learning rate η

Output: Prediction model $g_M(\mathbf{x})$

1 $g_0(\mathbf{x}) = \text{argmin}_{\hat{y}} \sum_{i=1}^{n} L(y_i, \hat{y})$
2 **for** $m = 1, \dots, M$ **do**
  `// Compute pseudo-residuals for all samples`
3 **for** $i = 1, \dots, n$ **do**
4 $r_{i,m} = -\frac{\partial L(y_i, g_{m-1}(\mathbf{x}_i))}{\partial g_{m-1}(\mathbf{x}_i)}$
5 **end**
6 Fit decision tree for pseudo-residuals $r_{i,m}$ and create terminal regions $R_{j,m} \quad \forall j = 1, \dots, J_m$
7 **for** $j = 1, \dots, J_m$ **do**
8 $\hat{y}_{j,m} = \text{argmin}_{\hat{y}} \sum_{\mathbf{x}_i \in R_{i,j}} L(y_i, g_{m-1}(\mathbf{x}_i) + \hat{y})$
9 **end**
  `// Update prediction model`
10 $g_m(\mathbf{x}) = g_{m-1}(\mathbf{x}) + \eta \sum_{j=1}^{J_m} \hat{y}_{j,m} I(\mathbf{x} \in R_{j,m})$
11 **end**

Gradient boosting, see algorithm 3, works as follows. Given is a data set $\{\mathbf{x}_i, y_i\}_{i=1,\dots,n}$ with n samples and $\mathbf{x} \in \mathbb{R}^p$. The loss function of predicting $\mathbf{x}$ by model $g(\mathbf{x}) = \hat{y}$ is $L(y, g(\mathbf{x}))$. Furthermore, M is the maximum number of weak prediction models, in particular the maximum number of decision trees with maximum depth d. The learning rate is $\eta \in [0, 1]$. First, an initial prediction model $g_0(\mathbf{x})$ is computed, i.e., finding a value for $\hat{y}$ that minimizes the loss value over all samples. This is a constant, see line 1. For a regression problem, this is basically the mean over all labels y_i, $i = 1, \dots, n$. Starting with the first iteration, the pseudo-residuals for every sample $\mathbf{x}_i$ are computed, by taking the negative gradient of the loss function w.r.t. $\mathbf{x}_i$ (line 3–5). Then, a decision tree is fitted to the data set, predicting the pseudo-residuals. This yields in terminal regions $R_{j,m}$ or end leafs of the tree (line 6). Similar to line 1, the loss function is minimized to find the best values $\hat{y}_{j,m}$ for every terminal region (line 7–9). As the last step of the iteration, the prediction model $g_m(\mathbf{x})$ is updated by adding the weighted decision of the decision tree to the previous prediction model $g_{m-1}(\mathbf{x})$. This reduces the loss, i.e., the prediction error and in the next iteration, a new decision tree is fitted to predict the residual-gradient and the process continues as described. Extensions of the presented gradient boosting aim to improve the algorithm in terms of runtime and accuracy. For example, the stochastic gradient boosting [44] does not iterate over all samples of the data set (cf. algorithm 3 line $1,\ 3-5$), only over a randomly sampled subset. XGBoost [20],

short for extreme gradient boosting, is an optimized library of gradient boosting to work in parallel or include some regularization techniques, for instance.

2.4.2 Metrics for Evaluation

The meta classification model outputs a probability for a predicted segment having an $IoU = 0$. For all probabilities equal or greater than 0.5, it is predicted that the IoU is equal to 0, i.e., it is a false positive segment. Then the accuracy $ACC \in [0, 1]$ is the ratio of correct predictions to all predictions. The accuracy corresponds to a single decision threshold. In contrast to that, the area under receiver operating curve ($AUROC$) and area under precision-recall curve ($AUPRC$) are obtained by varying the decision threshold of the classification output [34]. $AUROC$ takes the true positives (TP) and false positives (FP) in terms of their rates into account and thus, essentially measures the overlap of distributions corresponding to negative and positive samples; this score does not place more emphasis on one class over the other in case of class imbalance. $AUPRC$ ignores true negatives and emphasizes the detection of the positive class, i.e., here a correct detected false positive is a true positive in terms of classification.

For the evaluation of the meta regression model, the coefficient of determination R^2 is used. R^2 is a typical evaluation metric for regression tasks and is defined as follows. For the definition of R^2, let be $\gamma_i,\ i = 1, \dots, n$ the true observations, $\hat{\gamma}_i,\ i = 1, \dots, n$ the corresponding regression results and $\bar{\gamma} = \frac{1}{n}\sum_{i=1}^{n} \gamma_i$ the mean of observations. Then the coefficient of determination is

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\gamma_i - \hat{\gamma}_i)^2}{\sum_{i=1}^{n}(\gamma_i - \bar{\gamma}_i)^2}\,. \tag{2.86}$$

For a perfect fitted model, that always predicts the correct values, it is $R^2 = 1$. A model that always predicts the mean, has a coefficient of determination of zero, i.e., $R^2 = 0$. Theoretically, R^2 can be negative if the model has worse predictions.

Numerical experiments are presented in [113]. The performance of MetaSeg is demonstrated on semantic segmentations of street scenes with the Deeplab model. For meta classification, an accuracy of $ACC = 81.91\%$ and $AUROC$ of 87.71% on a validation set is achieved. For meta regression the coefficient of determination is $R^2 = 74.97\%$, also on a validation set. The performance of meta regression, i.e., the prediction quality estimation is shown in figure 2.21.

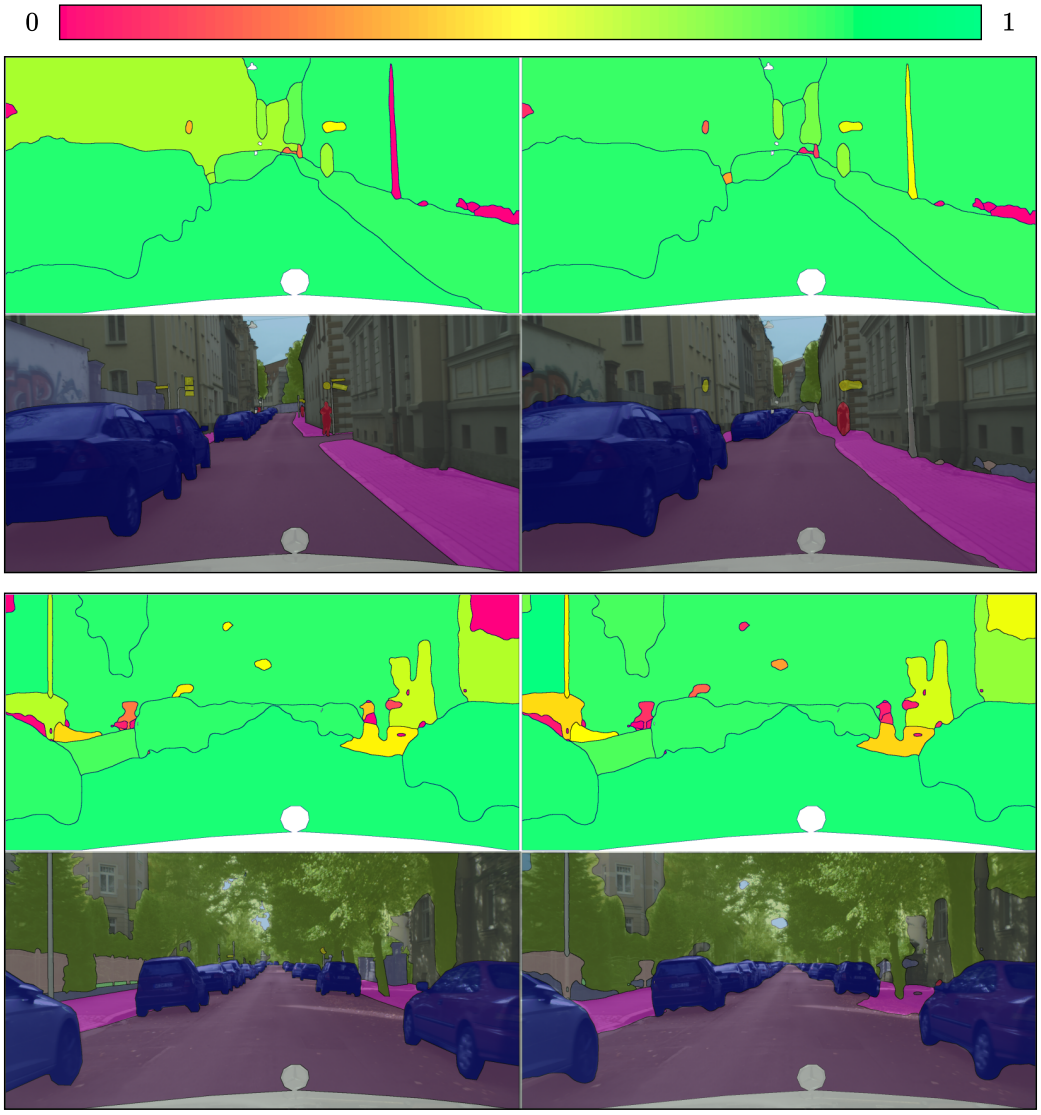


Figure 2.21: Two demonstrations (top and bottom four panels) of MetaSeg's performance of predicting IoU_{adj}. Each visualization contains the ground truth (bottom left), the semantic segmentation (bottom right), the true IoU for every predicted segment (top left) and the estimated IoU for every segment (top right). The colorbar at the top is for the segment-wise IoU values.

3 Region-Based Active Learning using Priority Maps for Image Segmentation

In order to achieve a high accuracy of DL models, it is important, among other things, to have a large amount of annotated and varying training data. The annotation of data is time consuming and costly, especially for the task of semantic segmentation it requires very high human annotation effort. In this chapter, a method is presented to reduce the annotation effort and at the same time maintaining a high model accuracy. The method uses the prediction quality estimation of MetaSeg to select image regions to be labeled, i.e., unseen image regions where it is assumed that the segmentation model is not able to infer them correctly. In addition, this will be combined with a cost estimation to be even more cost efficient in terms of annotation effort.

3.1 Introduction

In recent years, semantic segmentation, the pixel-wise classification of the semantic content of images, has become a standard method to solve problems in image and scene understanding. Examples of applications are advanced driver assistant systems (ADAS), automated driving (AD) and environment understanding [149, 19, 139], biomedical analyses [112] and further computer visions tasks. CNNs are commonly used in semantic segmentation, see section 2.3.1. In order to maximize the accuracy of a CNN, a large amount of annotated and varying data is required, since with an increasing number of samples the accuracy increases only logarithmically [130]. For instance, in the field of AD, fully and precisely annotated street scenes require an enormous and tiring annotation effort. Also biomedical applications, in general domains that require expert knowledge for annotation, suffer from high annotation costs. Hence, from multiple perspectives (annotation) cost reduction while maintaining model accuracy is highly desirable.

Active Learning. One possible approach to solve this problem is *active learning* (AL), which basically consists of alternatingly annotating data and training a model with the currently available annotations. The key component in this algorithm that can substantially leverage the learning process is the so-called query or acquisition strategy. The ultimate goal is to label the data that leverages the model accuracy most, while paying with as small labeling costs as possible. A typical AL process is as follows: it is assumed that a model and a data pool with unlabeled data is given. First, a small subset of samples of the data pool is labeled, which is then used to train the CNN. The model infers all the unlabeled data and based on an acquisition function, a pre-defined number of samples are labeled. An example for acquisition is to label the data that has the highest uncertainties in terms of the highest entropy of a predicted sample. The iterative process ends when a stopping condition is reached, e.g., if the model achieves a pre-defined accuracy or the annotations effort becomes too high. For an introduction to AL methods, see e.g., [120].

Image Annotation. The annotation effort of images depends on the task they will be used for. For classification, only one label must be assigned for the whole image. For object detection, simplified, the boxes of relevant object classes as well as their respective class need to be set. For semantic segmentation each pixel requires a label. Semantic segmentation data sets are generally labeled with a polygon-based annotation tool, i.e., the objects are described by a finite number of vertices connected by edges such that the latter form a closed loop [29], [97]. In tools such as labelme [117], the human annotator defines a segment as closed polygon by clicking around the edges of the object. Additionally, the class of the segment has to be assigned. Figure 3.1 shows a visualization of image segmentation ground truth and clicks for annotation. As it can be seen, the number of clicks for annotation depends on the number of ground truth objects, the complexity of the scene, i.e., how many class objects are given and the shape of the contours. Objects with curve structures require more clicks to approximate the shape with a polygon compared to objects without curve structures like a bus which is approximately a rectangle; a straight line requires only the start and end coordinate and thus only two clicks. However, it is also obvious that some image regions require more clicks for annotation than other e.g., in figure 3.1 the image center – mainly consisting of pixels belonging to class 'street' – requires significantly fewer clicks than the right or upper part of the image. The image from figure 3.1 is from the Cityscapes data set [29] which is a common data set for semantic segmentation. The annotation of an image (from the Cityscapes data set) takes in average $1{,}590$ clicks and up to two hours. Furthermore, an image has on average 95 ground truth segments. Thus, for the remainder of this

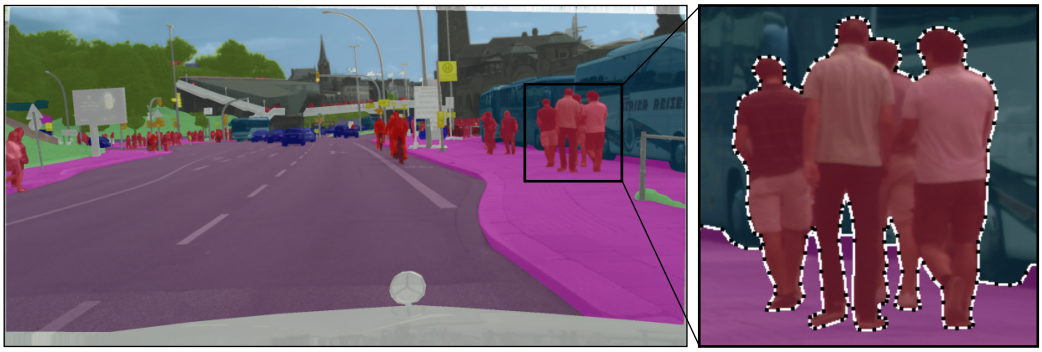


Figure 3.1: Visualization of an image and the corresponding ground truth. Each color represents a class. The right part illustrates the annotation clicks of the pedestrians. The segment boundaries are shown in white, the clicks are shown as black dots.

chapter, the annotation effort or costs are associated with the number of clicks for annotating an image. In section 3.4 more details for an evaluation of costs in terms of clicks will be presented.

In this chapter, a new region-based active learning method is presented. Due to the motivation above, only image regions, in what follows called boxes, are queried. The meta regression of MetaSeg for prediction quality estimation is used to identify predicted image regions with a low estimated quality in terms of segment-wise IoU. Furthermore, a click estimation method is introduced and incorporated in order to take the estimated annotation costs into account. Both selection criterions are aggregated and combined. This outlined procedure is called *MetaBox+*.

The contributions are summarized as follows:

- defining a new region-based active learning method that queries image regions for annotation based on one or multiple combined selection criterions called *priority maps*,
- using the segment-wise prediction quality estimation by MetaSeg as priority map, in order to select poorly predicted image regions,
- using a new and practical method to estimate the annotation costs in terms of clicks and combining this with the priority map via MetaSeg, which yields the novel AL method MetaBox+,
- exhausting numerical experiments on the Cityscapes data set [29] with two models, namely FCN8 [82] and Deeplab [19], and defining new evaluation metrics for region-based active learning and an in depth-study of the experiments.

The first section gave an introduction into the topic and active learning. Furthermore, details about the annotation process of images have been presented. In section 3.2, the related work is reviewed. Afterwards, in section 3.3, a new AL method is presented that prioritizes image regions to be labeled. Experiments and results are given in section 3.4, where the new method is evaluated and compared to baseline methods. The chapter ends with a discussion in section 3.5.

3.2 Related Work

First AL approaches (before the deep learning era) for semantic segmentation, for instance based on conditional random fields, go back to [137, 72, 64, 95]. In general, uncertainty sampling is one of the most common query strategies [140, 46, 114, 54, 4], besides that there also exist approaches on expected model change [137] and reinforcement learning-based AL [12]. In recent years, approaches to deep AL for semantic segmentation have been introduced, primarily for two applications, i.e., biomedical image segmentation and semantic street scene segmentation. The approaches in [147, 51, 100, 85], are specifically designed for medical and biomedical applications, mostly focusing on foreground-background segmentation. Due to the underlying nature of the data, these approaches refer to annotation costs in terms of the number of labeled images. The methods presented in [84, 127, 66, 12] use region-based proposals. All of them evaluate the model accuracy with $mIoU$ (cf. equation (2.64)).

The method in [127] is designed for multi-view semantic segmentation data sets, in which objects are observed from multiple viewpoints. The authors introduce two new uncertainty-based metrics and aggregate them on superpixel (SP) level (SPs can be viewed as visually uniform clusters of pixels). They measure the costs by the number of labeled pixels. Furthermore, they have shown that labeling on SP level can reduce the annotation time by up to 25%. However, in applications of semantic street scene segmentation the scenes are not given from multiple views.

In [66] a new uncertainty metric on SP level is defined, which includes the information of the (Shannon) entropy [122], combined with information about the contours in the original image and a class-similarity metric to put emphasis on rare classes. The queried regions consist of neighboring SPs and are not in a pre-defined (equal) shape or size. In [66], the number of labeled pixels define the costs. As already motivated, the annotation costs in terms of labeled pixels do not reflect the human annotation effort as good as the number of required clicks – this will also be shown

in section 3.4 in detail. Furthermore, the human annotation effort of labeling image regions of varying shapes and sizes is not studied properly. The click estimation approach in MetaBox+ also takes contours into account, but not those of the image, but those of the segmentation, as will be explained in the next section. Moreover, this chapter analyzes the annotation effort in a real-world scenario.

The authors of [12] utilize the same cost metric, i.e., the number of pixels as it is done in [66]. The latter work uses reinforcement learning to find the most informative regions, which are given in quadratic format of fixed size. This procedure aims at finding regions containing instances of rare classes.

The method in [84] queries quadratic regions of fixed size from images. In contrast to the methods discussed before, the costs are measured by annotation clicks. They use a combination of uncertainty measure (Vote [32] and Shannon entropy[122]) and a clicks-per-polygon-based cost estimation, which is regressed by a second DL model. However, instead of using an uncertainty measure to identify high informational regions, MetaBox+ uses information about the estimated segmentation quality in terms of the segment-wise IoU (see section 2.4). Entropy is a common measure to quantify uncertainty. However, in semantic segmentation, one can observe increased uncertainty on segment boundaries, while uncertainty in the interior is often low. This is in line with the observation that, neural networks in general provide overconfident predictions [50, 57]. The new region-based active learning method solves this problem by evaluating the segmentation quality of whole predicted segments. Furthermore, also the cost estimation method differs substantially from the one presented in [84]. While the authors of [84] use another DL model to regress on the number of clicks per image region, here the estimate of number of clicks are inferred directly from the prediction of the segmentation network.

3.3 A new Method for Region-Based Active Learning

This section first describes a region-based AL method which queries fixed size and quadratic image regions by means of priority measures and priority maps. Afterwards, the priority measures based on MetaSeg and a cost estimation are described, which leads to the new AL method. This method is subdivided into a process of two steps: first, the segmentation quality is estimated by using the segment-wise meta regression of MetaSeg. Second, a cost estimation of the click number required to label a region is incorporated. At the end of this section, baseline methods are presented.

3.3.1 Acquisition with Priority Maps

For the AL method, a CNN is assumed as semantic segmentation model with pixel-wise softmax probabilities as output. The corresponding segmentation mask is the pixel-wise application of the argmax function to the softmax probabilities. The data set is given as data pool $\mathcal{P}$. The set of all labeled data is denoted by $\mathcal{L}$ and the set of unlabeled data by $\mathcal{U}$. At the beginning, no data is labeled, i.e., $\mathcal{L} = \emptyset$. The pixel-wise labels to be set are over a given number of classes $c \in \mathcal{C} = \{1, \dots, q\}, q \geq 2$.

A generic AL method can be summarized as follows: Initially, a small set of data from $\mathcal{U}$ is labeled and added to $\mathcal{L}$. Then, two steps are executed alternatingly in a loop. First, the segmentation model is trained on $\mathcal{L}$. Thereafter, a chosen amount of unlabeled data from $\mathcal{U}$ is queried according to a query strategy, labeled and added to $\mathcal{L}$. In region-based AL, newly labeled regions are added to $\mathcal{L}$ instead of whole images. An image $\mathbf{x}$ remains in $\mathcal{U}$ as long as it is not entirely labeled. In order to avoid multiple queries of the same region, a region that is contained in both $\mathcal{U}$ and $\mathcal{L}$ is tagged with a query priority equal to 0. In the remainder of this section, the query function is described in detail and the appropriate concept of priority in the given context is introduced.

Region-Based Queries. The query strategy is a key ingredient of an AL method. In general, most query function designs strive for maximally leveraging training progress, i.e., achieving high validation accuracy after short time with reduced labeling costs. Thus, it is aimed for querying regions of images, leading to a region-specific concept of query priority. In what follows, only *measures of priority* are computed by means of the softmax output of the CNN. To this end, let

$$f : [0,1]^{w \times h \times 3} \to [0,1]^{w \times h \times q} \tag{3.1}$$

be a function given by a segmentation network providing softmax probabilities for a given input image $\mathbf{x}$, where w denotes the image width, h the height and q the number of classes. This can also be written as $f(\mathbf{x}) = \hat{\mathbf{y}}^{prob}$, which is the softmax (probability) output.

A *priority map* can be viewed as another function

$$g : [0,1]^{w \times h \times q} \to [0,1]^{w \times h} \tag{3.2}$$

that outputs one priority score per image pixel. The output of g can be viewed as a heatmap that indicates priority. A higher score of priority should presumably

correlate with the attractiveness of the corresponding ground truth. A typical example for g is the pixel-wise entropy E which for a chosen pixel (i, j) is given by

$$E(\hat{y}_{i,j}) = -\frac{1}{\log(q)} \sum_{c=1}^{q} \sum_{c=1}^{q} \hat{y}_{i,j,c}^{prob} \log(\hat{y}_{i,j,c}^{prob}) \,, \tag{3.3}$$

where $\hat{y}_{i,j,c}^{prob} = f(\mathbf{x})_{i,j,c} \in [0, 1]$ for a given input $\mathbf{x} \in [0, 1]^{w \times h \times 3}$. Note that, if an image pixel has already been labeled, the corresponding pixel value of the priority map is overwritten by 0.

The AL method queries regions that are square-shaped (*boxes*) and of fixed width $b \in \mathbb{N}$. A box-wise overall priority score is obtained via aggregation. To this end, the scores are summed up. That is, given a box $B \subset [0, 1]^{w \times h}$, the aggregated score is given by

$$g_{agg}(\hat{\mathbf{y}}^{prob}, B) = \sum_{(i,j) \in B} g_{i,j}(\hat{\mathbf{y}}^{prob}) \,. \tag{3.4}$$

Given the set $\mathcal{B}$ of all possible boxes of width $b \in [0, 1]^{w \times h}$, the *aggregated priority map* is defined as

$$g_{\mathcal{B}}(\hat{\mathbf{y}}^{prob}) = \{g_{agg}(\hat{\mathbf{y}}^{prob}, B) : B \in \mathcal{B}\} \,, \tag{3.5}$$

which can be viewed as another heatmap resulting from a convolution operation with a constant filter. Given t aggregated priority maps, for the sake of brevity named $h^{(1)}(\hat{\mathbf{y}}^{prob}, B), \ldots, h^{(t)}(\hat{\mathbf{y}}^{prob}, B)$, a *joint aggregated priority map* is defined by

$$h(\hat{\mathbf{y}}^{prob}, B) = \prod_{s=1}^{t} h^{(s)}(\hat{\mathbf{y}}^{prob}, B) \,, \tag{3.6}$$

where all aggregated priority maps are normalized. However, in what follows no distinction is made between joint (aggregated) priority maps and singleton (aggregated) priority maps as this follows from the context. Furthermore, it will be only referred to priority maps while performing calculations on priority score level.

Algorithm. In summary, the AL method proceeds as follows. Initially, a randomly chosen set of m_{init} entire images from $\mathcal{U}$ is labeled and then moved to $\mathcal{L}$. Afterwards, the AL method proceeds as previously described in the introduction of this section. Defining the set of all *candidate boxes* as

$$C = \{(\hat{\mathbf{y}}^{prob}, B) := f(\mathbf{x}),\, \mathbf{x} \in \mathcal{U},\, B \in \mathcal{B}\} \,, \tag{3.7}$$

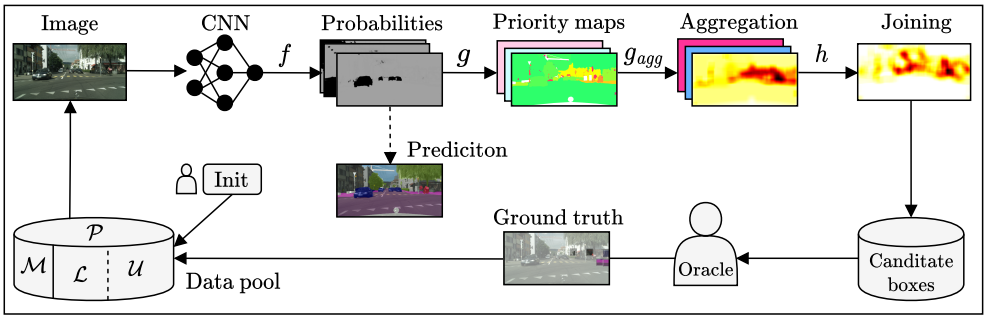


Figure 3.2: Illustration of the region-based AL method. Based on the network probabilities for a predicted image, the priority maps are computed, followed by the aggregation and joining. The candidate boxes are selected by their highest priority scores. After labeling the image regions, the data pool is updated and the process continues by predicting unlabeled images to select the next regions to be labeled.

in each iteration a chosen number m_q of *non-overlapping* boxes $Q = \{(\hat{y}_{i_j}^{prob}, B_j) : j = 1, \ldots, m_q\} \subset C$ are queried, with the highest scores $h(\hat{\mathbf{y}}^{prob}, B)$, i.e.,

$$\begin{aligned} &(\hat{\mathbf{y}}^{prob}, B) \in Q,\ (\hat{\mathbf{y}}^{prob'}, B') \notin Q \\ \Longrightarrow\ &h(\hat{\mathbf{y}}^{prob}, B) \geq h(\hat{\mathbf{y}}^{prob}, B') \text{ or } (B \cap B' \neq \emptyset \text{ and } \hat{\mathbf{y}}^{prob} = \hat{\mathbf{y}}^{prob'})\,. \end{aligned} \tag{3.8}$$

A sketch of the whole AL loop is depicted by figure 3.2.

3.3.2 Joint Priority Maps Based on Prediction Quality Estimation and Click Estimation

For the new active learning method MetaBox+, it remains to specify the priority maps $h^{(s)}(\hat{\mathbf{y}}^{prob}, B), s = 1, 2$ defined in the previous section. For the prediction quality estimation, the method MetaSeg is used which provides a quality estimate in $[0, 1]$ for each segment predicted by f, see section 2.4. This aims at querying ground truth for image regions that presumably have been predicted badly. Mapping predicted qualities back to each pixel of a given segment and thereafter aggregating the values over boxes, yields to the priority map $h^{(1)}(\hat{\mathbf{y}}^{prob}, B)$.

On the other hand, queried image regions shall be easy or cheap to label. Therefore, the required clicks to annotate a box B are estimated. From this, another priority map $h^{(2)}(\hat{\mathbf{y}}^{prob}, B)$ is defined which contains high values for regions with low estimated numbers of clicks and vice versa.

Finally, the boxes are queried according to the product of priorities, i.e.,

$$h(\hat{\mathbf{y}}^{prob}, B) = h^{(1)}(\hat{\mathbf{y}}^{prob}, B) \cdot h^{(2)}(\hat{\mathbf{y}}^{prob}, B) \; . \tag{3.9}$$

In what follows, the priority maps $g^{(1)}(\hat{\mathbf{y}}^{prob})$ and $g^{(2)}(\hat{\mathbf{y}}^{prob})$ are described more precisely, where the aggregated priority maps $h^{(1)}(\hat{\mathbf{y}}^{prob}, B)$ and $h^{(2)}(\hat{\mathbf{y}}^{prob}, B)$ are constructed as in section 3.3.1.

Priority via MetaSeg. For the prediction quality estimation, the meta regression model in MetaSeg has to be trained, see section 2.4. Since MetaSeg is based on metrics that take the prediction's dispersion measures into account, only data with ground truth should be used, that has not been used in training the neural network. Therefore, an initial data set $\mathcal{M}$ of n_{meta} samples is selected. At least, it should contain ground truth segments of every class $c \in \mathcal{C}$. Otherwise, MetaSeg would not be able or even trained to make estimations for those classes. This data set is fixed for the whole AL process. Since the network f is trained after each update of the data pool $\mathcal{P}$, the predictions change and the meta regression model for MetaSeg is re-trained as well.

To predict the quality of network predictions via MetaSeg, the following steps are performed after updating the semantic segmentation model:

1. infer the current CNN's predictions for all images in $\mathcal{M}$,
2. compute the metrics for each predicted segment (from 1.),
3. train MetaSeg to predict the IoU by means of the metrics (from 2.),
4. infer the current CNN's predictions for the unlabeled data $\mathcal{U}$,
5. compute the metrics for each predicted segment that belongs to $\mathcal{U}$ (from 4.),
6. apply MetaSeg in inference mode (from 3.) to each predicted segment from $\mathcal{U}$ (from 4.) and its metrics (from 5.) to predict the IoU.

For each unlabeled, i.e., not entirely labeled image, MetaSeg provides a segmentation quality heatmap $\zeta(\hat{\mathbf{y}}^{prob})$ by registering the predicted IoU values of the predicted segments for each of their corresponding pixels. An example of the segmentation quality heatmap is given in figure 3.3. The corresponding priority map, as defined in equation (3.2), is obtained via $g^{(1)}(\hat{\mathbf{y}}^{prob}) = \mathbf{1} - \zeta(\hat{\mathbf{y}}^{prob})$. Hence, regions of $g^{(1)}(\hat{\mathbf{y}}^{prob})$ containing relatively high values are considered as being attractive for acquisition.

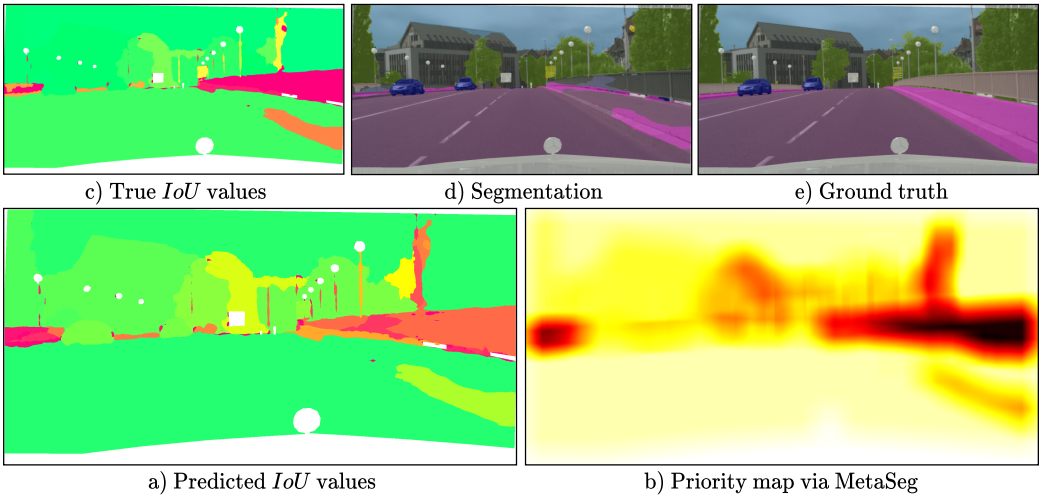


Figure 3.3: Prediction of the IoU values and priority map via MetaSeg. The figure consists of the predicted IoU values for the predicted segments (a), the corresponding aggregated priority map (b), the true IoU values for the predicted segments (c), the predicted segments (d) and the ground truth (e). Low IoU values are visualized in red, middle ones in yellow and high values in green. In the priority map, darker regions correspond to higher priority for acquisition. Overall, for white regions no ground truth is available.

Note, the main contribution of this chapter is the AL method MetaBox+, that takes the here presented quality estimation into account as well as the following presented click estimation. However, only taking the priority map via MetaSeg into account is referred to the AL method *MetaBox*.

Priority via Estimated Number of Clicks. The second priority map $g^{(2)}(\hat{\mathbf{y}}^{prob})$ is an estimation of the annotations costs. Multi-class semantic segmentation data sets are generally labeled with a polygon-based annotation tool, i.e., the objects are described by a finite number of vertices connected by edges such that the latter form a closed loop [29, 97]. If the ground truth is given only pixel-wise, an estimate of the number of required clicks can be approximated by applying the Ramer-Douglas-Peucker (RDP) algorithm [38, 110], to the segmentation contours. The RDP algorithm transforms a curve consisting of line segments into a similar curve with fewer points. The maximum distance between the original and approximated curve is controlled by a parameter ϵ, the lower ϵ the smaller is the maximum distance.

To estimate the true number of clicks required for annotation in the AL process, this number is correlated with how many clicks it approximately requires annotating the predicted segmentation (provided by the current CNN) using the RDP algorithm. The approximation accuracy of the RDP algorithm is controlled by a parameter ϵ.

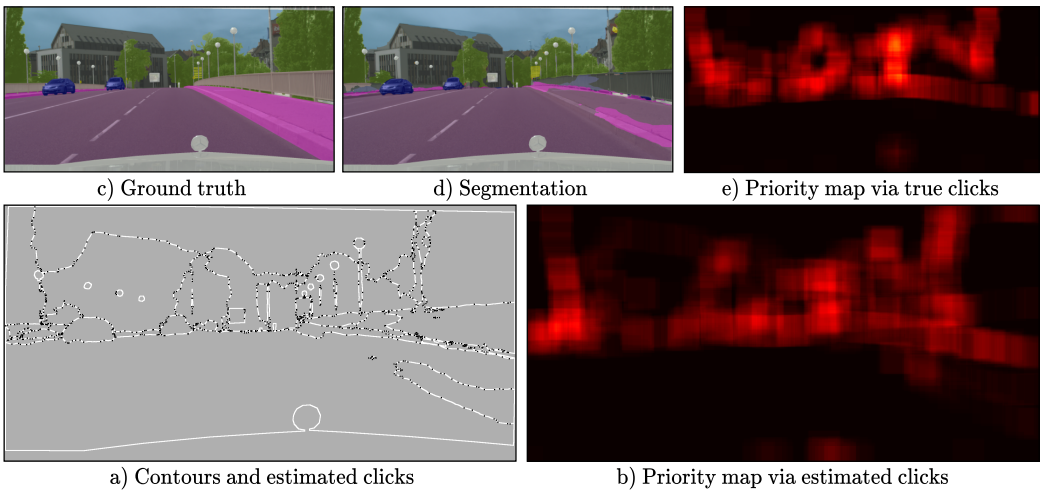


Figure 3.4: Click estimation and priority map via estimated number of clicks. The figure consists of the predicted segment's contours (white) and estimated clicks (black) (a), the corresponding aggregated priority map (b), the ground truth (c), the predicted segments (d) and the priority map of true annotation clicks for visualization purposes (e). In the priority maps, darker regions correspond to higher priority for acquisition. In c) and d) for white regions no ground truth is available.

For this purpose, a cost map $\xi(\hat{\mathbf{y}}^{prob})$ is defined via $\xi(\hat{y}_{i,j}^{prob}) = 1$ if there is a polygon vertex in pixel (i, j) and $\xi(\hat{y}_{i,j}^{prob}) = 0$ else. Since regions with low estimated costs are prioritized, the priority map is given by $g^{(2)}(\hat{\mathbf{y}}^{prob}) = \mathbf{1} - \xi(\hat{\mathbf{y}}^{prob})$. Following the construction in section 3.3.1 yields the aggregated priority map $h^{(2)}(\hat{\mathbf{y}}^{prob}, B)$. A visual example of the cost estimation is given in figure 3.4.

Tests with the RDP algorithm applied to ground truth segmentations have shown that on average, the estimated number of clicks is fairly close to the true number of clicks provided by the Cityscapes data set. Therefore, it is assumed that over the course of AL iterations, the model accuracy increases, approaching a level of segmentation quality that is close to ground truth, then the described cost estimation on average will approach the click numbers in the ground truth.

As already mentioned, *MetaBox* uses only the priority via MetaSeg and *MetaBox+* uses both the joint priority of MetaSeg and the estimated number of clicks. An overview of the different steps of the new AL method is given by figure 3.2 and an exemplary visualization of the different stages of MetaBox+ is shown in figure 3.5.

Further Priority Maps and Baseline Methods. For the sake of comparison, a priority map based on the pixel-wise entropy as in equation (3.3) is defined. Analogously to MetaBox and MetaBox+, the methods *EntropyBox* and *EntropyBox+* are introduced:

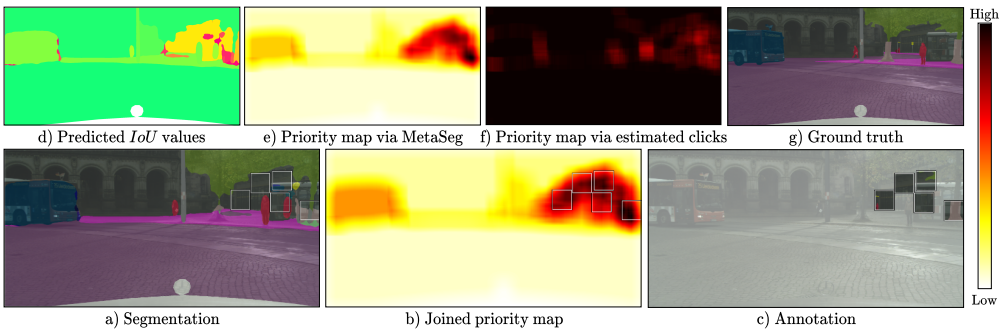


Figure 3.5: The figure consists of the predicted segments (a), the joined priority map (b) and the acquired annotation (c). The joined priority map (b) is based on the priority map via MetaSeg (e) and the priority map via estimated number of clicks (f). High values represent prioritized regions for labeling: in e) regions with low predicted IoU values are of interest and in f) regions with low estimated clicks. Additionally, d) shows the predicted IoU values via MetaSeg and g) the ground truth. Low IoU values are visualized in red, middle ones in yellow and high values in green. In the priority map, darker regions correspond to higher priority for acquisition. Overall, for white regions no ground truth is available.

EntropyBox uses only the priority via entropy and EntropyBox+ uses the joint priority of the entropy and the estimated number of clicks. The method EntropyBox+ is similar to the method presented in [84]. The corresponding authors also use a combination of the entropy and a cost estimation, but the cost estimation is computed by a second DL model. Furthermore, as a naive baseline, a random query function is considered that performs queries by means of random priority maps. This is called *RandomBox*.

3.4 Experiments and Results

This section first introduces new evaluation metrics for region-based AL or rather region-based annotation. To this end, different types of clicks required for labeling are discussed and how they can be taken into account for defining annotation costs. Afterwards, the experiment settings are described. Finally, the numerical results are presented and different query strategies as introduced in the previous section are compared w.r.t. accuracy and robustness.

3.4.1 Metrics for Evaluation

In semantic segmentation, annotation is usually generated with a polygon-based annotation tool. A segment of a given class is represented by a polygon, i.e., a closed path of edges. This path is constructed by a human labeler clicking at the corresponding vertices. Here, these vertices polygon clicks are termed as $c_p \in \mathbb{N}$. These are the clicks that have been motivated in section 3.1. Since quadratic image regions (boxes) are queried, the following additional types of clicks are introduced:

- intersection clicks $c_i(B) \in \mathbb{N}_0$, occur due to the intersection between the contours of a segment and the box boundary,
- box clicks $c_b(B) \in \mathbb{N}_0$, specify the quadratic box itself,
- class clicks $c_c(B) \in \mathbb{N}$, specify the class of the annotated segment.

For an annotated image, the class clicks correspond to the number of segments. Just like the polygon clicks, they can also be considered for the cost evaluation of fully annotated images.

For the evaluation of a data set $\mathcal{P}$ (with fully labeled images), $c_p(\mathcal{P}) \in \mathbb{N}$ is the total number of polygon clicks and $c_c(\mathcal{P}) \in \mathbb{N}$ the total number of class clicks. Let $\mathcal{L}_0$ be the initially annotated data set (with fully labeled images) and Q the set of all queried and annotated boxes, then the cost metrics are defined as:

$$cost_A = \frac{c_p(\mathcal{L}_0) + c_c(\mathcal{L}_0) + c_p(Q) + c_i(Q) + c_c(Q)}{c_p(\mathcal{P}) + c_c(\mathcal{P})}\,, \tag{3.10}$$

$$cost_B = \frac{c_p(\mathcal{L}_0) + c_p(Q) + c_i(Q) + c_b(Q)}{c_p(\mathcal{P})}\,, \tag{3.11}$$

$$\text{with } c_\sharp(Q) = \sum_{B_j \in Q} c_\sharp(B_j), \quad \sharp \in \{p, i, b, c\}\,.$$

In addition to that,

$$cost_P \tag{3.12}$$

defines the costs as amount of labeled pixels w.r.t. the whole data set.

The amount of required clicks depends on the annotation tool. The box clicks $c_b(B)$ are not necessarily required: with a suitable tool, the chosen image regions (boxes) are suggested and the annotation process restricted accordingly. Required are the polygon clicks $c_p(B)$ and the intersections clicks $c_i(B)$ to define the segment

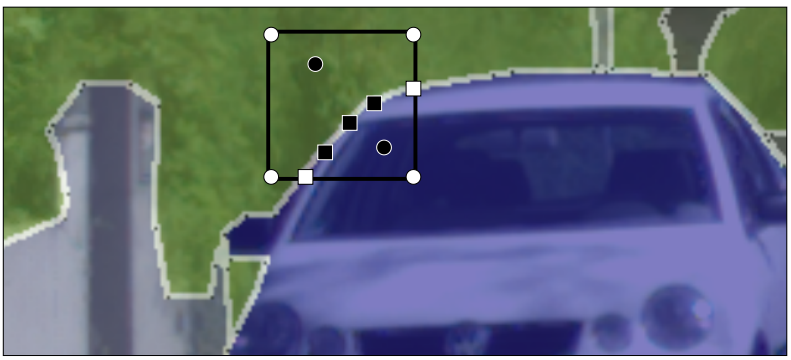


Figure 3.6: Illustration of possible click types that can be taken into account for a cost definition. The black squares depict true annotation clicks obtained from the data (here 3 polygon clicks within the box). The other ones represent additional types of clicks: the ones required for annotating intersection points of segment contour (white squares, here 2 intersection clicks), the ones for defining the box itself (white circles, 4 box clicks, one for each corner) and one click per segment to specify the class of the segment (black circles, here 2 class clicks: for car and vegetation).

contours as well as the class clicks $c_c(B)$ to define the class of the annotated segment. Cost metric $cost_A$ (equation (3.10)) is based on this consideration.

Cost metric $cost_B$ (equation (3.11)) is introduced in [84]. Cost metric $cost_B$ is mostly in accordance with $cost_A$, except for two modifications. The box clicks $c_b(B) = 4$ are taken into account, while the class clicks $c_c(B)$ are omitted. Theoretically, both metrics can become greater than 1. First, fully labeled images do not require intersection clicks c_i. Second, ground truth segments that are labeled by more than one box produce multiple class clicks c_c to specify the class. In the following, the cost metrics $cost_A$, $cost_B$, $cost_P$ are given in percent of the costs for labeling the full data set without considering regions. An illustration of the click types is shown in figure 3.6.

3.4.2 Experiment Settings

For the experiments, the Cityscapes [29] data set has been used. It contains images of urban street scenes with 19 classes for the task of semantic segmentation. Furthermore, the polygons and annotation clicks are given. The training set with 2,975 samples has been chosen as data pool $\mathcal{P}$. For all model and experiment evaluations, the validation set containing 500 samples has been used.

The two CNN models *FCN8* [82] (with width multiplier 0.25 introduced in [118]) and Deeplabv3+ [19] with an Xception65 [23] backbone, (short: Deeplab) have

been used. More details to both networks are given in section 2.3.1. Note, FCN8 is the same architecture as FCN, but with fewer filters to reduce the training time.

Using all training data, also referred to as full set, yields to an accuracy of $mIoU = 60.50\%$ on the validation data set with the FCN8 model and a $mIoU = 76.11\%$ for the Deeplab model. The images ares not resized, i.e., the original resolution of height $h = 1{,}024$ and width $w = 2{,}048$ were taken. In each AL iteration, the CNN model is trained from scratch. The training is stopped, if no improvement in terms of validation $mIoU$ is achieved over 10 consecutive epochs. All experiments started from an initial data set of 50 samples. For experiments with MetaSeg-based queries (MetaBox, MetaBox+), 30 additional samples were taken to train MetaSeg. In each AL iteration, $6{,}400$ boxes were queried with a width of $b = 128$, which corresponds to 50 full images in terms of the number of pixels.

Each experiment was repeated three times. For each method, in the following the average of the $mIoU$ and the mean over the cost metrics ($cost_A$, $cost_B$, $cost_P$) of each AL iteration are presented. All CNN trainings were performed on NVIDIA Quadro P6000 GPUs. In total, the Deeplab model was trained 180 times, which required approximately $8{,}000$ GPU hours and the FCN8 model 290 times, which required approximately $3{,}500$ GPU hours. This amounts to $11{,}500$ GPU hours in total. On top of that, a few additional GPU hours were consumed for the inference, as well as a moderate amount of CPU hours for the query process.

To train the CNN models with only parts of the images, the labels of the unlabeled regions were set to ignore values. Both models (FCN8 and Deeplab) are initialized with pre-trained weights of Imagenet [36]. For training the FCN8 model, the Adam optimizer [71] was used with learning rate, alpha and beta set to 0.0001, 0.99, and 0.999, respectively. The batch size was 1; no data augmentation was used. For the training of the Deeplab model, it was proceeded as in [19]: the decoder output stride was set to 4, train crop size to 769×769, atrous rate to $6, 12, 18$ and output stride to 16. To consume less GPU memory resources, a batch size of 4 was used, and the batch norm parameters were not fine-tuned. For the training in the AL iterations, a polynomial decay learning rate policy was taken:

$$\eta^{(i)} = \eta_{base} \cdot \left(1 - \frac{s^{(i)}}{s_{tot}}\right)^{p},$$

where $\eta^{(i)}$ is the learning rate in step $s^{(i)} = i$, $\eta_{base} = 0.001$ the base learning rate, $p = 0.8$ the learning power and $s_{tot} = 150{,}000$ the total number of steps. For training with the full set the same learning rate policy with a base learning rate $\eta_{base} = 0.003$

was used. Following these settings, an accuracy of $mIoU = 76.11\%$ (mean of 5 runs) is achieved. The original model achieves a $mIoU$ of 78.79% (with batch size of 8).

As meta regression model, a gradient boosting was used with 100 estimators, max depth 4 and learning rate 0.1, see section 2.4.1. MetaSeg was trained with the predictions of 30 images. Tests have shown, this is sufficient to achieve results ins terms of $R^2 \approx 0.75$ similar to those presented in [113]. Using more images to incorporate into the training of MetaSeg would increase the initial costs of annotation.

3.4.3 Evaluation

In what follows, the results of the MetaSeg-based methods MetaBox and MetaBox+, the entropy-based methods EntropyBox and EntropyBox+ as well as the random method RandomBox are compared. EntropyBox+ is a similar method as presented in the best performing region-based active learning method in [84]. Note, this is a similar re-implementation; the code for the original method is not available.

Comparison of MetaBox and EntropyBox. First, the methods that do not include the cost estimation are compared. As can be seen in figure 3.7, MetaBox outperforms EntropyBox in terms of annotation costs required to achieve 95% full set $mIoU$: for the FCN8 model, MetaBox produces click costs of $cost_A = 38.63\%$ while EntropyBox produces costs of $cost_A = 44.60\%$. For analogous experiments with the Deeplab model, MetaBox produces click costs of $cost_A = 14.48\%$ while EntropyBox produces costs of $cost_A = 19.61\%$. Furthermore, for the Deeplab model both methods perform better compared to RandomBox, which requires costs of $cost_A = 22.04\%$. However, for the FCN8 model RandomBox produces the least click costs of $cost_A = 34.54\%$. Beyond the 95% (full set $mIoU$) frontier, all three methods perform very similar on the FCN8 model. For the Deeplab model, RandomBox does not significantly gain accuracy while MetaBox and EntropyBox achieve the full set $mIoU$ requiring approximately the same costs of $cost_A \approx 36.56\%$.

Figure 3.8 shows a visualization of prioritized regions for annotation. In general, high entropy values are observed on the boundaries of predicted segments. Therefore, EntropyBox queries boxes, which overlap with the contours of predicted segments. Since MetaBox prioritizes regions with low predicted IoU values, queried boxes often lie in the interior of predicted segments. Furthermore, EntropyBox produces higher costs in each AL iteration compared to MetaBox and RandomBox. RandomBox produces relatively small but very consistent costs per AL iteration.

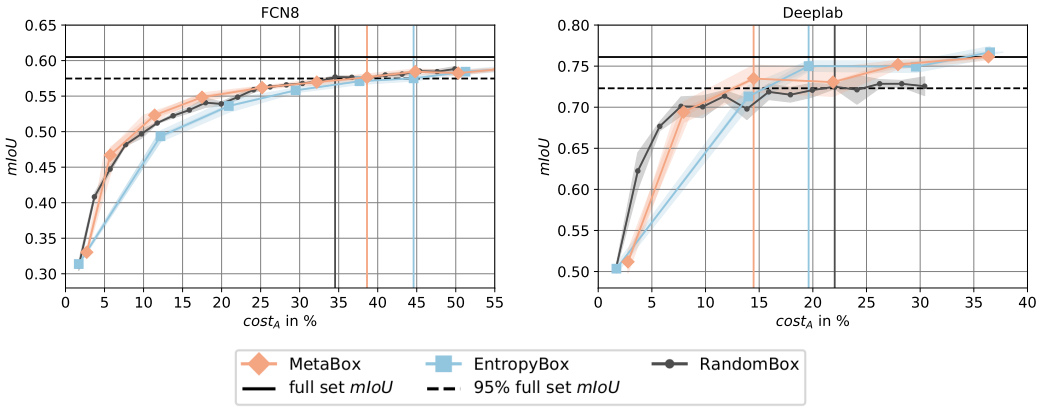


Figure 3.7: Results of the AL experiments for MetaBox, EntropyBox and RandomBox. Costs are given in terms of cost metric $cost_A$. The vertical lines indicate where a corresponding method achieves 95% full set $mIoU$. Each method's curve represents the mean over three runs.

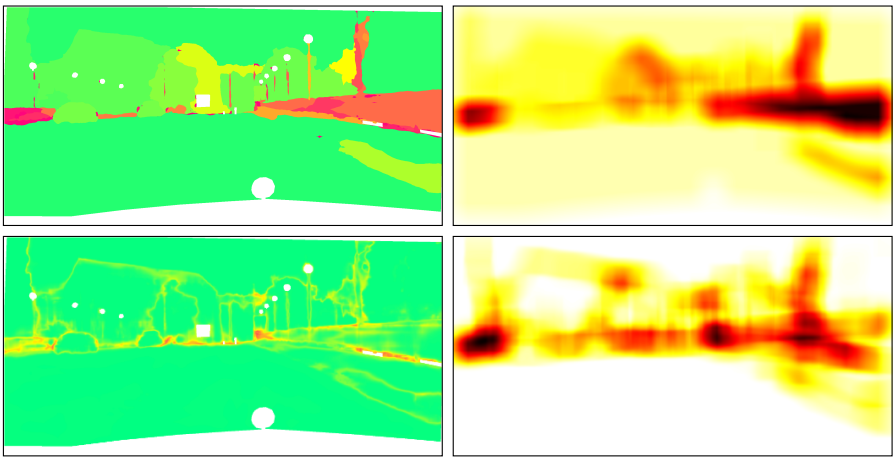


Figure 3.8: Comparison of query strategies based on MetaSeg and entropy. In the MetaSeg priority map (top left), low estimated IoU values are colored red and high ones green. Accordingly, in the entropy priority map (bottom left) low confidence is colored red and high confidence is colored green. The (line-wise) corresponding aggregations are given in the right-hand column. The higher the priority, the darker the color.

Comparison of MetaBox+ and EntropyBox+. Incorporating the estimated number of clicks improves both methods MetaBox and EntropyBox, see figure 3.9. For the FCN8 model, EntropyBox+ still produces more clicks $cost_A = 40.06\%$ compared to RandomBox. On the other hand, MetaBox+ requires the lowest costs with $cost_A = 32.01\%$. For the Deeplab model, EntropyBox+ and MetaBox+ produce almost the same click costs ($cost_A = 10.25\%$ and $cost_A = 10.47\%$, respectively) for achieving 95% full set $mIoU$. By taking the estimated costs into account, the produced costs per AL iteration are lower for both methods. Although, in comparison

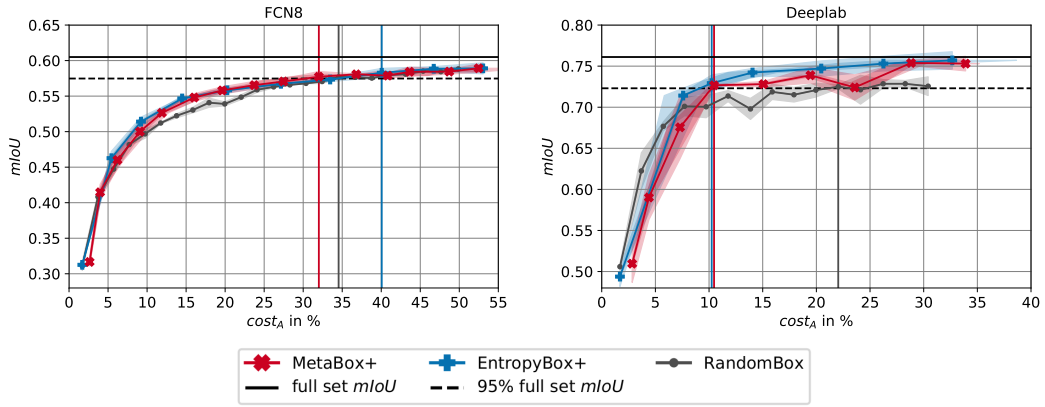


Figure 3.9: Results of the AL experiments for MetaBox+ and EntropyBox+. Costs are given in terms of cost metric $cost_A$. The vertical lines indicate where a corresponding method achieves 95% full set $mIoU$. Each method's curve represents the mean over 3 runs.

with EntropyBox and MetaBox, the methods EntropyBox+ and MetaBox+ require more AL iterations, both methods perform better in terms of required clicks to achieve 95% full set $mIoU$. Due to the incorporation of cost estimation, both EntropyBox+ and MetaBox+ query regions with lower costs and, as indicated by the results, with more efficiency. Segmentation results based on using MetaBox+ are shown in figure 3.10.

In general, one can observe that the Deeplab model gains accuracy quicker than the FCN8 model. This can be attributed to the fact that the FCN8 framework does not incorporate any data augmentation [125], while the Deeplab framework uses state-of-the-art data augmentation and provides a more elaborate network architecture, cf. section 2.3.1.

Robustness and Variance. Considering figure 3.11, where the experiments for each CNN model are shown in one plot, it can be observed that all methods show a clear dependence on the CNN model. Note that EntropyBox+* and MetaBox+* refer to methods that are equipped with the true costs from the Cityscapes data set. Incorporating true instead of estimated clicks is elaborated in a paragraph in the following. In the experiments with the FCN8 it can be observed that results show only insignificant standard deviation over the different trainings, see figure 3.7 and figure 3.9.

For the Deeplab model, the methods show a significant standard deviation over trainings, especially the methods MetaBox, MetaBox+ and RandomBox, see figure 3.12. In the first AL iterations, RandomBox rapidly gains accuracy at low costs.

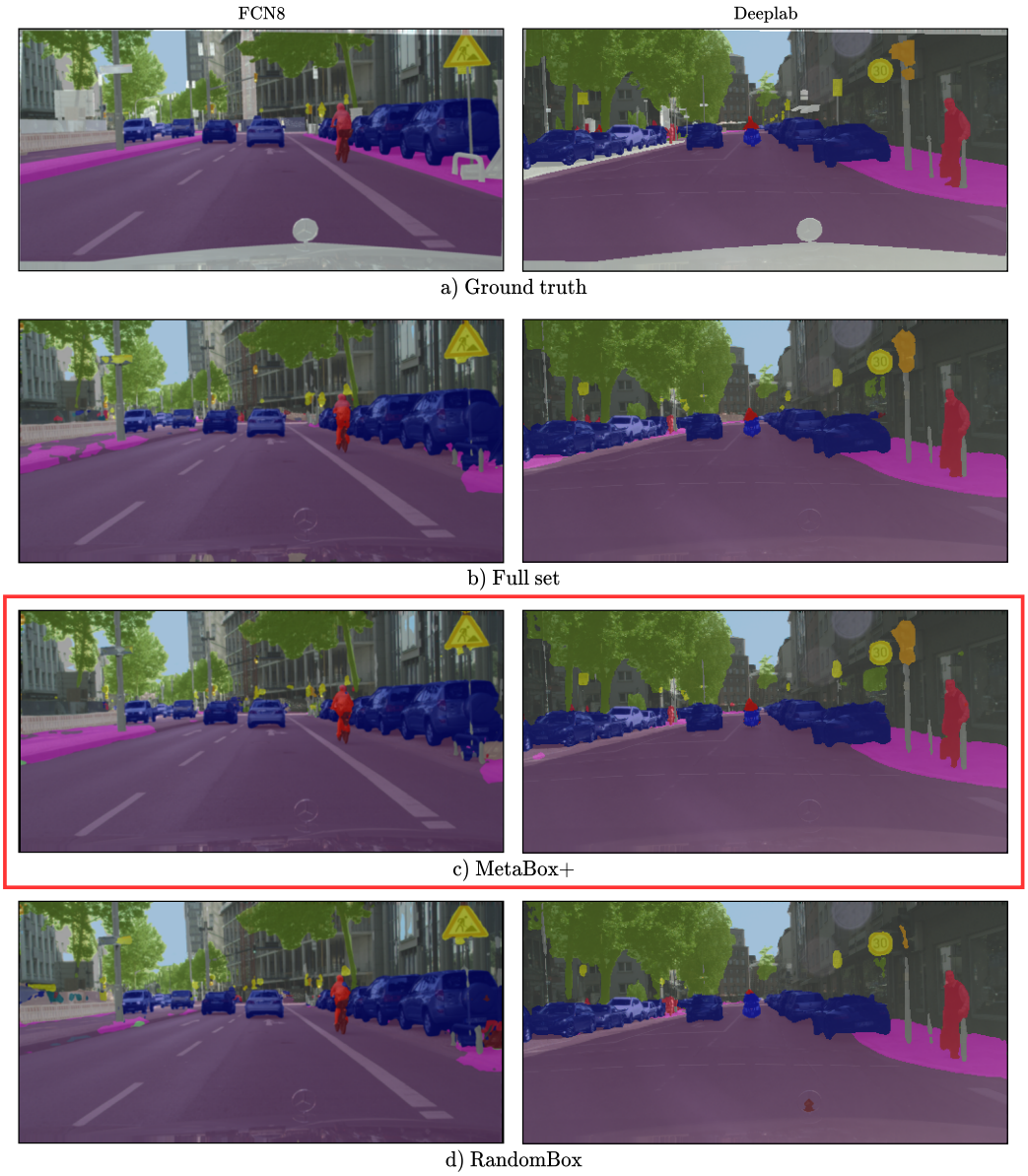


Figure 3.10: Comparison of segmentation results for FCN8 (left) and Deeplab (right). The ground truth is shown in a), results by using the full set for training in b). In c) the results of the segmentation models by using MetaBox+ are shown, and in d) the results for using RandomBox. In c) the segmentation models are used that achieve 95% full set $mIoU$ with an annotation effort of only 32.01% for the FCN8 model and 10.47% for the Deeplab model. The segmentation results for RandomBox are taken from models, producing the same annotation effort as MetaBox+. Annotation effort is stated in terms of the click-based metric $cost_A$.

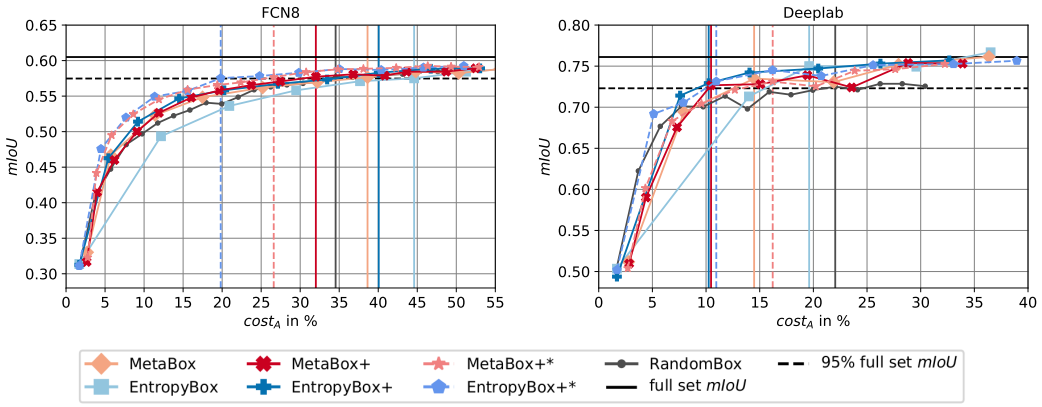


Figure 3.11: Summary of the results of the AL experiments with the FCN8 model (left) and the Deeplab model (right). The costs are given in cost metric $cost_A$. The vertical lines display where the 95% full set $mIoU$ are achieved. MetaSeg-based method's have more initial costs due to the data $\mathcal{M}$ required for training MetaSeg. Each method's curve represents the mean over 3 runs.

However, in the range of 95% full set $mIoU$ it rather fluctuates and only slightly gains accuracy. Beyond the 95% full set $mIoU$ frontier, the methods MetaBox(+) and EntropyBox(+) still improve at a decent pace. MetaBox+ and EntropyBox+ nearly achieve the full set $mIoU$ with approximately the same costs.

Furthermore, when investigating the variation of results w.r.t. two different CNN models, it can be observed that the discrepancy between the FCN8 and the Deeplab model is roughly 8 percentage points (pp) smaller for MetaBox+ than for EntropyBox+. This shows that MetaBox+ tends to be more robust w.r.t. the choice of CNN model.

Comparison of Cost Metrics. In the previous evaluation, only the cost metric $cost_A$ is considered. A comparison of cost metrics for both CNN models is given in table 3.1. Except for RandomBox, the required costs to achieve 95% full set $mIoU$ is up to 3 pp lower when considering $cost_B$ instead of $cost_A$. Considering the proportion of labeled pixels $cost_P$ makes the costs seem significantly lower. Noteworthily, for the FCN8 model EntropyBox requires only costs of $cost_P = 10.08\%$ while RandomBox does require costs of $cost_P = 28.57\%$, which is roughly a factor of 3 higher. Comparing this with $cost_A = 44.60\%$ it becomes clear, that these 10% of the pixels in the data set constitute to almost half of the actually required click work. This comparison highlights the importance of cost measurement (definition of a cost metric) and that the annotation of image regions requires different human annotation effort.

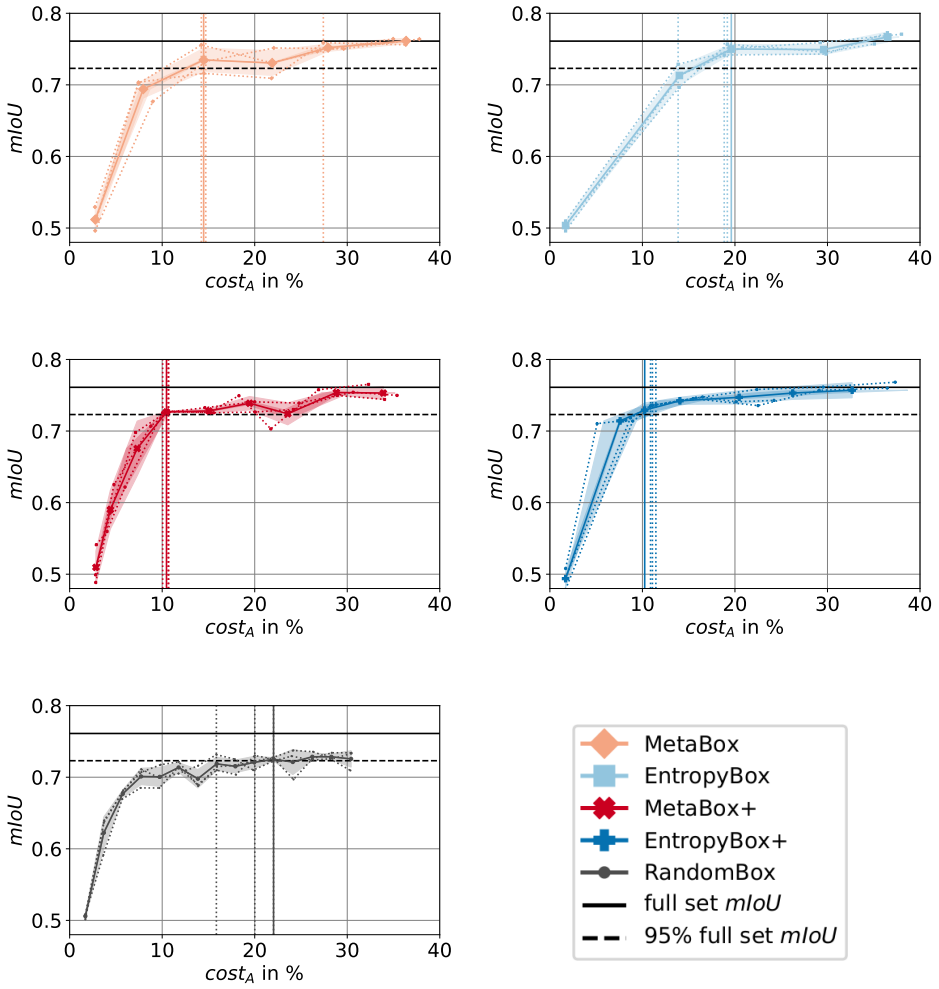


Figure 3.12: Results of single AL experiments (consisting of 3 runs each) for each AL method with the Deeplab model. Costs are given in terms of cost metric $cost_A$. In each plot, the single runs are given as dotted lines, the mean (over costs and $mIoU$) as a solid line. The vertical lines show where each run achieves 95% full set $mIoU$.

CNN model	Cost metric	RandomBox	EntropyBox			MetaBox		
				+	+*		+	+*
FCN8	$cost_A$	34.54	44.60	40.06	*19.82*	38.63	**32.01**	*26.61*
	$cost_B$	35.76	40.74	37.55	*18.87*	35.58	**30.85**	*25.74*
	$cost_P$	28.57	**10.08**	13.45	*10.08*	12.77	17.81	*16.13*
Deeplab	$cost_A$	22.04	19.61	**10.25**	*10.95*	14.48	10.47	*16.21*
	$cost_B$	22.77	17.92	**9.85**	*10.60*	13.56	10.43	*15.85*
	$cost_P$	18.49	5.04	**5.04**	*6.72*	6.05	7.73	*11.09*

Table 3.1: Annotation costs in % (row) produced by each method (column) to achieve 95% full set $mIoU$. Cost metrics $cost_A$ (equation (3.10)) and $cost_B$ (equation (3.11)) are based on annotation clicks while cost metric $cost_P$ (equation (3.12)) indicates the amount of labeled pixels. The costs w.r.t. the cost metric of the best performing methods are highlighted. The strategies EntropyBox+* and MetaBox+* represent a hypothetical 'optimum' by knowing the true costs. Each value was obtained as the mean over 3 runs.

Click Estimation. To evaluate the here presented cost estimation method (section 3.3.2), this is compared to the provided clicks in the Cityscapes data set by considering the latter as a 'perfect' cost estimation. That is, EntropyBox and MetaBox are supplied with the true costs and these methods are termed EntropyBox+* and MetaBox+*. A comparison of the different click estimations and the true clicks is given in table 3.1 as well as in figure 3.11. For the FCN8 model, the experiments show that knowing the true costs in most cases improves the results: EntropyBox+* produces costs of $cost_A = 19.82\%$. This is the half of the costs of EntropyBox+. MetaBox+* produces costs of $cost_A = 26.61$, which is 6 pp fewer costs compared to MetaBox+. For the Deeplab model, using true rather than estimated costs does not lead to better results. However, in terms of cost metric $cost_A$, EntropyBox+* produces 1 pp more costs than EntropyBox+. MetaBox+* produces even 6 pp more costs than MetaBox+. Similarly, such an increase can also be seen w.r.t. the other cost metrics $cost_B$ and $cost_P$, see table 3.1.

3.5 Discussion

In this chapter a novel AL method MetaBox+ was introduced, which is based on the estimated segmentation quality, combined with a practical cost estimation. The methods MetaBox and MetaBox+ were compared to entropy-based methods. Using a combination of entropy and the presented cost estimation shows also remarkable results. The experiments include in-depth studies for two different CNN models, comparisons of cost metrics in terms of annotation click estimates, three different query types (Random, Entropy, MetaSeg) as well as a study on the robustness. The

new method MetaBox+ leads to robust reductions in annotation cost, resulting in requiring 10–30% annotation costs for achieving 95% full set $mIoU$. The presented experiments were conducted using a query function that minimizes the product of two targets, i.e., minimizing the annotation effort and minimizing the estimated segmentation quality for a given query region.

It is still an open question, whether a weighted sum of priorities instead of a product (cf. equation (3.6)) or any other function for joining would lead to additional improvements of the methods. Furthermore, it would be interesting to vary the number of queried boxes per AL iteration, since each method produces different annotation costs for a fixed number of box queries. Moreover, further priority maps can be defined and incorporated.

Another approach to improve the method is to incorporate pseudo-labels, i.e., to label regions of high estimated quality with the predictions of the CNN model. Some experiments have been done on this idea: in each AL iteration, predicted segments with a high predicted $IoU \geq 0.90$ were pseudo-labeled, i.e., the segments were added to the ground truth. Furthermore, only object-like and rare classes have been pseudo-labeled such as pedestrians, traffic signs or bicyclists. Apart from this, once pseudo-labels have been set, those were fixed for the remaining AL iteration or in each AL iteration the pseudo-labels were queried again, i.e., no pseudo-labels were taken from the previous AL iteration. Both variants were tested, but in these initial experiments, no significant accuracy gain was observed. One guess is that segments with a high predicted IoU only reinforce the network in what it has already learned, or that the predicted IoU values were not as high as the true ones. Nevertheless, this semi-supervised approach is interesting but needs more investigation.

It was shown that MetaSeg and the prediction quality estimation is a valuable ingredient in region-based active learning. This motivates to do further studies as motivated above. However, semantic segmentation is costly in terms of computational run time, especially in the study of active learning. Besides the here presented use of MetaSeg, it was used in further applications [115, 13, 83]. Noteworthily, all the works are related to image data. The question is, if these concepts can be applied to other sensor data such as lidar? This requires a method similar to MetaSeg for lidar data, which is presented in the next chapter and is named *LidarMetaSeg*.

4 False Positive Detection and Prediction Quality Estimation for Point Cloud Segmentation

Semantic segmentation has become as standard method for scene understanding. Much has happened in the field of lidar point cloud segmentation in recent years, and the accuracy of new network architectures and models is continuously improving. However, the literature lacks methods for uncertainty quantification and reliability measures in this field. In applications of ADAS and AD, safety and reliable models are just as important as their accuracy. In this chapter, a method for prediction quality estimation and false positive detection of lidar point cloud semantic segmentation is presented. The method is an extension and adaptation of the (image-based) MetaSeg to lidar point cloud data.

4.1 Introduction

In the field of AD, scene understanding is essential. One possible solution for the semantic interpretation of scenes captured by multiple sensor modalities is lidar point cloud segmentation [91, 30, 151, 146] (in the following lidar segmentation for brevity) where each point of the point cloud is assigned to a class of a given set. A segment is an area of points of the same class. Compared to camera images, a lidar point cloud is relatively sparse, but provides accurate range information, cf. figure 2.1. Furthermore, since the lidar sensor in general is rotating, 360 degrees of the environment are considered. In recent years, the performance of lidar segmentation networks has increased enormously [108, 91, 30, 151, 146], but there are only few works on uncertainty quantification [30]. To tackle this problem, in this chapter a new post-processing tool, called *LidarMetaSeg* is introduced, which estimates the segment-wise (i.e., per connected component of the predicted segmentation) prediction quality in terms of segment-wise intersection over union [63] (IoU) of the lidar segmentation model, see also figure 4.1. This provides not only uncertainty

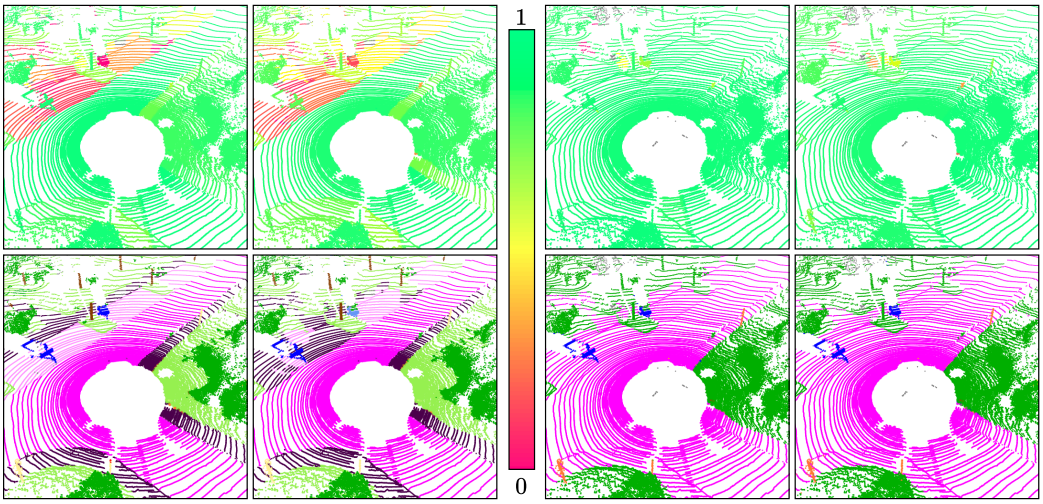


Figure 4.1: Visualizations of LidarMetaSeg on classes (a) and categories (b). Each visualization contains the ground truth (bottom left), the lidar segmentation (bottom right), the lidar segmentation quality (top left) as the IoU of prediction and ground truth and its estimation obtained by LidarMetaSeg (top right). The higher the IoU, the better the prediction quality.

quantification per predicted segment, but also an online assessment of prediction quality.

The method LidarMetaSeg works as follow: first, the point cloud and the corresponding softmax probabilities of the network are projected to a spherical 2D image representation, which are then used to compute different types of dispersion measures resulting in different dispersion heatmaps. To estimate uncertainty on segment level, the dispersion measures are aggregated w.r.t. each predicted segment. For each predicted segment, the IoU with the ground truth is computed, which is called *segment-wise* IoU. Hence, the aggregated dispersion measures with additional information from the point cloud input are used to create a set of handcrafted features. The latter are used in post-processing manner as input for training a *meta classification model* to detect false positive segments, i.e., classify between IoU equal or greater than 0 and a *meta regression model* to estimate segment-wise IoU. Thus, this yields not only point-wise uncertainty quantification, given by the dispersion heatmaps, but also a false positive detection as well as a segmentation quality estimation on segment level. The method is evaluated in an in-depth study in which the performance of meta regression and meta classification is provided class-wise and category-wise. Categories are groups or sets of pre-defined classes. For example, the classes 'person' and 'bicyclist' belong to the category 'human'. The difference between predicted and ground truth segment classes and the corresponding segment categories are shown figure 4.1. Class-wise evaluation is motivated by the fact that in

real-world applications and from a safety point of view, reliability is more important for some predicted classes than for others. Also, in some scenarios, it may be less critical to incorrectly predict a class if it belongs to the same category. For example, the classes 'person' and 'cyclist' belong to the same category 'human'. A reasonable assumption is that prediction quality estimation is easier for categories instead of classes.

The contributions are summarized as follows:

- introducing a new method for segment-wise uncertainty quantification in the lidar segmentation domain to detect false positive segments and to have a prediction quality estimation per predicted segment, by adapting and extending MetaSeg to point cloud data,
- numerical experiments on the SemanticKITTI data set [3] and nuScenes data set [10] with three models, namely RangeNet++ [91], SalsaNext [30] and Cylinder3D [151],
- presenting an in-depth evaluation of the metrics as well as studies of the meta regression and meta classification performance class-wise and category-wise.

First, in this section an introduction into the topic was given. A review of related work is given in section 4.2. Afterwards, in section 4.3, the new method for false positive detection and prediction quality estimation is presented. Experiments and results to evaluate the new method are given in section 4.4. This chapter concludes with a discussion in section 4.5.

4.2 Related Work

As already explained in section 2.3, state-of-the-art lidar segmentation models are based on CNNs and can be grouped into two main approaches: projection-based (2D) and non-projection-based (3D) networks, cf. [53]. Projection-based networks like [145, 91, 30] use a spherical (2D) image representation of the point cloud. The predicted semantic classes on the image are thereafter reinserted along the spherical rays into the 3D point cloud. This may contain some post-processing steps, like the k-nearest neighbor (kNN) approach, see [91]. Due to the representation of point clouds as projected images, the networks employed for lidar segmentation have architectures that often resemble image segmentation architectures. The non-projection-based networks, e.g., [109, 134, 151], process the point cloud directly in 3D space with or without different 3D representation approaches. For example,

in [134], the network operates on the 3D point cloud without introducing an additional representation while in [151] the authors perform a 3D cylinder partition. A combination of a 2D and 3D representation of the point cloud is used in [146]. All current architectures, using a 2D or 3D representation or a combination of both, provide the segmentation of the point cloud. Therefore, it is also possible to output the probabilities, which is the only requirement for LidarMetaSeg.

Concerning uncertainty quantification in deep learning, Bayesian approaches like Monte Carlo (MC) dropout [45] are commonly used, e.g., in image-based object detection [101], image semantic segmentation [67] and also in lidar object detection [41]. In object detection and instance segmentation, so-called scores containing (un)certainty information are used, while this is not the case for semantic segmentation. The network SalsaNext [30] is for lidar segmentation and makes use of MC dropout to output the model (epistemic) and observation (aleatoric) uncertainty. For epistemic uncertainty, a sample is predicted $n_e \in \mathbb{N}$ (e.g., $n_e = 30$) times and the average of the variances is taken for every prediction of a point. Aleatoric uncertainty is computed based on propagated noise though the network via assumed density filtering, see [47].

The idea of meta classification and regression to detect false positives and to estimate the segment-wise prediction quality was first introduced in the field of semantic segmentation of images [113], called MetaSeg. MetaSeg is described in section 2.4. The work presented in [116] goes in a similar direction, but for brain tumor segmentation. MetaSeg was further extended in other directions, i.e., for controlled false negative reduction [13], for time dynamic uncertainty estimates for video data [83], for taking resolution-dependent uncertainties into account [115] and as part of an active learning method, see chapter 3. Inspired by the possibility of representing the point could as a 2D image, the method LidarMetaSeg is an extension and further development of the original work. Therefore, MetaSeg [113] is the most related work to the here presented approach LidarMetaSeg, which is together with SalsaNext [30] the only work in the direction of uncertainty quantification in lidar segmentation.

With MC dropout, SalsaNext follows a Bayesian approach to quantifying the model and the observation uncertainty. The uncertainty output is point-based and not segment-based, as in the here presented approach. Also for MC dropout, the model has to infer one point cloud multiple times. LidarMetaSeg requires only a single network inference and estimates uncertainties by means of the network's class probabilities. In a 2D representation, these pixel-wise uncertainty estimates can be viewed as uncertainty heatmaps. From those heatmaps, aggregated uncertainties are computed for each predicted segment, therefore clearly going beyond the stage

of pixel-wise uncertainty estimation. In contrast to MetaSeg for image segmentation, LidarMetaSeg not only use the network's output but also utilize information from the point cloud input, such as the intensity and range features provided for each point of the point cloud.

4.3 A new Method for False Positive Detection and Prediction Quality Estimation

LidarMetaSeg is a post-processing method for lidar semantic segmentation for false positive detection and to estimate the segment-wise prediction quality. It consists of a meta classification and a meta regression model that for each predicted segment classifies whether it has an IoU equal to or greater than 0 with the ground truth and predicts the segment-wise IoU with the ground truth, respectively. The method works as follows: in a pre-processing step, each sample, i.e., the point cloud data, the corresponding network probabilities and the labels are projected to a spherical 2D image representation. In a next step and based on the projected data, dispersion measures and other features are computed like it is done for image data in [113, 115, 13], see also section 2.4. Afterwards, the segments of a given semantic class are identified and the pixel-wise values from the previous step on a segment level are aggregated. In addition, the IoU of each predicted segment with the ground truth of the same class is computed. This results in a structured data set, which consist of the coefficients of the aggregated dispersion measures as well as additional features and of the target variable – the $IoU \in [0, 1]$ for the task of meta regression or the binary variable $IoU = 0, > 0$ ($IoU = 0$ as indicator for a false positive) for the task of meta classification – for each segment. A classification and a regression model is fitted to this data set. In the end, the results are re-projected from the image representation to the point cloud.

4.3.1 Pre-processing

A sample of input data for LidarMetaSeg is assumed to be given on point cloud level and contains the following:

- *point cloud* $\mathbf{p} \in \mathbb{R}^{n\times 5}$, $p_j = (x_j, y_j, z_j, i_j, r_j)$ with $x_j, y_j, z_j \in \mathbb{R}$ Cartesian coordinates, intensity $i_j \in \mathbb{R}_+$ and range $r_j \in \mathbb{R}_+$ for $j = 1, \ldots, n$ with n the number of points in the lidar point cloud,

- *ground truth* or *labels* $\mathbf{y}_p \in \mathcal{C}^n$ with $\mathcal{C} = \{1, \dots, q\}$ the set of q given classes,
- *probabilities* $\hat{\mathbf{y}}_p^{prob} = f(\mathbf{p}) \in [0,1]^{n \times q}$ of a lidar segmentation network f, given as softmax probabilities,
- *prediction* $\hat{\mathbf{y}}_p = \underset{c \in \mathcal{C}}{\operatorname{argmax}}\ \hat{\mathbf{y}}_p^{prob},\ \hat{\mathbf{y}}_p \in \mathcal{C}^n$.

In order to have a spherical 2D image representation of the point cloud, the transformations from equation (2.50) and equation (2.51) are applied, see section 2.3.1. To get an image representation where each point correspond to one pixel and vice versa, the number of channels, the angular resolution and the horizontal FOV of the lidar sensor are needed. To this end, the height h is defined as the number of channels and the width w as the quotient of the horizontal FOV ψ and the angular resolution ϕ, i.e.,

$$w = \left\lceil \frac{\psi}{\phi} \right\rceil, \tag{4.1}$$

see section 2.3.1 and figure 2.2. Thus, the image representation – using the explicit sensor information – has as many entries or pixels as the point cloud can have maximum number of points. Unfortunately, there are still some technical reasons, due to which it can happen that multiple points are projected to the same pixel, e.g., ego-motion compensation or overlapping channel angles. More details concerning such projection errors can be found in [136]. With the projection proposed above this happens rarely enough, so that this event is negligible.

The projected 2D representation is defined as:

- image representation of point cloud $\mathbf{p}$ $\mathbf{F} = (\mathbf{F}^x, \mathbf{F}^y, \mathbf{F}^z, \mathbf{F}^i, \mathbf{F}^r),\ \mathbf{F} \in \mathbb{R}^{w \times h \times 5}$,
- ground truth or labels $\mathbf{y} \in \mathcal{C}^{w \times h}$,
- probabilities $\hat{\mathbf{y}}^{prob} \in [0,1]^{w \times h \times q}$,
- prediction $\hat{\mathbf{y}} \in \mathcal{C}^{w \times h}$.

Note, apart from the image representation $\mathbf{F}$, this is the same notation as defined in section 2.4.

The proposed image projection yields a sparse image representation. However, the post-processing approach LidarMetaSeg is based on connected components of the segmentation. In order to identify connected components (segments) of pixels in the 2D image resulting from the projection, these gaps are filled by setting any empty entry $z := (u_j, v_j)$ (entries without a corresponding point in the point cloud) to a value of one of its nearest neighbors that received a value from the projection. An

example of such a filled image representation is shown in figure 4.2 (left). In the following, only filled image representations are considered. The information which pixel received its value via projection and which one via fill-in is stored in a binary mask of width w and height h denoted by $\boldsymbol{\delta}$ where 1 represents a projected point and 0 a filled entry, i.e.,

$$\delta_z = \begin{cases} 1, & z \text{ corresponds to a projected point,} \\ 0, & \text{else.} \end{cases} \tag{4.2}$$

For simplicity, the filled image representations $\mathbf{F}$ (that are input quantities for the segmentation network) are referred as feature measures / feature measure heatmaps.

4.3.2 Dispersion Measures and Segment-wise Aggregation

The pixel-wise dispersion measures, i.e., for a pixel or entry on image representation $z = (u_j, v_j)$ are *entropy* E_z (equation (2.74)), *probability difference* D_z (equation (2.75)) and *variation ratio* V_z (equation (2.76)).

Additionally, the feature measures *coordinates*, *intensity* and *range* at position z are given by the image representation

$$F_z^\sharp, \ \sharp \in \{x, y, z, i, r\}. \tag{4.3}$$

In summary, this yields to the set of dispersion and feature measure heatmaps

$$\mathcal{M} = \{\mathbf{E}, \mathbf{D}, \mathbf{V}, \mathbf{F}^x, \mathbf{F}^y, \mathbf{F}^z, \mathbf{F}^i, \mathbf{F}^r\}. \tag{4.4}$$

The segment-wise aggregation follows section 2.4. In order to take the underlying point cloud data into account, the *point cloud segment size* $SP = |k|_{\boldsymbol{\delta}}$ for a predicted segment k is added as metric. Furthermore, the IoU and adjusted IoU_{adj} is restricted to the original point cloud data and thus restricted by $\boldsymbol{\delta}$:

- IoU: let $\mathcal{K}_{\mathbf{x}}|_k$ be the set of all $k' \in \mathcal{K}_{\mathbf{x}}$ that have non-trivial intersection with k and whose class label equals the predicted class for k, then

$$IoU(k) = \frac{|k \cap K'|_{\boldsymbol{\delta}}}{|k \cup K'|_{\boldsymbol{\delta}}}, \qquad K' = \bigcup_{k' \in \mathcal{K}_{\mathbf{x}}|_k} k', \tag{4.5}$$

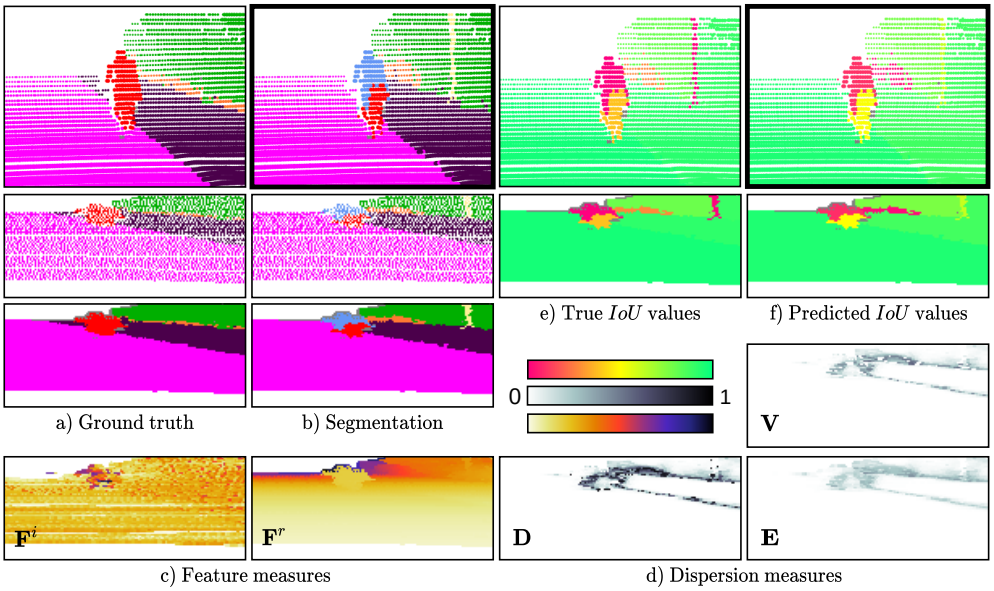


Figure 4.2: Visual examples from LidarMetaSeg. Ground truth (a) and segmentation (b) are shown from the pre-possessing part: point cloud (top), the corresponding sparse (center) and filled (bottom) image representations. Feature and dispersion measure heatmaps are visualized in c) and d), respectively. The true and predicted segmentation quality in terms of segment-wise IoU values are depicted in e) and f), respectively - in the bottom part for the image representation and in the top part the corresponding visualizations after the re-projection to the point cloud. The prediction of the point cloud and the corresponding prediction quality estimation is highlighted. Note, for a better visualization, the colormap for predicted classes differs to the colormaps of the remaining visualizations in this chapter.

- the adjusted IoU_{adj} does not count pixels in the ground truth segment that are not contained in the predicted segments, but in other predicted segments of the same class: let $Q = \{q \in \hat{\mathcal{K}}_{\mathbf{x}} : q \cap K' \neq \emptyset\}$, then

$$IoU_{\text{adj}}(k) = \frac{|k \cap K'|_{\boldsymbol{\delta}}}{|k \cup (K' \setminus Q)|_{\boldsymbol{\delta}}}. \tag{4.6}$$

In summary, $86 + 2q$ metrics are defined: the (relative) mean and variance metrics $\tau M, \tau M_{in}, \tau M_{bd}, \tau \bar{M}, \tau \bar{M}_{in}\ \forall \tau \in \{\mu, \upsilon\},\ \forall M \in \mathcal{M}$, the (relative) size metrics $S, S_{in}, S_{bd}, \bar{S}, \bar{S}_{in}, SP$ as well as $N_c, P_c\ \forall c \in \mathcal{C}$. An example of the pixel-wise dispersion measures as well as the segment-wise IoU_{adj} values and its prediction is shown in figure 4.2.

4.4 Experiments and Results

For numerical experiments, two data sets have been used: SemanticKITTI [3] and nuScenes [10]. For meta classification and regression, XGBoost is employed. Due to the reason mentioned in section 2.4, the target variable for the meta regression (and classification) is the adjusted IoU_{adj}.

First, the settings of the experiments for both data sets are described. Next, the results for the false positive detection and the segment-wise prediction quality estimation when using all metrics presented in the previous section are evaluated. Afterwards, an analysis of the metrics is conducted. Thereafter, a class-wise and category-wise evaluation for the meta regression is presented. The section ends with an analysis of the meta classification model.

4.4.1 SemanticKITTI

The SemanticKITTI data set [3] contains street scenes from and around Karlsruhe, Germany. It provides 11 sequences with about $23\,\mathrm{K}$ samples for training and validation, consisting of 19 classes from 6 categories as follows:

- **ground:** road, parking, sidewalk, other-ground,
- **structure:** building,
- **vehicle:** car, bicycle, motorcycle, truck, other-vehicle,
- **nature:** vegetation, trunk, terrain,
- **human:** person, bicyclist, motorcyclist,
- **object:** fence, pole, traffic-sign.

The data is recorded with a Velodyne HDL-64E lidar sensor, which has 64 channels and a (horizontal) angular resolution $\phi = 0.08°$. Furthermore, the data is recorded and annotated with 10 Hz and each point cloud contains about $120\,\mathrm{K}$ points. The authors of the data set recommend using all sequences to train the lidar segmentation model, except sequence 08, which should be used for validation.

For the experiments three pre-trained lidar segmentation models have been used, two projection-based models, i.e., RangeNet++ [91] and SalsaNext [30], and one non-projection-based model, i.e., Cylinder3D [151], which followed the recommended data split. All three models are reviewed in section 2.3.1. For RangeNet++ and

SalsaNext, the softmax probabilities are given for the 2D image representation prediction. Since it assumed that softmax probabilities are given on point cloud level, this representation is considered as the starting point and the softmax probabilities are re-projected to the point cloud.

After the re-projection from the 2D image representation prediction to the point cloud, both models have an additional kNN post-processing step to clean the point cloud from undesired discretization and inference artifacts [91], which may results in changing the semantic class of a few points. To take this post-processing step into account, the values of the cleaned points in the corresponding softmax probabilities of the point cloud are set to 1 and all other values to 0. Therefore, the softmax condition (the sum of all probability values of a point is equal to 1 and all values are between 0 and 1, cf. equation (2.5)) is met and the adjusted prediction is equal to the argmax of the probabilities. Other approaches are not expected to significantly change the results since the dispersion measures are aggregated and the number of modified points is small. For Cylinder3D the probabilities and predictions are directly provided on point cloud level. Pre-requird steps as for RangeNet++ and SalsaNext are not necessary.

Following the method from section 4.3, the image representation of the point cloud data is of size $(w, h) = (4{,}500, 64)$, cf. equation (4.1). Most deep learning models tend to overfit. Therefore, only samples for LidarMetaSeg are used, which are not part of the training data of the segmentation network, as overfitted models affect the dispersion measures. Thus, only sequence 08 is used for the experiments. Computing the connected components and metrics yields approx. $3.4\,\mathrm{M}$ segments for each network. Most of the segments are very small. Therefore, a similar segment exclusion rule as in MetaSeg [113] is done, where segments with empty interior, $S_{in} = 0$, are excluded. Here, segments consisting of less than 10 lidar points, i.e., $SP < 10$, are excluded, also shown in gray color in figure 4.2. Hence, the number of segments to approx. $0.45\,\mathrm{M}$ is reduced but with retaining 99% of the data measured in terms of the number of points. The dependence of the results under variation of the exclusion size SP was tested. The results were very similar to the results which will be presented in the following.

The performance of LidarMetaSeg on predicted categories is also evaluated. To this end, for every predicted point, the probabilities corresponding to a given category are summed up. Thus, the probabilities $\hat{\mathbf{y}}_p^{prob}$ are in $[0, 1]^{n \times q'}, q' < q$ with q' number of categories. The ground truth is treated analogously. Excluding the predicted segments with $SP < 10$ reduces the number of segments from approx. $2.9\,\mathrm{M}$ to

		RangeNet++		SalsaNext		Cylinder3D	
		training	validation	training	validation	training	validation
Classification $IoU_{\text{adj}} = 0, > 0$							
ACC	LMS	92.26%(±0.25%)	**85.50**%(±3.12%)	92.43%(±0.21%)	**86.26**%(±2.75%)	92.97%(±0.21%)	**88.37**%(±2.89%)
	LMS w/o features	90.64%(±0.30%)	85.10%(±3.21%)	90.77%(±0.23%)	86.02%(±2.86%)	91.68%(±0.27%)	88.22%(±2.93%)
	Entropy	79.37%(±0.37%)	79.21%(±3.17%)	80.14%(±0.33%)	80.06%(±2.75%)	85.35%(±0.30%)	85.24%(±2.78%)
	LMS ∪ MCDO			92.48%(±0.20%)	86.26%(±2.78%)		
	LMS Categories	*95.88%(±0.13%)*	*89.84%(±2.42%)*	*95.42%(±1.33%)*	*89.52%(±2.05%)*	*96.04%(±0.13%)*	91.76%(±1.94%)
	naive baseline	78.42% (*83.11%*)		79.36% (*82.34%*)		84.53% (*88.46%*)	
AUROC	LMS	97.19%(±0.12%)	**90.58**%(±1.89%)	97.22%(±0.10%)	91.16%(±1.75%)	96.86%(±0.10%)	**90.78**%(±2.36%)
	LMS w/o features	95.97%(±0.13%)	89.85%(±1.96%)	95.95%(±0.10%)	90.81%(±1.80%)	95.52%(±0.18%)	90.57%(±2.46%)
	Entropy	81.45%(±0.22%)	79.21%(±3.17%)	80.74%(±0.31%)	80.84%(±2.35%)	83.52%(±0.34%)	83.80%(±2.91%)
	LMS ∪ MCDO			97.25%(±0.09%)	**91.17**%(±1.75%)		
	LMS Categories	*98.97%(±0.05%)*	*93.85%(±1.68%)*	*98.79%(±0.01%)*	*94.07%(±1.51%)*	*98.66%(±0.01%)*	93.85%(±1.95%)
AUPRC	LMS	91.29%(±0.23%)	**73.47**%(±2.52%)	90.98%(±0.25%)	74.35%(±2.69%)	86.09%(±0.31%)	**64.54**%(±5.38%)
	LMS w/o features	87.73%(±0.20%)	72.00%(±2.67%)	87.34%(±0.27%)	73.70%(±2.91%)	81.11%(±0.51%)	64.29%(±5.38%)
	Entropy	49.16%(±0.59%)	48.48%(±5.75%)	48.16%(±0.61%)	48.24%(±5.99%)	47.25%(±0.48%)	47.91%(±4.53%)
	LMS ∪ MCDO			91.04%(±0.04%)	**74.39**%(±2.62%)		
	LMS Categories	*95.58%(±1.69%)*	*78.28%(±3.66%)*	*95.02%(±0.19%)*	*79.01%(±4.34%)*	*91.10%(±0.34%)*	67.31%(±6.01%)
Regression IoU_{adj}							
R^2	LMS	79.34%(±0.19%)	**66.69**%(±2.06%)	78.19%(±0.16%)	66.08%(±2.23%)	74.04%(±0.31%)	**61.57**%(±2.94%)
	LMS w/o features	75.78%(±0.18%)	64.91%(±1.87%)	74.91%(±0.18%)	65.31%(±2.21%)	70.78%(±0.30%)	61.51%(±2.96%)
	Entropy	51.45%(±0.37%)	50.88%(±2.74%)	48.54%(±0.57%)	48.21%(±4.25%)	50.90%(±0.45%)	50.30%(±3.27%)
	LMS ∪ MCDO			78.26%(±0.14%)	**66.13**%(±2.27%)		
	LMS Categories	*83.33%(±0.17%)*	*67.58%(±2.66%)*	*81.99%(±0.19%)*	*68.65%(±2.71%)*	*77.29%(±0.25%)*	62.89%(±2.64%)

Table 4.1: Results for meta classification and regression, averaged over 10 runs for SemanticKITTI. The numbers in the brackets denote standard deviations of the computed mean values. *LMS Categories* as well as the *values* in brackets of naive baseline are the results using the categories. The best results in terms of ACC, $AUROC$, $AUPRC$ and R^2 on the validation data are highlighted, *LMS Categories* are excluded in this consideration.

approx. 0.29 M, also retaining about 99% of the data points. In the following, and unless highlighted, the evaluation is presented w.r.t. predicted classes.

For training and validation of LidarMetaSeg the connected components and metrics from the predictions of sequence 08 are split into 10 disjoint sub-sequences. These sub-sequences are used for a 10-fold cross validation. A cross validation over all samples would yield highly correlated training and validation splits as all sequences are recorded with 10 Hz. The results for the meta classification and regression are given in table 4.1. For evaluation, the metrics from section 2.4.2 are used. For meta classification and regression, XGBoost was used with 750 estimators, maximum depth of 6 and learning rate 0.05. For all three models, a validation accuracy between 85.50% and 88.37% is achieved, see row 'ACC LMS' (short for LidarMetaSeg). The accuracy of random guessing ('ACC naive baseline') is between 78.42% and 84.53% which directly amounts to percentage of segments with an $IoU_{\text{adj}} > 0$.

For predicted categories, see '*LMS Categories*' a validation accuracy between 89.52% and 91.76% is achieved while the naive baseline is between 82.34% and 88.46%. This observation already hints at a heterogeneity in class-wise performance which is studied later in this section.

Using the metrics of the previous section (LMS) for the meta classification yields $AUROC$ values above 90% and $AUPRC$ up to 74.35%. For predicted categories, the

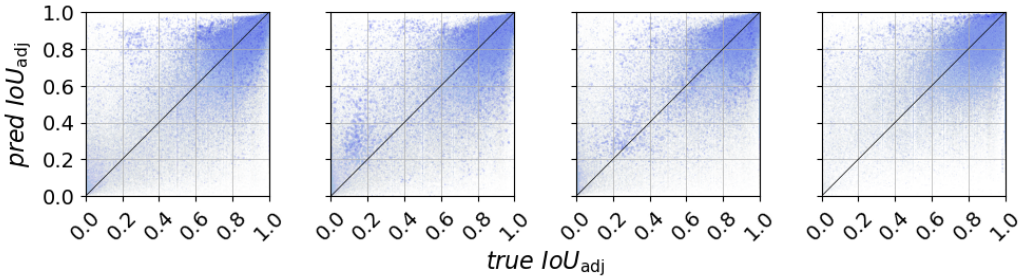


Figure 4.3: True IoU_{adj} vs predicted IoU_{adj} for RangeNet++, SalsaNext, Cylinder3D on SemanticKITTI as well as Cylinder3D on nuScenes, from left to right.

$AUROC$ and $AUPRC$ values are up to 94.07% and 79.01%, respectively. For the meta regression R^2 values between $61.57 - 66.69\%$ are achieved and $62.89 - 68.65\%$ for predicted categories. Evaluating the results w.r.t. categories instead of classes yields always higher values – up to 3 pp – in terms of ACC, $AUROC$ and $AUPRC$ for meta classification and R^2 for meta classification, see columns '*LMS Categories*'. Figure 4.3 depicts the quality of predicting the IoU_{adj}. A visualization of estimating the IoU_{adj} is shown in figure 4.1 and in the video[1].

4.4.2 NuScenes

The nuScenes data set [10] contains street scenes from two cities, Boston (US) and Singapore. It provides 700 sequences for training and 150 sequences for validation. Each sequence contains about 40 samples which amounts to a total of $34\,\text{K}$ key frames. The dataset has 16 classes from 5 categories as follows:

- **movable-object:** barrier, traffic-cone,
- **vehicle:** bicycle, bus, car, construction-vehicle, motorcycle, trailer, truck,
- **human:** pedestrian,
- **flat:** drivable-surface, other-flat, sidewalk, terrain,
- **static:** manmade, vegetation.

The data is recorded and annotated with 2 Hz. The lidar sensor has 32 channels and an angular resolution of $0.33°$. Every point cloud contains roughly $35\,\text{K}$ points. For the experiments the pre-trained Cylinder3D was used with the recommended data split. RangeNet++ and SalsaNext were not tested since the corresponding pre-trained models are not available.

[1] `https://youtu.be/907jJSRgHUk`

		Cylinder3D	
		training	validation
Classification $IoU_{\text{adj}} = 0, > 0$			
ACC	LMS	92.96%(±0.08%)	**91.00**%(±0.60%)
	LMS w/o features	92.31%(±0.13%)	90.62%(±1.16%)
	Entropy	89.91%(±0.14%)	89.86%(±1.17%)
	LMS Categories	*97.88%(±0.01%)*	*95.48%(±0.09%)*
	naive baseline	89.85% (*95.08%*)	
AUROC	LMS	94.81%(±0.10%)	**90.05**%(±1.11%)
	LMS w/o features	93.47%(±0.10%)	89.09%(±1.29%)
	Entropy	82.83%(±0.18%)	82.81%(±1.44%)
	LMS Categories	*99.16%(±0.01%)*	*94.43%(±1.05%)*
AUPRC	LMS	70.82%(±0.54%)	**50.25**%(±2.97%)
	LMS w/o features	65.49%(±0.42%)	48.26%(±3.31%)
	Entropy	35.16%(±0.33%)	35.29%(±2.88%)
	LMS Categories	*89.15%(±5.97%)*	*48.16%(±6.54%)*
Regression IoU_{adj}			
R^2	LMS	57.93%(±0.37%)	**48.84**%(±1.81%)
	LMS w/o features	54.73%(±0.19%)	47.62%(±1.64%)
	Entropy	39.51%(±0.28%)	39.33%(±2.57%)
	LMS Categories	*68.73%(±0.42%)*	*50.19%(±3.91%)*

Table 4.2: Results for meta classification and regression, averaged over 10 runs for nuScenes. The numbers in the brackets denote standard deviations of the computed mean values. *LMS Categories* as well as the *values* in brackets of naive baseline are the results using the categories. The best results in terms of ACC, $AUROC$, $AUPRC$ and R^2 on the validation data are highlighted, *LMS Categories* are excluded in this consideration.

The image projection is of size $(w, h) = (1{,}090, 32)$. Computing the connected components for all samples of the 150 validation sequences yields approx. $1.5\,\text{M}$ segments. Excluding all small segments containing less than 10 points, i.e., $SP < 10$, reduces that number to $0.34\,\text{M}$. Still, this retains 99% of the data in terms of points. For predicted categories, the same exclusion reduces the number of segments from approx. $0.75\,\text{M}$ to approx. $0.11\,\text{M}$, also retaining about 99% of the data points. A 10-fold cross validation was performed where always 90% of the 150 sequences were taken i.e., 135 sequences, for training and the remaining 10%, i.e., 15 sequences for validation of the meta models. The results are presented in table 4.2. For meta classification and regression a XGBoost was used with 200 estimators, maximum depth of 6 and learning rate 0.1. 89.85% of all segments have an $IoU_{\text{adj}} > 0$. With the meta classification, an accuracy of 91.00%, $AUROC$ of 90.00% and $AUPRC$ of 50.25% is achieved, see 'LMS' rows. For the meta regression, it is $R^2 = 49.19\%$ on validation data. The quality of predicting the IoU_{adj} is shown in figure 4.3 (right). Overall, the evaluation metrics are 2 – 4 pp higher for predicted categories compared to predicted classes, see columns '*LMS Categories*'.

4.4.3 Metric Selection

So far, results based on all metrics from section 4.3 have been presented, indicated by LMS in table 4.1 and table 4.2. To analyze the effects of the metrics on performance, the experiments were repeated for several sets of metrics.

Feature Measures. First tested were the performance of the meta classification and regression model without the feature measures, i.e., the metrics based on the point cloud input features, see row 'LMS w/o features'. The performance in terms of ACC, $AUROC$, $AUPRC$ and R^2 for all experiments are up to 2 pp lower compared to when incorporating feature measures.

Entropy. Since the entropy is commonly used in uncertainty quantification, all experiments were tested with only using the mean entropy μE, see 'Entropy' rows. The performance for the meta classification is up to 12 pp lower compared to LMS, for the meta regression R^2 decreases by up to 18 pp. These results hint that using multiple uncertainty measures significantly improve the performance in terms of ACC and R^2.

Bayesian Uncertainties. The projection-based model SalsaNext model follows a Bayesian approach as already mentioned in section 4.1: the lidar model provides a model (epistemic) and observation (aleatoric) uncertainty output for the point cloud's 2D image representation prediction, estimated by MC dropout (MCDO). Following the procedure from [30] yields those uncertainties. This ends up in epistemic epi_z and aleatoric ale_z uncertainty values for each pixel position z. The same aggregated measures are computed as for the measures $M \in \mathcal{M}$. Adding these new metrics to the previous metrics LMS is referred to as 'LMS $\cup$ MCDO'. The additional uncertainties do not improve the meta classification and regression performance significantly, see table 4.1. SalsaNext were not tested on nuScenes since the pre-trained model is not available. For comparability of results, only publicly available pre-trained models were used.

Greedy Heuristic. Inspired by forward-stepwise selection for linear regression, different subsets of metrics by performing a greedy heuristic are analyzed: starting with an empty set of metrics and iteratively adding a single metric that maximally improves the performance – ACC for the false positive detection and R^2 for the prediction quality estimation. This greedy heuristic is performed for both, meta

	# metrics	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	124
SemanticKITTI	RangeNet++																
	ACC	81.13	82.56	83.35	83.69	83.96	84.22	84.40	84.62	84.78	84.91	85.01	85.09	85.09	85.18	85.25	85.50
	Added	μV	P_{16}	$v\bar{F}^z_{in}$	P_{14}	P_{13}	$v\bar{F}^y$	P_{15}	μF^r_{bd}	μF^i	vF^z	N_1	P_{11}	vF^i	μD_{bd}	P_1	all
	R^2	56.74	58.20	59.01	59.87	60.72	61.57	62.07	62.69	63.24	63.70	64.06	64.43	64.72	64.91	65.15	66.69
	Added	μV	P_{14}	P_{16}	P_{17}	P_{15}	P_9	vF^y_{bd}	$v\bar{F}^z$	μF^i	P_{13}	P_3	vF^x	μF^r	N_{18}	SP	all
	SalsaNext																
	ACC	82.05	83.49	84.10	84.70	84.93	85.16	85.35	85.52	85.65	85.76	85.86	85.96	85.96	86.03	86.11	86.26
	Added	μV	P_{16}	P_{15}	P_{14}	vF^y_{bd}	P_{13}	$\mu\bar{D}$	vF^z	N_1	P_{12}	$\mu\bar{V}^z_{in}$	P_{10}	P_{11}	vF^i_{bd}	vF^x	all
	R^2	56.18	58.56	59.59	60.71	61.67	62.35	62.87	63.36	63.77	64.11	64.40	64.62	64.86	65.02	65.21	66.08
	Added	μD	P_{14}	P_{17}	P_{15}	P_{13}	vF^r	vF^z	P_{12}	P_1	P_9	P_2	$\mu\bar{V}^i$	vF^y	N_{10}	$\mu\bar{V}^z$	all
	Cylinder3D																
	ACC	86.13	86.76	87.19	87.41	87.60	87.75	87.84	87.93	88.02	88.08	88.17	88.26	88.26	88.29	88.33	88.37
	Added	μD	P_{15}	$v\bar{F}^r$	P_{17}	P_{11}	vD	P_{13}	P_{10}	$v\bar{F}^z$	N_1	SP	μF^r_{bd}	N_{15}	P_{14}	μF^i	all
	R^2	53.10	54.54	55.71	56.76	57.64	58.36	58.90	59.31	59.68	60.03	60.32	60.66	60.95	61.12	61.33	61.57
	Added	P_{14}	μE_{bd}	P_{17}	P_{15}	SP	$v\bar{F}^x$	vF^i	P_6	vF^y	N_9	P_{12}	vD	P_{13}	P_5	S	all

	# metrics	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	118
nuScenes	Cylinder3D																
	ACC	90.06	90.27	90.44	90.53	90.61	90.67	90.73	90.8	90.85	90.87	90.89	90.91	90.91	90.93	90.95	91.00
	Added	μD	$\mu\bar{V}^r$	P_9	P_{16}	μF^x_{bd}	vF^r	P_{12}	P_4	$v\bar{E}$	$\mu\bar{V}^i$	N_{11}	P_{15}	μF^z_{bd}	P_3	vF^z_{bd}	all
	R^2	40.88	43.25	44.27	45.14	45.79	46.32	46.80	47.16	47.4	47.62	47.79	48.02	48.1	48.24	48.33	48.84
	Added	μV_{bd}	P_{16}	$v\bar{F}^r$	P_3	P_{13}	μF^r	P_4	vF^z	μF^y_{bd}	μF^i_{bd}	N_{14}	N_{10}	vF^z	P_{10}	P_1	all

Table 4.3: Metric selection using a greedy method that in each step adds one metric that maximizes the meta classification / regression performance in terms of ACC / R^2 in %. All results, SemanticKITTI (top) and nuScenes (bottom) are calculated on the data set's metrics' validation set.

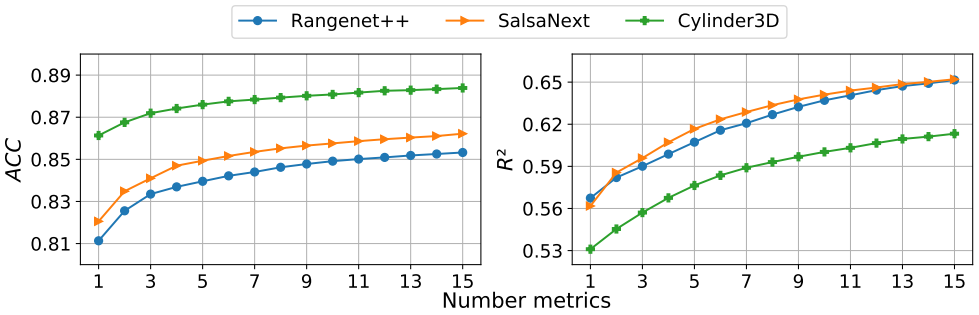


Figure 4.4: Performance of the meta classification (left) and the meta regression (right) model on SemanticKITT depending on the number of metrics, which are selected by the greedy approach.

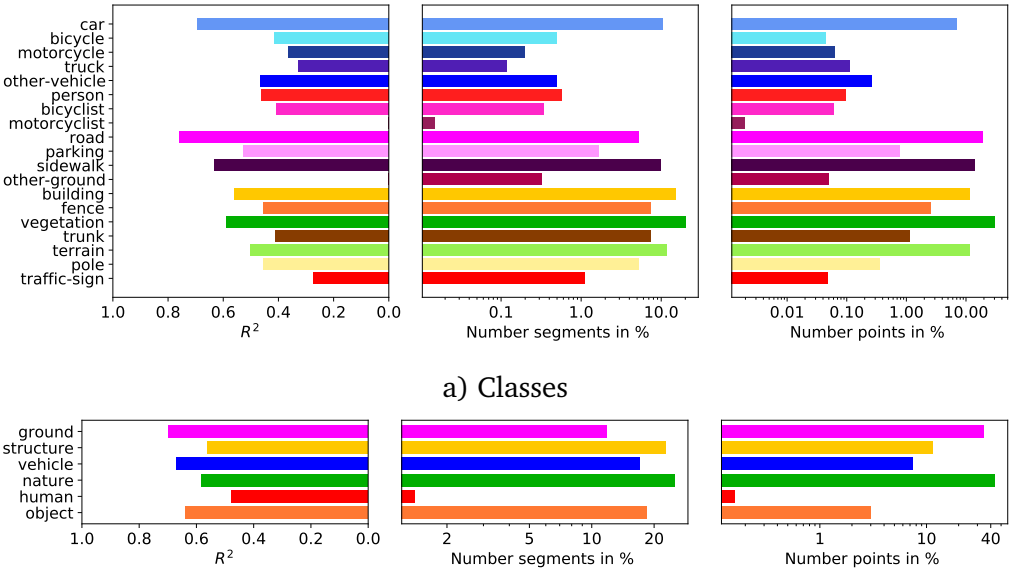
classification and meta regression. The results in terms of ACC and R^2 are shown in table 4.3 and figure 4.4 (for SemanticKITTI). For the meta classification, one can observe a comparatively big accuracy gain during adding the first five metrics, then the accuracy increases rather moderately. For the meta regression, this performance gain in terms of R^2 spreads wider across the first ten iterations, before the improvement per iteration becomes moderate. Furthermore, the results show that a small subset of metrics is sufficient for good meta models. Nearly the same performance can be achieved for both tasks with 15 metrics selected by the greedy heuristic compared to when using all metrics (LMS). Considering table 4.3, the mean variation ratio μV and the mean probability difference μD in most cases constitute the initial choices. Moreover, the mean probabilities P_c, $c \in \mathcal{C}$, are also frequently subject to early

incorporation. In addition to only using the mean entropy for meta classification and regression as shown above, the results clearly show that using multiple metrics increases the performance significantly.

4.4.4 Class- and Category-wise Evaluation

In this section, an in-depth study of the performance of the method is presented. The different numbers of ground truth instances and points per class affect the performance of the segmentation model w.r.t. a given class [104]. Hence, a class-wise and category-wise evaluation of meta regression performance is presented. It will be shown that these performances of LidarMetaSeg are affected by this imbalance as well.

Evaluations per class and category for the meta regression, i.e., the prediction quality estimation, are given in table 4.4. Figure 4.5 a) visualizes the class-wise results for the SalsaNext network and the SemanticKITTI data set. The figure consist of R^2 values, the distribution over predicted segments and points per class and the distribution. The figure shows a high correlation between the R^2 values and each of both distributions. Figure 4.5 a) reveals that classes with higher number of predicted segments and points have in general higher R^2 values, see e.g., 'car' or 'road'. On the other hand, predicted segments belonging to classes underrepresented in the prediction, such as 'bicycle', 'motorcycle' or 'person' (most classes listed in the upper part of the figure), have lower R^2 values. For the class 'motorcyclist' with the fewest predicted segments and points, $R^2 = 0$ is obtained, i.e., a reliable prediction quality estimation for this class is not possible. However, for the class 'other-ground', a moderate number of predicted segments and points are available, but the R^2 is still 0. A closer look at table 4.4 shows that there exists a few predictions of the class 'other-ground'. In comparison to other classes belonging to the same category, that number of predictions is substantially lower. Indeed, 'other-ground' is difficult to distinguish from the other classes in the same category. Also the model's IoU_c for that class is only about 4% (not listed here), which is one of the lowest class-wise IoU values for that network and data set. For its ground truth points, the model mostly predicts in favor of other classes of the same category. Analogously to the class-wise evaluation in figure 4.5 a), figure 4.5 b) depicts the category-wise evaluation, also for SalsaNext and SemanticKITTI. The heterogeneity of the category-wise R^2 values is lower compared to the class-wise R^2 values. All R^2 values are between 47.70% ('human') and 69.73% ('ground'), which correspond to a range of about 17 pp. Even when excluding the class-wise R^2 values of 0, the range of the class-wise R^2 values is about 48 pp. More precisely, the values are enveloped by 27.54% for 'traffic-sign'



a) Classes

b) Categories

Figure 4.5: Barplot of predicted classes (a) and categories (b) of R^2 values, number of segments w.r.t. all segments and number of points w.r.t. all points. Values for number of segments and points are given in $\%$; the x-axes are scaled logarithmically. The visualizations are from SalsaNext and SemanticKITTI.

and 75.75% for 'road'. In the here presented experiments, higher category-wise mean IoU values than class-wise mean IoU are observed. This hints that the classes may potentially be confused with other classes of the same category, while the confusion of classes across different categories is less likely. This is in accordance with typical segmentation benchmarks or [90], and is also reflected in the meta regression R^2 values. For example, for category 'object', it is $R^2 = 63.92\%$, while the mean over its classes yields $R^2 = 39.41\%$, which is substantially lower. Figure 4.6 and figure 4.7 show the quality of predicting the class-wise and category-wise IoU_{adj} for SalsaNext and SemanticKITTI. Almost the same observations as discussed up to now can be made for the models RangeNet++ and Cylinder3D on SemanticKITTI, see table 4.4. Also, the R^2 values for the classes 'motorcyclist' and 'other-ground' are 0. For RangeNet++ and class 'truck' the R^2 is equal to 0 as well. Similar observations as discussed previously can be made for Cylinder3D and nuScenes. No class-wise R^2 values of 0 are obtained. The class 'bicycle' has the lowest number of predicted segments and points, but also the lowest R^2 value. In summary, the results show a clear correlation between predicted class and category distributions to the performance of the meta regression. Thus, higher class- and category-wise R^2 values can be expected when more predicted segments and points of these classes or categories are available.

	RangeNet++			SalsaNext			Cylinder3D		
	R^2	% segments	% points	R^2	% segments	% points	R^2	% segments	% points
ground	**66.89**	**11.09**	**34.09**	**69.82**	**11.84**	**34.38**	**57.97**	**12.11**	**34.33**
road	72.03	5.07	19.55	75.75	5.23	19.42	66.85	5.55	19.49
parking	52.97	1.55	0.78	52.86	1.65	0.77	55.92	1.89	0.77
sidewalk	60.83	9.22	13.77	63.18	9.88	14.18	51.86	9.98	14.12
other-ground	0.00	0.32	0.04	0.00	0.32	0.05	0.00	0.25	0.01
structure	**52.40**	**14.99**	**11.29**	**56.02**	**15.07**	**11.52**	**51.95**	**14.80**	**11.49**
building	52.40	14.99	11.29	56.02	15.07	11.52	51.95	14.8	11.49
vehicle	**69.73**	**17.20**	**7.31**	**66.87**	**17.18**	**7.38**	**64.10**	**18.29**	**7.33**
car	68.54	10.60	6.98	69.32	10.50	6.91	58.53	10.03	6.68
bicycle	36.04	0.26	0.03	41.56	0.50	0.04	35.44	0.74	0.06
motorcycle	58.13	0.20	0.06	36.51	0.20	0.06	50.00	0.30	0.08
truck	0.00	0.13	0.10	32.94	0.12	0.11	44.34	0.16	0.10
other-vehicle	53.27	0.43	0.17	46.44	0.50	0.26	43.04	0.96	0.43
nature	**58.63**	**26.11**	**44.19**	**58.13**	**25.21**	**43.58**	**56.87**	**24.59**	**43.89**
vegetation	60.64	21.28	30.31	58.82	20.24	30.87	64.00	19.79	30.95
trunk	54.99	8.30	1.06	41.06	7.45	1.16	30.44	7.43	1.05
terrain	50.81	11.75	12.69	50.10	11.74	11.43	45.99	13.74	11.73
human	**64.76**	**1.22**	**0.14**	**47.70**	**1.40**	**0.16**	**53.54**	**1.95**	**0.17**
person	59.27	0.53	0.08	46.20	0.57	0.10	48.76	0.85	0.11
bicyclist	42.38	0.32	0.06	40.94	0.34	0.06	46.53	0.35	0.07
motorcyclist	0.00	0.04	0.00	0.00	0.01	0.00	0.00	0.00	0.00
object	**63.94**	**18.03**	**2.97**	**63.92**	**18.49**	**2.99**	**61.88**	**19.07**	**2.85**
fence	50.47	6.66	2.60	45.31	7.34	2.61	48.02	7.14	2.49
pole	55.13	5.49	0.34	45.38	5.31	0.35	33.85	4.95	0.32
traffic-sign	34.69	1.03	0.04	27.54	1.10	0.05	17.21	1.10	0.05

a) SemanticKITTI

	Cylinder3D		
	R^2	% segments	% points
movable-object	**26.34**	**6.47**	**1.11**
barrier	26.95	1.82	1.07
traffic-cone	15.22	0.64	0.05
vehicle	**40.40**	**29.03**	**6.42**
bicycle	12.70	0.12	0.01
bus	48.83	0.63	0.54
car	38.53	7.44	3.79
construction-vehicle	36.52	0.37	0.09
motorcycle	40.25	0.21	0.05
trailer	42.35	0.70	0.24
truck	36.00	1.94	1.74
human	**30.73**	**1.80**	**0.22**
pedestrian	30.73	1.80	0.22
flat	**53.15**	**21.33**	**56.98**
drivable-surface	58.96	9.12	38.24
other-flat	47.61	2.26	1.26
sidewalk	38.97	13.25	8.33
terrain	37.15	13.76	9.34
static	**56.77**	**37.86**	**35.27**
manmade	48.12	27.82	20.90
vegetation	51.39	18.11	14.13

b) nuScenes

Table 4.4: Results for class-wise and category-wise meta regression, averaged over 10 runs. The categories are given in bold face. The corresponding classes of a category are listed below. For every network, the R^2 values, the number of predicted segments per class and category w.r.t. to all predicted segments and the number of points per class and category w.r.t. to all predicted points are given. All values are given in %.

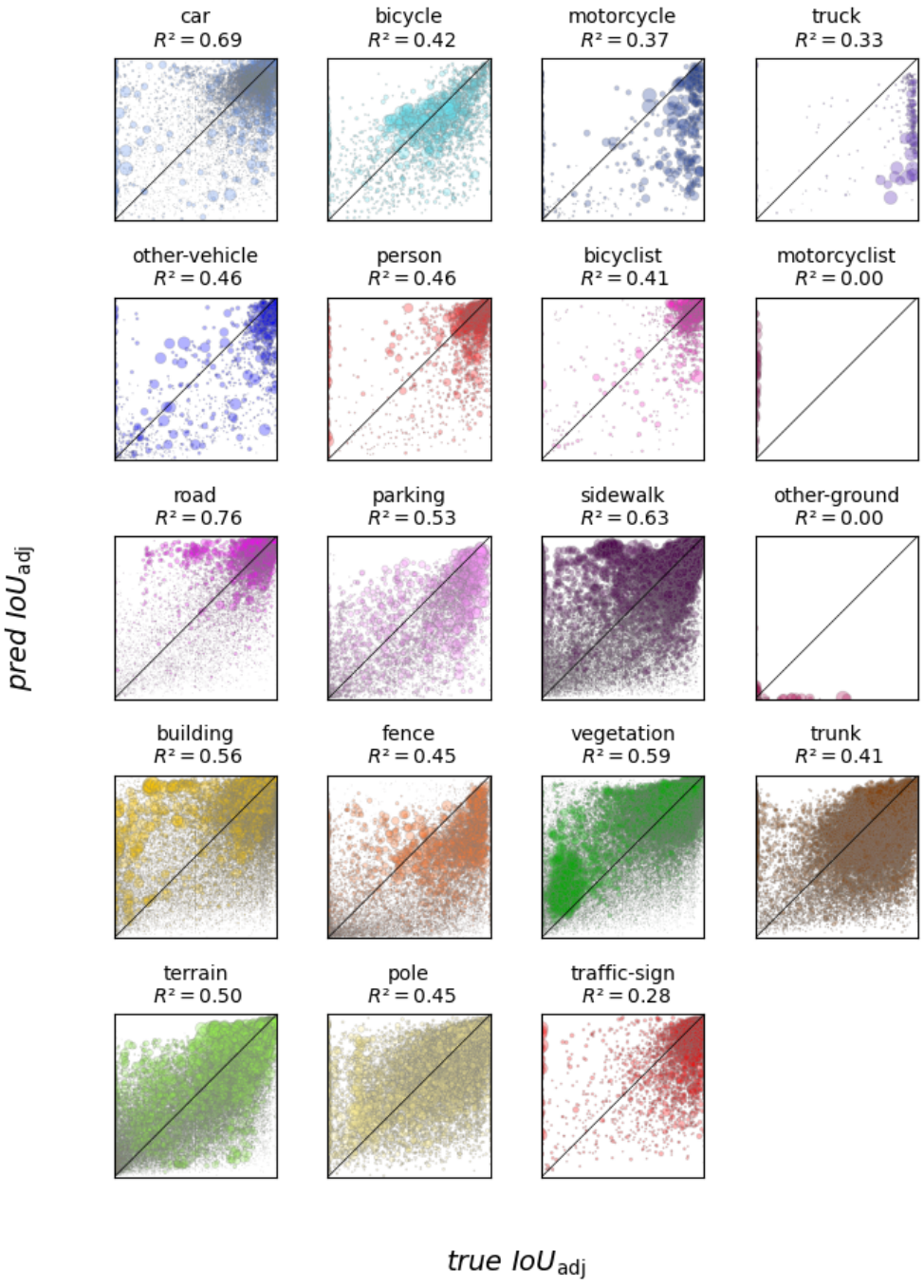


Figure 4.6: True IoU_{adj} vs predicted IoU_{adj} for classes of SemanticKITTI and SalsaNext. The marker size represents the predicted segment size.

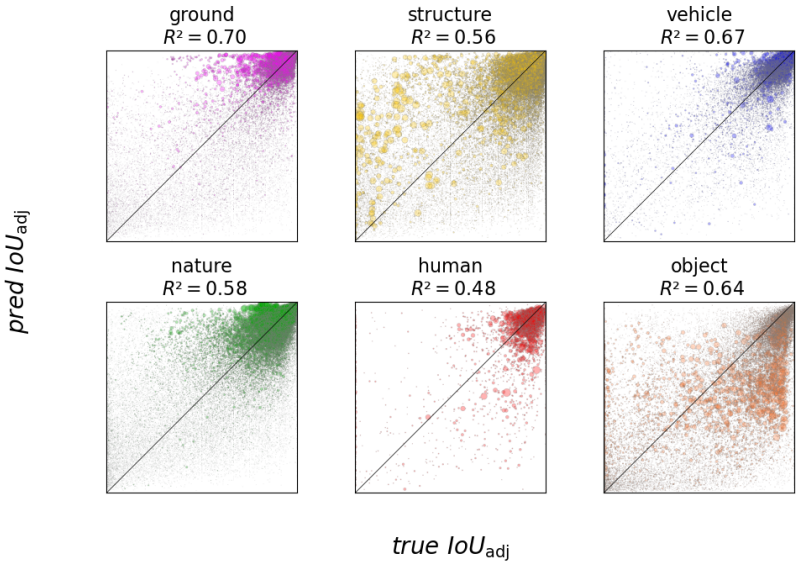


Figure 4.7: True IoU_{adj} vs predicted IoU_{adj} for categories of SemanticKITTI and SalsaNext. The marker size represents the predicted segment size.

4.4.5 Confidence Calibration

The false positive detection is based on a meta classification model, which classifies whether the predicted IoU_{adj} is equal or greater than 0. In order to demonstrate the reliability of the classification model, it is shown that the confidences are well calibrated. Confidence scores are called *calibrated*, if the confidence is representative for the probability of correct classification, cf. [52].

The meta classification model estimates for each predicted segment k the probability $\hat{y}^{prob}_{\text{fp},k} \in [0,1]$ for being false positive, i.e., for $y_{\text{fp},k} = 1$ (and $y_{\text{fp},k} = 0$ else, i.e., $IoU > 0$). The probabilities for all meta classified segments of the validation data are grouped into $M = 10$ interval bins $I_m = (\frac{m-1}{M}, \frac{m}{M}]$. B_m is the set of indices of segments whose confidence falls into the interval $I_m = (\frac{m-1}{M}, \frac{m}{M}]$. The accuracy of a bin B_m is the relative amount of true predictions, i.e.,

$$ACC(B_m) = \frac{1}{|B_m|} \sum_{k \in B_m} \mathbb{1}_{\{\hat{y}_{\text{fp},k} = y_{\text{fp},k}\}} , \tag{4.7}$$

where $\hat{y}_{\text{fp},k} \in \{0,1\}$ is the predicted binary meta class label for k being a false positive segment. Besides, the average confidence within bin B_m is the mean of its probabilities

$$CONF(B_m) = \frac{1}{|B_m|} \sum_{k \in B_m} \hat{y}^{prob}_{\text{fp},k} . \tag{4.8}$$

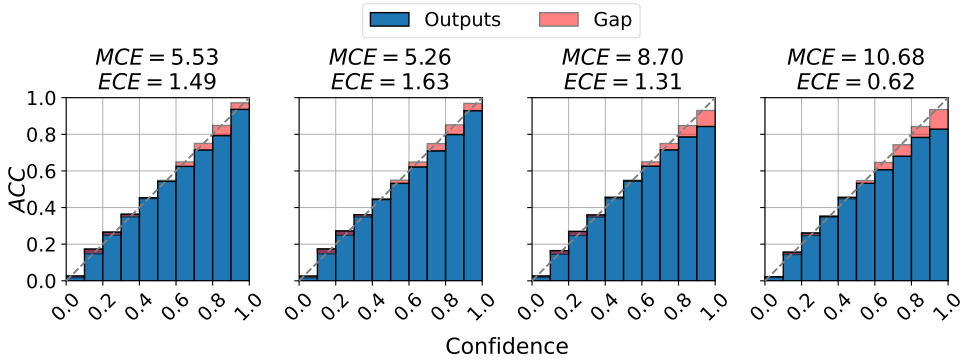


Figure 4.8: Reliability diagrams with MCE and ECE for the meta classification model: RangeNet++, SalsaNext, Cylinder3D on SemanticKITTI as well as Cylinder3D on nuScenes, from left to right. MCE and ECE are given in $\%$.

The closer the accuracy $ACC(B_m)$ and the confidence $CONF(B_m)$ are to each other, the more probabilistically reliable is the corresponding classification model. This is visualized in a so-called reliability diagram. For the evaluation of calibration, the maximum calibration errors (MCE) is defined as

$$MCE = \max_{m \in \{1,\dots,m\}} \left| ACC(B_m) - CONF(B_m) \right| . \tag{4.9}$$

It is the maximum absolute difference between the accuracy and the confidence over all bins. The expected calibration errors (ECE) is

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} \left| ACC(B_m) - CONF(B_m) \right| , \tag{4.10}$$

where n is total number of segments.

The reliability diagrams and the MCE as well as the ECE for all previously discussed meta classification models are computed for segments with $SP \geq 10$ and shown in figure 4.8. The smaller the gaps, i.e., the closer the outputs are to the diagonal, the more reliable and well calibrated is the model. The MCE and ECE are between 5.26 – 10.68 and 0.62 – 1.63, respectively. For predicted categories, the values are almost the same: for MCE, the deviation is in a range of ± 1 pp and for ECE in a range of ± 0.1 pp. The results indicate well calibrated and reliable meta classification models.

4.5 Discussion

In this chapter, the method MetaSeg was adapted and extended to lidar point cloud segmentation. LidarMetaSeg is one of the first works of uncertainty quantification in the field of lidar segmentation. It allows detecting false positive segments and to estimate the prediction quality estimation. LidarMetaSeg is applicable to any state-of-the-art lidar segmentation model. First, the point cloud data is projected on image level. Then dispersion and features measures per pixel are aggregated on predicted segment level and used to train a meta classification and regression model. The performance of both meta tasks was presented. The evaluation has shown that the use of more metrics always yields to the best results. This holds for all considered evaluation metrics – ACC, $AUROC$, $AUPRC$ and R^2. The results also show that adding epistemic and aleatoric uncertainties on top of the dispersion measures based on the softmax probabilities improves neither meta classification nor meta regression performance. Moreover, an in depth-study per class and category of predicted segments was presented, which clearly showing a correlation between meta regression performance and the predicted class and category distributions. In future work, augmentation methods such as presented in [55] can be used, that combines different point clouds or injects objects such as persons / pedestrians or vehicles from the one point cloud into another one. Thus, more training data for LidarMetaSeg can be generated, especially for underrepresented classes or categories, in order to improve the performance of meta classification and meta regression.

The effectiveness of LidarMetaSeg was demonstrated on street scene scenarios, and it can be assumed that this method can be adapted to other lidar segmentation tasks and applications, e.g., indoor segmentation or panoptic segmentation.

As already mentioned, the (image-based) MetaSeg was used or extended in further directions. Such usages or extensions of LidarMetaSeg into further directions are also possible. With regard to the previous chapter 3, it would be interesting to incorporate LidarMetaSeg into an active learning method for lidar point cloud segmentation. The annotation of point clouds is even more complex than for images [3, 10].

5 HD Lane Map Creation based on Trail Map Aggregation

In addition to the real-time perception of the environment, high definition (HD) maps are playing an increasingly important role of ADAS and AD. HD maps provide information about the road system and are referred to as backup solution when the perception system is occluded or fails. In the context of AD, HD maps have to exist for almost everywhere. This requires an automated process for creating HD maps. This chapter presents a procedure to create HD maps of lanes based on detected and tracked vehicles from perception sensor data as well as the own ego vehicle, using multiple observations of the same location. This procedure – named map aggregation – is fully automated, applicable for every environment and does not require any prior map information. Moreover, further ideas how to extend the map aggregation system are presented, aiming for the next ADAS map aggregation system.

5.1 Introduction

In automated driving (AD), HD maps play an important role. While the sensor system, e.g., consisting of camera or radar, is responsible for the real-time perception of the environment, HD maps provide information about the road system, the geometry ahead and the functional meaning of driving lanes. For example, they allow understanding that a certain lane is a dedicated bus lane. Thus, they define where the vehicle is allowed to drive, where traffic signs are located and what is the speed limit, for instance. Furthermore, HD maps are often referred to as a backup solution when a sensor is occluded or fails.

Creating an HD map is a challenging task. Usually, map providers utilize multiple surveillance vehicles equipped with high-cost sensors, e.g., high resolution camera and lidar. Then they drive all roads in the relevant area to map and aggregate this data in an offline process. Sometimes, even additional information is annotated manually by a human labeler. Another issue arises due to the dynamic nature of

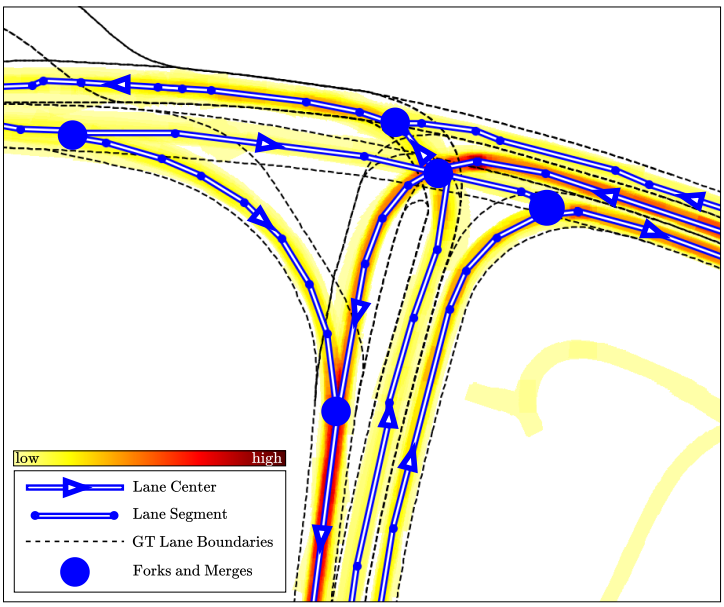


Figure 5.1: Example of an HD lane map provided by the novel procedure. The number of vehicle trajectories are visualized from yellow (low numbers) to red (high numbers). The lane centers with direction indicator of lane (center) segments are shown in blue. Forks and merges are marked with blue circles. For visualization purposes, the lane boundaries (black dashed) from ground truth are added.

the road environment, in which static HD maps are frequently outdated by road geometry changes over time. It is also a challenging task to keep them up-to-date.

Creation or generation of HD maps can be done in multiple ways. They can be annotated manually, which is in general very costly due to the human annotation effort. Thus, this manual approach is not scalable, since HD maps for ADAS and AD are needed almost everywhere and have to be kept up-to-date.

The procedure to create HD maps in this chapter is based on detected and tracked vehicles, i.e., estimated trajectories of observed road users, called *road users trails* as well as the driving path of the ego vehicle, called *ego vehicles' trails*. The data of the trails is obtained by multiple recordings, i.e., it assumed that locations are recorded more than once. The procedure consists mainly of two steps: an aggregation part wherein the trails are aggregated in a map representation and an extraction part in which the lanes are extracted as lane centers. A visualization of the outcome of the new method is shown in figure 5.1. The method presented in this chapter, is developed for the following scenario: a whole fleet or production series vehicles are equipped with a low-cost sensor system, e.g., radar that allows object detection. Road user (vehicles) are detected by an object detection and tracking model. Real-time object detection works with radar data [39, 33] and lidar data [75, 92, 141]. Furthermore, the detections are defined as bounding boxes and require little memory,

which makes a remote transition possible. With an over-the-air (OTA) connection, the trails, i.e., detected and tracked vehicles are sent to a cloud server in which the data is collected. In this scenario, a single vehicle that is equipped with the aforementioned system is able to detect multiple other road users, that enables to derive lane information. By having a whole fleet of those vehicles, a bunch of trails per lane are given and highly precise lanes can be provided. Furthermore, by incorporating uncertainty quantification methods, this becomes even more reliable. Then an algorithm such as the following presented HD map generation procedure is executed to create HD maps. Through the OTA connection, the HD maps can then be transmitted to the vehicles.

The contribution is summarized as follows:

- presenting a new procedure that post-processes ego-car's GPS data together with DL-based perception detection and tracking to aggregate this data in a map format,
- introducing an extraction algorithm that is developed to extract the lane centers from the aggregated data in a structured data format,
- an evaluation on real-world data on an own created data set and an analysis of multiple factors that affect the procedure,
- an outlook presenting the steps towards a next generation map aggregation system, including methods to improve and enrich the HD map data.

This chapter is organized as follows: after the given introduction in this section, related work is reviewed in the next section, section 5.2. The procedure to create HD maps is presented is section 5.3. Experiments and results are given in section 5.4. This chapter concludes with a discussion in section 5.5, also including an outlook to future work.

5.2 Related Work

Some approaches of generating HD maps go back to using GPS signals from multiple devices as kind of crowd sourcing data [11, 133]. For a map provider, this approach is difficult with regard to data protection and the availability of this data. Further information on whether the signal comes from a vehicle may be missing. Other approaches train DL models on various sensor data to predict the road geometry [59, 153]. In [59], a DL model is trained on lidar data, especially the intensity

values are used to predict the road geometry in terms of lane boundaries as well as forks and merges. The approach is demonstrated on highways. The approach in [153] uses different partially processed sensor input data to infer the road geometry in a graph representation. As input for the DL model, birds-eye-view images of the lidar intensity values, RGB images and the semantic segmentation as well as detected vehicles are used. The methods presented in [89, 88] use the trajectories of detected and tracked road users for lane intersection estimation. The authors do not use a DNN, they use probabilistic generative models, more precisely a two-stage Markov chain Monte Carlo [21] sampling approach. This sampling approach is used to estimate the coarse intersection parameters in terms of number of lanes and arms of the intersection. Different intersection models are sampled from a Markov chain. The authors of [89] extend their work in [88] by a non-linear least square formulation to refine the results. They argue that the trajectories in terms of detected and tracked vehicles can be based on camera, radar or lidar. Noteworthily, the evaluation is mostly done on simulated data and the trajectories are purely based on simulated vehicles. It is not only important to create HD maps but also keep them up-to-date. Possible solutions for this are presented for instance in [105, 69].

The procedure to create HD maps in this chapter is based on trails of detected road users as well as the ego vehicles trails. Data from multiple recordings of a location is used, i.e., it is assumed that locations are recorded more than once. Here, the trails from road users are detected and tracked vehicles from lidar point cloud data, but any other sensor data could be provided, that yields 3D detections. Compared to the other methods reviewed above, the method presented here is scalable, since not only the driving paths as GPS signals are used as the approaches in [11, 133] do; also other road users observed and detected via the perception sensors of the ego vehicle are taken into account. Thus, a single ego vehicle can provide data about multiple lanes and all that information can be aggregated to create the HD map. By deriving the lanes from trails, a DL model for this is not required. Furthermore, HD map ground truth data (for training a DL model) is not required as in [59, 153]. Moreover, in high traffic scenarios, the road properties such as lane markings are not visible and the presented procedure benefits from the traffic participants in terms of more trails. Besides that, on many roads there are no lane markings, or due to a poor condition of the road, they are hardly visible or not at all. The related works closest to the procedure in this chapter are the approaches [89] and [88] that make use of generative modeling in order to estimate lane intersections. The HD map generation procedure presented in this chapter uses trajectories of detected and tracked road users as well. However, the procedure is based on map aggregation and not on probabilistic generative models as in [89, 88]. Compared to their work,

here multiple recordings are taken into account. Furthermore, also the driving path of the ego vehicle is taken into account. Moreover, an in-depth study on real-world data and not on simulated data is presented. The new approach is also suitable for intersections, but also for other scenarios such as forks and merges on highways or inner city roundabouts. The presented procedure is not restricted to any scenario or environment and is designed to be used anywhere.

5.3 A new Method for HD Lane Map Generation

This section describes the procedure for HD lane map creation based on trail map aggregation. First, 3D object detection and tracking is applied to sensor data. The detections represented by cuboids are transformed into the global coordinate system (GCS). The cuboids are then aggregated in a map representation. Next, an extraction algorithm is applied to this aggregated data to get the lane information in a structured data set, resulting in the final HD lane maps. In what follows, a lane center line or for the sake of brevity *lane center* represents a lane between two forks and merges (or the start and end of a lane) by its sequence of lane center coordinates. The section between two coordinates in the sequence (not necessarily equidistant for different coordinates) is given by a line segment, which is here called lane (center) segment. Thus, the collection of all lane centers can be viewed as a directed graph, where the lane segments are the edges and their connection points, their start and end points as well forks and merges are the nodes, see figure 5.1. For the procedure, it is assumed that the ego vehicle is equipped with a sensor system e.g., camera, radar or lidar, that allows for performing 3D object detection and tracking. Furthermore, the ego car's location at the time of recording in terms of GPS is required. In the following, the outlined steps of the procedure are described in detail.

5.3.1 Detection and Tracking

The trails of road users are based on tracked detections of vehicles. Therefore, the availability of a 3D bounding box detection and a tracker is assumed, to identify the road users over multiple frames. It is also assumed that the orientation of the cuboids is given, i.e., having access to a prediction indicating which side of the bounding box belongs to the front side of the corresponding vehicle. A tracker confidence and a flag signaling whether the tracked vehicles are stationary or moving are required, too.

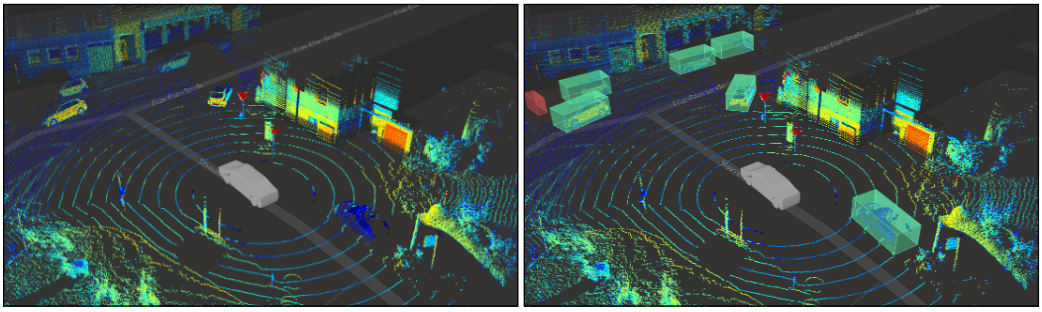


Figure 5.2: Example of a lidar point cloud (left) and lidar point cloud including detected objects (right). The ego vehicle is located in the center (gray) and detected vehicles are shown by cuboids (green).

Once other road users are detected, the position and orientation can be derived from the GPS position of the ego vehicle and the calibration of the sensor system. For example, if using a lidar sensor, which provides dense and spatial data with range information, the cuboids from the local coordinate system can be transformed to the global coordinate system. Figure 5.2 shows a point cloud and detected vehicles by cuboids.

A track or *trail* t is defined as a trajectory of cuboids $c_j,\ j = 1, \ldots, p$, i.e., $t = \{c_1, \ldots, c_p\}$. Each trail t is enriched with additional meta information:

- $t_{conf} \in [0, 1]$ tracker confidence,
- $t_r \in \mathbb{R}^p$ the euclidean distance from the ego vehicle to every cuboid,
- $t_{len} \in \mathbb{R}$ the euclidean distance between the center points of the first and last cuboid of a trail.

Cuboids and trails will be aggregated depending on their meta information, aiming for aggregating less uncertain data. Own experiments have shown, that in long range more objects are not detected, i.e., more false negatives occur. Besides that, even if an object is detected in long range, the IoU with the ground truth is lower compared to detected objects in near range. That means, the localization in long range is less accurate. In addition, tracking of an object over multiple frames is generally more reliable than tracking an object over a few frames because more prior knowledge is included. For example, using a Kalman filter for tracking, a longer track gets a better approximation of the covariance prediction $\mathbf{P}_t^-$ for the measurement accuracy, see section 2.3.2.

The ego vehicle's trail is defined as well, which is provided by the localization and the dimension of the vehicle.

5.3.2 Aggregation

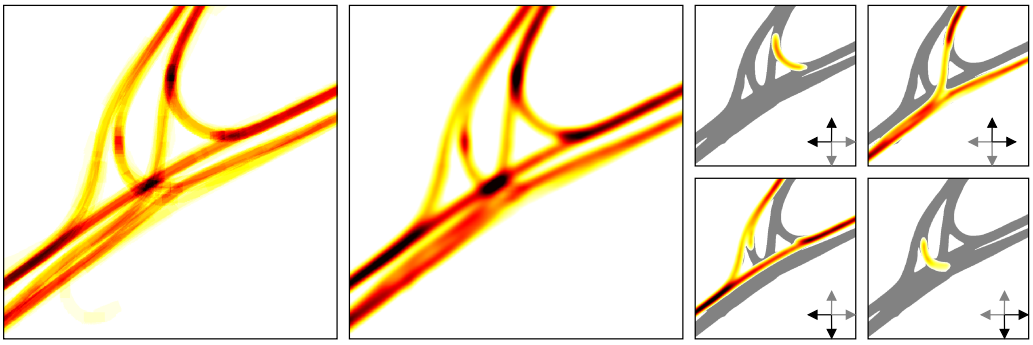
For the aggregation of a given location, a *density map* $\mathbf{D} \in \mathbb{N}_0^{m \times n}$ is defined and supposed to accumulate the road users trails as well as the one of the ego vehicle in terms of a density, projected from a birds-eye-view onto the 2D plane, cf. figure 5.3. Each element of $\mathbf{D}$ corresponds to a region of $10 \times 10\,\text{cm}^2$ in the GCS. Initially, all values of $\mathbf{D}$ are 0. For every trail t that covers the given location, it will proceed as follows. For every cuboid c_j of a trail t of a road user in motion and depending on thresholds for the meta information t_{conf}, t_r, t_{len}, defined as *aggregation parameters*, the cuboids are projected into the density map and the values of $\mathbf{D}$ are incremented that overlap with the cuboids. A value in $\mathbf{D}$ is incremented only once per trail. After processing the cuboids of the first trail, a value in $\mathbf{D}$ can either be 0 or 1.

As already mentioned, the orientation of the cuboids is required. The orientation is used to construct direction-dependent density maps, for the sake of brevity *direction density maps* that are used in the sub-sequent extraction part and are vital for the method. When constructing the graph representation, the consideration of the density map $\mathbf{D}$ (and not the direction density maps) can lead to connecting lanes with opposite driving direction once they cross each other. The orientation in GCS can be represented by the yaw rate $\phi \in [0, 360]$ (deg). The orientation is discretized into four classes and used to define four direction density maps $\mathbf{D}_d \in \mathbb{N}_0^{m \times n}, d = 1, \ldots, 4$. Iteratively all trails and cuboids w.r.t. to the orientation of the cuboids are mapped to the corresponding direction density maps such that $\mathbf{D}_d$ contains the cuboids with ϕ as follows:

- $\mathbf{D}_1$: $\phi \in [0, 90)$ (north to east / top to right),
- $\mathbf{D}_2$: $\phi \in [90, 180)$ (east to south / bottom to right),
- $\mathbf{D}_3$: $\phi \in [180, 270)$ (south to west / bottom to left),
- $\mathbf{D}_4$: $\phi \in [270, 360)$ (west to north / top to left).

For the later extraction part, each direction density map has an opposite one. $\mathbf{D}_1$ and $\mathbf{D}_3$ are opposite to each other as well as $\mathbf{D}_2$ and $\mathbf{D}_4$. Examples of (direction) density maps are given in figure 5.3.

Cuboids and trails are aggregated only if their meta information are above or below certain values, the aggregation parameters. For example, only the cuboids and trails are aggregated that are close to the ego vehicle, i.e., $t_r \leq 50\,\text{m}$, requiring a high tracker confidence $t_{conf} \geq 0.75$ and having a minimal length i.e., $t_{len} \geq 20\,\text{m}$. These



a) Density map $\mathbf{D}$ b) Smoothed density map $s(\mathbf{D})$ c) Smoothed direction density map $s(\mathbf{D}_d)$

Figure 5.3: Density map visualizations from aggregation part. The density map $\mathbf{D}$ and the corresponding smoothed density map $s(\mathbf{D})$ are shown in a) and b), respectively. In c) the four direction density maps $s(\mathbf{D}_d)$ are shown. Additionally, density map $s(\mathbf{D})$ is added in gray to all direction density maps $s(\mathbf{D}_d)$ for a better visualization.

particular aggregation parameters are termed as 'conservative'. More details on aggregation parameters and studies on them are given in section 5.4.

After mapping the trails of road users, also the cuboids of the ego vehicle's trails are mapped and aggregated to the (direction) density maps $\mathbf{D}$ and $\mathbf{D}_d$, $d = 1 \ldots, 4$ without any meta information, but with double increment weight, since the own trails are obviously more reliable than the detected road users' trails.

The present procedure argues to generate reliable HD maps by having a high number of observation, i.e., trails. As pre-processing of the extraction, the values in $\mathbf{D}_d, d = 1, \ldots, 4$ are set to 0 where the values are not greater or equal to a given minimal number of trails $num_t \in \mathbb{N}$. This parameter is intended to avoid false positive lanes or unwanted detections. For example, if a specific location is recorded with moderate traffic ten times and in a specific area there is only one detection out of ten recordings, then this might be a false positive. To this end, the minimal number of trails num_t ensures a certain number of detections in relation to the recordings.

An additional pre-processing of the extraction is a smoothing of the density map $\mathbf{D}$ and the direction density maps $\mathbf{D}_d$. This is done by applying a convolution with stride $s = 1$, kernel size $k = 20$, fixed kernel weights of 1 and zero padding. The (direction) density maps $\mathbf{D}$, $\mathbf{D}_d$ and the smoothed (direction) density maps $s(\mathbf{D})$, $s(\mathbf{D}_d)$ have the same dimensions. The smoothed density maps are more suitable for the sub-sequent extraction of lane centers than the density maps. For example, if having only the trail of one vehicle, then the corresponding values in the density

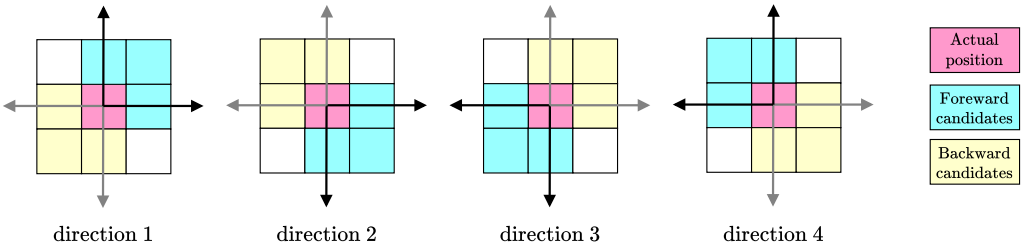


Figure 5.4: Illustration of candidates from whose values are to be chosen from actual position w.r.t. to the direction d of the density map.

map $\mathbf{D}$ are equal to 1. In the smoothed density map $s(\mathbf{D})$ the values would be different: the values in the corresponding center of the trail are higher than the values to the side of the trail. Thus, the center of the trail can be interpreted as a collection of local maxima in affine sub-spaces orthogonal to the trail. Examples of $\mathbf{D}$ and $s(\mathbf{D})$ are given in figure 5.3. In what follows, only the smoothed maps will be used. Therefore, the term 'smoothed' is omitted.

5.3.3 Extraction

For the extraction of lane centers, first a pre-extraction algorithm on each direction density map is employed, second a cutting process and third a connection step.

The *pre-extraction* as first part is an iterative process. For every direction density map $s(\mathbf{D}_d)$, $d = 1, \ldots, 4$ it is proceeded as follows: The highest value in $s(\mathbf{D}_d)$ is picked as starting point. From this position, the next highest value from the candidates in the current direction is chosen, as depicted in figure 5.4.

For example, if density map $\mathbf{D}_3$ (south to west / bottom to left) is processed, then the candidate with the highest value to left, bottom left or bottom is chosen. Pre-extracting w.r.t. to the discretized orientation / direction d is called forward extraction. After the forward extraction, a backward extraction is performed where it is proceeded as in the forward process with the same starting point, but in the opposite direction (not opposite density map!) of d, see figure 5.4. In the mentioned example, the values to the top, top right and right from the actual position are candidates to be checked. The forward and backward extraction stops if

1) the actual position is at the end of the map,

2) the next candidates' values are 0 which means there is no trail information,

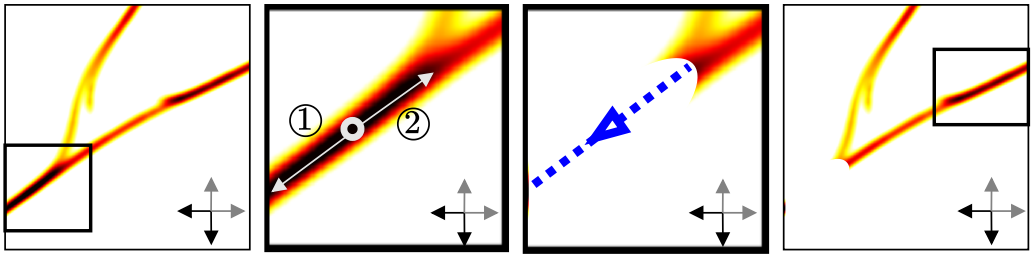


Figure 5.5: Illustrations of the first part of the extraction. Iteratively, the lane centers are extracted from the direction density maps. Here, ① illustrates the forward and ② the backward extraction. After extracting the first lane center of the highlighted area (first illustration and zoom in second and third illustration), the values around the extracted lane center are set to 0 as indicator for 'processed'. The method continues with the next lane center extraction by picking the highest value on the updated map that lies in the highlighted region of the last illustration.

3) the next highest candidate value is much smaller than the mean of the already extracted values.

The idea of the last stopping condition $3)$ is to avoid lane mixing in case of parallel lanes with same direction. After the pre-extraction of the first lane center, the values close to the extracted values, i.e., in a radius corresponding to $2\,\mathrm{m}$ are explicitly set to 0 and it is continued with picking the highest value on the updated density map $s(\mathbf{D}_d)$. This iterative process ends when all values in $s(\mathbf{D}_d)$ are 0. Figure 5.5 illustrates the first iteration of the pre-extraction. Pending is the motivation of the backward extraction. Without the backward extraction, it will often happen that the next highest value on the updated density map is just behind (in terms of driving direction) the last picked one from the previous iteration. This would yield more iterations and short lane centers. The backward extraction avoids this and makes the extraction in this chapter faster and yields better results.

After performing the previously defined steps, the following intermediate observation can be made: due to the pre-extraction, where the values of the region around extracted lane centers are set to 0, the start and end coordinates of extracted lane centers are not connected to the start or end coordinates of adjacent lane centers, see figure 5.6 (left). To overcome this, a *cutting* (second) and a *connection* (third) step is applied. In the cutting part, the start and end coordinates of all unconnected lane centers are checked if their values are within the region's (e.g., radius corresponding to $0.5\,\mathrm{m}$) highest values of their corresponding direction density map. If not, they are cut off, see figure 5.6 (center). In order to have a directed graph representation, adjacent lane centers have to be connected. In the *connection* part, the previously described pre-extraction part is applied to every position of a lane center start or

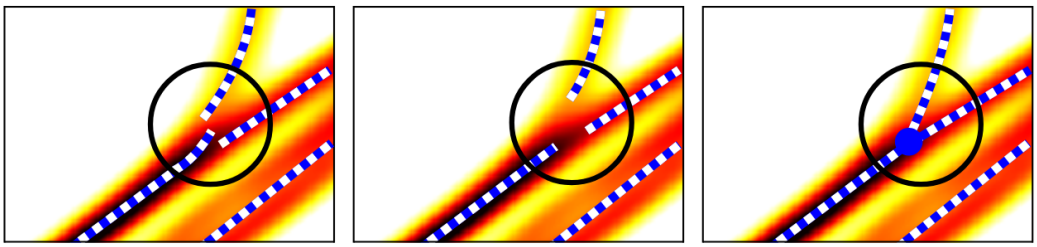


Figure 5.6: Illustrations of the second and third part of the extraction. The results of the pre-extraction is shown on the left side. The lane centers are not connected. The centered illustration shows the outcome of the cutting part. The values of the lane center start and end coordinates do not lie in the range of the neighbors highest values. The last illustration on the right side shows the results of the third, the connection part. The start or end coordinates of all three lane centers in the highlighted region are the same and represent a fork and merge node in a graph representation.

end coordinate that is unconnected. However, now it is operated on the density map $s(\mathbf{D})$ without distinguishing the direction of the density map. Furthermore and in addition to the stopping conditions $1) - 3)$ above, the process stops if

$4)$ in the very near range of the current position (e.g., radius corresponding to $0.5\,\text{m}$) there is a plausible lane center.

A lane center in near range is plausible, if its direction is not the opposite direction of the trail seeking a connection. The results of the connection part is shown in figure 5.6 (right).

Finally, each lane center is given as a list with its coordinates, ordered w.r.t. to the driving direction. Due to the above extraction method, the coordinates are connected and may contain redundant coordinates for the lane center representation. In order to reduce the complexity of the graph, redundant coordinates are removed with a Ramer-Douglas-Peucker (RDP) algorithm [110, 38]. The remaining coordinates for each lane center define the lane segments. The connection of lane segments as well as forks and merges represent nodes and the lane segments represent edges in a graph representation. The whole procedure for aggregation and extraction is summarized in algorithm 4.

Algorithm 4: Procedure for HD lane map generation

Input: Set of trails $\{t^1, \ldots, t^T\}$ with $t^i = \{c_1 \ldots, c_p\}$ with meta information $t^i_{conf}, t^i_r, t^i_{len}$ and aggregation parameters as well as $num^i_t,\ i = 1, \ldots, T$

`// Aggregation`

Create (direction) density maps $\mathbf{D}$ and $\mathbf{D}_{d=1,\ldots,4}$ depending on aggregation parameters

Set all value in $\mathbf{D}, \mathbf{D}_d < num_t$ to 0

Get $s(\mathbf{D}), s(\mathbf{D}_{d=1,\ldots,4})$ by smoothing $\mathbf{D},\ \mathbf{D}_d$

`// Pre-extraction`

for $d = 1, \ldots, 4$ **do**

  while $s(\mathbf{D}_d) \neq \mathbf{0}$ **do**

    Get $\max s(\mathbf{D}_d)$ and corresponding position $pos = \text{argmax}\, s(\mathbf{D}_d)$

    while *Stopping condition* $1) - 3)$ *is not met* **do**

      Forward extraction w.r.t. d starting at pos

    end

    while *Stopping condition* $1) - 3)$ *is not met* **do**

      Backward extraction w.r.t. d starting at pos

    end

    Update $s(\mathbf{D}_d)$ by setting values around extracted coordinates to 0

  end

end

`// Cutting`

for *every unconnected lane center start coordinate* **do**

  Follow lane center forward and stop when the value of current position is one of the highest in near range and delete coordinates up to the actual position

end

for *every unconnected lane center end coordinate* **do**

  Follow lane center backward and stop when the value of current position is on of the highest in near range and delete coordinates up to the actual position

end

`// Connection`

for *every unconnected lane center end coordinate* pos_e **do**

  while *Stopping condition* $1) - 4)$ *is not met* **do**

    Forward extraction w.r.t. d on $s(\mathbf{D})$ starting at pos_e

  end

end

for *every unconnected lane center start coordinate* pos_s **do**

  while *Stopping condition* $1) - 4)$ *is not met* **do**

    Backward extraction w.r.t. d on $s(\mathbf{D})$ starting at pos_s

  end

end

Apply RDP algorithm

Output: Lane centers

5.4 Experiments and Results

For the experiments, an own data set was created. Public data sets such as Argoverse [14], nuScenes [10], Lyft Level 5 [61] or Waymo Open Motion [131] provide HD maps but do not provide multiple recordings per location as needed here. The own created data set covers a region of $1 \times 2.5\,\text{km}^2$ in Wuppertal, Germany, including urban and sub-urban environment as well as highway. Each street w.r.t. to driving direction is recorded up to 5 times. The ego (recording) vehicle is equipped with a Velodyne Alpha Prime lidar sensor and an Applanix, providing the dGPS information. In order to evaluate the HD map generation procedure, an HD map covering the region of the created data set is used as ground truth. The ground truth provides information about the lane boundaries as well as the lane centers. In total, $18\,\text{km}$ of lanes are evaluated. The region of the data set is shown in figure 5.8 (bottom right).

For object detection an ensemble of two networks has been used, namely PointPillars [75] and Part-A^2 [123], see section 2.3.2. Both networks are trained on an internal, proprietary data set. For tracking detected 3D bounding boxes a Kalman filter [70] is used. The tracker confidence t_{conf} is defined as the mean of the underlying bounding box confidence scores.

The experiments aim to answer the following questions.

a) How do different aggregation parameters affect the overall quality and quantity of the lane center estimates?

b) What is the impact on the lane center quality and quantity when using only the trails of the ego vehicle, only the trails of other road users and when using both?

c) How does the quality and quantity of the lane center estimates develop when increasing the number of recordings?

5.4.1 Metrics for Evaluation

For evaluation, the given task of lane center detection is considered from a classification and a regression perspective. The estimated lane centers are adjusted, such that the coordinates have a fixed euclidean distance of $1\,\text{m}$ to each other, i.e., they are equidistant. The same is done for the ground truth (gt) lane centers.

For classification, an estimated lane center coordinate is considered correct and counted as true positive (tp) if it lies between the gt lane boundaries. Otherwise, it is considered incorrect and counted as false positive (fp). The gt lane center coordinates that do not match with estimated lane center coordinates between the lane boundaries are false negatives (fn). Based on that, precision P is computed as

$$P = \frac{\sum_{j_1}^{m_1} tp_j}{(\sum_{j_1}^{m_1} tp_{j_1} + \sum_{j_2}^{m_2} fp_{j_2})}, \tag{5.1}$$

with $m_1 + m_2$ number of estimated lane center coordinates and recall

$$R = \frac{\sum_{j_1}^{m_1} tp_{j_1}}{(\sum_{j_1}^{m_1} tp_{j_1} + \sum_{j_3}^{m_3} fn_{j_3})}, \tag{5.2}$$

with $m_1 + m_3$ number of gt lane center coordinates. Precision is a measure of quality indicating how much of the estimates are correct. Recall is a measure of quantity and implies the proportion of estimated (detected) lane centers.

Viewing the lane center estimation as a regression task, the mean absolute error MAE and root mean squared error $RMSE$ are computed as:

$$MAE = \frac{1}{m}\sum_{j}^{m} e_j, \quad RMSE = \sqrt{\frac{1}{m}\sum_{j}^{m} e_j^2}, \tag{5.3}$$

with $m = m_1 + m_2$ where the error e_j is the distance between an estimated lane center coordinate to the gt lane center. Furthermore, the MAE_{TP} and $RMSE_{TP}$ is defined only for correct estimated lane centers, i.e., for true positives coordinates:

$$MAE_{TP} = \frac{1}{m_1}\sum_{j}^{m_1} e_j, \quad RMSE_{TP} = \sqrt{\frac{1}{m_1}\sum_{j}^{m_1} e_j^2}. \tag{5.4}$$

In order to evaluate an experiment with n sets of parameters, the average precision AP is used, see section 2.3.3.

5.4.2 Evaluation

In the following, the aggregation parameters are defined as:

- 'conservative': $t_{conf} \geq 0.75$, $t_r \leq 50\,\text{m}$,
- 'normal': $t_{conf} \geq 0.50$, $t_r \leq 100\,\text{m}$,
- 'aggressive': $t_{conf} \geq 0.25$, $t_r \leq 150\,\text{m}$.

In all experiments, it is $t_{len} \geq 20\,\text{m}$. With a lower value t_{len} there would be more false positives in the detection and tracking. A higher value does not lead to a significant difference in the following presented results.

Aggregation Parameters. Figure 5.7 (top) shows the results using different parameters for aggregation 'aggressive', 'normal' and 'conservative' as well different minimal number of trails $num_t \in \{1, 2, 5, 10\}$. The numerical results are presented in table 5.1.

Comparing the results in terms of precision and recall for 'aggressive', 'normal' and 'conservative' and for a given num_t, the values are in the same range but 'conservative' has a little higher precision and a lower recall than 'aggressive', cf. figure 5.7. The parameter num_t has a higher impact on the results than the other ones (t_{conf}, t_r). For example, when considering the 'normal' aggregations, the precision and recall goes from 0.9280 and 0.8153 for $num_t = 1$ to 0.9799 and 0.4087 for $num_t = 10$, respectively. This is a difference of about 0.05 for precision and 0.40 for recall, which shows that the impact on recall is much higher than on precision. However, the results show that with a minimal number of trails $num_t = 2$, a precision and recall of 0.9596 and 0.7703 is achieved, respectively. This indicates that the procedure does not require a large number of trails.

Furthermore, it can be observed that the higher the precision, the lower are MAE and $RMSE$ due to lower numbers of false positives. For correct detected lane centers, MAE_{TP} is about $0.23\,\text{m}$ and $RMSE_{TP}$ is about $0.37\,\text{m}$. The errors in terms of regression are mainly caused by parking areas or curves. In areas where vehicles are parking to the side of a road, the vehicles tend to drive more to the inner of the street. In curves, the vehicle tend to drive more to the inside as well, cf. figure 5.1. Overall and evaluating all parameter sets, an average precision of $AP = 0.8085$ is achieved.

Ego Vehicle vs. Road Users. The same experiments as in the previous paragraph were repeated for only using the trails of the road users and only using the trails of the ego vehicle. Since there are no defined parameters for the ego vehicle and weight is double incremented, the plot of the ego vehicle in the aggregation, figure 5.7 shows only three precision-recall values ($num_t = 2, 5, 10$) for the ego vehicle. It was previously shown, that the impact of the thresholds to the tracker confidence t_{conf} and range t_r are not as high as the minimal number of trails num_t. Therefore, the numerical results are only given for 'normal' aggregations, see table 5.2. The results show that the ego vehicle achieves the best results in terms of precision

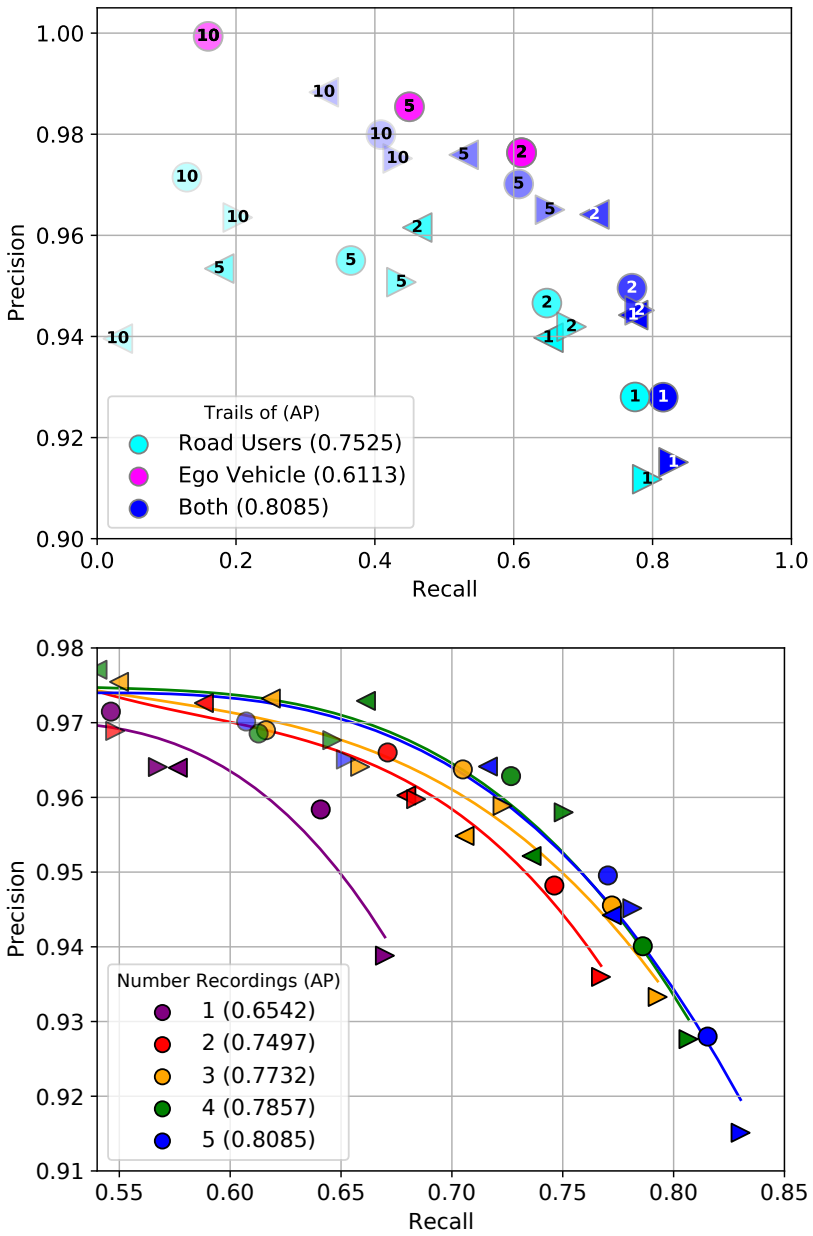


Figure 5.7: Results of the HD lane map creation procedure. Top: results depending on using only the trails of road users, only the ego vehicle's trail and using both. Bottom: results depending on the number of observations i.e., how often every road is recorded. The lines represent an approximation of the results. The markers in both plots show the parameters of aggregation: left arrow 'conservative', circle 'normal', right arrow 'aggressive'. The brighter the markers, the lower the minimal of trails is assumed to be. The minimal number of trails num_t is also shown in the markers (only in top part). The average precision (AP) for each experiment is given in brackets in the legends.

Aggregation Parameters	num_t	Precision	Recall	MAE	$RMSE$	MAE_{TP}	$RMSE_{TP}$
$t_{conf} \geq 0.25$ $t_r \leq 150\,\text{m}$ 'aggressive'	1	0.9151	0.8302	0.5494	1.6531	0.2295	0.3698
	2	0.9452	0.7814	0.4542	1.3892	0.2251	0.3657
	5	0.9651	0.6522	0.3758	1.1110	0.2249	0.3653
	10	0.9752	0.4329	0.3193	0.7962	0.2293	0.3656
$t_{conf} \geq 0.50$ $t_r \leq 100\,\text{m}$ 'normal'	1	0.9280	0.8153	0.4950	1.4947	0.2291	0.3675
	2	0.9496	0.7703	0.4482	1.3637	0.2253	0.3643
	5	0.9702	0.6072	0.3597	0.9872	0.2324	0.3696
	10	0.9799	0.4087	0.2997	0.6599	0.2329	0.3698
$t_{conf} \geq 0.75$ $t_r \leq 50\,\text{m}$ 'conservative'	1	0.9442	0.7721	0.4437	1.3233	0.2354	0.3737
	2	0.9641	0.7162	0.3839	1.1066	0.2327	0.3697
	5	0.9759	0.5279	0.3338	0.8022	0.2414	0.3818
	10	0.9883	0.3264	0.2720	0.4774	0.2454	0.3768

Table 5.1: Results depending on aggregation parameters. To compare the results when using only the trails of the ego vehicle and only of the road users see column 'normal' in table 5.2.

Aggregation Parameters	num_t	Precision	Recall	MAE	$RMSE$	MAE_{TP}	$RMSE_{TP}$
Ego Vehicle 'normal'	1	0.9763	0.6113	0.3243	0.8375	0.2450	0.3798
	2	0.9763	0.6113	0.3243	0.8375	0.2450	0.3798
	5	0.9854	0.4498	0.3060	0.5818	0.2615	0.3975
	10	0.9994	0.1599	0.2742	0.3915	0.2733	0.3896
Road Users 'normal'	1	0.9280	0.7748	0.5300	1.5423	0.2508	0.3983
	2	0.9466	0.6478	0.4772	1.4051	0.2349	0.3829
	5	0.9550	0.3653	0.4188	1.2197	0.2170	0.3604
	10	0.9715	0.1293	0.3065	0.7682	0.2122	0.3579

Table 5.2: Results depending on using only the trails of the ego vehicle and only the trails of road users.

$0.9736 - 0.9994$, but the recall achieves a maximum of 0.6113. This means, only about 61% of all lanes are detected. The data sets consists of roads with up to three lanes in each driving direction. During recordings, no care was taken to drive on each lane at least once. Otherwise, i.e., having a recording and thus, having an ego vehicle trail for every lane would yield a baseline to the procedure. The precision for using the trails of the road users is up to 0.05 lower than the precision of 'ego vehicle' but the maximum recall is 0.7721 and thus, about 0.16 higher, cf. table 5.1 and table 5.2. The average precision (over all aggregations) is $AP = 0.6031$ for the ego vehicle and $AP = 0.7525$ for the road users. This clearly shows that the use of road users leads to more lane detections and make the approach scalable. The best results are still achieved in terms of average precision when using both: the ego vehicle and the road users where the average precision is $AP = 0.8085$.

Number Recordings. The results for a varying number of recordings per location are shown in figure 5.7 (bottom). In general, one can observe that the more recordings are available, the better are the results in terms of precision and recall.

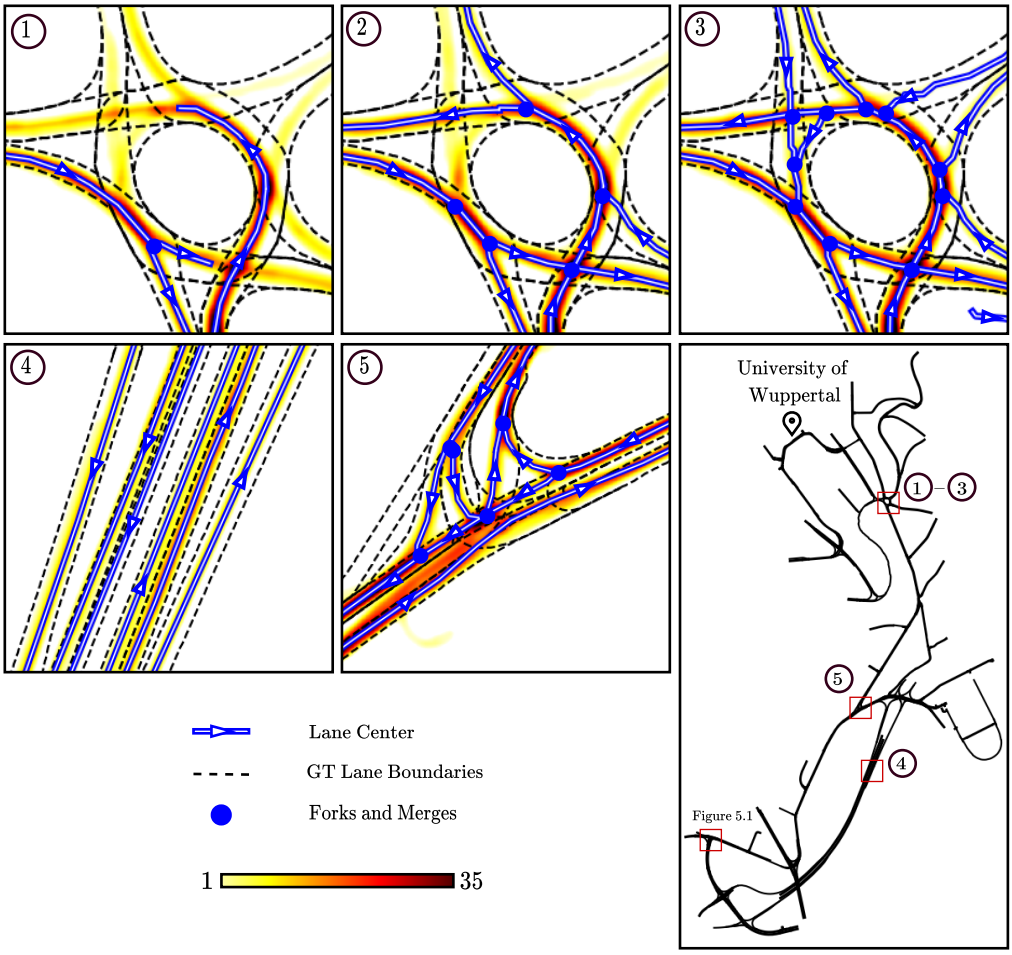


Figure 5.8: Region of the created data set and visual examples of HD lane maps by the new map aggregation procedure. Part ①–③ shows results of different parameters for aggregation for a roundabout: ① 'conservative', ② 'normal' and ③ 'aggressive'. In part ④, a highway scenario is shown as the outcome of 'normal' parameters. Part ⑤ shows the corresponding area of figure 5.3, figure 5.5 and figure 5.6.

The values for average precision are $AP_1 = 0.6542 < AP_2 = 0.7497 < AP_3 = 0.7732 < AP_4 = 0.7857 < AP_5 = 0.8085$ where the subscript represents the number of recording. This shows also the improvements with increasing number of recordings. While the increase from one to two recordings is high, this increase saturates with the number of recordings and there are only minor improvements.

The experiments show a general trade-off between precision and recall, depending on the parameters. If they are too conservative, this leads to a very high precision but a low recall. If they are too aggressive, the results have a high recall but a little lower precision. The 'normal' aggregation parameters are a good trade-off between both. Visual results of different scenes are shown in figure 5.8.

5.5 Discussion and Future Work

A novel approach of HD lane map generation based on trail map aggregation was presented. A procedure was shown, consisting of an aggregation and extraction method that is build on a 3D bounding box detection and tracking. The impact of different aggregation parameters on the performance were studied, the impact of number of recordings as well as only using the trails of the ego vehicle, only the trails of road users and the combination of both. Real-world data was used for all experiments. With a low number of recordings, 77% of all lanes can be detected with a high precision of 0.95. The lane centers are on average accurate up to $23\,\mathrm{cm}$. Taking other road users into account makes the procedure scalable. On top of that, the procedure is not restricted to any environment or street scenarios, which is a clear advantage over related work. For future work, it is planned to add the lane width and/or to determine the lane boundaries. In addition, specific environments will be studied in more detail and the procedure will be compared with related work in specific environments.

The procedure has shown promising results of automated HD map generation. In order to make this procedure scalable and applicable in AD and ADAS, more work has to be done. Companies like Mobileye which are also working on map aggregation systems [93], have several teams working on this topic. Thus, the presented procedure is a starting point of a future map aggregation system that is as follows.

Map Aggregation System. In order to get trails in terms of detected and tracked vehicles from almost everywhere in the world, it is assumed that production series vehicles are equipped with low-cost perception and localization sensor systems. Furthermore, these vehicles require an OTA connection to a cloud service of the map aggregation provider. Low-cost perception and localization systems mean that for the recognition of the surrounding cheap sensors such as radar or cameras are used as well as a GPS for the localization. The here presented procedure was evaluated with an expensive lidar and dGPS sensor. However, the procedure only requires a localization and a 3D bounding box detection which can also be provided by GPS and a radar and/or camera system, respectively. In such a future map aggregation system, the tracks are sent OTA to the cloud service, in which the data is collected and processed as described in this chapter. Besides the HD lane map generating, the cloud service should also be able to provide an HD map update method or only provide HD lane maps based on the actual data in order to continuously provide

up-to-date map data. Furthermore, not only lanes can be provided but also more map features such as sidewalks based on detected pedestrians. Moreover, the HD map data can be enriched with additional meta information such as average and standard deviation of the traffic, the velocity or acceleration. The procedure presented in this chapter as well as these presented possibilities (future work) are already incorporated into the filled patent application [27].

Classification of Points of Interest and Road Types. Another future work is to enrich the HD map data with classified and detected points-of-interest (POI) such as roundabouts or zebra crossing and classified road types such as highway, rural road or city, both based on aggregated map data. Examples of road types are illustrated in figure 5.9. The knowledge of POI or road types is used in ADAS such as intelligent speed control. In this example, the information of a roundabout in a few hundred meters w.r.t. to the actual location of the vehicle, provides the system the information to decrease the velocity. A lot of use cases in which such information about ROI or road types are useful and make AD safer are obvious. The idea of POI and road type classification based on map aggregation is incorporated into the filled patent application [28] and is as follows.

It is assumed that 3D object detection, e.g., with radar sensor can be employed to get the trails of multiple classes such as vehicles (v) and pedestrians (p) in GCS using a GPS. Then those trails are aggregated as described in the beginning of section 5.3.2 but per class c, i.e., $\mathbf{D}^v, \mathbf{D}^p$, with a lower grid resolution e.g., $2 \times 2\,\mathrm{m}^2$ (instead of $10 \times 10\,\mathrm{cm}^2$) and in more than four direction density maps, e.g., $n_d = 8$

$$\mathbf{D}_d^c = \left[\frac{d-1}{n_d} * 360, \frac{d}{n_d} * 360 \right), \; d = 1, \ldots, n_d, \; c \in \{v, p\}\,.$$

The direction density maps are normalized to $[0, 1]$ w.r.t. the number of recordings. Anolougysly, velocity maps $\mathbf{V}_d^c$ are created by taking the average velocity per grid cell. An example for the aggregated data in terms of ratio and average velocity of vehicles and pedestrians per cell are shown in figure 5.10. Stacking the data in this particular example yields a $4 \times 8 = 32$ dimensional feature vector per cell. A region of $m \times n$ cells is then represented as an aggregated feature map $\mathbf{F} \in \mathbb{R}^{m \times n \times 32}$. Based on that, the idea is to train a CNN that learns to predict the presence of these POIs and the road types. Each cell can be classified or POIs and road types can be detected.

The following examples motivate the assumption that this approach would work; therefore see figure 5.9. For a rural road (b), one would only expect a single trail

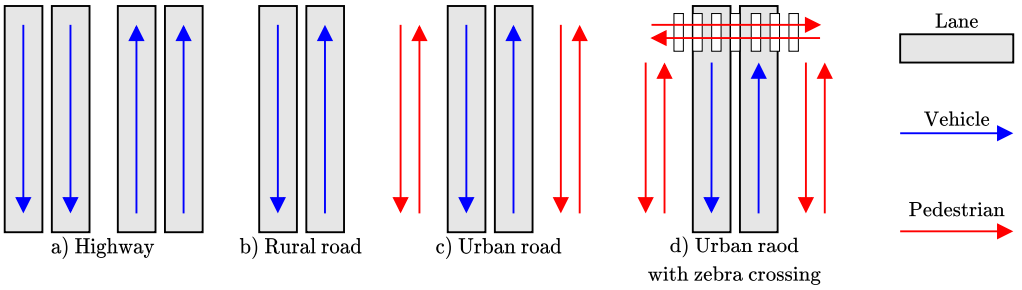
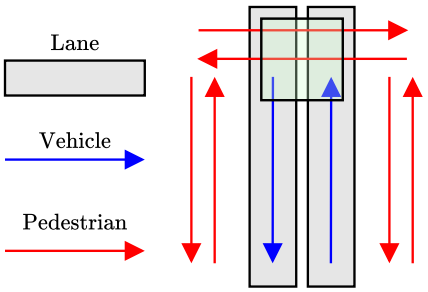


Figure 5.9: Four examples of road types. A highway (a) has generally at least two lanes in each direction and no sidewalks. Rural roads (b) have mostly only one lane in each direction, and sidewalks for pedestrians mostly do not exist. In urban roads (c), (d) pedestrians and sidewalks are assumed to be present. Furthermore, zebra crossings (POI) as depicted in d) can occur.



Direction	↑	↗	→	↘	↓	↙	←	↖
Ratio vehicle	0.5	0	0	0	0.7	0	0	0
Velocity vehicle	50 km/h	0	0	0	45 km/h	0	0	0
Ratio pedestrian	0	0	0.3	0	0	0	0.4	0
Velocity pedestrian	0	0	5 km/h	0	0	0	4 km/h	0

Figure 5.10: The highlighted region in green (center) represents a grid cell. The corresponding aggregated features ratio and average velocity for vehicles and pedestrians for eight discretized directions are shown right.

going in each direction. Also, the distance between both directions will be smaller compared to a highway scenario (a). For a city road, additional movement patterns for pedestrians will likely occur, see d) and figure 5.10. Moreover, the aggregated velocity will provide a clear indicator. The average velocity on highways and rural roads is generally significantly higher than in the city.

Lane Map Aggregation from Lane Markings and Trails. An other idea to improve the lane estimates is to take detected lane markings into account and combine this information with trails, see figure 5.11.

Trails are based on dynamic objects, i.e., moving vehicles. The higher the traffic is, the more trails will be given. Lane markings are static objects which might be hard to detect in high traffic. However, once lane markings are detected they define the lane boundaries, what is actually missing in the HD lane map generation process. Estimating lanes based on trails and lane markings is a promising approach to improve the presented procedure. The following description focuses on two slightly

different approaches. In the first one, a *cross-check* is focused, in which the lane information coming from vehicles (trails) and lane markings is compared. This is already claimed in the filled patent application [25]. The second approach deals with a *low-cost sensor system* and proposes a method to get reliable lane information from lane markings and trails in the GCS. The second approach is claimed in the filled patent application [26].

In general, both ideas follow a similar procedure, see figure 5.11. Sensor data is recorded with one or more recording vehicles. The sensor data should allow performing lane marking detection e.g., with camera or lidar data and to perform object detection – here vehicles – e.g., with camera, lidar or radar data. Furthermore, a localization system is required. Based on the detected lane markings and vehicles for both, tracking and aggregation is applied. This yields lane estimates from lane markings and trails in GCS. In the next part, both estimates are fused and cross-checked to output reliable lane estimates.

The main idea of [25] is the cross-check part to get drivable lanes. The lane marking-based lanes indicate where lanes are according to the available information about lane markings. Lane markings do not necessarily have to coincide with drivable lanes. In many situations, lane markings could be visible, but the lane would still not be drivable because there is an obstacle, a construction area, or a prohibition to use the lane. The trail-based lanes give indication on where other road users have driven and thereby a hint on which lane might actually be usable. However, the detection of the lanes itself based on trails might not be as reliable. By combining the two methods, one can receive the accurate position of lanes by lane markings together with the information whether these lanes can actually be used. In addition, the lane estimates can be overall made more robust by combining a lane marking detection algorithm based on one sensor with a trail detection algorithm based on another sensor with another working principle. For example, the lane marking detection could be based on camera while the trail detection could be based on lidar. Those sensors have different failure modes. When this is combined in the described system, reliable lanes can be obtained in most circumstances.

The main idea of [26] is to create reliable lanes with a low-cost sensor system such as camera for lane marking detection, radar for object detection and a normal GPS for localization. Reliable lane estimates from low-cost sensors with potentially noisy data include following physical checks. Physical checks for trails: jitter introduced from the sub-optimal localization system can be filtered out when aggregating the trails in the map. This is done by making reasonable physical assumptions about the driving behavior of other vehicles, such as maximum accelerations, braking's, yaw

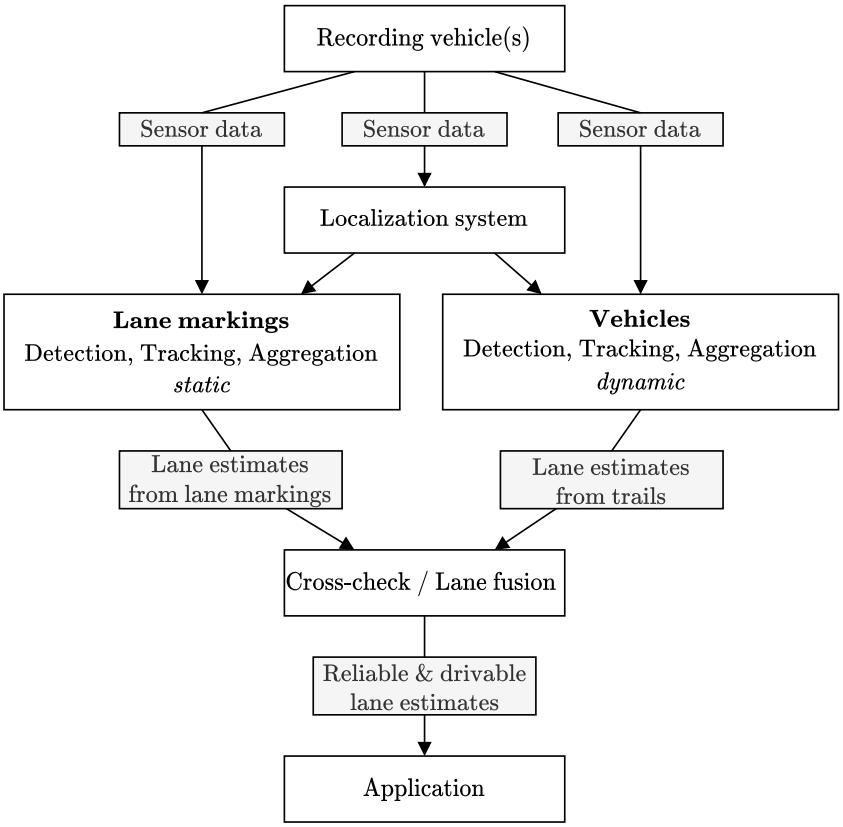


Figure 5.11: Diagram of getting reliable and drivable lanes from lane markings and trails by aggregating detections of lane markings and vehicles. Both lane estimates are cross-checked / fused to get reliable lanes.

rates and similar. Additionally, the trails of other vehicles coming out of the detection algorithm can be first transformed into the GCS (using the simple localization info available) and then tracked in this coordinate frame using a tracker. The use of the tracker after the data is transformed in the global map coordinate system smooths and reduces the introduced jitter. Physical checks for lane markings: again, jitter is filtered out from the localization system when aggregating the lane markings in GCS. Reasonable physical assumptions are made about the driving behavior of the ego-vehicle (using information such as ego-vehicle velocity and yaw rate). This allows to propagate the lane markings using a tracker to the expected next position and thereby reduce the introduces jitter. Note that the above proposed tracking is different from a simple tracking, as the physical constraints are applied on the ensemble of trails, i.e., by aggregation, rather than individual trajectories. This is different from just applying a tracker with an underlying physical model for each tracked object. The same applies for the aggregation of lane markings, where again physical sanity checks are applied on the ensemble of lane markings to aggregate, in addition to any tracking of individual lane markers.

6 Conclusion & Outlook

In this thesis, uncertainty quantification and its applications for multimodal semantic segmentation have been presented. In particular, this includes uncertainty quantification and its applications in terms of active learning for semantic segmentation of images and uncertainty quantification for lidar point clouds as well the application of uncertainty quantification for 3D object detection to create semantic HD maps.

First, a new region-based active learning method for image semantic segmentation was presented. This new method queries image regions by means of priority maps consisting of priority scores that are based on the prediction and probabilities of the images. The first priority map and selection criterion is the segment-wise prediction quality estimation of unlabeled inferred images to identify poorly predicted image regions. By these means and compared to related work – which is commonly based on uncertainty quantities such as entropy – the information about what the segmentation model is estimated to predict correctly and especially what it is estimated to predicted falsely is used. An other priority map consists of a cost estimation. In general, the annotation effort to label data for semantic segmentation of images is high, and it was shown that measuring the costs in terms of annotation clicks is a reasonable approximation of true human annotation effort. To this end, a practical cost estimation approach was introduced that is based on the borders of the segmentation masks. In numerical experiments with two different segmentation models, the new proposed AL method MetaBox+ were evaluated and compared to baseline methods that are based on the entropy with and without a combined cost estimation as well as randomly picking image regions for labeling. By applying MetaBox+, 95% of the full set $mIoU$ was achieved with annotation costs of only 32.01% for FCN8 and 10.47% for Deeplab. Combined to the baseline methods, MetaBox+ showed the best results in terms of required costs to achieve 95% of the full set $mIoU$.

In the second part of this thesis, a new method for uncertainty quantification on predicted segment level for lidar point cloud segmentation was presented. In particular, the new method called LidarMetaSeg detects false positive segments and provides a segment-wise prediction quality estimation in terms of segment-wise IoU. LidarMetaSeg is an adaptation and extension of MetaSeg to lidar point cloud

segmentation. LidarMetaSeg first projects the point cloud data meaning the point cloud itself, the ground truth, the prediction and the probabilities to a 2D image representation. With some adjustments w.r.t. to the point cloud representation and the enrichment of feature measures in addition to dispersion measures that are based on input point cloud features, MetaSeg can be applied. In numerical experiments, the performance of false positive detection and prediction quality estimation were shown. To this end, three different networks and two data sets were used. For meta classification and meta regression, values up to $AUROC = 91.16\%$ and $R^2 = 66.13\%$ were achieved, respectively. Furthermore, a class-wise and category-wise evaluation were presented that clearly show a correlation between predicted class and category distributions and the meta regression performance.

The last presented method in this thesis is a procedure to create HD maps. The underlying method does not process image or lidar point cloud segmentation as the previous here presented methods. But it uses DL object detection and tracking methods from lidar point cloud data in order to create HD maps, which can be seen as a segmented fine-grained grid map. The procedure aggregates detected and tracked road users as well as the driving path of the ego vehicle from multiple recordings in a map representation. In order to reduce possible detection errors in the aggregation, uncertainty measures were taken into account. Furthermore, a high number of observation in terms of trails correlate with the true lane. Thus, by a high number of trails, uncertainties are reduced. In the second part of the procedure for generating HD maps, the lane centers are extracted from the aggregated data. The resulting HD map contains lane centers. The procedure was evaluated on an internal data set. In terms of a trade-off between precision and recall, 77% of all lanes were detected with a precision of 95%. Several ideas how to improve and enrich the map aggregation system were presented. However, these ideas have not been tested yet, but promising results are expected here.

In conclusion, the respective results and findings of the presented methods of uncertainty quantification and its applications motivate suggestions and ideas for improvement and further development of the respective methods. A class-wise (or category-wise) evaluation of MetaSeg similar to what were presented for LidarMetaSeg can be incorporated to the priority maps or will be helpful to the outlined semi-supervised approach. In the mentioned semi-supervised approach, it sounds reasonable to pseudo-label segments with a high predicted IoU. This requires not only a powerful meta regression model, but also a knowledge about the class-wise performance.

Apart from that, an active learning method for lidar point cloud segmentation is of interest. The incorporation of the prediction quality estimation into the presented AL method MetaBox+ has shown remarkable results. Therefore, incorporating the estimated prediction quality estimation via LidarMetaSeg into an AL method for lidar data is expected to yield promising results.

In the HD map generation, only object detection and tracking is used. In future work, segmented lidar point clouds could be incorporated. Furthermore, in this scenario, the aggregation of data can depend on the meta classification and regression information e.g., only segments with a predicted IoU over given threshold will be aggregated. Aggregating reliable road segments (using LidarMetaSeg) provides a prior knowledge about the streets and lanes. Object detection networks are not trained on predicting the street, since it is not an obvious object. Moreover, the segmented point clouds can be used to enrich the HD map with further classes e.g., static classes like buildings or sidewalks. On the other hand, the provided HD map data can be incorporated into detection and segmentation models. This incorporation provides prior knowledge and might improve the accuracy.

List of Figures

List of Tables

List of Algorithms

List of Notations

Throughout this thesis, scalars, parameters or variables representing a single value are denoted by lower-case or upper-case letters e.g., $a,\ A$. Vectors or matrices are given in bold face e.g., $\mathbf{a},\ \mathbf{A}$. Unless otherwise indicated, the notation is as follows.

Fundamentals and general

f	neural network (function)
$\ell,\ L$	loss function
η	learning rate
σ	(sigmoid) activation function
$J_{\mathbf{w}}(\mathbf{f})$	Jacobian matrix of function $\mathbf{f}$ at $\mathbf{w}$
$\frac{\partial f_i}{\partial w_j}$	partial derivative of f_i w.r.t. to variable w_j
$\mathbf{x}$	input sample
$\mathbf{y}$	label of sample $\mathbf{x}$
$\hat{\mathbf{y}}^{prob}$	probabilities
$\hat{\mathbf{y}}$	prediction
$\mathbf{w}$	weights of a neural network
$\mathbf{p}$	point cloud
z	pixel $z = (i, j)$
w	width
h	height
$\mathcal{C}$	set of classes or label space
c	class index
q	number of classes

Region-based active learning

ζ	prediction quality map
ξ	cost map
$\mathcal{P}$	data pool
$\mathcal{L}$	labeled data
$\mathcal{U}$	unlabeled data
$\mathcal{M}$	data for MetaSeg
$cost_A$	click-based cost metric A
$cost_B$	click-based cost metric B
$cost_P$	pixel-based cost metric

LidarMetaSeg

ψ	vertical field of view (lidar sensor)
ϕ	horizontal field of view (lidar sensor)
$\mathbf{E}$	entropy heatmap
$\mathbf{D}$	probability difference heatmap
$\mathbf{V}$	variation ratio heatmap
$\mathbf{F}$	feature measure heatmap
$\mathcal{M}$	set of dispersion and feature measures
$\hat{\mathcal{K}}_{\mathbf{x}}$	set of predicted segments / connected components
$\mathcal{K}_{\mathbf{x}}$	set of ground truth segments / connected components
MCE	mean calibration error
ECE	expected calibration error

HD map generation

$\mathbf{D}$	density map
$s(\mathbf{D})$	smoothed density map
d	direction of density map
MAE	mean absolute error
$RMSE$	root mean squared error

Evaluation metrics

TP	number of true positives
FP	number of false positives
TN	number of true negatives
FN	number of false negatives
ACC	accuracy
$AUROC$	area under receiver operating curve
$AUPRC$	area under precision recall curve
R^2	coefficient of determination
AP	average precision
mAP	mean average precision
$MOTA$	multiple object tracking accuracy
$MOTP$	multiple object tracking precision
IoU	intersection over union
$mIoU$	mean intersection over union
IoU_{adj}	adjusted intersection over union
IoU_{pred}	predicted IoU per predicted segment
IoU_{true}	true IoU per predicted segment

List of Abbreviations

AD	automated driving
ADAS	advanced driver assistant systems
AI	artificial intelligence
AL	active learning
ASPP	atrous spatial pyramid pooling
BN	batch normalization
CNN	convolutional neural network
DL	deep learning
EM waves	electromagnetic waves
ERM	empirical risk minimization
FCN	fully convolutional network
FOV	field of view
GPS	global positioning system
HD map	high definition map
KL	Kullback–Leibler divergence
kNN	k-nearest neighbor
lidar	light detection and ranging
MC dropout	Monte Carlo Dropout
ML	machine learning
MLP	multilayer perceptron
NN	(artificial) neural network
OTA	over-the-air
radar	radio detection and ranging
RDP algorithm	Ramer-Douglas-Peucker algorithm
ReLU	rectified linear unit
RGB image	red-green-blue image
SD map	standard definition map
SGD	stochastic gradient descent
SP	superpixel
UQ	uncertainty quantification

Bibliography

[1] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, et al. “A review of uncertainty quantification in deep learning: Techniques, applications and challenges”. In: *Information Fusion* 76 (2021), pp. 243–297 (cit. on pp. 1, 41).

[2] John Aldrich. “RA Fisher and the making of maximum likelihood 1912-1922”. In: *Statistical science* 12.3 (1997), pp. 162–176 (cit. on p. 13).

[3] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. “Semantickitti: A dataset for semantic scene understanding of lidar sequences”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9297–9307 (cit. on pp. 75, 81, 94).

[4] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. “The power of ensembles for active learning in image classification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9368–9377 (cit. on pp. 2, 52).

[5] Philipp Bender, Julius Ziegler, and Christoph Stiller. “Lanelets: Efficient map representation for autonomous driving”. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE. 2014, pp. 420–425 (cit. on pp. 3, 8).

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166 (cit. on p. 29).

[7] Keni Bernardin and Rainer Stiefelhagen. “Evaluating multiple object tracking performance: the clear mot metrics”. In: *EURASIP Journal on Image and Video Processing* 2008 (2008), pp. 1–10 (cit. on p. 39).

[8] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. “Understanding batch normalization”. In: *Advances in neural information processing systems* 31 (2018) (cit. on p. 29).

[9] Vanessa Buhrmester, David Münch, and Michael Arens. “Analysis of explainers of black box deep neural networks for computer vision: A survey”. In: *Machine Learning and Knowledge Extraction* 3.4 (2021), pp. 966–989 (cit. on p. 1).

[10] Holger Caesar, Varun Bankiti, Alex H Lang, et al. “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631 (cit. on pp. 75, 81, 84, 94, 107).

[11] Lili Cao and John Krumm. “From GPS traces to a routable road map”. In: *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. 2009, pp. 3–12 (cit. on pp. 97, 98).

[12] Arantxa Casanova, Pedro O Pinheiro, Negar Rostamzadeh, and Christopher J Pal. "Reinforced active learning for image segmentation". In: *arXiv preprint arXiv:2002.06583* (2020) (cit. on pp. 52, 53).

[13] Robin Chan, Matthias Rottmann, Fabian Hüger, Peter Schlicht, and Hanno Gottschalk. "Controlled False Negative Reduction of Minority Classes in Semantic Segmentation". In: 2020 (cit. on pp. 41, 71, 76, 77).

[14] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, et al. "Argoverse: 3d tracking and forecasting with rich maps". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8748–8757 (cit. on p. 107).

[15] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848 (cit. on pp. 26, 28).

[16] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Semantic image segmentation with deep convolutional nets and fully connected crfs". In: *arXiv preprint arXiv:1412.7062* (2014) (cit. on pp. 23, 26).

[17] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587* (2017) (cit. on p. 26).

[18] Liang-Chieh Chen, Huiyu Wang, and Siyuan Qiao. "Scaling Wide Residual Networks for Panoptic Segmentation". In: *arXiv preprint arXiv:2011.11675* (2020) (cit. on p. 25).

[19] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *CoRR* abs/1802.02611 (2018) (cit. on pp. 26, 49, 51, 62, 63).

[20] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794 (cit. on pp. 44, 45).

[21] Siddhartha Chib. "Markov chain Monte Carlo methods: computation and inference". In: *Handbook of econometrics* 5 (2001), pp. 3569–3649 (cit. on p. 98).

[22] Jung Uk Cho, Seung Hun Jin, Xuan Dai Pham, Jae Wook Jeon, Jong Eun Byun, and Hoon Kang. "A real-time object tracking system using a particle filter". In: *2006 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2006, pp. 2822–2827 (cit. on p. 37).

[23] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 1800–1807 (cit. on pp. 27, 62).

[24] Pascal Colling. "Zuverlässigkeitsmaße für Neuronale Netze in der Bildsegmentierung". MA thesis. University of Wuppertal, Department of Mathematics and Informatics, 2017 (cit. on pp. 2, 41).

[25] Pascal Colling and Peet Cremer. *Methods and Systems for Estimating Lanes for a Vehicle (Cross-Checking)*. European Patent Application EP21200261.2, 2021 (cit. on p. 116).

[26] Pascal Colling and Peet Cremer. *Methods and Systems for Estimating Lanes for a Vehicle (Low-Cost Sensor System)*. European Patent Application EP21200267.9, 2021 (cit. on p. 116).

[27] Pascal Colling, Dennis Müller, and Lutz Roese-Koerner. *Method for Generating High Definition Maps, and Cloud Server and Vehicle*. European Patent Application GB2201080.5, 2022 (cit. on p. 114).

[28] Pascal Colling, Dennis Müller, Lutz Roese-Koerner, and Christian Nunn. *"Method of determining a point of interest and/or a road type in a map, and related cloud server and vehicle*. European Patent Application GB2201078.9, 2022 (cit. on p. 114).

[29] Marius Cordts, Mohamed Omran, Sebastian Ramos, et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 50, 51, 58, 62).

[30] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. "SalsaNext: Fast, Uncertainty-aware Semantic Segmentation of LiDAR Point Clouds for Autonomous Driving". In: *arXiv preprint arXiv:2003.03653* (2020) (cit. on pp. 3, 28, 31, 73, 75, 76, 81, 86).

[31] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314 (cit. on p. 19).

[32] Ido Dagan and Sean P Engelson. "Committee-based sampling for training probabilistic classifiers". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 150–157 (cit. on p. 53).

[33] Andreas Danzer, Thomas Griebel, Martin Bach, and Klaus Dietmayer. "2D Car Detection in Radar Data with PointNets". In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*. IEEE, 2019, pp. 61–66 (cit. on p. 96).

[34] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240 (cit. on p. 46).

[35] Bert De Brabandere, Davy Neven, and Luc Van Gool. "Semantic instance segmentation with a discriminative loss function". In: *arXiv preprint arXiv:1708.02551* (2017) (cit. on p. 3).

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cit. on pp. 1, 63).

[37] Juergen Dickmann, Jens Klappstein, Markus Hahn, Nils Appenrodt, Hans-Ludwig Bloecher, Klaudius Werber, and Alfons Sailer. "Automotive radar the key technology for autonomous driving: From detection and ranging to environmental understanding". In: *2016 IEEE Radar Conference (RadarConf)*. IEEE. 2016, pp. 1–6 (cit. on p. 8).

[38] David H Douglas and Thomas K Peucker. “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature”. In: *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973), pp. 112–122 (cit. on pp. 58, 105).

[39] Maria Dreher, Emec Ercelik, Timo Bänziger, and Alois C. Knoll. “Radar-based 2D Car Detection Using Deep Neural Networks”. In: *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*. IEEE, 2020, pp. 1–8 (cit. on p. 96).

[40] Vincent Dumoulin and Francesco Visin. “A guide to convolution arithmetic for deep learning”. In: *arXiv preprint arXiv:1603.07285* (2016) (cit. on pp. 21, 23, 24).

[41] Di Feng, Lars Rosenbaum, and Klaus Dietmayer. “Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 3266–3273 (cit. on p. 76).

[42] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001 (cit. on p. 44).

[43] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232 (cit. on p. 44).

[44] Jerome H Friedman. “Stochastic gradient boosting”. In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378 (cit. on pp. 44, 45).

[45] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059 (cit. on p. 76).

[46] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. “Deep Bayesian Active Learning with Image Data”. In: *CoRR* abs/1703.02910 (2017). arXiv: 1703.02910 (cit. on pp. 2, 52).

[47] Jochen Gast and Stefan Roth. “Lightweight probabilistic deep networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3369–3378 (cit. on p. 76).

[48] Hossein Gholamalinezhad and Hossein Khosravi. “Pooling methods in deep neural networks, a review”. In: *arXiv preprint arXiv:2009.07485* (2020) (cit. on p. 23).

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on p. 21).

[50] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint* (2014) (cit. on p. 53).

[51] Marc Gorriz, Axel Carlier, Emmanuel Faure, and Xavier Giro-i Nieto. “Cost-effective active learning for melanoma segmentation”. In: *arXiv preprint arXiv:1711.09168* (2017) (cit. on p. 52).

[52] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. “On calibration of modern neural networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1321–1330 (cit. on p. 92).

[53] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. “Deep learning for 3d point clouds: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* (2020) (cit. on p. 75).

[54] Lukas Hahn, Lutz Roese-Koerner, Peet Cremer, Urs Zimmermann, Ori Maoz, and Anton Kummert. “On the Robustness of Active Learning”. In: *EPiC Series in Computing* 65 (2019), pp. 152–162 (cit. on pp. 2, 52).

[55] Frederik Hasecke, Martin Alsfasser, and Anton Kummert. “What Can be Seen is What You Get: Structure Aware Point Cloud Augmentation”. In: *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2022 (cit. on p. 94).

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 29).

[57] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. “Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 41–50 (cit. on pp. 11, 53).

[58] Dan Hendrycks and Kevin Gimpel. “A baseline for detecting misclassified and out-of-distribution examples in neural networks”. In: *arXiv preprint arXiv:1610.02136* (2016) (cit. on p. 2).

[59] Namdar Homayounfar, Wei-Chiu Ma, Justin Liang, Xinyu Wu, Jack Fan, and Raquel Urtasun. “Dagmapper: Learning to map by discovering lane topology”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2911–2920 (cit. on pp. 3, 97, 98).

[60] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on p. 19).

[61] John Houston, Guido Zuidhof, Luca Bergamini, et al. “One thousand and one hours: Self-driving motion prediction dataset”. In: *arXiv preprint arXiv:2006.14480* (2020) (cit. on p. 107).

[62] Eyke Hüllermeier and Willem Waegeman. “Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods”. In: *Machine Learning* 110.3 (2021), pp. 457–506 (cit. on pp. 1, 40, 41).

[63] Paul Jaccard. “The distribution of the flora in the alpine zone. 1”. In: *New phytologist* 11.2 (1912), pp. 37–50 (cit. on pp. 2, 37, 73).

[64] Suyog Dutt Jain and Kristen Grauman. “Active image segmentation propagation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2864–2873 (cit. on p. 52).

[65] Felipe Jiménez, José Eugenio Naranjo, José Javier Anaya, Fernando García, Aurelio Ponz, and José María Armingol. “Advanced driver assistance system for road environments to improve safety and efficiency”. In: *Transportation research procedia* 14 (2016), pp. 2245–2254 (cit. on p. 1).

[66] Tejaswi Kasarla, Gattigorla Nagendar, Guruprasad M Hegde, Vineeth Balasubramanian, and CV Jawahar. “Region-based active learning for efficient labeling in semantic segmentation”. In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 1109–1117 (cit. on pp. 52, 53).

[67] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. “Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding”. In: *arXiv preprint arXiv:1511.02680* (2015) (cit. on p. 76).

[68] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. “Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding”. In: *CoRR* abs/1511.02680 (2015). arXiv: 1511.02680 (cit. on p. 2).

[69] Kitae Kim, Soohyun Cho, and Woojin Chung. “Hd map update for autonomous driving with crowdsourced data”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1895–1901 (cit. on p. 98).

[70] Youngjoo Kim and Hyochoong Bang. “Introduction to Kalman filter and its applications”. In: *Introduction and Implementations of the Kalman Filter, F. Govaers, Ed. IntechOpen* (2019) (cit. on pp. 33, 36, 37, 107).

[71] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980 (cit. on p. 63).

[72] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. “Introducing geometry in active learning for image segmentation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2974–2982 (cit. on p. 52).

[73] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012) (cit. on p. 1).

[74] Anders Krogh and John Hertz. “A simple weight decay can improve generalization”. In: *Advances in neural information processing systems* 4 (1991) (cit. on p. 20).

[75] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. “Pointpillars: Fast encoders for object detection from point clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12697–12705 (cit. on pp. 33, 96, 107).

[76] Yann LeCun et al. “Generalization and network design strategies”. In: *Connectionism in perspective* 19.143-155 (1989), p. 18 (cit. on p. 21).

[77] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6 (1993), pp. 861–867 (cit. on p. 19).

[78] Justin Liang, Namdar Homayounfar, Wei-Chiu Ma, Yuwen Xiong, Rui Hu, and Raquel Urtasun. "Polytransform: Deep polygon transformer for instance segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9131–9140 (cit. on p. 25).

[79] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. "Sparse convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 806–814 (cit. on p. 35).

[80] Shujian Liu, Haibo Zhou, Chenming Li, and Shuo Wang. "Analysis of Anchor-Based and Anchor-Free Object Detection Methods Based on Deep Learning". In: *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE. 2020, pp. 1058–1065 (cit. on p. 35).

[81] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37 (cit. on p. 34).

[82] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on pp. 25, 51, 62).

[83] Kira Maag, Matthias Rottmann, and Hanno Gottschalk. "Time-Dynamic Estimates of the Reliability of Deep Semantic Segmentation Networks". In: *2020 IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. 2020 (cit. on pp. 41, 44, 71, 76).

[84] Radek Mackowiak, Philip Lenz, Omair Ghori, Ferran Diego, Oliver Lange, and Carsten Rother. "CEREALS - Cost-Effective REgion-based Active Learning for Semantic Segmentation". In: *CoRR* abs/1810.09726 (2018). arXiv: 1810.09726 (cit. on pp. 52, 53, 60, 62, 64).

[85] Dwarikanath Mahapatra, Behzad Bozorgtabar, Jean-Philippe Thiran, and Mauricio Reyes. "Efficient Active Learning for Image Classification and Segmentation using a Sample Selection and Conditional Generative Adversarial Network". In: *CoRR* abs/1806.05473 (2018). arXiv: 1806.05473 (cit. on p. 52).

[86] Seyed Mojtaba Marvasti-Zadeh, Li Cheng, Hossein Ghanei-Yakhdan, and Shohreh Kasaei. "Deep learning for visual tracking: A comprehensive survey". In: *IEEE Transactions on Intelligent Transportation Systems* (2021) (cit. on p. 37).

[87] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955". In: *AI magazine* 27.4 (2006), pp. 12–12 (cit. on p. 1).

[88] Annika Meyer, Jonas Walter, and Martin Lauer. "Fast Lane-Level Intersection Estimation using Markov Chain Monte Carlo Sampling and B-Spline Refinement". In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2020, pp. 71–76 (cit. on pp. 3, 98).

[89] Annika Meyer, Jonas Walter, Martin Lauer, and Christoph Stiller. "Anytime lane-level intersection estimation based on trajectories of other traffic participants". In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 3122–3129 (cit. on pp. 3, 98).

[90] Andres Milioto, Jens Behley, Chris McCool, and Cyrill Stachniss. “Lidar panoptic segmentation for autonomous driving”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 8505–8512 (cit. on p. 89).

[91] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. “Rangenet++: Fast and accurate lidar semantic segmentation”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 4213–4220 (cit. on pp. 3, 28, 30, 73, 75, 81, 82).

[92] Kazuki Minemura, Hengfui Liau, Abraham Monrroy, and Shinpei Kato. “LMNet: Real-time multiclass object detection on CPU using 3D LiDAR”. In: *2018 3rd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*. IEEE. 2018, pp. 28–34 (cit. on p. 96).

[93] Mobileye. *Mobileye REM ™*. `https://www.mobileye.com/our-technology/rem/`. Accessed: 2022-04-20 (cit. on p. 113).

[94] Rohit Mohan and Abhinav Valada. “Efficientps: Efficient panoptic segmentation”. In: *International Journal of Computer Vision* 129.5 (2021), pp. 1551–1579 (cit. on p. 25).

[95] Agata Mosinska, Jakub Tarnawski, and Pascal Fua. “Active learning and proofreading for delineation of curvilinear structures”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2017, pp. 165–173 (cit. on p. 52).

[96] Alexander Neubeck and Luc Van Gool. “Efficient non-maximum suppression”. In: *18th International Conference on Pattern Recognition (ICPR’06)*. Vol. 3. IEEE. 2006, pp. 850–855 (cit. on p. 34).

[97] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, and Peter Kontschieder. “The mapillary vistas dataset for semantic understanding of street scenes”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 4990–4999 (cit. on pp. 50, 58).

[98] Davy Neven, Bert De Brabandere, Marc Proesmans, and Luc Van Gool. “Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8837–8845 (cit. on p. 25).

[99] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning deconvolution network for semantic segmentation”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1520–1528 (cit. on p. 24).

[100] Firat Özdemir, Zixuan Peng, Christine Tanner, Philipp Fürnstahl, and Orcun Goksel. “Active Learning for Segmentation by Optimizing Content Information for Maximal Entropy”. In: *CoRR* abs/1807.06962 (2018). arXiv: `1807.06962` (cit. on p. 52).

[101] Onur Ozdemir, Benjamin Woodward, and Andrew A Berlin. “Propagating uncertainty in multi-stage bayesian convolutional neural networks with application to pulmonary nodule detection”. In: *arXiv preprint arXiv:1712.00497* (2017) (cit. on pp. 25, 76).

[102] Rafael Padilla, Sergio L Netto, and Eduardo AB Da Silva. “A survey on performance metrics for object-detection algorithms”. In: *2020 international conference on systems, signals and image processing (IWSSIP)*. IEEE. 2020, pp. 237–242 (cit. on p. 38).

[103] Sankar K Pal, Anima Pramanik, Jhareswar Maiti, and Pabitra Mitra. “Deep learning in multi-object detection and tracking: state of the art”. In: *Applied Intelligence* 51.9 (2021), pp. 6400–6429 (cit. on p. 37).

[104] Yancheng Pan, Fan Xie, and Huijing Zhao. *Understanding the Challenges When 3D Semantic Segmentation Faces Class Imbalanced and OOD Data*. 2022. arXiv: 2203.00214 [cs.CV] (cit. on p. 88).

[105] David Pannen, Martin Liebner, Wolfgang Hempel, and Wolfram Burgard. “How to keep HD maps for automated driving up to date”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 2288–2294 (cit. on p. 98).

[106] Fabian Poggenhans, Jan-Hendrik Pauls, Johannes Janosovits, Stefan Orf, Maximilian Naumann, Florian Kuhnt, and Matthias Mayr. “Lanelet2: A high-definition map framework for the future of automated driving”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 1672–1679 (cit. on pp. 3, 8).

[107] Lutz Prechelt. “Automatic early stopping using cross validation: quantifying the criteria”. In: *Neural Networks* 11.4 (1998), pp. 761–767 (cit. on p. 19).

[108] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660 (cit. on pp. 3, 32, 34, 73).

[109] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *arXiv preprint arXiv:1706.02413* (2017) (cit. on pp. 3, 75).

[110] Urs Ramer. “An iterative procedure for the polygonal approximation of plane curves”. In: *Computer graphics and image processing* 1.3 (1972), pp. 244–256 (cit. on pp. 58, 105).

[111] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018) (cit. on p. 29).

[112] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241 (cit. on p. 49).

[113] Matthias Rottmann, Pascal Colling, Thomas-Paul Hack, Robin Chan, Fabian Hüger, Peter Schlicht, and Hanno Gottschalk. “Prediction error meta classification in semantic segmentation: Detection via aggregated dispersion measures of softmax probabilities”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–9 (cit. on pp. 2, 41, 44, 46, 64, 76, 77, 82).

[114] Matthias Rottmann, Karsten Kahl, and Hanno Gottschalk. “Deep bayesian active semi-supervised learning”. In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2018, pp. 158–164 (cit. on pp. 2, 52).

[115] Matthias Rottmann and Marius Schubert. “Uncertainty measures and prediction quality rating for the semantic segmentation of nested multi resolution street scene images”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 0–0 (cit. on pp. 41, 71, 76, 77).

[116] Abhijit Guha Roy, Sailesh Conjeti, Nassir Navab, and Christian Wachinger. “Inherent brain segmentation quality control from fully convnet monte carlo sampling”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2018, pp. 664–672 (cit. on p. 76).

[117] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. “LabelMe: a database and web-based tool for image annotation”. In: *International journal of computer vision* 77.1-3 (2008), pp. 157–173 (cit. on p. 50).

[118] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. “Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation”. In: *CoRR* abs/1801.04381 (2018) (cit. on pp. 27, 62).

[119] Warren S Sarle. “Neural networks and statistical models”. In: (1994) (cit. on p. 14).

[120] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009 (cit. on pp. 2, 50).

[121] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014 (cit. on p. 9).

[122] Claude Elwood Shannon. “A mathematical theory of communication”. In: vol. 5. 1. ACM New York, NY, USA, 2001, pp. 3–55 (cit. on pp. 14, 52, 53).

[123] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. “From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network”. In: *IEEE transactions on pattern analysis and machine intelligence* (2020) (cit. on pp. 33, 35, 107).

[124] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. “Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1874–1883 (cit. on p. 31).

[125] Connor Shorten and Taghi M Khoshgoftaar. “A survey on image data augmentation for deep learning”. In: *Journal of big data* 6.1 (2019), pp. 1–48 (cit. on p. 66).

[126] Ravid Shwartz-Ziv and Amitai Armon. “Tabular data: Deep learning is not all you need”. In: *Information Fusion* 81 (2022), pp. 84–90 (cit. on p. 44).

[127] Yawar Siddiqui, Julien Valentin, and Matthias Nießner. “Viewal: Active learning with viewpoint entropy for semantic segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9433–9443 (cit. on p. 52).

[128] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on p. 25).

[129] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958 (cit. on p. 20).

[130] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. “Revisiting unreasonable effectiveness of data in deep learning era”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 843–852 (cit. on p. 49).

[131] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, et al. “Scalability in perception for autonomous driving: Waymo open dataset”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2446–2454 (cit. on p. 107).

[132] Shunqiao Sun, Athina P Petropulu, and H Vincent Poor. “MIMO radar for advanced driver-assistance systems and autonomous driving: Advantages and challenges”. In: *IEEE Signal Processing Magazine* 37.4 (2020), pp. 98–117 (cit. on p. 8).

[133] Luliang Tang, Xue Yang, Zihan Kan, and Qingquan Li. “Lane-level road information mining from vehicle GPS trajectories based on naïve bayesian classification”. In: *ISPRS International Journal of Geo-Information* 4.4 (2015), pp. 2660–2680 (cit. on pp. 97, 98).

[134] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. “Kpconv: Flexible and deformable convolution for point clouds”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 6411–6420 (cit. on pp. 75, 76).

[135] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288 (cit. on p. 20).

[136] Larissa T Triess, David Peter, Christoph B Rist, and J Marius Zöllner. “Scan-based Semantic Segmentation of LiDAR Point Clouds: An Experimental Study”. In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2020, pp. 1116–1121 (cit. on p. 78).

[137] Alexander Vezhnevets, Joachim M Buhmann, and Vittorio Ferrari. “Active learning for semantic segmentation with expected change”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 3162–3169 (cit. on p. 52).

[138] Diego Vidaurre, Concha Bielza, and Pedro Larranaga. “A survey of L1 regression”. In: *International Statistical Review* 81.3 (2013), pp. 361–387 (cit. on p. 20).

[139] Jingdong Wang, Ke Sun, Tianheng Cheng, et al. “Deep High-Resolution Representation Learning for Visual Recognition”. In: *TPAMI* (2019) (cit. on p. 49).

[140] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. “Cost-effective active learning for deep image classification”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 27.12 (2016), pp. 2591–2600 (cit. on pp. 2, 52).

[141] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. “Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8445–8453 (cit. on p. 96).

[142] Zhangjing Wang, Yu Wu, and Qingqing Niu. “Multi-sensor fusion in automated driving: A survey”. In: *IEEE Access* 8 (2019), pp. 2847–2868 (cit. on p. 5).

[143] Greg Welch, Gary Bishop, et al. “An introduction to the Kalman filter”. In: (1995) (cit. on p. 36).

[144] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. “Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1887–1893 (cit. on p. 3).

[145] Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. “Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 1–19 (cit. on p. 75).

[146] Jianyun Xu, Ruixiang Zhang, Jian Dou, Yushi Zhu, Jie Sun, and Shiliang Pu. “RPVNet: A Deep and Efficient Range-Point-Voxel Fusion Network for LiDAR Point Cloud Segmentation”. In: *arXiv preprint arXiv:2103.12978* (2021) (cit. on pp. 3, 73, 76).

[147] Lin Yang, Yizhe Zhang, Jianxu Chen, Siyuan Zhang, and Danny Z Chen. “Suggestive annotation: A deep active learning framework for biomedical image segmentation”. In: (2017), pp. 399–407 (cit. on p. 52).

[148] Xue Ying. “An overview of overfitting and its solutions”. In: *Journal of Physics: Conference Series*. Vol. 1168. 2. IOP Publishing. 2019, p. 022022 (cit. on p. 19).

[149] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. “Pyramid scene parsing network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2881–2890 (cit. on p. 49).

[150] Ding-Xuan Zhou. “Universality of deep convolutional neural networks”. In: *Applied and computational harmonic analysis* 48.2 (2020), pp. 787–794 (cit. on p. 21).

[151] Hui Zhou, Xinge Zhu, Xiao Song, Yuexin Ma, Zhe Wang, Hongsheng Li, and Dahua Lin. “Cylinder3d: An effective 3d framework for driving-scene lidar semantic segmentation”. In: *arXiv preprint arXiv:2008.01550* (2020) (cit. on pp. 3, 28, 32, 73, 75, 76, 81).

[152] Yin Zhou and Oncel Tuzel. “Voxelnet: End-to-end learning for point cloud based 3d object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4490–4499 (cit. on p. 32).

[153] Jannik Zürn, Johan Vertens, and Wolfram Burgard. “Lane Graph Estimation for Scene Understanding in Urban Driving”. In: *arXiv preprint arXiv:2105.00195* (2021) (cit. on pp. 3, 97, 98).