UNIVERSITY OF APPLIED SCIENCES, DRESDEN

—

Semantic Lane Segmentation of LiDAR Point Clouds using Convolutional Neural Networks

MASTER THESIS

A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE

BY
CARL SCHWEDES

—

supervised by
Prof. Dr. Marco Block-Berlitz
Prof. Dr. Marco Hamann

June 17, 2020

Abstract

This approach focused to characterize the model for *semantic lane segmentation* of *LiDAR* point clouds by using *convolutional neural networks*. An introducing research shows state-of-the-art developments in semantic segmentation and outlines most recent developments by targeting demands of autonomous driving industry. *Perception algorithms* of the modern self-driving vehicle requires high *reliability* to system functionality. The use of modern *multimodal datasets* (such as nuScenes) enabled a detailed investigation for *ground-plane* segmentation. Under the use of extended map layers, *semantic classes* have been derived to specifically render *road semantics* and lane *traffic-direction* to point cloud information. For efficiently and effectively processing point cloud information, sensor data has been converted in *multi-channel range-image* representation (x,y,z,i,r,ϕ), by using convolutional neural network *U-Net* (showing state-of-the-art performance in semantic segmentation). Hereby, segmentation performance has been investigated within a detailed qualitative and quantitative study. The provided approach showed to be in the position to well distinguish between the defined semantic classes. Segmentation performance indicated, the neural network had been able to *derive descriptive features* from training data to correctly *decode road semantics* in predefined point labels. The suggested metric demonstrated to have the potential to provide *real-time validation* and integration in *sensor-fusion application* to further improve robustness of *autonomous driving*.

Acknowledgement

Firstly, special thanks goes to my professor Dr. Marco Block-Berlitz who enabled the possibility to write a master thesis with focus in the automotive industry in collaboration with HELLA Aglaia Mobile Vision GmbH.

Further, I like to thank my colleagues and supervisors from HELLA Aglaia, which supported me in plenty of ways throughout the time of this thesis. Further, special thanks goes to Dr. Hans-Arne Driescher, Philipp von Radziewsky, Eduard Feicho and Dennis Kroenninger from the deep learning group as well as to Matthias Wissing and all others. Thank you very much for all of your help and support throughout this time, and plenty of advices, which were mainly responsible to create and consistently improve this work.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

BRDF Bidirectional Reflectance Distributive Function.

CE Cross Entropy.

CNN Convolutional Neural Network.

FC Fully Connected Layer.

FCN Fully Convolutional Network.

FDR False Discovery Rate.

FN False Negative.

FOV Field of View.

FP False Positive.

GD Gradient Descent.

GPS Global Positioning System.

ILSVRC ImageNet Large Scale Visual Recognition Challenge.

IMU Inertial Measurement Unit.

IoU Intersection over Union.

IQR Inter Quartile Range.

kNN k-Nearest-Neighbour.

LiDAR Light Detection and Ranging.

MLP Multilayer-Perceptron.

PDF Probability Density Function.

R-CNN Region-CNN.

ROI Region of Interest.

RPN Region Proposal Network.

SGD Stochastic Gradient Descent.

TN True Negative.

TP True Positive.

TPR True Positive Rate.

1 Motivation and Introduction

Translating perception of environment into meaningful representations is a fundamental milestone of autonomous driving, where data from multiple of sensors need to be analysed in real-time. A system which does not depend on global data (maps, car2x, etc.) requires a complex sensor setup and high-level understanding of the surroundings to accurately navigate in urban scenarios [17, 1].

One of the key properties of modern imaging sensors is the fact that most of the desired information for environmental perception is not directly measured. Instead, complex algorithms are used to derive these data from raw representations (e.g. camera image or 3D point cloud). This introduces a new level of complexity at the sensor fusion stage where simple sensor characteristics (e.g. resolution, dynamic range, accuracy) do not directly translate to more complex perception data (e.g. existence probability of cars, pedestrians or reliability of free-space estimation). This is especially true for machine learning based algorithms, where it is currently not possible to derive these sensor characteristics from first principles. Therefore, the need for defining appropriate metrics to characterise the perception result arises.

Finding an appropriate metric to validate the performance of a perception system is a hard challenge. In context of vision-based semantic segmentation, small variations in environmental conditions can have crucial impact to classification performance [20]. The model needs to be proven to achieve acceptable performance under a variety of environmental conditions before it can be used in real-world applications [20]. For appropriately describing model performance, many approaches focus to use measures such as intersection over union (IoU) or other commonly used metrics, which are directly related to the labelled training data [17, 1]. In [20] a detailed evaluation of the model performance is seen under a variety of environmental conditions, where also the use of different network architectures is underlined. Several attempts describe applications for path proposals [13] and ego-lane segmentation, using camera [30, 11] and Light Detection and Ranging (LiDAR) [21] sensors but not addressing other aspects of ground layers in detail. Undertaking segmentation, other approaches focus to spatial relationships but not consider temporal or semantic information. [17] used consecutive frames as well as semantic information, such as occurrence of traffic signs or driving direction of vehicles, to perform semantic lane segmentation and showed that including temporal and semantic aspects could improve the decision making process, which is indicating that lane semantics are not only encapsulated in road parts.

Autonomous driving highly depends on analysis of multimodal data, the fusion of multiple sensor outputs. A single sensor has not the capabilities to be aware of all possible scenarios from diverse environments (e.g. urban scenarios or broad spectrum of varying environmental conditions).

Hereby, sensor fusion needs a more detailed description of the perception performance. A typical example is the performance degradation over distance for many perception algorithms. While the performance can be very good and reliable at close distance with many measurements (e.g. hundreds of LiDAR pings) on the object, it is obvious, that this is not the case at larger distances with only a few measurements on the object of interest. In sensor fusion algorithms, this is typically called the a-priory information about the sensor (or more specific on the overall setup of sensor and algorithm) and is used in the fusion step to weight the results of the different sensors according to their overall probability.

The purpose of this thesis is to characterise the approach for semantic lane segmentation under the use of a 360° LiDAR scanner and convolutional neural networks.

In (2), an introducing research in the field of deep learning and development of convolutional neural networks, by using camera as well as LiDAR sensor data, outlines substantial achievements defining the state-of-the-art and contemporary technologies. The self-driving car industry is mostly inspired by recent developments in applying machine learning applications to autonomous vehicles.

Fundamental building blocks of neural network architectures are shown in (3). Where further, in (3.3) theoretical aspects of LiDAR technology are discussed in detail, which is used to help to develop appropriate metrics for evaluating semantic ground-plane segmentation. Moreover, a discussion of the impact of different environmental parameters as well as the availability of certain information in the training data will support the choice of appropriate metrics and motivate their use in sensor fusion applications.

In (4), state-of-the-art approaches are highlighted to mark the most recent developments in the field of semantic segmentation. Herein, the applicability of machine learning algorithms is pointed out to meet requirements from the autonomous driving task. Further, U-Net as the chosen backbone network to perform semantic lane segmentation is motivated to be used as architecture for point cloud processing.

The generation of ground truth data to provide semantic classes is shown in (5), which is based on the well-known nuScenes dataset. Further, point cloud information gets preprocessed and projected to range-image representation which defines the input data for the chosen state-of-the-art deep learning network.

Previously introduced metrics will be applied to show the applicability of the suggested metrics with a discussion and detailed evaluation in (6).

Lastly, (7), discusses the overall results with respect to sensor-fusion applications and summarizes this thesis.

2 Related Work

During the past decade plenty of research papers from the deep learning community have been published to address the issues of classification, object detection as well as semantic segmentation. The competition has been introduced by the: *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [42] which addresses several tasks from the field of computer vision. Hereby, the ImageNet dataset contains more than 20,000 different category classes, where a set of well-chosen samples from 1,000 classes is used to train and evaluate learning algorithms from submitted approaches. In processing sensory data (e.g. image-sensor), the use of Convolutional Neural Networks (CNNs) gained a lot of attention due to significant improvements in the object detection task.

The pionieering project **AlexNet** (**2012**) [48] drew attention towards the use of deep CNN by exceeding *state-of-the-art accuracy* (26.2%) of more than 10.8 percent (**15.4%**). It was tried to stack multiple convolutional layers on top of each other to form a hierarchically deep structure of connected neurons. Performing convolution, AlexNet used 3x3 kernels as well as large kernel sizes of 7x7, which have not been seen very often in following approaches.

The Oxford research group started to only use small kernel sizes of 3x3 in their **VGGNet** architecture (**2014**) [43]. A total number of 19 convolutional layers have been stacked on top of each other, where it had firstly been possible to break the 10% error mark (**7.3%**). This finally drew attention to the use of deep CNN architectures in the field of visual recognition.

In the following years a lot of research work along convolutional neural networks had been undertaken to better perform in object-detection and classification challenge. Further, pushing the state-of-the-art, several developments and modifications in changing network structure had achieved significant improvements, which could also outperform human performance.

GoogLeNet (**2015**) [44] introduced the *inception module*, a fundamental building block for convolutional neural networks. The overall network structure with 22 deep convolutional layers has carefully been designed, which allowed to increase *depth* and *width* of the network while *keeping computational cost constant*. The convolutional block, arranges locally sparse units from dense components (stacked next to each other, rather than on top of each other).

Inception Module: Working with very deep neural networks is computational expensive and comes along with several drawbacks, such as overfitting or effect of vanishing gradient. In processing features with varying sizes, different kernel-sizes are useful to extract more relevant information, rather than just using one single kernel size. The inception module has been designed in such a way to increase depth and width of the network, while keeping computational cost constant. It was to find out, if an optimal local sparse structure can be approximated by the use of dense components [44]. Hereby, rather than executing several filter operations sequentially (3x3, 5x5, max-pooling, ...) the inception module considers executing kernels in parallel. To reduce dimensionality of features, 1x1 convolution is introduced, which renders the feature-blocks of local activations to a dense representation. This enables the network to provide a robust pipeline for appropriate handling larger numbers of activation layers and an efficient implementation to intuitively process visual information at various scales simultaneously [44].

By participating the ILSVRC, a new state-of-the-art performance could be achieved by pushing error rate to **6.7%**.

ResNet (**2015**) [37] introduced one of the most fundamental logical building blocks for CNN architectures. By increasing depth of network, accuracy has been seen to saturate and is then rapidly degrading, which causes higher training errors. ResNet addresses vanishing gradient problem by implementing residual learning, which had mainly been responsible for pushing new state-of-the-art among the ILSVRC. An error rate of **3.57%** could be achieved, which is even better than human performance (super human performance) [37]. The overall shape of the network includes about 150 convolutional layers by exclusively using 3x3 convolutions in combination of residual skip-connections. The principle of residual learning and mechanism of identity skip-connection is explained in (3.2.5) in detail.

NASNet (**2018**) [29] literally fundamentally changed the way of building convolutional neural networks for image classification models, where the underlying architecture is directly learned rather than engineered. The main contribution of NASNet is to enable transferability, this is done by searching for the: *"best architectural building block"*. Herein, the convolutional layer is firstly trained on a small dataset and then transferred to a larger one. It is tried to build a new convolutional architecture with multiple of copies of the original *cell*. The new architecture enabled to further push error rate below 3% mark (**2.13%**) and had been applied on the ImageNet dataset [29].

The early developments in building deep neural networks, basically addressed issues of the classification and object detection task, which has primarily been pushed within the ImageNet Large Scale Visual Recognition Challenge [42]. After significant improvements could be achieved by the deep learning community it was tried to transfer the knowledge from **detection** to the **segmentation** task. Here, element-wise classification is focused, rather than to identify individual objects in size and position. Fusing **global** features with fine grained information from **local** spatial relationships, enables **semantic segmentation** to perform a pixelwise classification.

Further, CNN architectures have been developed to address those issues by targeting the image domain. Meanwhile, several landmark papers have been published to investigate semantic segmentation as well as instance segmentation. **Fully Convolutional Networks (FCNs)** [35] as well as **U-Net** [41] and **Mask-RCNN** [22] are well known architectures which also introduced new concepts of a network design. Semantic segmentation reqiures detailed local knowledge to perform a pixel- or pointwise classification. Therefore, it is important to provide architectures which are in the position to handle *global* as well as *local* features. In [39] the very commonly seen *encoder-decoder* architecture is outlined which is often used as segmentation standard. Hereby, the encoder network gets extended by a decoder network (called deconvolutional network), which acts as the so called *segmentation head*. Replacing fully connected layers by convolutional layers, at the end of the encoder network, enables CNN architectures to output a *heatmap*. By adding further layers, the decoder network is in the position to provide accurate pixelwise classification (see (4.1) for more details about fully convolutional networks addressing semantic segmentation).

Single stage approaches mainly predominated development of neural architectures in image domain. Hereby, processing of information takes place within a single encoding pipeline, as it is the case for previously mentioned approaches. **Region-CNN (R-CNN) series** [45, 36, 40, 7, 22] adopted detection networks towards *two stage* approaches by including first stage processing as **Region Proposal Network (RPN)** with following second stage as refinement layer of the identified *important* regions from first stage. This treatment has the advantage of isolating Region of Interest (ROI) from background noise, which is usually seen to be most of the information by processing natural data. Further, training

can be focused to important regions which are more likely to contain the actual information of defined category classes. **Mask R-CNN** [22] further adopted the two stage architecture by adding additional layer for binary mask with the purpose to perform instance segmentation. After applying first stage RPN, the second stage outputs class and box labels with an additional binary map mask for each ROI individually.

To better handle problem of background noise, **RetinaNet** [24] motivated the use of *focal-loss* under the use of *single stage* detectors which claims to resolve problems have been addressed in two stage architectures as single stage pipeline. Here, the *cross-entropy* function is modified in such a way that class imbalance is addressed by *weighted loss*, where the optimization algorithm is *forced* to give attention to foreground samples. Combining two stage pipelines in a single stage architecture potentially enables the network to perform faster by using a more simple procedure [24].

Processing **3d point cloud** information introduces a new level of complexity to the classification task. Applying common mechanisms such as **convolution** to unordered sets of points is not possible in the first place and requires substantially different training routines. Furthermore, the **loss of spatial structure** (predefined pixel-grid) does not include context from local neighbourhood, which is especially important for the task of **semantic segmentation**. Several approaches which address the domain of 3d LiDAR point clouds have been summarized in (4.3) to give a detailed overview of the most recent developments.

Several treatments of processing point cloud data have been developed throughout the past years. Voxelization introduces the concept of 3d convolution which renders data voluminous in a 3d grid, such as in [26, 34]. In general, voxelization is seen to be computational expensive and infeasible to provide fast encoding. Further, [15] showed to transform voxel grid into a set of 3d cylindrical pillars which enabled fast encoding under the use of a conversion from 3d point cloud to pseudo 3d multi-channel image. By exceeding the state-of-the-art by a large margin, achieving runtime frequencies of up to 105Hz, [15] seems to provide an appropriate encoder for point cloud domain. In [33] and [25] a series of seminal papers introduced analyses of unprocessed point cloud data which benefits from fast execution and shows state-of-the-art performance, but suffers from loss of spatial structure. Other approaches focus to projection of point cloud to the image domain, such as in [15, 27, 19, 18, 10, 4, 3]. The developments from 2d image detection networks are reused and transferred to the segmentation task, by integrating the activation path of a decoding network, which enables the new architecture to perform point-wise classification.

Pointing out semantic understanding, previous works such as [17] could show that spatial relationships are not the only source of information a network uses for classification. Here, semantic lane segmentation is processed with additional and self-annotated classes from CityScapes dataset [31], where the actual lane semantics have been identified for urban scenarios. Information such as driving-direction of vehicles and occurrence of traffic-signs could improve the segmentation results, where it seems that lane semantics are not only encapsulated in road parts.

There are several papers where path proposals [13] and ego lane segmentation [30, 11] has been performed on camera sensors as well as laser scanners such as in [21] but is not addressing other aspects of ground layers. Further, the approach to classify lane semantics has not been undertaken very much in detail, mostly due to the loss of suitable datasets which could provide rich annotations for lane level geometry. Datasets which also provide highly detailed semantic classes are urgently needed to further push development towards a full scene understanding.

3 Theory

Modern machine learning originates from the middle of the 20th century and occurred in several waves of scientific interest. During the past decades, machine learning experienced large changes in the use of practical applications for many learning algorithms. Limitations to computational power as well as the loss of the existence of appropriate data made the use of machine learning and especially which is called **deep learning** these days, infeasible. Modern deep learning has the potential to fundamentally change many aspects in the field of computer vision. The following chapter outlines the most significant components in the development of neural networks to make data processing, in terms of modern deep learning, feasible and more efficient. Further, outlining the processing of *sensory data* from **LiDAR** laser scanners, theoretical as well as practical aspects on the quality of range measurements are investigated by considering the application of autonomous driving.

3.1 Artificial Neural Networks - ANN

3.1.1 Perceptron

The perceptron model was firstly implemented by Frank Rosenblatt [57] after mathematicians Warren McCulloch and Walter Pitts [58] introduced the *neuron* as logical building block which mainly marks the *cybernetic wave* (1940s-1960s) in very early years of developments of learning mechanisms [32]. McCulloch, Pitts [58] as well as Hebb [52] were working on theories about biological learning, where later Rosenblatt actually implemented the first artificial neuron, the perceptron Figure (3.1), which allowed to train a single neuron [32].

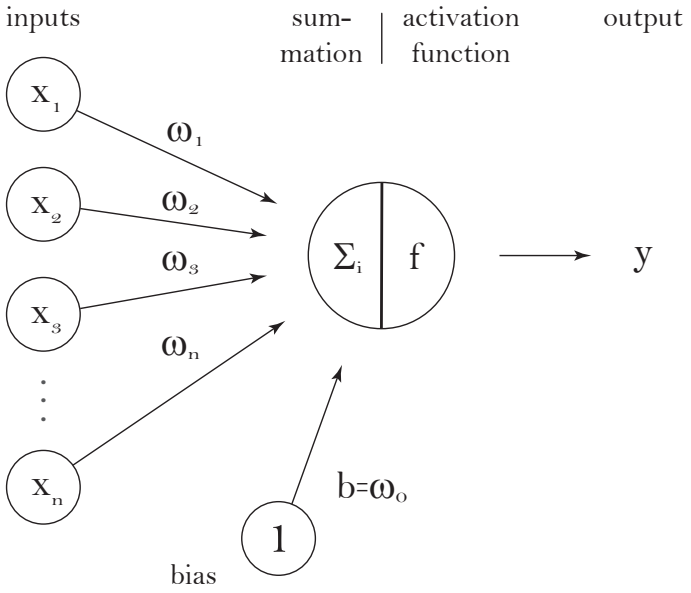


Figure 3.1: *Interpretation of the perceptron model [57].*

The perceptron defines a **linear model** where n input values $(x_1, \cdots, x_n)$ are associated with a defined category output y. The early implementation allowed to find the corresponding

weights $(w_1, \cdots, w_n)$ by training, to predict the correct category output y from input examples - $f(\vec{x}, \vec{w}) = y_j = x_1 w_1 + \cdots + x_n w_n$ [32].
Linear models come along with a couple of disadvantages, e.g. they are not in the position to learn the *XOR* function. It is easy to show that a linear system is not descriptive enough to separate the sample space of an XOR unit into well-defined category regions [32]. Finding solutions for more complex problems, which are beyond linear space, the model's complexity simply had to be increased. As the second wave, *connectionism wave* (1980s-1995s), in developing learning mechanisms, [55] introduced the back-propagation mechanism to train more complex neural networks which are consisting out of one or more hidden layers. Until today, this is one of the most widely used learning algorithm applied in deep learning [32].

3.1.2 Multilayer Perceptron - MLP

Feedforward neural networks, also called Multilayer-Perceptron (MLP), define one of the most widely used network architectures in machine learning applications. Convolutional neural networks are mainly used in image processing is one very well-known application of this type of network.

The structure of a MLP can simply be explained as concatenating multiple of perceptron models to one larger network, which now contains multiple of layers. Figure (3.2) illustrates the shape of a standard feedforward neural network, which contains an input layer, one or more hidden layers and an output layer.

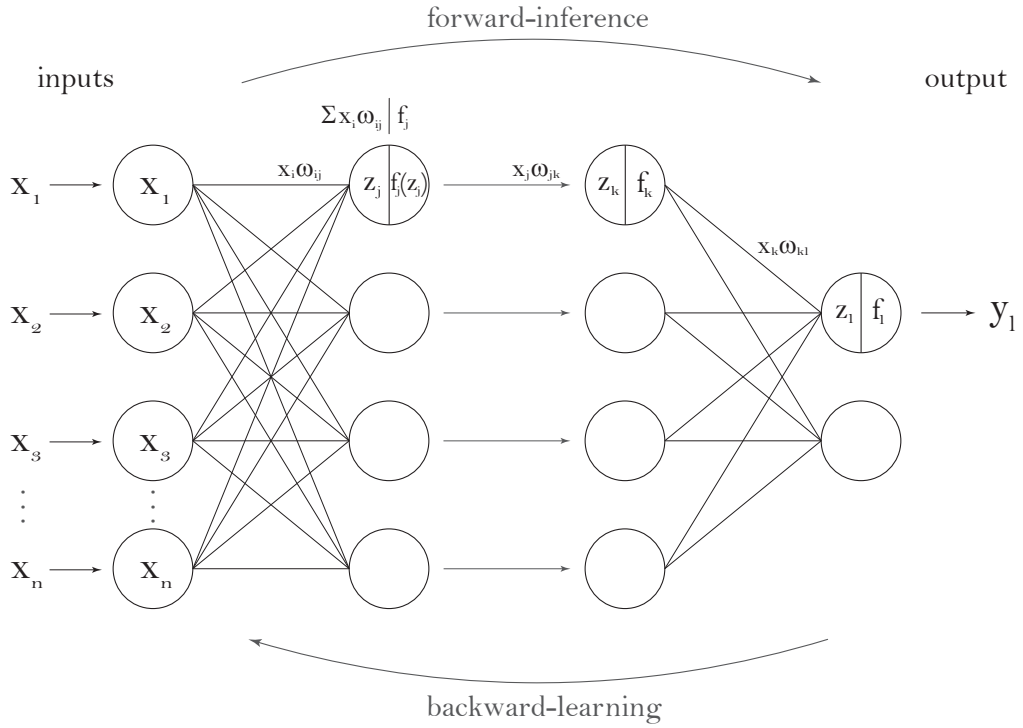


Figure 3.2: *Model of Multilayer-Perceptron.*

Where the previously introduced model of the perceptron, (3.1.1) is limited to linear space, a MLP composes a chain of functions, by putting more hidden layers in between the two known input and output layers, $y = f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ [32], see Figure (3.3). The depth of the network is defined by the length of the chain. The first layer is the input layer,

$$y_l = f_l\left(b_l + \Sigma_k f_k\left(b_k + \Sigma_j \underbrace{f_j\left(\underbrace{b_j + \Sigma_i x_i \omega_{ij}}_{z_j}\right)}_{x_j} \omega_{jk}\right) \omega_{kl}\right)$$

Figure 3.3: *Multilayer-Perceptron - concatenated functions from layers.*

where data gets fed into the network. The last layer is the output layer. The name *hidden layer* is used because the output of the hidden units is not directly seen in the network, as it is not the case for the input and output layer [32].
Within supervised learning, labels from training data, simply determine how the output of the last layer (prediction) need to look like. During training, plenty of samples are presented to the network, where the predicted output $f^*(x)$ gets consequently compared to the actual true value of $f(x)$. The difference of $f(x)$ and $f^*(x)$ is called the *error*, which needs to be minimized during training. The training of a feedforward neural network is basically guaranteed by applying the well-known back-propagation algorithm which can efficiently calculate the partial derivatives of f. Performing the actual training step, an *optimization algorithm* (e.g. gradient-descent) tries to minimize the error value through weight adjustments, by feeding the error backwards through the network.
Lastly, it is about the optimization algorithm to find the best parameters to match $y = f(x)$ with $\hat{y} = f^*(x)$ [32]. Today's deep learning mechanism can differentiate between at least a thousand categories simultaneously, where early linear models could only handle binary classification problems (0/1) [32]. In [32] it is stated that a neural network, which is trained with at least 5,000 labelled examples per category class, can achieve *acceptable performance*. *Human performance* is exceeded with training sets containing at least 10 million labelled samples. Such large quantities are very rare and training would require massive computing power. Therefore, an important area of research is to achieve appropriate results with datasets which are much smaller than this, to finally gain from large quantities in unsupervised learning [32].

3.1.3 Backpropagation

The backpropagation algorithm has been introduced by [55] and is basically computing the partial derivatives of a chain of concatenated functions. Figure (3.4) shows a *directed acyclic graph* which basically represents the structure of a feedforward neural network.

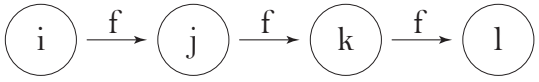


Figure 3.4: *A directed acyclic graph illustrates a chain of concatenated functions: $j = f(i), k = f(j), l = f(k)$ [32]. To compute the partial derivatives, $\frac{\partial l}{\partial i}$, the chain rule is applied to obtain Equation (3.1) [32].*

$$\frac{\partial l}{\partial i} = \frac{\partial l}{\partial k}\frac{\partial k}{\partial j}\frac{\partial j}{\partial i} \tag{3.1}$$

The flow of information is forwards, from left to right and cannot go backwards at any point in the network. Network structures with backward connections, which allow a recurrent flow

of information - means to also give the possibility to re-enter neurons from previous layers, in a loop wise manner; are called recurrent neural networks. Algorithm (1) illustrates the **forwardpass** (here: written in pseudocode), which is the commonly used algorithm to process inference for a feedforward neural network. For inference, data is fed into the network to calculate *bias* and *weight* values, which getting used to lastly update a corresponding loss function $L(\hat{y}, y)$ with the predicted value $\hat{y} = f^*(x)$ and the actual true value from label data $y = f(x)$.

Algorithm 1: Forward propagation in pseudocode representation. Information gets fed into the network where *bias* and *weight* values are updated accordingly. The corresponding *loss* value is calculated by function $L(\hat{y}, y)$ [32]. Accordingly, the loss value is used by an *optimization algorithm* to perform the actual step of learning with the help of *backpropagation algorithm*, Algorithm (2).

Data:
Number of layers: l
weight matrices: $W^{(i)}, i \in 1, \dots, l$
bias parameters: $b^{(i)}, i \in 1, \dots, l$
input: x
Result: output y
$h^{(0)} = x$
for $k = 1, \dots, l$ **do**
  $z^{(k)} = b^{(k)} + W^{(k)} h^{(k-1)}$
  $h^{(k)} = f(z^{(k)})$
end
$\hat{y} = h^{(l)}$
$J = L(\hat{y}, y)$

At this point it is also worth to notice that the backpropagation algorithm is only responsible for calculating the gradients, where another *optimization algorithm*, such as gradient-descent or adam-optimizer, is using these gradients to perform the actual learning task through weight adjustments.
An illustration of the backpropagation algorithm in Algorithm (2) is showing an example written in pseudocode, which is outlining the most relevant procedures to accordingly backpropagate calculated gradients.

Algorithm 2: Backward propagation, pseudocode example [32].

gradient calculation at output layer with loss J from forward propagation
$g \leftarrow \nabla_{\hat{y}} J$
for $k = l, l-1, \dots, 1$ **do**
  gradient calculation at a layer's output, of *prenonlinear activations* f
  $g \leftarrow \nabla_{z^{(k)}} J = g f'(\dot{z}^{(k)})$
  gradient calculation for *weights* and *biases*
  $\nabla_{b^{(k)}} J = g + \nabla_{b^{(k)}}$
  $\nabla_{w^{(k)}} J = g h^{(k-1)\mathrm{T}} + \nabla_{w^{(k)}}$
  backpropagate gradients to next lower-level layer (*hidden layer's activations*)
  $g \leftarrow \nabla_{h^{k-1}} J$
end

3.1.4 Loss Functions

Cross Entropy Loss

The loss-function, also often called error- or cost-function, $J = Loss(\hat{y}, y)$, computes the loss value between the predicted output $\hat{y}$ and the true label value y from training data [32]. In general, Cross Entropy (CE) is used in binary classification, the equation for discrete probability distributions is shown in Equation (3.3):

$$H(p, y) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log p(y_i) + (1 - y_i) \log(1 - p(y_i)). \tag{3.2}$$

In training a neural network with N number of samples, with probability $\frac{1}{N}$, y_i represents the actual true values from training labels, where $p(y_i)$ are the predictions. Without considering size of sample space, cross-entropy can be noted as follows [24]:

$$H(p, y) = H(p_t) = -\log p_t. \tag{3.3}$$

Respectively, $log(p)$ determines if the sample belongs to some category class, on the other hand $\log(1 - p)$ determines the probability of not being part of it [24]:

$$p_t = \begin{cases} p & \text{if y = 1} \\ 1 - p & \text{otherwise.} \end{cases} \tag{3.4}$$

Focal Loss

Using cross entropy directly, comes with some drawbacks. That is because of imbalance of positive and negative samples in natural data. The data which gets collected by some sensor (e.g. camera or LiDAR) usually contains much fewer positive samples than negative ones. This is because desired objects cover only a small area in the entire frame, where the rest is dominated by negative background noise. Classification is mainly affected by this imbalance where it is problematic to classify the real positive samples. An approach to better handle imbalance of examples is the focal loss function, Equation (3.5). Herein, the loss function is down-weighting negative examples (easy samples) to focus training on positive ones (hard samples). In general, focal loss is the weighted cross-entropy loss with $\gamma > 0$.

$$FL(p_t) = -(1 - p_t)^{\gamma} \log p_t \tag{3.5}$$

The modulating factor $(1 - p_t)^{\gamma}$ is near 1 if the predicted value for p_t is small (close to 0), the computed loss will be unaffected. Whereas, if p_t is close to 1, the factor will be almost 0. Therefore, the loss for correct classified examples is down weighted. Furthermore, α is added as an additional factor to the equation of focal loss, which is also handling class imbalance of datasets, see Equation (3.6). It could be seen that there was an improvement in detection accuracy compared to the equation without α [24]:

$$\alpha_t = \begin{cases} \alpha & \text{if y = 1} \\ 1 - \alpha & \text{otherwise} \end{cases}$$

$$FL(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log p_t \tag{3.6}$$

3.1.5 Optimization Algorithms

Gradient Descent

Training a Deep Learning Algorithm comes along with an optimization task, (e.g. to optimize some function $y = f(x)$). Function y is also called loss-, error-, or cost-function. Herein, optimization describes the procedure of minimizing or maximizing $f(x)$. Minimizing often refers to finding the global minimum of $f(x)$, which can be achieved by computing the derivative $f'(x) = \frac{\partial y}{\partial x}$ [32]. The basic principle is illustrated bellow, Figure (3.5).

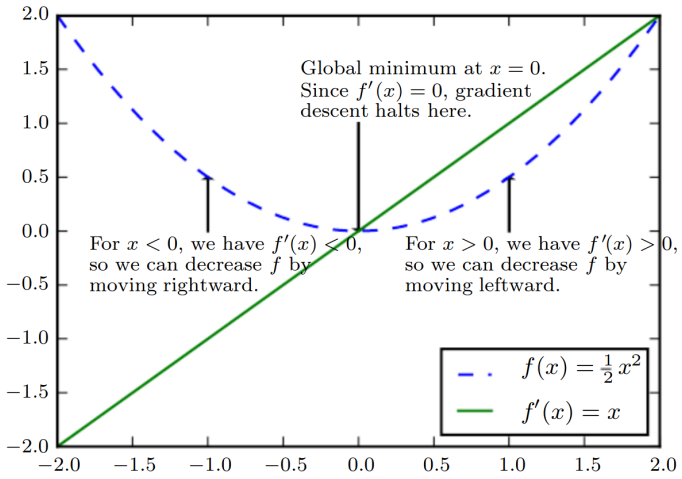


Figure 3.5: *Illustrating principle of gradient descent algorithm. [32]*

The derivative $f'(x)$ describes the slope of $f(x)$ at every point x and therefore implies how to scale changes in the input x to get the corresponding change in the output: $f(x+\epsilon) = f(x) + \epsilon f'(x)$ [32]. Herein, ϵ is the learning rate, which is a scalar value and describes a factor to scale step size. The negative gradient just tells the direction of how to change value x to come closer to the global minimum given from y. This algorithm of iteratively computing and updating $f(x)$ is called *Gradient Descent (GD)*. Due to its iterative nature, the procedure stops if the computed change in loss is below a certain threshold [32].
An example of how GD is finding a functions global minimum is shown in pseudocode in Algorithm (3), herein x is the parameter vector.

Algorithm 3: Gradient Descent [32]

Data:
Learning rate: ϵ
Initial parameters: x
while *step_size > threshold* **do**
  compute gradient with parameters: $\hat{f} \leftarrow \nabla_x f$
  update parameters: $\hat{x} \leftarrow x - \epsilon \hat{f}$
end

By increasing complexity of the objective function, complexity of optimization increases respectively. Generally, there are plenty of critical points, which makes it difficult to find an appropriate solution. Arbitrary numbers of local minima and maxima, saddle points, local minima which perform just as good as global ones, etc. are common situations in solving real world problems [32]. In Deep Learning, optimization just involves such complex functions, which can also be multidimensional, depending on type of input data. This is making the field of optimization to a broad area of research.

Stochastic Gradient Descent

The Stochastic Gradient Descent (SGD) algorithm is an extension to gradient descent and is one of the most widely used optimization algorithms among Deep Learning Applications. Whereas gradient descent uses the entire set of samples to perform an update iteration and therefore to minimize the objective function, SGD performs on a randomly sampled subset of m examples from the training set. An example in pseudocode, in Algorithm (4), is illustrating the algorithm. SGD massively reduces the required number of computational steps for updating parameters compared to standard gradient descent.

Algorithm 4: Stochastic Gradient Descent [32]

Data:
Learning rate: ϵ
Initial parameters: θ
while $step\_size > threshold$ **do**
  randomly sample m examples from training set $\{x^1, \cdots, x^m\}$
  compute gradients: $\hat{f} \leftarrow \nabla f = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$
  update parameters: $\theta \leftarrow \theta - \epsilon \hat{f}$
end

Further, due to very large training sets with up to millions of samples, gradient descent needs to iterate through the entire sample space to perform a single update to come closer to the global minimum. SGD is not always finding the optimal solution, but in general the approximation is as good as optimal, with a fraction of computational steps.

Adam Optimization

There are plenty of optimization algorithms available to be used in machine learning application. Common optimization algorithms used in today's learning applications are: SGD w/w/o momentum, RMSProp w/w/o momentum, AdaDelta and Adam [32]. It is hard to say if there is one algorithm which performs better than all others. The choice of which optimization algorithm to choose, mainly depends on the learning task or the objective of the user [32]. Algorithm (5) outlines the most relevant procedures the Adam algorithm uses for optimization. *"Adam"* is the short version for *"adaptive moments"*, where the algorithm updates with exponential moving averages ($\beta m + (1 - \beta)\nabla f$) of estimates of the first and second order momentum (mean, variance) of the gradient, ∇f, of the objective function [38]. The parameters $\beta_1, \beta_2 \in [0, 1)$ controlling the exponential decay rates of the moving averages [38]. Momentum helps the gradient to move faster into the "right" direction to find global minimum, where SGD w/o momentum has problems to find the optimal solution and tends to end up in local minima.

Algorithm 5: Adam - Adaptive Moments [38, 32]. The suggested default value for learning rate is 0.001 with decay rates of $\beta_1 = 0.9$ and $\beta_2 = 0.999$. $\odot$ defines the elementwise multiplication and $\delta = 10^{-8}$ is a constant for numerical stabilization.

Required:

Learning rate: ϵ

Exponential decay rates: $\beta_1, \beta_2 \in [1, 0)$

Stochastic objective function: $f(\theta)$

Initial parameter vector: θ_0

Init:

Initialize first moment estimate vector: $m_0 = 0$

Initialize second moment estimate vector: $v_0 = 0$

Initialize timestep $t = 0$

while *step_size* > *threshold* **do**

randomly sample m examples from training set $\{x^1, \cdots, x^m\}$

$t \leftarrow t + 1$

compute gradients: $\hat{f}_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

update biased first moment estimate: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\hat{f}_t$

update biased second moment estimate: $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)\hat{f}_t \odot \hat{f}_t$

compute unbiased first moment estimate: $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$

compute unbiased second moment estimate: $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$

update parameter: $\theta_t \leftarrow \theta_{t-1} - \epsilon \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\delta}$

end

3.2 Convolutional Neural Networks - CNN

3.2.1 Convolution and Deconvolution

Convolution

The convolution operation is widely used in the field of computer science. In vision-based applications it turned out to be one of the most important mathematical operation to process images. By applying convolution, the input signal(a discrete set of values), is getting convolved with a *kernel* function, which usually is an approximation of another function. In general, the convolution operator describes an elementwise multiplication of the input function f with a kernel function k, see Figure (3.6) for illustration.

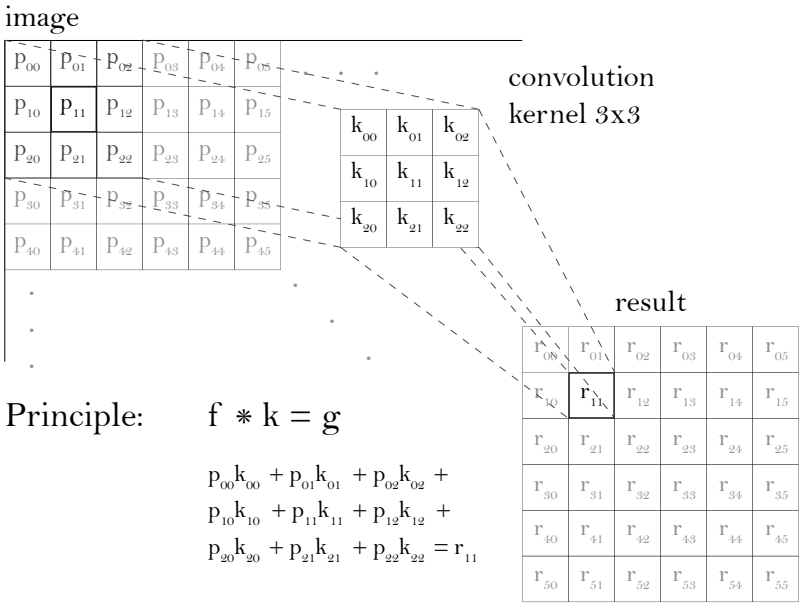


Figure 3.6: *Convolution operation* $g(x, y) = (f * k)[x, y]$.

The result is a new function g, the convolved input from f. Equation (3.7) demonstrates convolution $g(x, y) = (f * k)[x, y]$, where filter-size n is an odd number:

$$g(x, y) = \sum_{i=\lfloor -n/2 \rfloor}^{\lfloor n/2 \rfloor} \sum_{j=\lfloor -n/2 \rfloor}^{\lfloor n/2 \rfloor} f(x + i, y + j)k(i, j). \tag{3.7}$$

In convolutional neural networks, the specific look of the produced feature-maps is determined by the kernel coefficients. Where in classic computer vision predefined filters (e.g. edge- or corner-filters) are used to compute corresponding feature maps, a neural network first learns filter coefficients to produce complex features. In literature, the kernel-size is often seen as a $3x3$ array of discrete values, which turned out to achieve sufficient results for the most of applications.

The input image of a convolutional layer is in the size of $H_i x W_i x D_i$. H_i, W_i defines the spatial dimension and D_i is the dimensionality in channels of the input data. A set of N (nxn) filters is applied to the input to produce the corresponding stack of feature maps in dimensionality N, $HxWxN$. Convolutional layer and filter stack is illustrated in Figure (3.7).

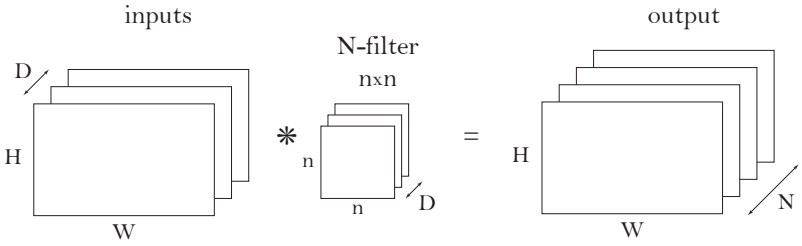


Figure 3.7: *Convolutional layer in CNN, illustration shows stack of filters and feature maps in convolutional layer.*

Deconvolution

Deconvolution (often referred as transposed convolution) describes the exact inverse process of convolution operation, $f = g * k$. Where convolution naturally shrinks the size of the input image, the purpose of deconvolution is to restore the original image size, demonstrated in Figure (3.8). Convolution comes with a loss of information which makes it impossible to restore the distorted image back to the original. In a convolutional neural network, deconvolution is usually applied after unpooling operation, see (3.2.2). Unpooling enlarges the activation map of a previous layer, which is yet a sparse map of activations. Unpooling is a non-learnable operation without filter, where it is tried to restore the spatial resolution of the original map before pooling will be applied (see Figure (3.9) and Figure (3.10) for illustrations of pooling and unpooling operations). After unpooling has been applied, transposed convolution is densifying the sparse map of activations with a stack of learnable filters, similar to convolutional layer [39]. Convolution compromises multiple of activations within a filter window into a single activation (many to one relationship), deconvolution on the other hand associates single activation with multiple outputs (one to many relationship), where deconvolution produces an enlarged but dense activation map.

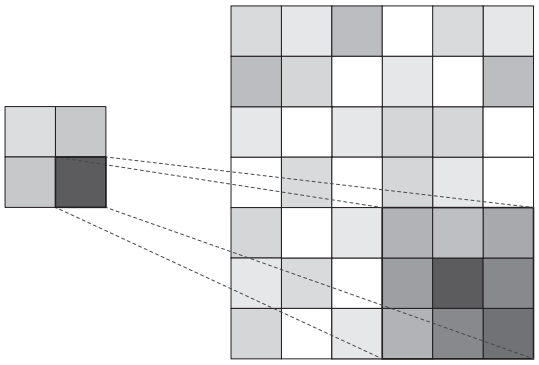


Figure 3.8: *Deconvolution example, illustrates the principle of deconvolution operation. A single activation gets enlarged to multiple of outputs. Deconvolution is densifying the sparse map of activations after unpooling.*

3.2.2 Pooling and Unpooling

Pooling

Pooling comes in two different types which are very common in use, average- and max-pooling. In general, applying pooling operation to a map of activations, comes with a loss

of spatial information. Herein, pooling helps to filter noisy map of activations, which results in a down sampled version of the original feature map. The number of activations inside of a receptive field (sliding window of a filter), is abstracted to a single representative value. Therefore, pooling obtains only the most robust activations in higher layers [39]. Figure (3.9) is illustrating the pooling operation. In applying max-pooling the maximum activation is chosen from a kxk kernel which is sliding over the entire feature map, similarly to convolution. Herein, a new feature map of activations is formed, only containing the maximum values where all the rest of information is discarded. Next to the maximum activation, pooling is also storing the grid position of the value. This is helpful information which supports unpooling operation in later layers, to place the value back again to the same position it originally came from.

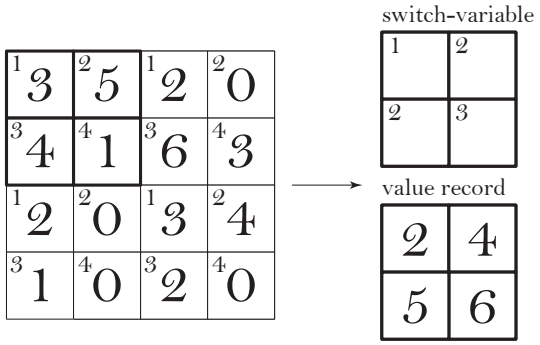


Figure 3.9: *Principle of pooling operation. Max-Pooling and Average-Pooling are commonly used method to process pooling. The illustration demonstrates max-pooling, where the maximum value within a 2x2 filter grid is selected and stored in the pooled- and down sampled-layer. To better reconstruct the original map with unpooling, the location coordinates of pooled layers are stored as switch-variables.*

Unpooling

Unpooling defines the reverse operation of pooling, which is basically used to reconstruct the original spatial resolution of the activation map, before pooling had been applied. Performing unpooling, the stored location coordinates of the pooled activation map are used to place each of the activations back to its original location [39]. The principle of unpooling is illustrated in Figure (3.10). Performing unpooling this way, allows a parameter free operation without the need of learning filters [39].

3.2.3 Fully-Connected

After a series of convolution and pooling operations a CNN often uses Fully Connected Layers (FCs) for classification, at the last layers of a network. The FC layer, at position k in a convolutional network, is summarizing all activations from the previous layer $k-1$. Herein, FC means that every neuron from layer $k-1$ is connected to each other in layer k, the FC layer. The size of the FC layer is often related to the actual number of classes the network model tries to distinguish (One-Hot-Encoding), see Figure (3.11). Using FC layers comes with the loss of spatial information, therefore, pixelwise classification is not possible anymore, which is not desirable in semantic segmentation.

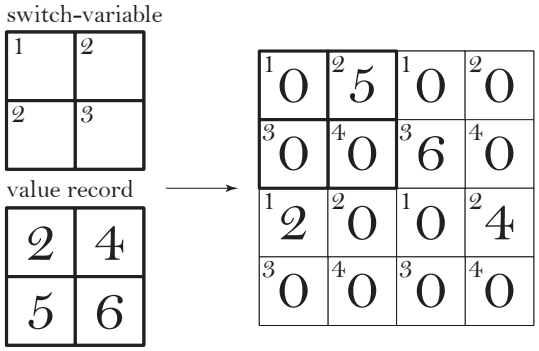


Figure 3.10: *Principle of unpooling operation. Reverse procedure of pooling. The down sampled layer from pooling is tried to get rebuild by up sampling and filling the gaps with* 0 *values.*

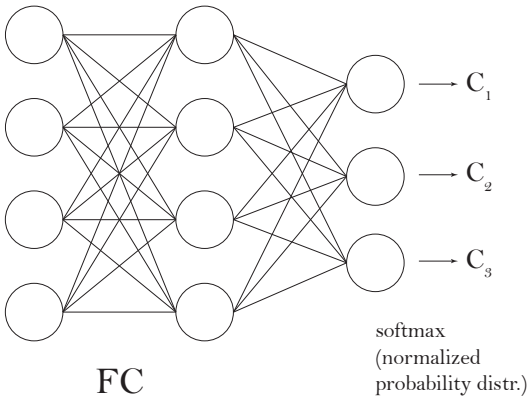


Figure 3.11: *Illustration shows structure of fully connected layers (FC). The last FC layer is downsized to the actual number of classes. Applying normalization operation by softmax function provides a probability distribution for the output neurons (One-Hot-Encoding).*

3.2.4 1x1 Convolution

The general purpose of introducing 1x1 convolution or dimensionality reduction is to reduce the effort of computational cost. Instead of using larger kernel sizes to process a stack of N channels of feature maps, 1x1 convolution is used in a similar way such as pooling. Certain channels become less relevant during training and contribute only a small amount of value to the overall process of classification. Herein, a channel-wise pooling is provided. The number of channels is reduced in a more compact representation of features, while the most robust activations getting retained.

The example illustration, Figure (3.12), shows the principle operation of applying 1x1 convolution to an input stack of features with N channels. Depending on the number of 1x1 filters, the stack-size of the resulting number of channels can also be increased.

3.2.5 Identity Skip-Connection

The principle of identity skip-connection has been introduced as logical building block of the ResNet architecture [37]. The general purpose was to construct very deep network architectures while prohibiting problems such as vanishing gradients through residual learning.

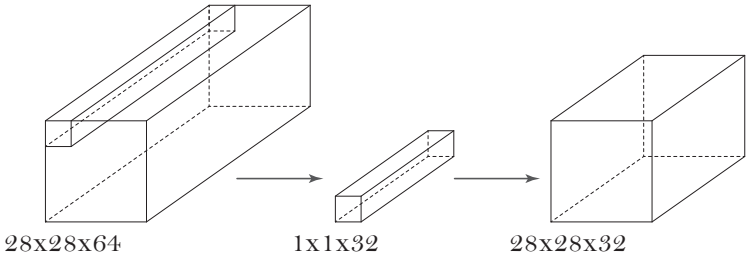


Figure 3.12: *Principle of dimensionality reduction. 1x1 convolution is applied to a stack of features-maps with N channels.*

With increasing depth of network, learning saturates and accuracy degrades rapidly which causes higher training errors. The logic behind skip-connections is to learn a residual mapping, $\mathcal{F}(x) = \mathcal{H}(x) - x$, by a stack of nonlinear layers, instead of learning the original mapping $\mathcal{H}(x)$ directly. Therefore, the desired mapping is denoted as $\mathcal{H}(x)$, which can further be recast into $\mathcal{F}(x) + x$ [37].

The reason behind this is the fact that it is easier to optimize a residual mapping and to minimize the residual to zero rather than to fit the original identity mapping for each stack of nonlinear layers directly. The new defined mapping $\mathcal{F}(x) + x$ can be implemented in feedforward neural networks as *skip-connection* or *shortcut connection*. Further, identity skip-connection is an easy to understand mechanism where simply the raw information from a lower layer in the network hierarchy is passed to a deeper layer. This mapping can be understood as *information forwarding*, which showed to be useful to handle degradation problem of deep architectures. Herein, neither computational complexity nor extra parameters are needed and the network is still trainable in an end-to-end manner.

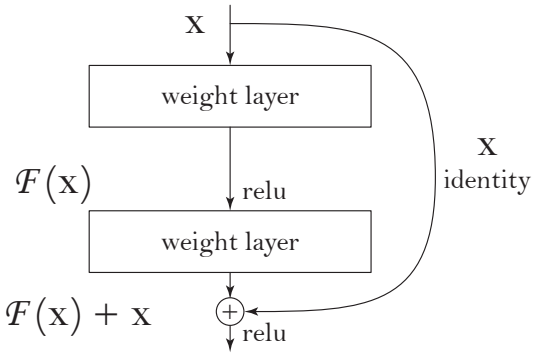


Figure 3.13: *Residual block, identity skip-connection [37].*

3.3 LiDAR - Theory and Practice

General Specifications - LiDAR Range Measurement

Light Detection and Ranging (LiDAR), provides accurate range measurements of the environment, mostly as a map of 3-dimensional coordinates and intensity values (*x,y,z,i*). Focusing recent developments addressing autonomous driving, there are basically two types of LiDAR systems used in practice, traditional LiDAR (rotating mechanics with mirror) and solid state LiDAR (MEMS mirror or flash lidar). In the scope of this thesis, only traditional LiDAR systems are considered.

Within the scope of this work, the Velodyne LiDAR HDL-32E scanner [49] has been used to undertake range measurements. A short summary of several system specifications of the scanner is given in Table (3.1) [49].

Table 3.1: *Specifications: Velodyne LiDAR HDL-32E [49]*

Model:	HDL-32E
Channels:	32
Range:	100m (effective range, about 1m - 70m)
Rate:	5Hz-20Hz
Accuracy:	±2cm
V-FOV:	41.33°(+10.67°to -30.67°)
H-FOV:	360°
Resolution (V/ inclination):	1.33°
Resolution (H/ azimuth):	0.08°-0.33°
Points/Sec.:	695,000 (single), 1,390,000 (dual)
Wavelength:	903nm
Beam divergence:	3.0mrad

The HDL-32E scanner provides a total number of 32 scanlines. Herein, vertical angular offset between consecutive scanlines is *linear* and fixed to a value of 1.33° per channel. The horizontal offset, basically depends on speed of rotation of the entire apparatus with a default value of 0.166°. The horizontal and vertical field of view is tried to illustrate in Figure (3.14) with the specifically integrated offset between consecutive lidar channels.

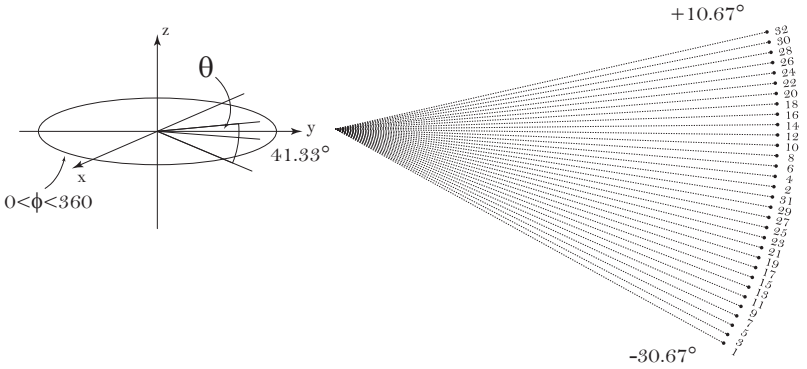


Figure 3.14: *Illustrating schematics of HDL-32E LiDAR. Horizontal angular resolution depends on mechanical rotation of apparatus, vertical angular resolution is fixed by laser-diods and lens.*

The general physical principle of LiDAR range measurements is explained in the following. The LiDAR scanner is emitting light-pulses (laser-beams) to the environment, where a part of the energy from the emitted pulse is backscattered to the receiver and registered as incoming signal peak. The energy footprint of the transmitted signal pulse follows the pattern of a non-uniform distribution and can be approximated by a two-dimensional gaussian function, where the emitted energy is the highest at the centre of the pattern:

$$f(x, y, \sigma_x, \sigma_y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2})} \tag{3.8}$$

σ_x, σ_y indicating lateral distribution of signal profile. The schematics of range measurement and signal footprint are illustrated in Figure (3.15).

Herein, the delta time, between emitting and receiving a signal, is called **time-of-flight** (tof). herein, the emitted laser is assumed to travel at the speed of light c, where the radial distance d between scan device and obstacle can easily be determined by:

$$d = \frac{c}{2}\Delta t_{tof} \tag{3.9}$$

The quality of measurement mainly depends on factors such as: calibration of scanner, environmental influences, obstacle properties (geometrical shape, surface characteristics) [50]. To theoretically describe scan characteristics, the quality of range measurements is investigated with respect to physical and geometrical aspects of the scan device and target surface as well as environmental influences.

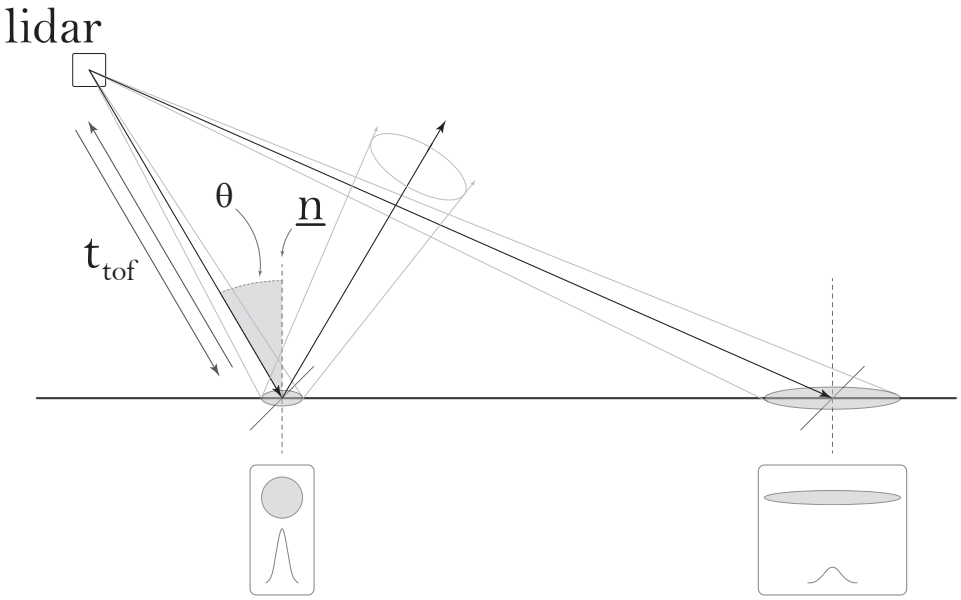


Figure 3.15: *Demonstrating reflection of light-pulse at a flat surface. Due to beam divergence, the footprint of the signal appears in a circular shape within perpendicular reflections and changes to an elongated shape if incidence angle differs from* 0° *[50].*

Physical Aspects

The pulse of a lidar scanner can be considered as a gaussian beam. The footprint of a light pulse appears in circular shape by hitting a perpendicular surface, 0°. At larger angles, the

footprint is more elongated, weaker in magnitude and larger in time, which is demonstrated by example in Figure (3.16) [14].

Herein, the emitted signal (red) and two received waveforms (blue and green) are shown, which have been measured at same distance but varying incidence angle (green $\alpha = 0°$, blue $\alpha = 80°$). The received waveforms are not just separated in time, with offset Δts, but also appear in different shapes and strength of magnitude, due to varying incidence angle. A bigger portion of the signal, which is hitting the oriented surface, returns sooner to the receiver, as the one which is not oriented. This introduces a bias in range measurement, where a different incidence angle leads to varying results in magnitude of the received signal, by keeping the same distance [14]. Other effects of instrumental calibration or environmental conditions are not considered here.

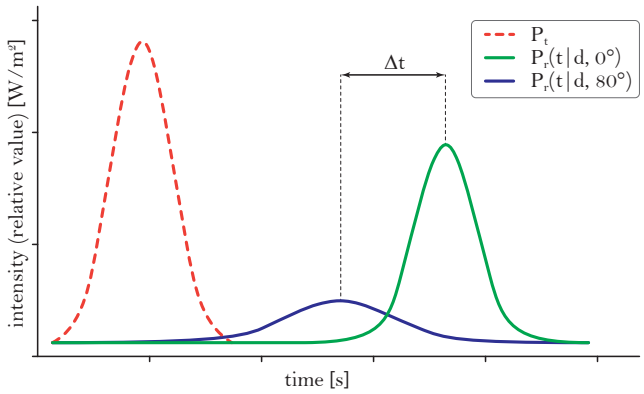


Figure 3.16: *Experimental example, shows waveforms of LiDAR signals with varying incidence angles but constant distance [14]. Transmitted signal: red; Received waveforms: green, perpendicular; blue,* 80° *incidence angle. Physical unit of laser beam intensity mostly referred as power per square meter,* $[W/m^2]$*. Here intensity is given as value between* $0 \ldots 1$*, where 1 associate to full power of used scanning device.*

Considering the experimental setup of the nuScenes test vehicle, the LiDAR scanner is roof-mounted, at a height of approximately $1.8m$ above ground. Further, ground is assumed as a flat surface, where the emitted light pulses from a laser scanner will hit ground at varying angles (illustrated in Figure (3.15)), which directly contributes to signal to noise ratio and therefore to reliability of measurement.

Beam divergence describes effect of increasing beam radius with distance from scan device. In Figure (3.17) the size of beam diameter is modelled with respect to distance, herein, progressing in radial distance shows that size of beam footprint is proportional to distance.

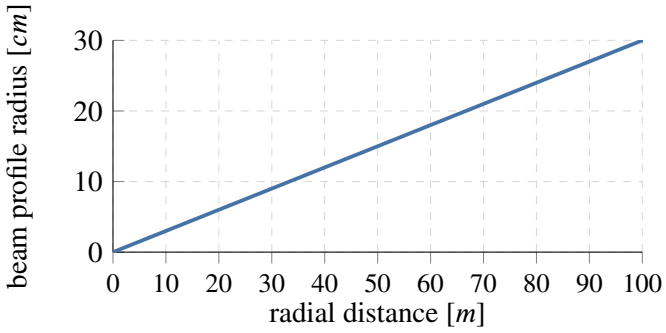


Figure 3.17: *Modelling beam divergence (HDL-32E: 3mrad) relative to radial distance.*

Without knowing the exact reflectance properties of the surface of an obstacle, it is tried to theoretically model the influence of the incidence angle α (to normal vector n at target surface) related to the signal to noise ratio of the received signal. A light pulse hits a target surface. Herein, the backscattering is described by the *Bidirectional Reflectance Distributive Function (BRDF)* [53]. Furthermore, the emitted and received intensity of the signal can be measured by the use of the laser range equation (radar equation), where the received signal power P_r is formulated as:

$$P_r = \frac{\pi \rho A_{tar}}{4d^2} P_t \cos\alpha \eta_{atm} \eta_{sys} \tag{3.10}$$

ρ reflectance properties of material surface, A_{tar} surface target area, d radial distance to scan device, P_t transmitted signal power, α incidence angle at surface normal vector, η_{atm} environmental influences (e.g. atmospheric transmission), η_{sys} system related parameters.

It can be seen that radial distance and cosine of angle α directly contributes to signal deterioration, as range and incidence angle increases, see Figure (3.18) [50].

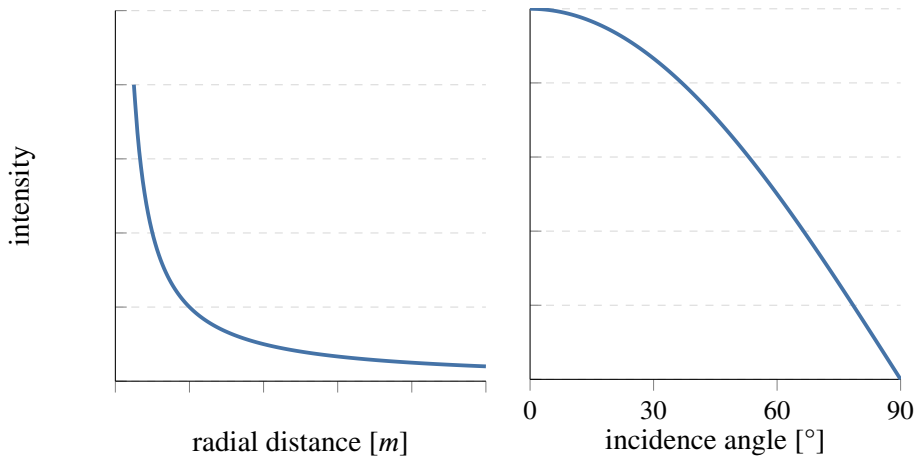
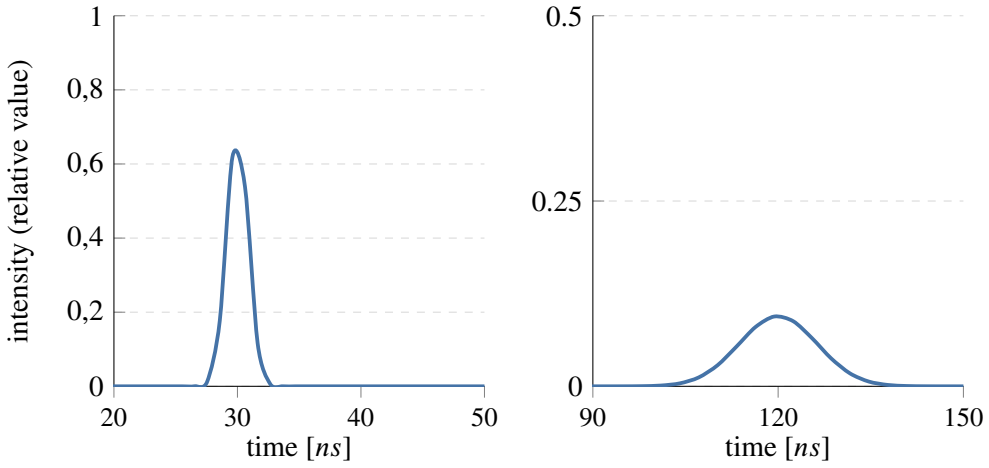


Figure 3.18: *Theoretically modelling intensity degradation with respect to radial distance and incidence angle of laser beam located at surface normal vector. Physical unit of laser beam intensity mostly referred as power per square meter,* $[W/m^2]$*. Here intensity is given as value between* $0 \ldots 1$*, where 1 associate to full power of used scanning device.*

Herein, the received signal intensity directly corresponds to distance and surface alignment as well as material properties, where signal quality decreases due to decreasing signal to noise ratio.

Effect of beam divergence causes the radius of the signal footprint to increase with distance. Considering the experimental setup of the test vehicle the LiDAR scan device is installed at height $h = 1.8m$ above ground. As it was shown in Figure (3.18), SNR decreases if incidence angle and distance increases. Herein, it can be demonstrated that the ability of a scan device to detect a target mostly depends on signal to noise ratio, which can mainly be determined by the ratio of emitted and received signal power, as mentioned before. Other effects such as instrumental- or environmental influences are not considered in here.

In Figure (3.19), amplitudes of two possible range measurements are illustrated. Example (a) demonstrates measurement of ground at a distance of $5m$, whereas example (b) shows the response of a measurement at about $20m$. The effect of beam divergence and growing incidence angle causes degradation of model performance. The signal response in (b) appears broader in time, separated from the actual true measurement position and smaller in magnitude, compared to (a). Therefore, detection probability would decrease if radial distance increases. This characteristic behaviour suggests introducing an a-priori threshold for removing unreliable measurements [50].



(a) close measurement

(b) distant measurement

Figure 3.19: *Example of two range measurements. Considering the experimental setup, the LiDAR installation position is approximately* $1.8m$ *above ground. (a) measured at close distance (small incidence angle and radius of beam footprint); (b) measured at far distance (large incidence angle and radius of beam footprint). Physical unit of laser beam intensity is referred as power per square meter,* $[W/m^2]$*. Here intensity is given as value between* $0 \dots 1$*, where 1 associate to the full power spectrum of the used scan-device.*

Geometrical Aspects

Geometrical properties influencing range measurements are discussed in the following. In Figure (3.20) offsets in horizontal and vertical direction of consecutive scan-points are seen. The signal power footprint of a laser beam is assumed ot be non-uniform and can be approximated by a two-dimensional gaussian function. Considering the HDL-32E lidar scanner, beam profiles are partly overlapping in horizontal direction (beam divergence of 3mrad). The

minimum geometrical size of an object which would be still detectable, does not depend on physical size but more on properties of surface reflectivity. For simplicity, the footprint of signal power is assumed to be uniform, where the area of the beam profile results in:

$$A_{bf} = \frac{\pi}{4}(d\gamma)^2 \tag{3.11}$$

d is denoted as radial distance and γ as the beam divergence. Further the area of the target object (Equation (3.12)) which gets hit by the laser beam is assumed to be equal to A_{bf}, $A_{bf} = A_{tar}$.

$$A_{tar} = \pi r^2 \tag{3.12}$$

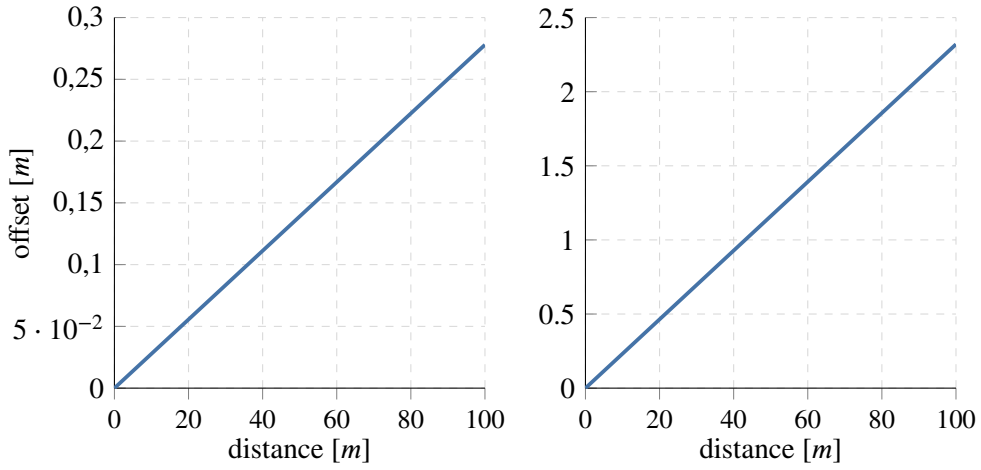
r defines the radius of the target object which is set to circular shape. Following, the power density of the signal at the target surface is approximated by:

$$\Phi_{tar} = \frac{P_t}{A_{bf}}\eta_{atm} \tag{3.13}$$

Atmospheric transmission η_{atm} mainly reduces signal power P_t, the amount of power of the emitted signal. Further, the amount of reflected energy from target surface results in:

$$P_{ref} = \rho\Phi_{tar}A_{tar}\eta_{atm} \tag{3.14}$$

Without considering η_{atm}, the amount of energy is getting reflected from target surface is primarily limited by *rho* - material reflectivity and size of surface area A_{tar}. Considering an object with $\rho = 5\%$ and $A_{bf} = A_{tar}$. The minimum size of an object with $\rho = 100\%$ would be still detectable is $\frac{A_{bf}}{20}$ [51].



(a) Horizontal Offset

(b) Vertical Offset

Figure 3.20: *Horizontal and vertical resolution of the Velodyne HDL-32E scanner. Illustration of offsets between consecutive measurement points in* [m]*, with respect to angular resolution. Horizontal angular resolution:* 0.166°*; Vertical angular resolution:* 1.33°.

3.4 Metrics

Intersection over Union

In the deep learning community, it is seen very often to describe the accuracy of a classifier in global metric. Accuracy describes the proportion of actually correctly predicted cases, which is the number of True Positive (TP) (predicted as positive sample and actually true) and True Negative (TN) (predicted as negative sample and actually labelled as such one) divided by the total population of samples: *Acc = (TP+TN) / total population*.

This sort of a performance indicator is best used for binary classification, whereas within multiclass classification, mostly an average accuracy is formed over the entire set of classes. This treatment is not always optimal, if the numbers of per class samples is very unbalanced, which is naturally seen in real data. In the sense of global measures, class imbalance is considered only poorly. Therefore, it is necessary to preserve a more meaningful metric for evaluating category classes which are easy and hard to predict.

In semantic segmentation, Intersection over Union (IoU) computes the overlap between the boundaries of 2 patches (rectangular box or any segmentation mask). Therefore, area of overlap is divided by area of union: $\frac{A \cap B}{A \cup B}$.

$$IoU = \frac{TP}{TP + FP + FN}. \tag{3.15}$$

Five-Number Summary

In terms of descriptive statistics, five-number summary provides compact representation to plot statistical values and measures of a dataset. There are several standardized plots, such as box-plot or violin-plot, see Figure (3.21). In general, information about the dataset is summarized in 5 meaningful values:

minimum:	lowest data point (except outliers)
maximum:	highest data point (except outliers)
Q1 (25th percentile):	median of lower half
Q2 (50th percentile):	median of dataset
Q3 (75th percentile):	median of upper half

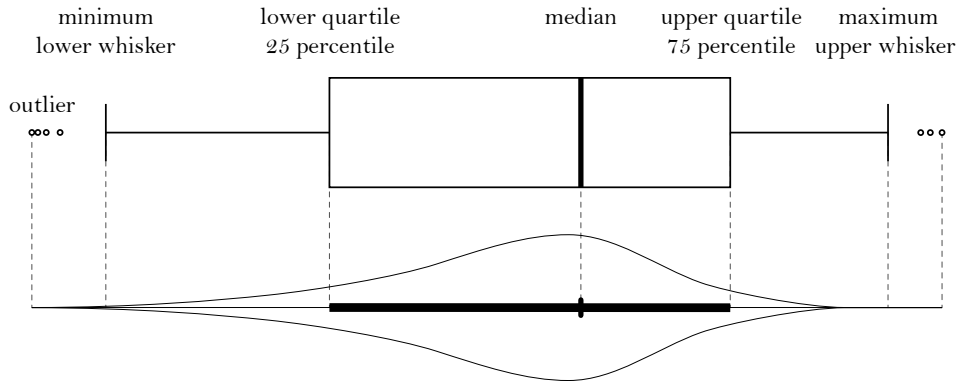


Figure 3.21: *Demonstrates box-plot elements with additional probability density distribution drawn as violin-plot (robust statistics, five-number summary).*

The information presented by five-number summary is not limited to describe the distribution of data along the median value, such as in Equation (3.4) but can also represent

any other statistical measure (e.g. mean and standard deviation). Considering box-plot, the inner-box (called inter quartile range (Inter Quartile Range (IQR))), represents the distribution of 50% of the dataset, where the thick line within the box can show median or mean value of the distribution. Further, defined ranges between minimum and first quartile (Q1), and maximum and third quartile (Q3) is described as whiskers. Herein, the second half of the dataset is represented which shows to contain a larger range of dispersion from median. Information such as dispersion, symmetry as well as skewness of distribution can be visualized by using box-plot representation. Skewness necessarily requires information of mean value to decide whether data is left- or right skewed (left skewed (negative skew) - mean is less or located rightwards from median; right skewed (positive skew) - mean is larger or located leftwards from median).

However, box-plot cannot visualize modality of distribution, which requires plotting of Probability Density Function (PDF) of the corresponding data. Herein, violin-plot is in the position to compare density functions from two categories directly. Modality of a distribution is shown by the number of local maxima. In terms of machine learning application, the performance values of a trained classifier can be plotted as probability density function, where the modality of the distribution is unimodal if the distribution reveals one distinct peak. Bimodal or multimodal distributions show the inability of the classifier to well differentiate between the defined category classes.

Multilabel-Confusion-Matrix

Rendering the prediction result in a multilabel confusion matrix offers class wise detection scores of defined semantic classes. Several metrics to describe performance of the investigated model can be calculated from the confusion matrix directly, where the distribution of true positive, false positive, false negative and true negative is available in compact representation. Hereby, Figure (3.22) shows the distribution of samples for class c1.

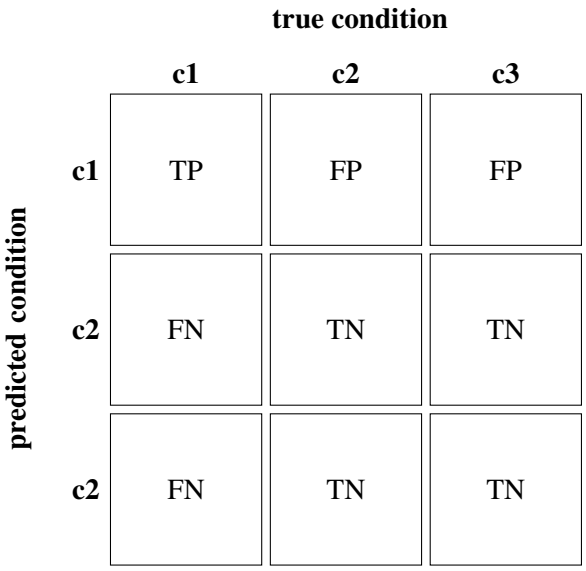


Figure 3.22: *Multilabel confusion matrix, example: distribution of class samples (c1): TP - true positive, FP - false positive (type I error), FN - false negative (type II error), TN - true negative.*

Furthermore, the confusion matrix can be plotted in normalized representation, where

the True Positive Rate (TPR) can directly be extracted as the score of TP of the corresponding class. This normalization is helpful in case of imbalance of class samples to better visualize rates of misclassifications. Considering a binary classification, TPR (sensitivity or recall) measures the number of positively identified data samples which are also actually labelled as being part of the positive class compared to the amount of TP and False Negative (FN) samples (the samples which are labelled as being part of the positive class but which are actually falsely identified as being negative (type II error)). On the other hand, False Discovery Rate (FDR) determines the number of falsely identified class samples (type I error). Applying FDR is especially helpful if comparison between multiple classes is undertaken. FDR controlled procedures give more weight to the rate of False Positive (FP) compared to other ones (e.g. false positive rate). Hereby, FDR compares the rate of FP against $FP+TP$, where a higher sensitivity in changes of FP samples is given (change in type I errors).

$$TPR = \frac{TP}{TP + FN} \tag{3.16}$$

$$FDR = \frac{FP}{FP + TP} \tag{3.17}$$

Multilabel confusion matrix directly shows if a classifier is in the position to well distinguish between the defined semantic classes, where the classification performance of the model can be interpreted quantitatively. In normalized form, true positive scores close to the value of 1.0, which are aligned along the main diagonal of the confusion matrix, shows good performance of the model. This implicitly contains small rates of misclassifications among the remaining row and column values.

4 LiDAR Semantic Segmentation

Demands of semantic segmentation introduced new level of complexity to the classification task. Whereas object detection primarily focused to predict type of category as well as position and size of objects, semantic segmentation is targeting fine grained elementwise classification. Pixel-to-pixel relationship associates pointwise information with a class label. In the following chapter, basic principles of the development of fully convolutional networks are introduced. Further milestone architectures mark the most recent developments for both image (**UNet**) and point cloud domain (**PointNet**).

4.1 Fully Convolutional Networks - FCN

The principle architecture of a fully convolutional network (FCN) has firstly been introduced in *Fully Convolutional Networks for Semantic Segmentation* [35]. Very common is the encoder-decoder architecture which allows an end-to-end dense learning, see Figure (4.1) for illustration [35].

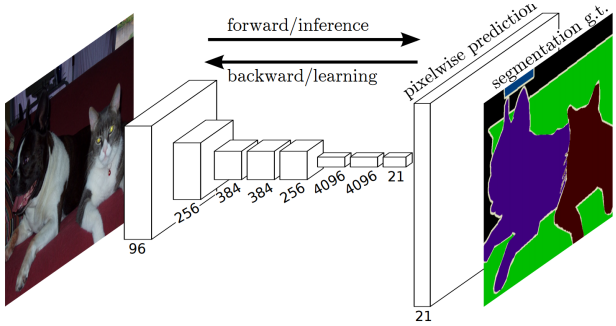


Figure 4.1: *Fully Convolutional Network [35]. Illustration demonstrates common encoder-decoder architecture. Pixelwise classification has been made possible through conversion of fully connected layers into convolutional layers only.*

CNNs from the object detection task usually end in fully connected layers. This allows classifying the input in some category class with bounding-box regression. In semantic segmentation, pixelwise classification is demanded, which cannot be achieved with the use of fully connected layers, where any spatial information for pixelwise classification would be lost. Therefore, transforming fully connected layers to convolutional layers enables a CNN to output a heatmap, for illustration see Figure (4.2). By adding further layers this can be used for up sampling with deconvolution operations to form class-specific activation maps, which enables the network to an efficient way of pixelwise end-to-end dense learning [35].

In FCNs the encoder and decoder are transferring the input to convolved feature maps. The activations (neurons) are connected throughout layers. That is because, convolution leaves a path of locally connected neurons from one layer to another, which means that a neuron from a specific layer l_i in the network, is connected to several neurons from previous layers. These path-like connections are called the *effective receptive field* of a neuron. In very low layers of the network (being closer to the input layer), convolution extracts very fine-grained local information in feature map with a small receptive field. On the other hand,

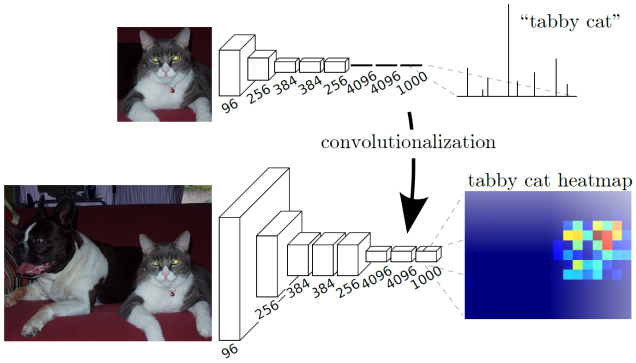


Figure 4.2: *Transforming fully connected layers to convolutional layers. Convolutionalization enables the network to output a heatmap to efficiently perform end-to-end dense learning [35].*

going down the *effective receptive field* (higher layers), there is less local information but the meaning of those features increased, where convolution responses to a much more compact representation (subsampled version) of the original input.

Applying convolution and pooling operations to activation maps, leads to new feature maps. Those maps containing only the most robust activations in higher layers, also see (3.2.2) [39].

In contrast to convolution, the reverse operation is defined as deconvolution or transposed convolution. Similarly to convolution, there is also a feature hierarchy provided of different shape level details. Features from lower layers are mainly used to describe the general shape of objects, whereas in higher layers more class-specific and fine-details are encoded in the activation map. Herein, class-specific information is directly integrated in the decoder, which is useful for fine grained semantic segmentation [39].

4.2 U-Net

U-Net has been introduced in [41] as FCN for the purpose of semantic segmentation, targeting the image domain. This approach focuses to the more efficient use of smaller sample sets rather than to work with larger quantities, comprised within a simple and intuitive model architecture, illustrated in Figure (4.3). The network is structured as encoder-decoder path, which is commonly seen in semantic segmentation. Herein, the encoder captures high-level spatial context to collect fine-grained and detailed information of the input data. On the contrary, the decoder network provides an expanding, up-sampling network path to enable precise localization [41]. The main idea behind U-Net is to extend the encoder network by the use of a decoder structure. Herein, pooling layers are replaced by up sampling operators (unpooling), which enables the expanding network path to restore the original resolution from corresponding encoder level. The concept of transposed convolution is essential for the task of semantic segmentation. Object detection networks are primarily used to find out the specific type of an object, which is present in the input data. Hereby, it mostly cares about *"what"* to find inside the data, whereas on the other hand, semantic segmentation is also interested about *"where"* to find samples from specific classes (means the precise location). Therefore, the use of transposed convolution helps to restore the location information after encoding step. Further, rather than to rely on pure up sampling layers in the decoder network, common convolution is applied to stacked feature maps, wherein learnable weights

getting introduced in the deconvolution path. Encoding uses fine-grained high-level features. Decoding suffers from restoring such information in the up-sampling step. Therefore, identity skip-connections are used to enable information forwarding into lower layers. For more details about skip-connections see (3.2.5) and reference to deep residual learning [37].

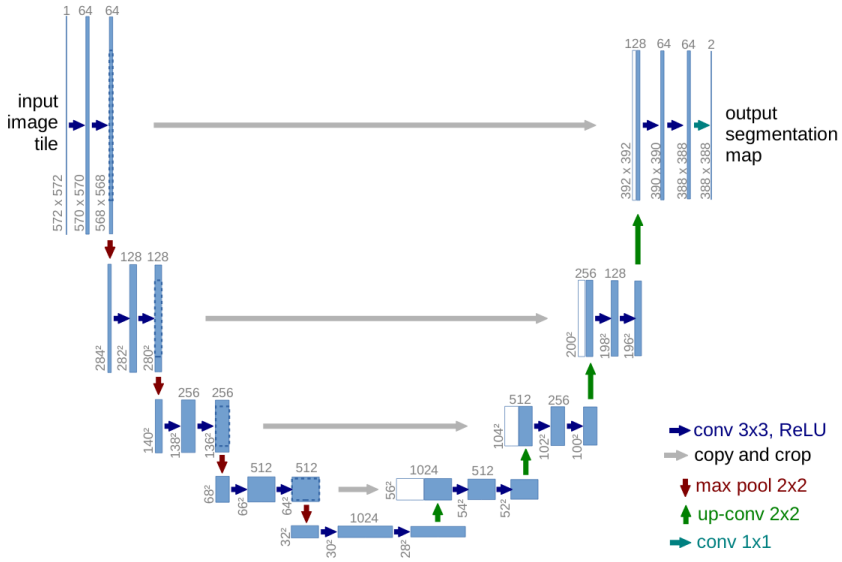


Figure 4.3: *U-Net architecture, uses encoder-decoder path with identity skip-connections to perform semantic segmentation on arbitrarily sized input images [41].*

4.3 Point Cloud Segmentation

In comparison to an image where the information is well defined in a pixel-grid with spatial neighbourhood, a point cloud is understood as an unordered set of points where no spatial structure is given in the first place. Most often, image processing in deep learning is undertaken by convolutional neural networks, where the spatial context is directly included to the processing, due to the nature of convolutional operation. By processing unordered sets, such as point clouds, applying convolutional operations directly is not possible due to the loss of a well-defined structure such as a pixel-grid. Whereas convolution is very fast and efficient in image processing, different ways of computing point clouds need to be investigated for both performance and efficiency. During the past decade there have mainly been three procedures to analyse point cloud data fast and efficient: *voxelization*, *unprocessed* point cloud and *projections*.

Therefore, an initial research in the field of semantic segmentation driven by deep learning is to highlight landmark papers showing the most recent developments and state-of-the-art performance.

Voxelization: Rendering point clouds voluminous in a 3d voxel grid is adding spatial structure. Quantization is generally coming with a loss of information, where basically the voxel size is determining the level of compression and the granularity of the result. Processing a point cloud in a voxel grid representation is computational expensive and results in a runtime complexity of $O(n^3)$ [26], which is generally seen to lack in performance and is not feasible for many applications.

SEGCloud [26] is a state-of-the-art paper which addresses voxelization and 3d convolution. It is showing voxelization in a *dense grid* with an additional post processing to refine the resulting segmentation in combination with a 3d fully convolutional neural network, Figure (4.4).

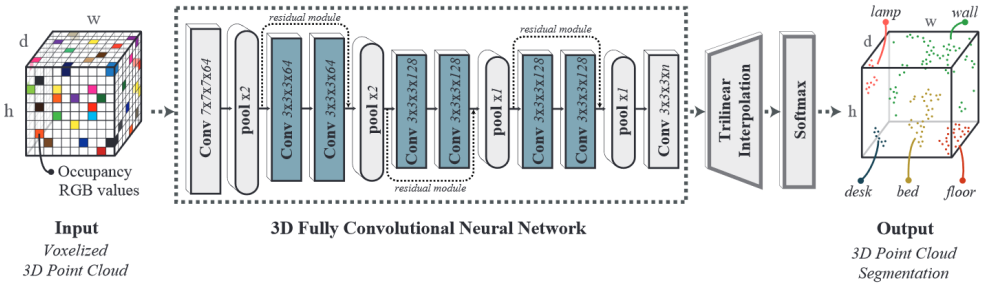


Figure 4.4: *Illustrating network architecture of SEGCloud approach, voxel-based point cloud processing by using 3d convolutional operations [26].*

The system accuracy basically depends on grid resolution. Excluding further post processing, the segmentation result would be just as good as the grid resolution. Hereby, every point in a voxel gets assigned the same label. Furthermore, a dense grid is not optimized for the natural sparsity of point clouds, where most of the voxels remain empty (containing no information).

Therefore, performance heavily suffers caused by runtime complexity of $\mathbf{O(n^3)}$ [26] (for static grid structure), where only small voxel resolutions are feasible ($\mathbf{64^3}$)[26], even with the use of modern high performance computing power.

[34] motivates using more efficient ways of treating voxel grids in a different *octree* based data structure and *overcomes* the *sparsity* problem of point clouds by introducing *dynamic grids* Figure (4.5). Here, uniform regions getting combined to a single cell. The octree only scales with the depth of the tree and therefore reduces runtime complexity to $\mathbf{O(n^2)}$ in both time and space. By using the introduced octree with a dynamic grid structure, much higher resolutions are feasible ($\mathbf{512^3}$ on modern GPUs). Furthermore, the authors claim that the conversion from a dense grid to a dynamic one could turn out in a new state-of-the-art approach for point cloud segmentation.

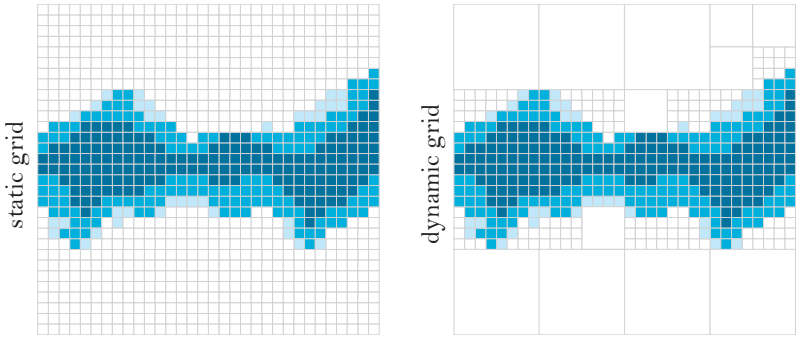


Figure 4.5: *Illustrating static grid from traditional voxel-based approaches compared to dynamic grid with a smaller number of empty voxels [26].*

Unprocessed: Processing raw point cloud data has the advantage of no computational cost of pre-processing, where the raw information (e.g. (x, y, z, i)) is used for further processing.

Point clouds can be understood as *unordered sets*, which are not structured into a predefined pixel grid (a more profound study on sets and how order invariance is achieved can be found in [33, 28]). Therefore, common convolution cannot be applied to point clouds in the first place, caused by the loss of spatial structure. Nonetheless, by doing so, two major issues need to be considered:

Problem 1: *variance to ordering*
Problem 2: *desertion of shape*

Figure (4.6) is illustrating a convolutional kernel (i) and three point clouds $(ii) - (iv)$ with varying order of indices as well as different shapes. As it is obvious, (iii) and (iv) sahre the same geographical coordinates but varying in order, whereas (ii) and (iii) are quite the opposite, ordering is equal but shape is different.

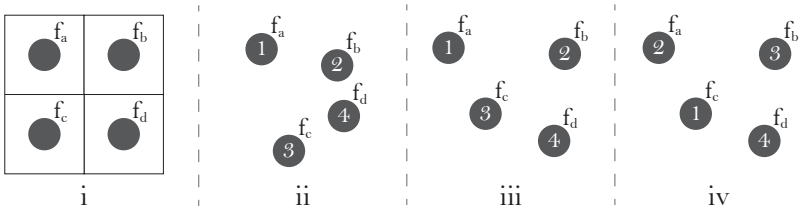


Figure 4.6: *Applying convolution to unprocessed point clouds [16].*

Basically, a point cloud with n elements contains $n!$ permutations for possible input variations. When a function f is applied, every permutation need to give the same result. Equation (4.1) is outlining the stated issues by applying f to point clouds $(ii) - (iv)$, where the equation fails to fulfil both requirements.

$$\begin{aligned} f(iii) &\neq f(iv) \\ f(ii) &= f(iii) \end{aligned} \tag{4.1}$$

Processing point clouds in a raw and unprocessed representation, a different approach needs to be investigated, to fulfill Equation (4.2). Therefore, a short overview of the most recent developments will be shown in the following.

$$\begin{aligned} f(iii) &= f(iv) \\ f(ii) &\neq f(iii) \end{aligned} \tag{4.2}$$

PointNet [33] is a seminal paper for analysing unprocessed point cloud data, see the network architecture in Figure (4.7). The crude three-dimensional point cloud information is used to perform a feature transformation with independently arranged multilayer perceptron's, where every point gets transformed separately. Forming of a *global feature-vector* is taking place, where the *symmetric max-pooling operation* is getting applied, to solve the problem of *permutation invariance*. Besides the standard network for object detection and bounding box regression a second sub-network is used to *combine global- and local-features* which is improving the segmentation process.

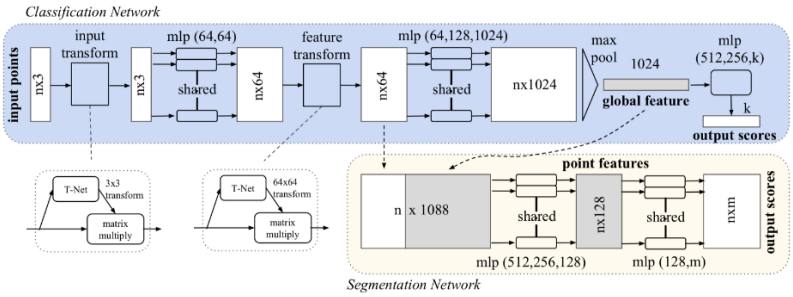


Figure 4.7: *PointNet [33]*

Nonetheless, PointNet is not considering any spatial relationships to neighboured points, as every point gets transformed individually. This outlines a major drawback for segmentation which lacks in considering local context.

PointNet++ [25] is the follow-up paper to [25], where local neighbourhood is getting considered in a hierarchical manner, to improve accuracy of segmentation. As it is illustrated in Figure (4.8), the approach follows an encoder-decoder architecture including identity skip-connections for the segmentation branch. This is commonly seen among convolutional neural networks.

A down sampling stream of *grouped layers* is used to perform segmentation hierarchically, which adaptively combines features from multiple scales. At first, a *neighbourhood search* is started over the entire point cloud on previously identified *centroid points*. Followed by a *pointnet-layer* where multiple of PointNet architectures getting used for independent feature transformation for every centroid with the corresponding neighbourhood.

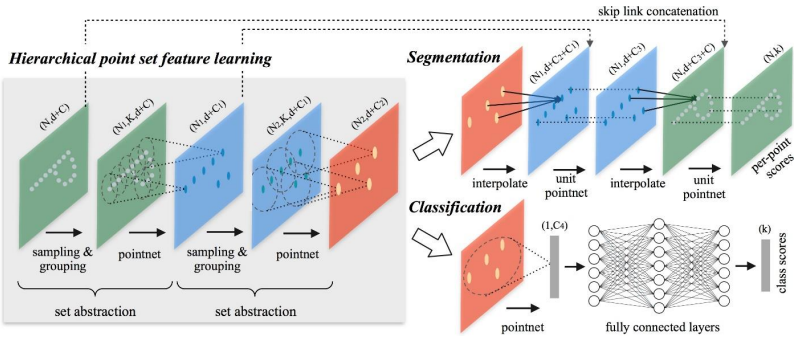


Figure 4.8: *PointNet++ [25]*

[25] shows an approach to capture local context and therewith addresses an issue of [33] but also lies back in performance due to computationally expensive neighbourhood queries.

PointCNN [16], marks a remarkable paper in processing point clouds, where it is tried to directly apply convolution to an unordered set of points.

As it is illustrated in Figure (4.9), the basic principle of PointCNN is to apply a *learned transformation* to $(ii) - (iv)$ to perform convolution correctly with arbitrary kernel grids. Equation (4.3) shows how to handle the stated issues in Equation (4.1), with the loss of a well-defined spatial grid.

$$f_{ii} = k * (\chi \times [f_a, f_b, f_c, f_d]^T) \tag{4.3}$$

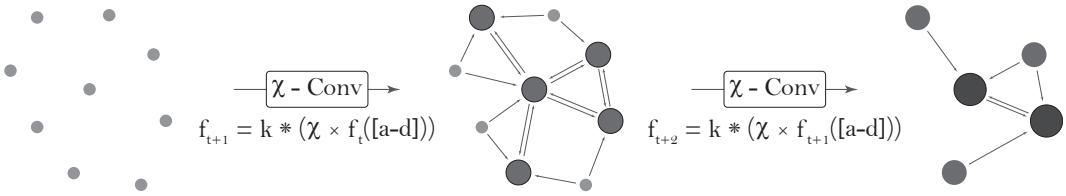


Figure 4.9: *Applying convolution to point clouds with the use of a learned transformation (χ) [16]. Further, coordinates are encoded in features, layer wise, similar to [33, 25].*

Herein, the learned χ transformation would find the *correct permutation* for the convolution operation and is lifting the coordinates into features, which is similar to [33], where instead of applying a symmetric function the χ *transformation* is used to solve for Equation (4.2).

As stated from the authors, the fundamental understanding of the χ transformation is far from perfect and still an interesting research topic for further improvements [16]. Nonetheless, the approach has already been in the position to be on-par or even outperform state-of-the-art architectures, such as PointNet [33].

Projections: Projecting 3d point cloud data to the image domain avoids computational overhead and renders the data with spatial structure. Various attempts have been arisen during the past years in object detection as well as semantic segmentation, where projections gained attention by performing on-par or even better than state-of-the-art [15, 27, 19, 18, 10, 3].

Point clouds mainly benefit from accurate distance information, which is not available, or only partly, in the 2d image domain. However, projecting 3d point clouds to 2d images does not need to come along with the loss of certain information. Similar to an image, with RGB channels, all the information from the 3d space can be stored in multiple of feature channels, where coordinates or other qualities from point clouds or sensor *(x,y,z, intensity, radius, ...)* can be converted in to a top-view or spherical-projection. Conversion of three-dimensional coordinates to flat 2d images is tried to illustrate in Figure (4.10).

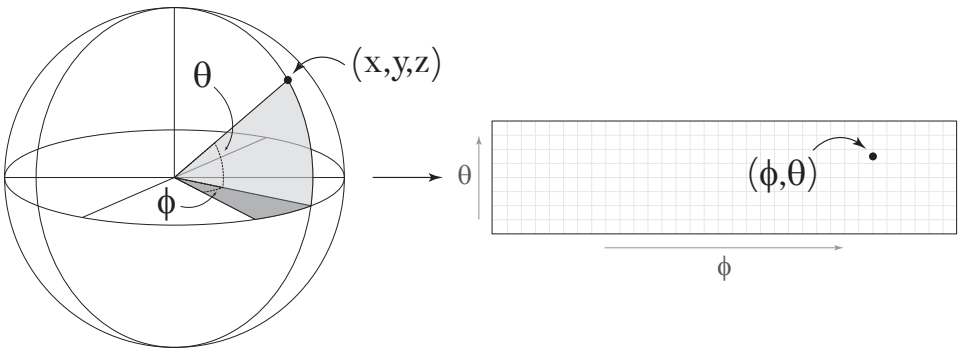


Figure 4.10: *Spherical projection of point cloud data into a 2d range-image representation [18]. Points from xyz cartesian-coordinates are transformed to polar-coordinates, angles θ and ϕ are used as information for pixel index.*

Projecting a point cloud to image space has the advantage of using established 2d algorithms as backbone network, which have been developed and optimized throughout the past years. This allows fast inference speeds with the advantage of processing features from 3d

world in 2d space. In [15, 27, 19, 18, 10, 3] spherical projection has been used to render the data compact and to overcome problems of sparsity. Discretisation into a 2d grid comes with a loss of information, where basically the resolution must be chosen in such a way that most points don't occupy same pixel positions for multiple of times. On the other hand, due to physical aspects of the used sensor, not every pixel position gets occupied by a measurement ping. This phenomenon is called *dropout-noise* [27, 19], which is mainly caused by: no reflection is registered by the sensor, angular resolution of laser beams (no consecutive pixel positions can be covered) or influence of incidence angle. Improving performance on these circumstances, a binary mask is applied to the feature-maps to simply exclude dead points.

4.4 LU-Net

Within the scope of this thesis, **U-Net** [41] is chosen to be the backbone network to perform semantic segmentation with. U-Net has been seen to be very fast and efficient in processing camera images for the task of segmentation. A FCN is used to produce a segmentation mask in the same size as the input image. [3] focused to process LiDAR point clouds in a spherical projection under the use of U-Net architecture, see Figure (4.11). The approach achieves state-of-the-art performance and forms the base of this thesis. Further, [3] included cartesian coordinates (x, y, z), intensity as well as radius information as feature maps.

Moreover, it is shown that previous work considered 5-channel or even only 2-channel range-images and could achieve results which are comparable to the state-of-the-art. The number of used feature-channels seems to be empirical for the target application and needs to be conducted by a larger set of experiments to find the best combination of channels [3]. Therefore, to not set focus to network engineering, an already optimized architecture has been used to perform semantic segmentation on point cloud data.

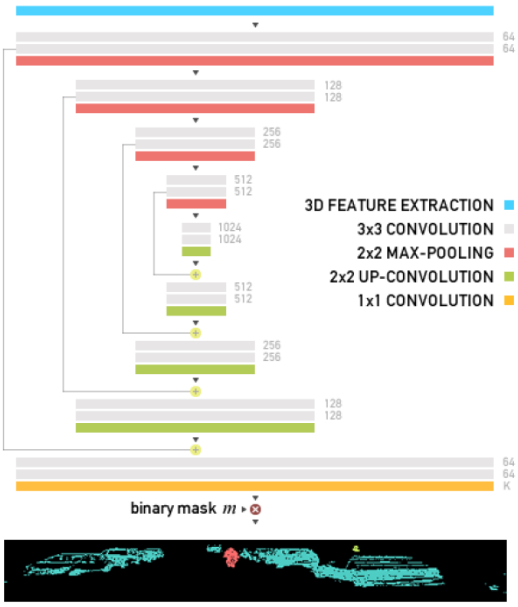
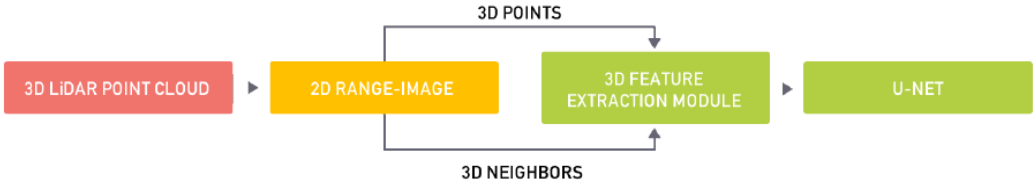


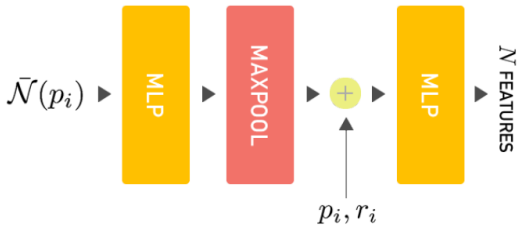
Figure 4.11: *Architecture of proposed LU-Net as backbone architecture containing processed segmentation mask [3, 41].*

For properly handling converted 3d data in range-image representation, a *high-level 3d*

feature extraction module has been introduced in [9, 3] (as mechanism of local point embedder to prevent segmentation of point cloud from overfitting), which allows to efficiently transform input data to N-dimensional feature vectors with the help of multi-layer perceptron's and maxpool operations. The overall pipeline is illustrated in Figure (4.12).



(a) Overall pipeline LU-Net, with feature extraction module.



(b) Detailed view of 3D feature extraction module, outputs N-dimensional feature vector for each LiDAR point.

Figure 4.12: *Overall pipeline of LU-Net (a) [3], 3D feature extraction module (b) [3].*

Introducing the high-level 3D feature extraction module enables the network to directly learn meaningful features for the task of semantic segmentation. Spatial context is introduced to point cloud processing with an *8-connected neighbourhood*. Herein, neighbourhood pixels are extracted from the range-image directly, which avoids computationally expensive pre-processing of point clouds, (e.g. k-Nearest-Neighbour (kNN) search in 3d space, etc.). The set of N neighbouring points $N(p_i)$ of point p_i are firstly processed by a MLP followed by operations of ReLU and batch normalization. The resulting set of transformed features is further processed by a max pooling operation where the resulting point feature is concatenated with p_i and r_i (reflectivity value of point i). Then, the resulting vector is processed by another MLP. This outputs an N-dimensional feature vector for every point p_i from range-image. The introduced module performs feature transforming on a multichannel range-image which can be used as a matrix in shape H x W x N for further learning procedures [3].

Addressing issues of *drop-out noise*, as it has been mentioned in [27, 19], a binary map mask is applied to the image to exclude specific *dead* points from the training procedure, which is excluding points don't correspond to any return of a LiDAR pulse (marked as black stains in range-image).

5 Methodology

The following chapter describes the proposed model for semantic lane segmentation in detail. The application of autonomous driving demands high reliability on perception of surroundings to appropriately perform motion control. The data from multimodal sensor-suite need to be accessed and validated in real-time to correctly react within the fast-paced environment. Autonomous driving mainly benefits from processing multimodal data, where a single sensor cannot fulfil the capabilities to provide a full scene understanding.

Modelling the architecture of a sensor-fusion application, Figure (5.1) demonstrates units for sensing, perception and decision making, which is illustrated within an abstract flow chart.

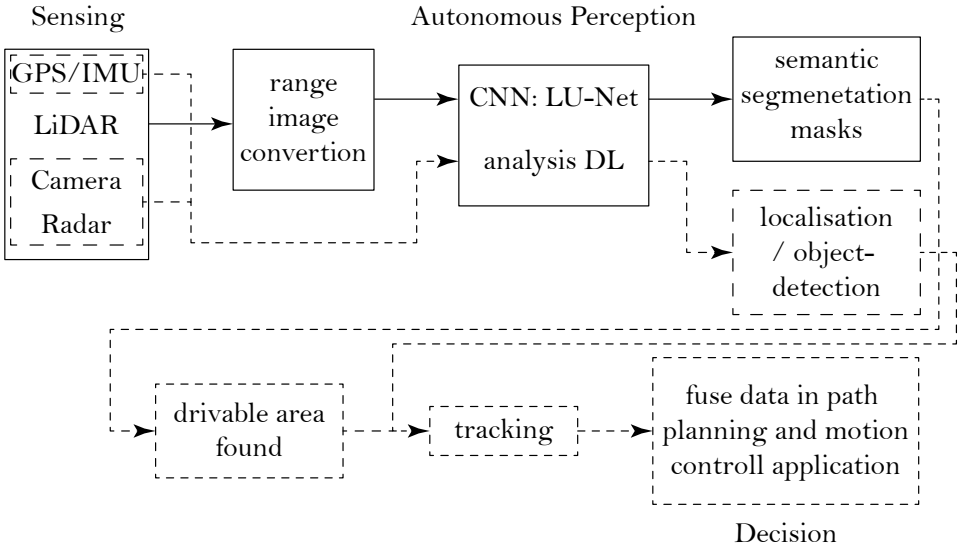


Figure 5.1: *Abstract model architecture, in multisensor application with additional tracking and control unit for identified objects.*

The choice of an appropriate dataset is discussed with respect to the underlying application of semantic lane segmentation. The generation of semantic classes from provided annotations is discussed and illustrated in detail. Here, semantic classes are derived from nuScenes dataset [5] to describe road-semantics in detail. The provided point cloud information is further projected to a 2d range-image. Hereby, state-of-the-art convolutional neural network *U-Net* is chosen as backbone architecture for processing converted point cloud data fast and efficient. A qualitative and quantitative evaluation of the model performance outlines the usability of suggested metrics for validating semantic lane segmentation.

5.1 Datasets

Throughout the last decade, several datasets have been published which had a large effect in further pushing developments of autonomous driving applications and substantially affected the work on object detection, tracking and segmentation tasks towards a full scene understanding.

Developments to further push detection could make huge steps forward in reducing error-rate and even exceeded human performance within the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [42, 48, 43, 37, 23].

Besides solving object detection, new and ambitious challenges arose during the past years with the demand of larger and much more detailed datasets. Several recently published datasets have been investigated, such as [6, 5, 8, 2, 12], which are in the position to provide detailed and annotation-rich samples to support semantic and scene understanding.

The well-known KITTI [47] dataset is the pioneering project in providing multimodal sets of data combined a rich set of annotated frames (15k) from stereo-camera (frontal view as well as dense LiDAR point clouds). Investigating recently published datasets, where much larger quantities of category classes as well as annotations are available, it seems feasible to effectively train deep neural networks. The extended dataset, which also provides information such as rich annotated map-layers can be used to address semantic segmentation or instance segmentation.

The visual recognition challenges for object detection, semantic segmentation and others, early started in focusing image domain. These day's, there is a transformation, where data from a full sensor-suite is needed (camera, LiDAR, radar), for achieving full scene understanding. Therefore, data fusion gains from multimodal data to attain full environmental perception.

The following summary in Table (5.1) contains a selection of the most recent benchmarks, which have been published throughout the past years and gained a lot of popularity among the development of autonomous driving applications.

dataset	year	scenes	cam.	lidar	radar	ann. frames	3d boxes	night/ rain	map-layers	classes	location
KITTI	2012	22	15k	15k	0	15k	200k	N/N	0	**8**	Karlsruhe
Waymo Open	2019	**2k**	**1M**	**200k**	0	**200k**	**12M**	**Y/Y**	0	4	3xUSA
CityScapes	2016	-	25k	0	0	25k	0	N/N	0	30	50 cities
BDD100k	2017	**100k**	**100M**	0	0	100k	0	**Y/Y**	0	10	NY, SF
ApolloScape	2018	-	144k	0	0	**144k**	**70k**	Y/N	0	**8-35**	China
semanticKITTI	2019	22	0	43k	0	43k	0	N/N	0	19	Karlsruhe
Argoverse	2019	113	490k	44k	0	22k	993	**Y/Y**	2	15	MIA, PT
NuScenes	2019	**1k**	**1.4M**	**400k**	**1.3M**	40k	**1.4M**	**Y/Y**	**11**	**23**	MA, SG
LyftLevel 5	2019	366	323k	46k	0	**46k**	1.3M	N/N	7	9	Palo Alto

Y - Yes, N - No, NY - New York, SF - San Francisco, MIA - Miami, PT - Pittsburgh, MA - Boston, SG - Singapore

Table 5.1: *Overview, selection of commonly used datasets for development of AD related applications [5]. 1^{st} section: sets without labels for ground-layers; 2^{nd} section: sets without lidar frames, but rich diverse views from various scene setups; 3^{rd} section: split from KITTI Odometry dataset; 4^{th} section: rich annotated map-layers, rich in diversity.*

The 1st section *defines datasets which don't contain annotations for ground classes. Although the Waymo Open Dataset holds* $12M$ *annotated bounding boxes and up to 200k LiDAR frames, unfortunately, there are only labels of 4 object classes provided without considering ground-layers.*

The 2nd section *contains datasets which are not necessarily helpful for this approach, because there are no LiDAR frames available. Nonetheless, these sets provide very diverse views from different scenarios and cities as well as several environmental conditions. It also gives a better overview of the historical developments of datasets from the autonomous driving domain.*

The 3rd section*, shows semanticKITTI, the annotated version of KITTI Odometry dataset, for semantic segmentation purposes. The KITTI split provides rich annotations for LiDAR frames as well as ground layers but does not contain samples for night cycles or different weather conditions.*

The 4th section *mainly shows sets containing rich annotated map-layers, which are especially useful for semantic segmentation of ground layers. The nuScenes dataset contains an overall number of 400k LiDAR point clouds with 40k annotated keyframes and various weather conditions as well as day and night cycle.*

Such extended and highly detailed sets of data are only available partly yet. In the past years new datasets arose such as: *Argoverse* [6], *NuScenes* [5] and *Lyft Level 5* [8], to contribute: complex and large-scale multimodal setups, covering 360°surround views from vision and range sensors, diverse scenery as well as richly annotated map-layer information, for rapidly pushing development of autonomous driving [5].

KITTI: [47], referenced in numerous papers, is a pioneering dataset in providing a multimodal setup of camera frontal view and dense point cloud LiDAR annotations (15k frames each). In autonomous driving it is one of the most widely used datasets. Also, other datasets such as *Cityscapes* [31] gained attention by providing fine segmentation labels from various city scenes.

Due to the rapidly increasing performance of visual recognition algorithms as well as evolving pace of autonomous driving applications, the need for extended and much more complex benchmarks arises.

Argoverse: [6] counts a 113 scenes setup with hd semantic maps recorded in locations of Miami and Pittsburgh. The sensor setup includes 2 roof-mounted 360°, 32 beams LiDAR sensors providing on average a ≈107,000 points point cloud at a frequency of 10 Hz. The ground-layer herein contains information such as: lane centrelines as well as traffic direction, ground-height and annotations for intersections. Polygons for road- or individual lane-layers are not available directly. Moreover, the width of a lane is determined through an average lane-width (separately defined for each scenery). Unlike to nuScenes (provides a binary rasterized 2d map in top-down view without graph structure for lane segments), Argoverse includes a 3d vector-map with ground-height and lane centreline connectivity [6].

NuScenes: [5] provides a broad set of annotations counting a 1,000 scenes setup from two very diverse cities, Boston and Singapore, which also includes several weather conditions as well as day and night cycle, similarly to [6]. The dataset contains $40k$ annotated keyframes and provides 11 semantic classes as a vectorized map in a 2d top-view. Annotations of vehicles are seen at distances of up to 80m from ego-vehicle [5], where the roof-mounted 360°and 32 beams LiDAR sensor produces a

point cloud with ≈35,000 points on average and 20 Hz spinning frequency. [5] contributes a large-scale multimodal dataset with a full sensor-suite across all vision and range sensorics (camera, radar and LiDAR) among rich semantic maps, which do not provide other datasets except [6] and [8]. Compared to [6], nuScenes map layers don't provide level of ground-height nor contain a graph like structure showing lane connectivity which could prevent additional work of pre-processing for map automation. Therefore, detailed sets of map polygons (e.g. individual lane fragments) are provided which implicitly contain width of road- and lane-segments as well as start- and end-tokens to extract flow of traffic-direction [5].

Lyft Level 5: [8] very much relies on nuScenes [5], by using the same tree-like data format associated with tokens. The set has been recorded in Palo Alto and provides semantic maps with plenty of road segments. Considering the sensor-suite, the recordings have been undertaken within a full 360°FOV for both sensors camera as well as LiDAR (7 wide-field-of-view cameras and up to 3 LiDAR sensors in parallel (40- and 64-beam range sensors)). The average point cloud counts ≈216,000 points per frames. Furthermore, a map expansion contains up to 7 semantic classes with detailed map layers. Different to [5], where areas of intersections covered by single polygons, [8] provides lane segments along intersections. This allows decision making for multi-path planning. Further, annotations are more flexible where intersections don't get assigned a single label.

Considering features of the investigated datasets and the scope of this work, *nuScenes* dataset [5] has been chosen, which offers the most diverse set of samples and annotations from plenty of various scenes. The extended dataset with additional semantic map records should give an appropriate benchmark for evaluating semantic lane segmentation approach. Besides labels for ground layers, [5] is also providing bounding boxes for object classes, which could be important for a detailed evaluation for investigating semantic contextual information. However, it is still to find out if nuScenes can provide appropriate labels for ground-truth data. This is because, hd-maps are only available in a 2d top-view with the loss of z-level information. Furthermore, to extract information such as traffic-direction, each polygonal lane fragment must be processed individually, which could potential lead to collisions in flow of traffic directions among areas representing complex infrastructure.

5.1.1 NuScenes

Data-Format

The format of the nuScenes dataset comes with a *relational database*, this is different from traditional datasets, but makes accessing of data fairly simple within a unified structure [5]. All aspects of the dataset including annotations and metadata, calibration, maps, scenes, sensor-types, etc., are accessible via *token-identifier* and ordered in a tree-like structure. The annotations are sampled at a frequency of $2Hz$ and between keyframes interpolation is used in combination with odometry data from the test-vehicle to provide accurate labels. An annotated sample is herein described as a *keyframe* [5]. Accessing sample data from a specific sensor, the *sequence of token identifier* must be determined *(scene-token → sample-token → sample-data-token)*. The structure of the relational database is illustrated in Figure (5.2), showing an example of accessing sample-data from a scene record.

Further, detailed hd-maps with semantic classes are provided with accurate localization and ego-poses. Every scene snippet contains log data which refers to ego-positions for the

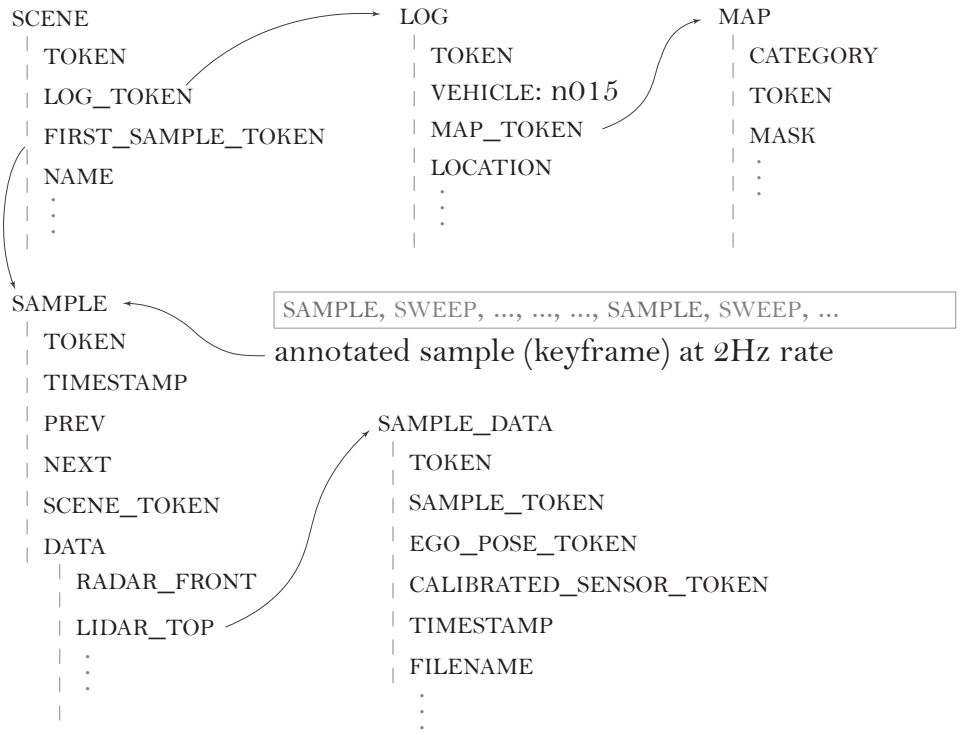


Figure 5.2: *NuScenes data format of the provided relational database. Illustrating sequence of token-identifiers to access sample-data.*

test-vehicle in global coordinates. The map data is herein provided as flat 2d binary mask from a top-view perspective without any z-level information.

NuScenes, as a multimodal dataset has been recorded with multiple of sensors at different rates of frequency. Achieving good sensor synchronization, data alignment between LiDAR and camera must be organized carefully. Considering that data annotations are provided at a frequency of $2Hz$, it is important that sensor frames share the same *keyframe* position, because of localisation and positioning of bounding boxes throughout streams from different sensors. Figure (5.3) demonstrates update order of LiDAR and camera frames, where it can be seen that only a few LiDAR frames share a position with a camera image.

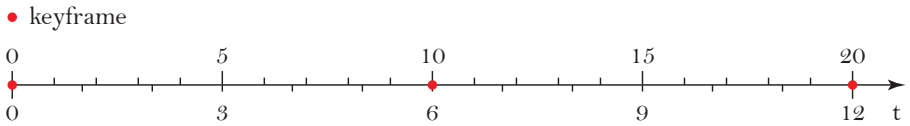


Figure 5.3: *NuSccenes data format, sensor synchronisation between LiDAR (20Hz) and camera (12Hz) [5].*

Furthermore, the dataset provides a broad spectrum of records from different scene locations as well as contains several environmental conditions. The data has been recorded in two cities, Boston and Singapore with a percentual proportion of: Boston 55%, Singapore 45% (One-North 21.5%, Queenstown 13.5% and Holland Village 10%). Different weather conditions such as rain (19.4%) as well as night-cycles (11.6%) are contained in the data, where this benchmark differs from other datasets which don't provide such information [5].

On average, there are annotations for 7 pedestrians and 20 vehicles per keyframe. Due to the very fine-grained classes, of 23 object classes and 11 semantic classes from map layers,

there is a large class imbalance existing of 1:10k. This is caused, because of some very rare classes providing only a few representative instances throughout the entire set of annotations, where other classes dominate with thousands of label examples. For training, this could be resolved by merging small collections to larger parent classes [5].

Coordinate-Systems

Localisation is a major concern in autonomous driving, where the map location of a vehicle needs be known as precise as possible. Several approaches performed localisation by using Global Positioning System (GPS) or Inertial Measurement Unit (IMU) data, which is prone to failure, as the GPS map outages and changes over time. Therefore, in the nuScenes dataset, a local localization method based on collected data from LiDAR sensor has been used to create a detailed hd-map, by applying point cloud registration in an offline step. Then, during recording, odometry information from the vehicle as well as LiDAR point clouds are used as measurement input for a particle filter algorithm (Monte Carlo Localization), to update position estimations. Herein, it is stated that a low error value of $\leq 10cm$ could be achieved, due to robustness of Monte Carlo Method [5]. The **estimated position**, as output from the localization algorithm, is provided as **ego-pose**, which contains the position and orientation of the **ego-vehicle** in global coordinates.

Position:	(x,y,z) in [m]	z-level always 0
Orientation:	(w, x, y, z)	given as quaternion

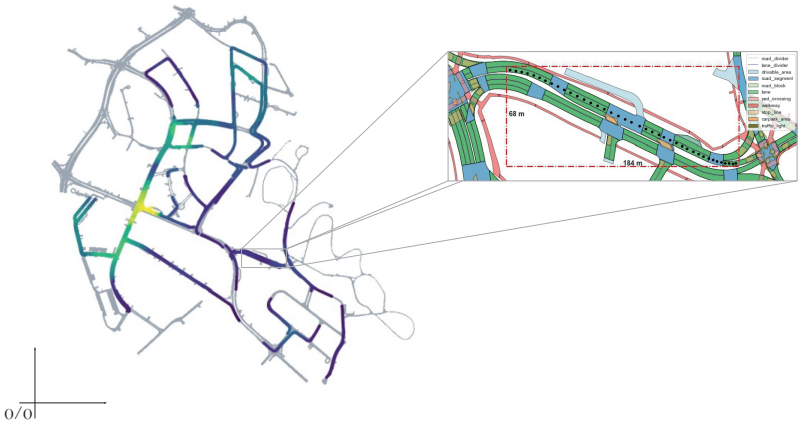


Figure 5.4: *Illustrating global to local coordinate transformation with nuScenes map example.* ***ego-positions*** *are marked with black dots within the zoom-view.*

For each *keyframe* an ego-pose record is provided with information of position and orientation of the ego-vehicle. The extrinsic and intrinsic parameters of calibrated sensors are given with respect to the ego-vehicle body frame in local coordinates [5].

Position:	(x,y,z) in [m]	
Orientation:	(w, x, y, z)	given as quaternion

For projecting point cloud data from sensor-frame to the semantic map layer, a global to local coordinate transformation is needed by using the extrinsic calibration parameters of the LiDAR scanner as well as ego-pose record. Rendering the sensor data is done in a top-down view perspective by default. As the sensor frame is not perfectly aligned in an up-right position, this might cause inaccurate representation of sensor data with respect to

the semantic map. Therefore, the map is rendered to the ego-frame of the vehicle (3d vehicle pose) by using *"flat vehicle coordinates"* (flat vehicle poses), which is aligned in parallel to the global z plane. Projecting between 3d vehicle poses and *flat vehicle poses*, the parameters of the calibrated sensor and ego-pose records are used [5].

Development Kit

The nuScenes datasets differs from other benchmarks by providing a relational database which can be used in combination of a python implemented development kit to effectively work with the data structure. Further, the development kit provides plenty of tools to render and process sensor data as well as polygonal information from semantic map layers. Herein, to easily access and visualize data, jupyter notebooks are provided which can be used in any web-browser.

Geometric information from semantic map layers is mainly provided as polygons, from a 2d top-view perspective. Processing complex polygons in combination with LiDAR point clouds can be computational expensive and requires a lot of computing power. Herein, any geometric shape from the nuScenes dataset is provided as *shapely object*. *Shapely* is another Python library, providing plenty of tools to process geometric shapes. Unfortunately, most of these functions are provided as CPU implementation and not optimized for speed to process large quantities of data. Therefore, pre-processing of nuScenes map layers turned out to be very time consuming and infeasible. This motivated the implementation of GPU accelerated versions of polygon queries, which is outlined in (5.1.2) and (6.1.2).

Further, the original version of the dataset (v1.0) contained several blacklisted scenes (46 in total) with inaccurate mapping of ego-pose on map layers. Here, a slight offset was noticed between LiDAR scans and the actual map layers. This has mainly been fixed in update (v.1.1), where only 3 scenes remained blacklisted *(scene-499, scene-515, scene-517)*. Considering the generation of ground-truth data, this offset led to false labelling and imperfect annotations, which could not be used for training.

Generating Ground-Truth

In the scope of this approach, the nuScenes dataset has been used for ground-truth generation which provides a multimodal sensor setup comprising 40,000 annotated LiDAR keyframes from 1,000 scene samples. Keyframes have been annotated by a frequency of 2Hz. Here, keyframes only have been used for the generation of ground-truth. This treatment of using annotated keyframes only mainly benefits from processing data of a larger spectrum of different scene samples, which adds more variation to the split for training and validation data.

The distribution of provided ground-truth samples is illustrated in Figure (5.5), which shows occurrence of single LiDAR pings from defined semantic classes (radial distance and angular component wrt position of ego-vehicle). As a mid-range sensor, the Velodyne HDL-32E provides a radial detection range of up to 100m. Most of the sample points from chosen ground classes are gathered in the near-field of the detection range, farther distances contain much less data, as it had been investigated in (3.3) by considering traditional LiDAR devices with static scan pattern.

Figure (5.6) conceptualizes an abstract model for segmenting ground-layers with semantic meaning, where classes for *ego-lane*, *opposite-lane* and *walkway* have mainly been defined.

The generation of an appropriate ground-truth for semantic map classes from nuScenes dataset, comes with several difficulties. Associating single LiDAR points with an actual

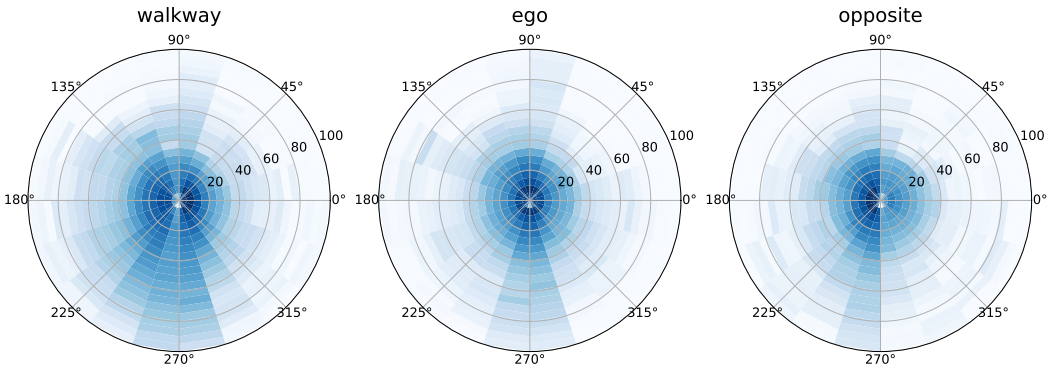


Figure 5.5: *Polar log scaled density map form annotated LiDAR pings from semantic classes. Labelled ground-truth data is mapped by using radial distance and angular component wrt position of ego-vehicle. Classes: walkway, ego-lane, opposite-lane.*

category label is difficult, hence semantic map layers are only available in a 2d top-view perspective, without providing any z-level information. Looking from a top-view perspective, this makes it harder to identify which actual point from LiDAR scanner belongs to a specific ground layer, as LiDAR pings from other classes, (e.g. vehicles, lamp-posts, overhanging parts of trees, etc.) can partly overlap parts from ground layers.

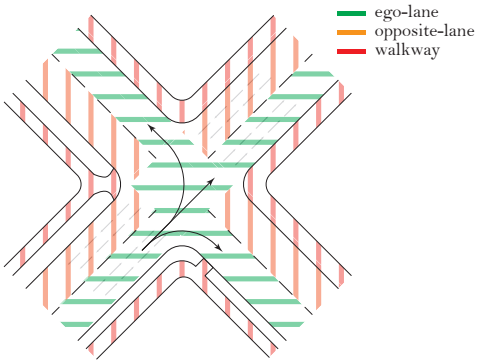


Figure 5.6: *Conceptualizing semantic lane segmentation containing traffic-direction. Three defined semantic classes (**walkway**, **ego-lane** and **opposite-lane**) are highlighted by polygonal patches.*

Ground is assumed as a flat and homogenous surface, which is usually seen for urban scenarios. Difficulties might arise by segmenting points from ground samples which belong to non-flat surfaces (such as road parts which are elevated, inclined or ramped wrt position of ego-vehicle).

There are several algorithms to pre-process point cloud information (such as filter techniques for ground-removal). Improving the generation of ground-truth by pre-processing point cloud data under the use of filter algorithms introduces threshold operations. Here, it is not guaranteed to process accurate labels throughout all scene samples which might lead to loss of information, if samples are processed inaccurately. Furthermore, threshold operations (by using classical algorithms) generally need to be adjusted manually, if conditions from surrounding environment would change too much. This introduce new complexity to

ground-truth generation by appropriately handling threshold values. Moreover, using filter algorithms to perform ground-removal, would require a detailed comparison to other methods if pre-processing clearly improves the generation of ground-truth data.

Therefore, a detailed pre-processing of point cloud data has been undertaken to minimize the influence of failure. Keeping pre-processing simple and efficient, the point cloud information has been tried to be reduced to a minimum set of remaining measurement values which are most likely to belong to ground layers. Herein, points belonging to bounding-box geometry and other points (which obviously cannot be part of ground layers, due to certain z-level height) are excluded and declared as part of background class. Considering the calibrated sensor-setup of the ego-vehicle, the position of the LiDAR sensor has been identified as approximately $1.8m$ above ground. Herein, looking from a top-view perspective, points which belong to a polygonal patch have been selected, where the position in z-level is used to separate ground from non-ground class labels.

In Figure (5.7), semantic map layers are illustrated with additional information of ego-poses (black dots) and bounding-boxes (rectangular shapes). The nuScenes map-expansion provides several semantic map layers *(drivable-area, road-block, road-segment, lane, walkway, stopline, etc.)*.

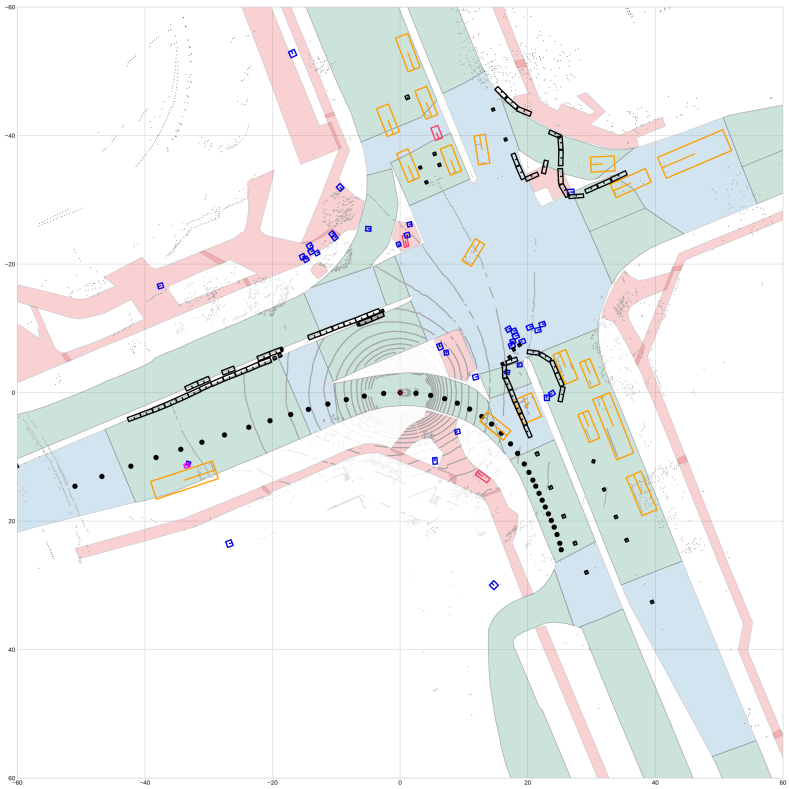


Figure 5.7: *NuScenes, semantic map layers containing ego-poses and bounding-boxes, (scene-0061) [5]. Several objects and markers are visualized: polygons of road-segments (green), intersections (blue), walkway (red); sequence of ego-poses (black dots); bounding boxes from object classes: vehicles (orange), pedestrians (blue) and movable objects (black); identifiers for traffic direction marked in yellow (from_edge_line_token) and blue (to_edge_line_token) [5].*

For the purpose of semantic lane segmentation, polygons from road-segments (green) and intersection areas (blue) need to be associated with corresponding *traffic direction* for

enhancing lane-semantics. This is implicitly encoded in map polygons by the use of a token-based system: *from-edge-line* → *to-edge-line*.

Correctly identifying traffic direction throughout all scene samples comes with difficulties. Considering situations containing complex areas of intersections and traffic-rules adds additional complexity, where latter is not explicitly encoded in nuScenes data. This is tried to outline in Figure (5.8), where areas of intersections are demonstrated with varying degree of complexity.

Without the use of global map information, the interpreting of the current scenario mostly depends on position of ego-vehicle. Hereby, it might be difficult to correctly identify direction of travel for road-segments throughout all scenarios. Introducing meaning of traffic-rules also adds further complexity. The training of neural networks might rely on falsely generated ground samples by not considering road regulations. Therefore, the model might learn an incorrect representation of the environment, which introduces a major difference in plain free-space estimation and semantic lane segmentation.

Providing multiple of alternative routes based on one current ego-position introduces multi-labelling and path planning. Hereby, the set of semantic classes would need to be extended to appropriately handle sub paths for intersection scenarios. Extending the set of semantic classes by classes for alternative routes, is very complex. Unfortunately, the nuScenes dataset is not providing such kind of semantic map annotations, where multipath labelling should not be a matter of this thesis and is left for future work.

Therefore, simplifying this more complex problem, a road part can either be classified as an area where the ego-vehicle is allowed to drive (without giving respect to traffic-rules) or where a road-segment shows opposite direction of travel, the ego-vehicle is not allowed to use. Furthermore, to avoid label conflicts intersections will be labelled as ego-lane too.

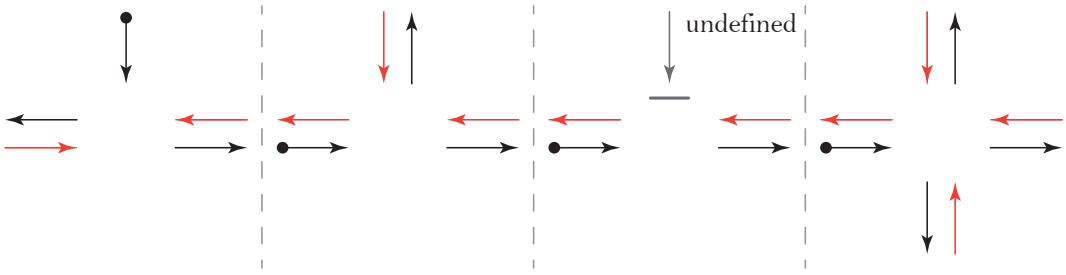


Figure 5.8: *Scenarios for road intersections with varying level of complexity. Interpretation of direction of travel mostly depends on ego-position and traffic-rules, where latter is not explicitly encoded in nuScenes data.*

Focusing areas of intersections, the nuScenes dataset provides plenty of road-segments, marked with the *intersection flag*, which is illustrated as map polygons (blue) in Figure (5.7). Edge line tokens from road-polygons implicitly encode traffic direction: *from_edge_line* (yellow dot)→ *to_edge_line* (blue dot).

To further derive the orientation of a road polygon, the coordinate system of every scene sample is aligned in such a way, that the forward position of the ego-vehicle points in direction of positive x-axis. Therefore, the identity vector $\vec{u} = (1, 0)$, is analog to the driving direction of the ego-vehicle. With the help of the law of cosines Equation (5.1), vectors $\vec{u}$ and $\vec{v} = to\_edge\_line - from\_edge\_line$ are used to find the actual orientation of a road-segment wrt ego-pose:

$$\alpha = \arccos \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| * \|\vec{v}\|}. \tag{5.1}$$

5.1.2 Parallel Computing Complexity

Computing ground-truth data for LiDAR point clouds can be computationally expensive. The average size of a point cloud in the nuScenes dataset is denoted as 35,000 points. Further, polygons from map layers are complex and contain up to several thousand of vertex positions.

Processing point in polygon queries, the *crossing number algorithm* [46] is used, to determine whether a point lies inside or outside of a polygonal shape, demonstrated in Figure (5.9). The algorithm simply defines a line for each point p of the point cloud, with the coordinates (p_x, p_y) and $(p_{+\infty}, p_y)$. Next, all intersections with edge lines are counted indirectly with the help of a switch variable, which is alternating between values 0 and 1. Herein, 0 represents even number of intersections, which means a point does not intersect with polygonal area, respectively 1 results in an odd number of intersections, where the point lies inside the polygon [46].

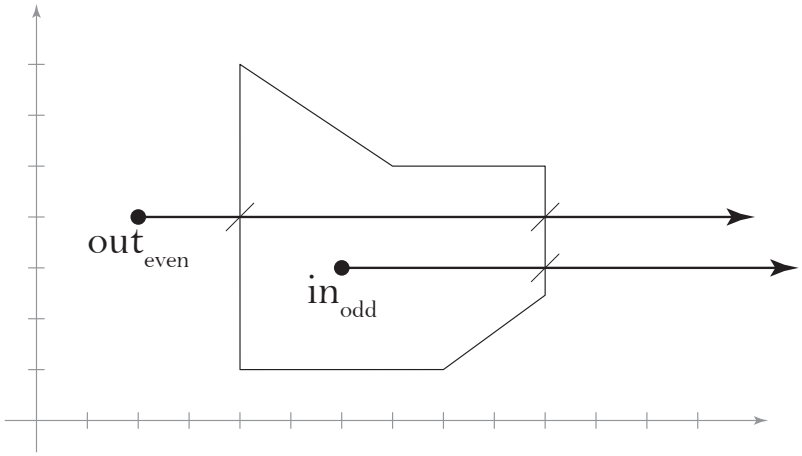


Figure 5.9: *Showing base concept of crossing number algorithm [46]. The algorithm determines intersection of point and polygonal area. Point lies within polygon (counting odd number of intersections); point lies outside the polygon (counting even number of intersections).*

A theory of parallel computing complexity describes the benefit of parallel execution measured as speedup compared to runtime of sequential execution.

There are several quantities to measure the speedup of a program if serial and parallel execution is compared. A very common metric describes *latency*, measured in *execution time* or *instructions per cycle*. Herein, latency describes the execution time t_1 of a procedure with respect to timing t_2 of the parallelized program. On the other hand, instructions per cycle defines the throughput of the architecture. Here, no parallelization work has been undertaken and speedup benefits from system architecture.

$$S_{latency} = \frac{t_1}{t_2} \tag{5.2}$$

In [56], Amdahl's law defines a theory of achieving large scale computing, where the overall runtime of a task benefits from parallel execution. Further, a task cannot be parallelized in full, where certain parts of an algorithm are executed just once, (e.g. for initialization purposes). Therefore, execution time t is given as part of *serial* and *parallel* runtime, $t = t_S + t_P$. Herein, the speedup is mainly limited by part of sequential runtime.

$$S_{latency}(s) = \frac{1}{1 - p + \frac{p}{s}} \tag{5.3}$$

Amdahl defined the theoretical speedup S as latency of the overall execution time. p is the fraction of the task which benefits from parallel execution. s is given as speedup with respect to the part which benefits from parallelization. Herein, it is stated that runtime of a parallelized task cannot be better than the serial part $1 - p$. Therefore, parallel computing, with plenty of processing units, mostly benefits from highly parallelizable tasks [56].

Super-linear speedup mainly describes cache effect, which is seen in modern parallel computing, where the speedup of S is larger than N, number of processors. This additional speedup is caused by different memory architectures and cache-level sizes, which reduces time of memory access significantly [54].

Expressed in big O notation, computing complexity of the crossing number algorithm for N number of points and M number of vertices from polygon, results in $\mathcal{O}(N\,M)$. An illustration of runtime complexity is provided in Figure (5.10) with respect to size of N, M and parallel factor $\frac{1}{k}$, where k is the number of available processing units. It is assumed that the number of vertices M is much smaller than N, where complexity can be simplified to $\mathcal{O}(N)$. Further, parallelization can improve complexity by a factor of $\frac{1}{k}$, $\mathcal{O}(\frac{1}{k}N)$. Runtime complexity would be reduced to $\mathcal{O}(1)$, if the number of processors is equal to N, if the task is fully parallelizable.

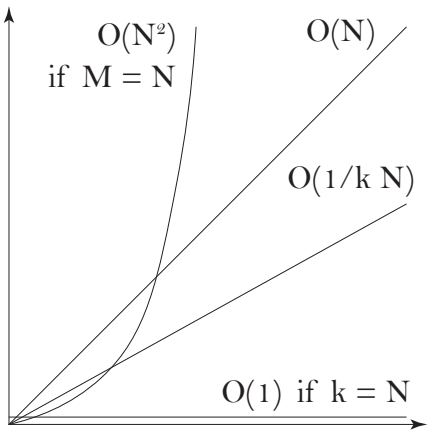


Figure 5.10: *Big O Notation, runtime complexity related to parallel computing. N size of point cloud, M number of vertices from polygon, k number of available processing units.*

5.2 Range-Image Generation

Point clouds commonly suffer a sparse representation and are understood as unordered set, which makes it harder to analyse if compared to an image with a well-defined grid structure. Pre-processing point clouds includes several advantages where the data can be transformed into a dense representation by converting it to the image domain. Further, transforming the data to a lower dimensional space makes processing much faster with the advantage of using already optimized neural network architectures which have been developed for the image domain. Conversion of data often comes with a loss of information, where the transformation suffers from discretization errors. However, approaches which use a conversion of three-dimensional data to a lower-dimensional representation, such as range-images or top-view perspectives, gained a lot of attention throughout the past years. Similar to an rgb-image with three colour channels it is possible to convert several information from the 3d space into a stack of multiple of feature channels, in such a way that no actual three-dimensional information is lost. The stack of feature channels is illustrated in Figure (5.11) where the coordinates (x,y,z), the intensity value of the range measurement (i), the calculated radius (r), the yaw-angle (ϕ) of the 360° surround view and the calculated labels from ground-truth generation are encoded (x,y,z,i,r,ϕ,l).

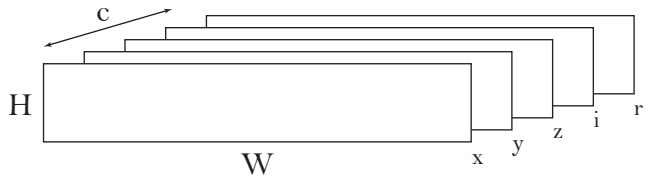


Figure 5.11: *Stack of feature-channels for range-image representation. Generated from 3d point cloud.*

Coordinates:	(x,y,z)	cartesian
Intensity:	(i)	magnitude of reflected pulse
Radius:	(r)	radial distance
Yaw-Angle:	(ϕ)	360° range
Class-Label:	(l)	0,1,2,3, ...

The average point cloud size in the nuScenes dataset counts about 35,000 points. Therefore, an adequate resolution of the underlying range-image needs to be chosen to avoid pixels which are occupied by several 3d points for multiple of times. Due to the Velodyne HDL-32E scanner, which provides 32 scanlines, the resolution of the range-image is determined by a size of *(1024x32)* pixels. The size of the image allows an absolute amount of 32,768 pixel positions, which covers most of the points from the average point cloud size and still renders the data in a dense representation.

To calculate a spherical projection from 3d cartesian coordinates (x, y, z) a transformation to spherical coordinates (r, θ, ϕ) needs to be achieved, as it is illustrated in Figure (5.12). Herein, the transformation is provided in such a way that the range-image is covered by angles from $(0°, \dots, 360°)$. The beginning and end position of the range-image, in a sense of 360° angular range, is basically determined by the position of the *spherical cut*. This decision mostly depends on the target application or the objective of the user and is tried to set to a point where it is less disruptive (here 0°).

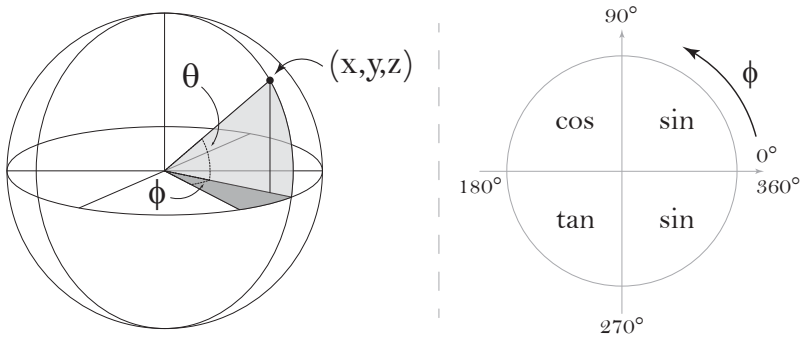


Figure 5.12: *Transformation of 3d cartesian coordinates* (x, y, z) *to spherical coordinates* (r, θ, ϕ)*. r - radial distance from ego-vehicle position,* θ *- inclination,* ϕ *- azimut.*

The radial distance is determined by the L2-Norm $\|(x, y, z)\|_2$, Equation (5.4), the euclidean distance between origin and position in 3d. Further, the angles in spherical coordinates θ (inclination) and ϕ (azimuth) are calculated by trigonometric functions Equation (5.6) and Equation (5.7).

$$r = \sqrt{x^2 + y^2 + z^2} \tag{5.4}$$

$$h = \sqrt{x^2 + y^2} \tag{5.5}$$

$$\phi = \arcsin \frac{y}{h} \tag{5.6}$$

$$\theta = \arccos \frac{z}{r} \tag{5.7}$$

The Velodyne HDL-32E scanner with 360° range provides horizontally and vertical angular resolutions of:

Horizontal:	$0.08° - 0.33°$	360° range
Vertical:	1.33°	41.33° range

Further, to associate the converted spherical coordinates with an actual pixel position from the 2d range-image, a discretization of the angular coefficients to width and height of the pixel grid need to be provided. Herein, the discrete pixel positions (u, v) are calculated with respect to the horizontally and vertical field of view as well as image size: $u = \frac{\phi}{\Delta\phi}$, $v = \frac{\theta}{\Delta\theta}$.

5.3 Training

Training of the nuScenes dataset comprises a 1,000 scenes setup with sequences of 20s. Due to the fact of ongoing detection and tracking challenges, ground-truth data for the test-split is currently not provided, where a smaller number of 850 scene samples remains for training and validation. Therefore, total number of scene samples have been divided in train/val split by a ratio of 4:1, with a resulting set of:

Training: 700 scenes
Validation: 150 scenes.

Within the scope of this thesis, [3] has been used to provide a convolutional neural network which has already been proven to perform well for the task of semantic segmentation. Herein, the entire point cloud is transformed into a range-image representation with feature channels (x,y,z,i,r,ϕ,l). ϕ defines the sensor's yaw-angle $0°, \dots, 360°$, to better handle segmentation of 360° surround views. This additional information is supplied to avoid any perspective ambiguities between front- and back-view, as it is explained in (6.2).The image resolution is set to (1024x32) pixels. Hereby, the image height is related to 32 scanlines from used laser scanner. The baseline architecture originates from U-Net [41], which is an encoder-decoder network and uses identity-skip connection to feed deconvolutional network with additional fine-grained information from feature encoding. Herein, it might be interesting to find out if segmenting the perspective projection of a point cloud, rendered to a dense representation, could still benefit from the use of an optimized architecture, which has recently been used in image processing.

Addressing issues affecting class imbalance, focal loss function Equation (5.8) (for more explanations about focal loss, see (3.1.4)) has been used in combination with Adam optimization with momentum (3.1.5).

$$\alpha_t = \begin{cases} \alpha & \text{if y} = 1 \\ 1 - \alpha & \text{otherwise} \end{cases}$$
$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log p_t \tag{5.8}$$

Hereby, classes counting only a few samples are getting up-weighted during process of gradient optimization, whereas categories with a much bigger sample size will be down-weighted, to less affect the detection of actually *interesting category classes*. From [24] the focusing parameter γ is set to default value of 2. Adam optimization is used with the recommended default parameters for learning-rate ($\epsilon = 0.001$) and decay rates ($\beta_1 = 0.9, \beta_2 = 0.999$), which are widely referenced in literature [32].

5.4 Semantic Lane Segmentation

The characterisation of the performance of a sensor model is a hard challenge and depends on many factors as it has been explained in (3.3). Describing model performance in general, it is tried to quantitatively evaluate class wise detection scores, which is summarized as box-plot and violin-plot. Further, multilabel confusion matrix is used where multiple of metrics can directly be derived from (e.g. *true positive rate* (TPR) and *false discovery rate* (FDR)).

The number of class samples of provided ground-truth *(ego-lane, opposite-lane, walkway)* is unbalanced by a factor of 4, considering the even larger size of background samples, there is an imbalance of (**1:7**), compared to ground classes:

Background:	75 mio.
Ego-Lane:	39 mio.
Opposite-Lane:	10 mio.
Walkway:	10 mio.

Herein, a metric which addresses class imbalance is introduced: **class wise intersection over union**, which is one of the most widely referenced metrics in literature to measure performance of semantic segmentation:

$$IoU_c = \frac{TP_c}{TP_c + FP_c + FN_c}. \tag{5.9}$$

Further, mean intersection over union ($mIoU$) is formed by calculating the average over all IoU_c values, per class:

$$mIoU = \frac{1}{|C|} \sum_c IoU_c. \tag{5.10}$$

Class imbalance is a general problem, which addresses both the evaluation of model performance and training of a network. Herein, the loss function should be in the position to minimize error and to maximize the used performance measure. By not covering class imbalance, a network might converge to a solution which is suboptimal and not in the position to appropriately predict any category classes, *(e.g. ego-lane, opposite-lane, walkway)*. In the scope of this approach, focal loss has been used to address the issue of class imbalance, see (3.1.4). Herein, it is tried to apply weight to the set of classes. Hard samples (from classes with only a few examples) are getting up-weighted for having a more direct influence on the optimization of the network parameters, whereas easy samples (from classes with a lot of examples) getting down-weighted, to have less influence to process of training. This treatment of up- and down-weighting is mainly motivated by the fact, that the number of actual *interesting samples* from category classes is much less compared to background samples. Therefore, by not handling class imbalance, a network might supply a perfect classifier for background information but cannot positively identify any category sample.

By the use of box-plot visualizations, important information of the underlying data can be rendered in compact form. Where basically five statistical values are considered to give an overview of distribution properties of the data (minimum value, maximum value, median (0.5 quartile), 0.25 quartile, 0.75 quartile), for more detailed explanation see (3.4). Furthermore, box-plot representation is not in the position to show modality of a distribution. Herein, box-and-whisker visualization is extended by the use of a probability density function, which is based on the sample wise IoU scores, where the modality (unimodal, bimodal, multimodal) of the distribution can directly be derived visually. By evaluating the classification performance of a semantic segmentation model, the modality of a specific distribution shows the ability of a classifier to distinguish between different category classes. Good performance should be achieved by unimodal or smooth bimodal distribution with a clear and single maximum.

Rendering results from classification in a multilabel confusion matrix directly offers class wise detection scores of defined semantic classes. Several metrics to describe performance of the provided model can directly be calculated from row and column wise operations, which has been explained more in detail in (3.4). Scores for TP, close to the value of 1.0, show that the classifier is in the position to well distinguish between the defined semantic classes, where occurrence of false negative (type II error) and false positive samples (type I error) is much lower compared to TP.

5.5 Model Evaluation

The characterisation of semantic lane segmentation model is very challenging and demands detailed investigations. Qualitative evaluation firstly highlights general performance by subjectively describing prediction results through the help of visual feedback. The influence of several environmental conditions is considered as well as different traffic scenarios from urban scenery, to show if the proposed model is in the position to appropriately distinguish between defined semantic classes.

Further, to investigate detection results of the proposed LiDAR application within a quantitative evaluation, the model performance can basically be described by the probability of four decisions:

Detection probability P_d is described by rate of true positives, where the predicted condition meets the actual true positive condition (**TP**).

False alarm probability P_{fa} is expressed as type I error, where the predicted positive condition fails to meet true positive condition interpreted as rate of false positives (**FP**).

Probability of miss P_m is defined as the complementary probability of P_d, defined as $P_m = 1 - P_d$, which is also called type II error and describes when the prediction fails to correctly identify actual true class value as false negatives (**FN**).

The probability of the remaining set of samples $(1 - P_{fa})$ which is labelled to not belong to the positive class and which is correctly predicted to be part of the negative class (**TN**).

The application of autonomous driving requires meaningful evaluation metrics. Providing appropriate performance indicators, *true positive rate* TPR as well as *false discovery rate* FDR is considered wrt radial distance as well as number of available LiDAR pings per class sample.

$$TPR = \frac{TP}{TP + FN} \tag{5.11}$$

$$FDR = \frac{FP}{FP + TP} \tag{5.12}$$

Further, distribution of local errors is investigated, to determine if the network prevents to falsely identify larger patches of entire road-segments, measured as local metric of TPR_l and FDR_l respectively.

6 Experiments and Results

6.1 NuScenes Dataloading

6.1.1 Implementation Details

The nuScenes dataset provides a relational database to effectively access and process sample data. The preloading of the database is supported by the nuScenes development kit. Herein, scene samples and map locations are encoded in a hierarchical manner, which can easily be accessed by the use of json-file format. Further, nuScenes provides several versions of the dataset. Herein, for the purpose of testing, *nuScenes mini* comprises a smaller selection of scenes from the much larger train and validation split of the entire dataset. By working with scene samples from the train and validation split, it is important to notice the stack of blacklisted scenes, in version (v1.0), which suffer from inaccurate mapping of ego-pose locations and therefore, causes generation of imperfect and slanted ground-truth samples. This has been improved as far as map-update (v1.1) provides new and rectified scene samples for all the blacklisted scenes, but three *(scene-499, scene-515, scene-517)*.

Generating labels for semantic ground classes has been seen to be complicated, due to the loss of appropriate z-level information, where map-data is rendered in a pure 2d flat representation from a top-view perspective. An extended map layer with additional information for level of ground height (as it is provided in [6]), should give appropriate improvement in generating annotation samples for semantic ground classes.

To limit the possibility to falsely annotate point samples, a pre-processing has been undertaken. Therefore, to speedup label generation, all parts of a point cloud, which obviously cannot belong to ground, are excluded and declared as background information. This included exclusion of point samples from object classes, which are encapsulated in bounding box geometry. Further, it is assumed that ground is defined as a flat and homogenous surface. Therefore, point data larger than a certain height in z-level is excluded and declared as part of background class.

Providing computation of traffic-direction is conceptualized in Figure (6.1) by the use of polygonal edge line tokens.

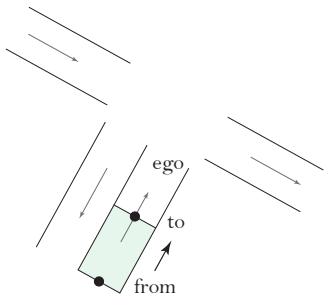


Figure 6.1: *Conceptualization for computing traffic direction. (from → to) edge-line-tokens used to derive polygonal orientation wrt ego-pose.*

The following computation of traffic direction for road-segments is achieved by applying the law of cosines to vectors $\vec{u}$ (driving direction of ego-vehicle) and $\vec{v}$ (orientation of road-segment). Herein, vector $\vec{v}$ is treated as position vector, as it is illustrated in Figure (6.2). The corresponding angle α indirectly determines the category label for the

road-segment. Determining direction of travel within straight road scenarios, angular bins of $45°, \dots, 0°, 360°, \dots, 315°$ and $135°, \dots, 225°$ are used, which should be sufficiently to describe ego- and opposite-lane direction. Due to areas of intersections, ranges between $45°, \dots, 135°$ and $225°, \dots, 315°$ need to be occupied twice.

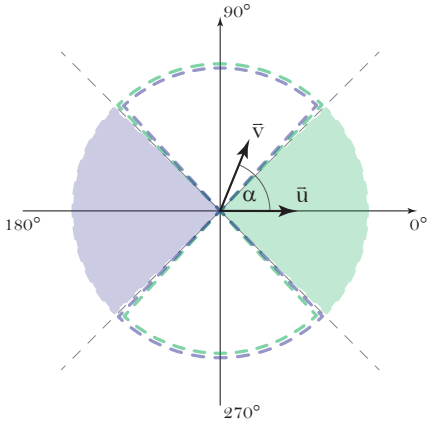


Figure 6.2: *Determining traffic direction from vectors $\vec{u}$, $\vec{v}$. Orientation of road-segment falls into predefined angular bins.*

The generation of ground-truth for an entire point cloud is computationally expensive. Hereby, non-road related LiDAR pings need to be separated from semantic map layers. Therefore, a point in polygon query decides whether a measurement point intersects with some polygonal map area. Geometrical shapes from nuScenes map layers are complex and contain up to multiple of thousands of vertices, where a single scene sample provides up to several hundreds of map polygons. Reducing computational overhead to a minimum, a thoughtful selection of map polygons must be found.

Applying the point in polygon search to an entire point cloud, where the point cloud contains many more points compared to polygon vertices, turned out to be infeasible if processed on CPU. To overcome drawbacks relied on CPU procedures, a GPU accelerated workflow was provided, with a speedup of more than 99%. Details of Crossing Number Algorithm and experimental results from parallelized execution are described in (6.1.2) and (6.1.3).

6.1.2 Parallel Computing Complexity

Executing point in polygon query has been implemented by the use of crossing number algorithm [46]. An example in pseudocode is given in Algorithm (6), which demonstrates the search problem.

Algorithm 6: Crossing Number Algorithm, pseudocode example.

```
Required: Polygon: 𝒫
Point of interest: 𝓅
switch-variable c: 0
number of vertices: n
for i = 0, …, n-1; j = n-1, 0, …, n-2 do
    if 𝓅 intersects with edge line from 𝒫 then
        c = not c
    end
end
return if c = 1, point 𝓅 is within polygon 𝒫, otherwise c = 0.
```

The average point cloud size in the nuScenes dataset is denoted as 35,000. Considering 40k annotated keyframes with complex polygons from semantic map layers turned out to be time consuming and infeasible to process in serial CPU mode.

Therefore, the implementation of a GPU accelerated version of Algorithm (6) was needed, to reduce the effort of execution time to an acceptable minimum. The implementation has been provided by the use of numba library (v.0.45.1), which defines a just-in-time compiler and allows parallel executing numpy objects for multiple of CPU or GPU processor units.

The numba library provides tools for copying variables and data structures (such as arrays) to GPU memory to enable fast memory access. Further, numba decorators (*jit*) are provided with interfaces to implement function signature and compiler instructions to convert python code to get executed on GPU.

6.1.3 Results and Evaluation

Figure (6.3) shows the results from label generation for semantic ground layers. The dataset provides a reasonable number of road-segments which are marked with the intersection flag. An intersection is a single polygonal patch and is not further subdivided into smaller parts for individual lane polygons. Hereby, it cannot be decided to apply either ego-lane or opposite-lane label without causing annotation errors. To resolve the situation in case of label conflict, intersection polygons are treated as ego-lane samples.

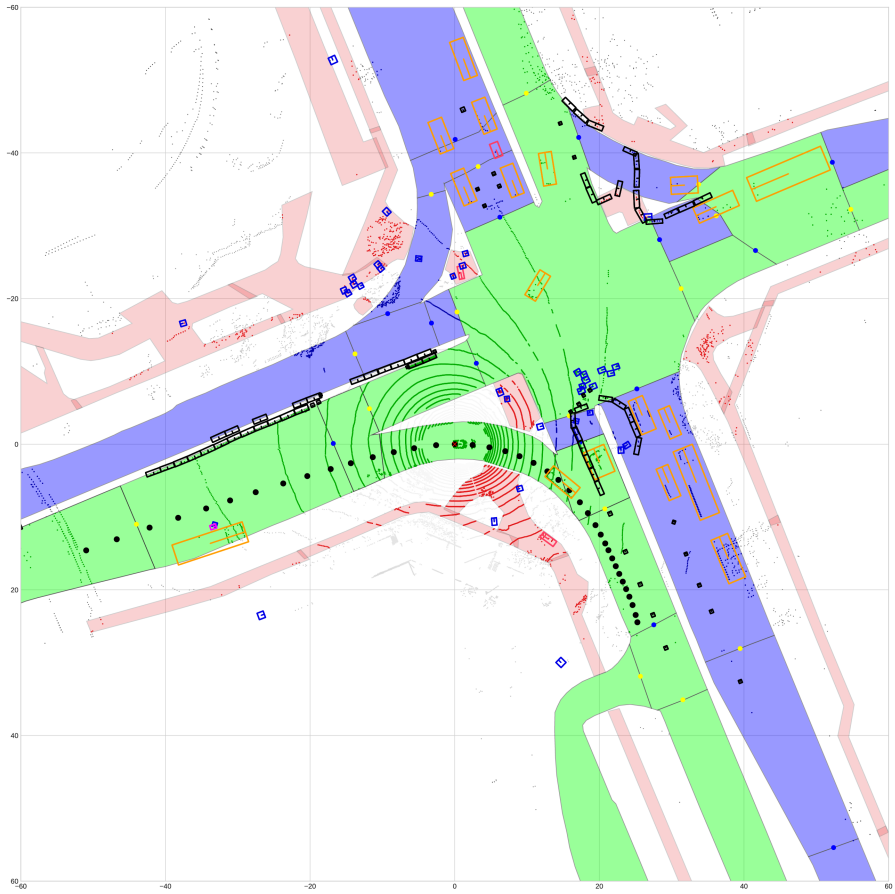
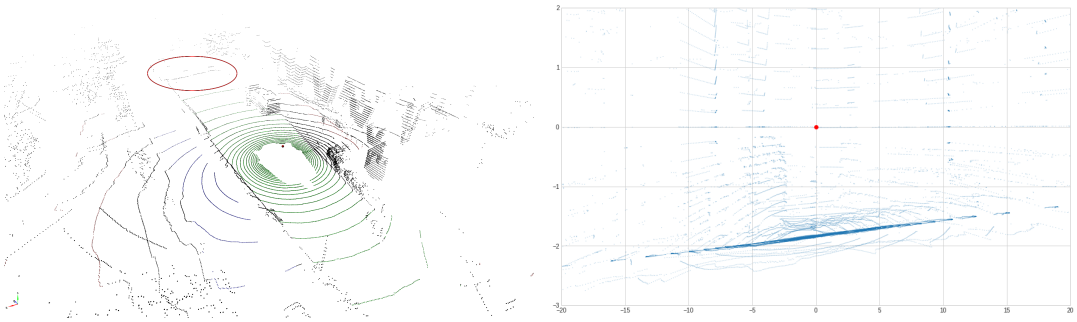


Figure 6.3: *Defined semantic map layers from nuscenes dataset (scene-0061) [5]. Category labels are defined as polygons ego-lane (green), polygons opposite-lane (blue) and walkway (red).*

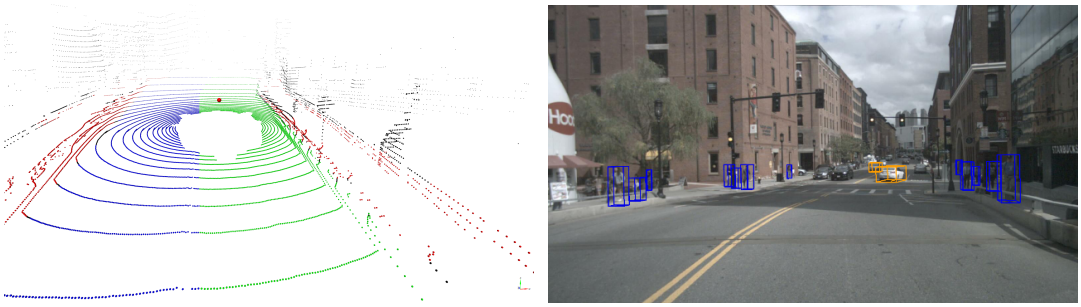
Ground is assumed as flat and homogenous surface, which is seen very often in urban scenarios. To provide appropriate labels from map layers it is difficult to handle non-flat, inclined and ramped parts of a road, where most ground pings don't necessarily lie within a surface parallel to xy-layer. The shape of an inclined road part is illustrated in Figure (6.4) (b). Assuming a flat surface of ground layers, label quality suffers from incorrect label generation where farther measurement pings from inclined road segments will be excluded due to increased z-level height, illustrated as red marks in Figure (6.4) (a). Furthermore, Figure (6.5) suffers from false labelling where several road-blocks have been falsely identified as class walkway (red points).



(a) Semantic classes, point cloud viewer.

(b) Side-view perspective, zx-plane.

Figure 6.4: *Showing semantic classes from generated ground truth. Additional side-view perspective is provided in zx-plane perspective, which is showing the silhouette of a non-flat ground surface (scene-0061).*

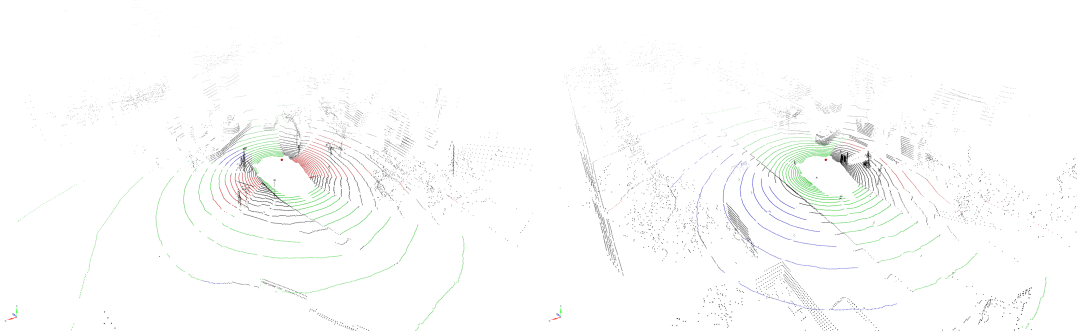


(a) Point cloud view.

(b) Camera image, front camera [5].

Figure 6.5: *NuScenes scene sample: scene-0103.*

In general, ground could be identified appropriately and pre-processing led to generation of correct labels. Examples are illustrated in Figure (6.6).



(a) Point cloud view. (b) Point cloud view.

Figure 6.6: *NuScenes scene samples: scene-0061.*

Parallel Computing Complexity

The computation of ground-truth data from the nuScenes dataset has been measured on both, CPU and GPU execution. Road geometry of semantic map classes is encoded in complex polygons, consisting out of multiple of thousands of vertices. Therefore, ground-truth generation has been seen very time consuming, on average 505.37 seconds for a single point cloud, computed on CPU. Considering a set 40k keyframes, from the nuScenes dataset, with an average of 10 complex polygons per scene sample, this turned out to be infeasible, where an accelerated implementation of the *point in polygon* query was needed. The results of CPU and GPU execution is demonstrated in Table (6.1). Compared to the execution on CPU with, 505.37 seconds per scene sample, the GPU implementation could achieve a speedup of about 700, which reduced the overall runtime to a fraction of CPU execution.

	time [s]
CPU	505.37
GPU	0.72
Numba impl.; v.0.45.1	

Table 6.1: *Processing ground truth data with GPU accelerated polygon query, implemented in numba python. GPU: GeForce RTX 2070*

6.2 Range Image Generation

6.2.1 Implementation Details

Similar to an rgb-image with three colour channels, the processed range-image contains several feature-channels *(x,y,z,i,r,ϕ,l)*. Avoiding loss of information by 3d $\rightarrow$ 2d conversion an arbitrary number of features could be stacked on top of each other which are used as input for a convolutional neural network to improve training.

The Velodyne HDL-32E laser scanner provides 32 channels of scanlines with a vertical offset of 1.33°. Each scanline is mapped linearly to indices from $0, \ldots, 31$. Processing spherical projection of the point cloud to discrete pixel positions, the horizontal and vertical angular range of the scanner (FOV_H, FOV_V) is used to determine discretization steps $\Delta\theta$ and $\Delta\phi$.

$$\Delta\phi = \frac{FOV_H}{image_{width}} \tag{6.1}$$

$$\Delta\theta = \frac{FOV_V}{image_{height}} \tag{6.2}$$

The angular projection of the range-image is chosen in such a way to map angles from $(0°, \ldots, 360°)$. The *spherical cut* can be set to an arbitrary position from $0, \ldots, 360$. Here, 0° has been chosen, where this marks a position which is less disruptive wrt the underlying application. Commonly, a mapping from ±180° is used to provide 360° spherical projections. The 0° mark is then right in front of the vehicle, respectively the ±180° mark lies right behind the vehicle.

The stack of feature-channels is extended by the use of yaw-angle ϕ of the spherical projection. Working with 360° surround views could possibly lead to problems such as *perspective disorientation*, which could have negative influence to training of a convolutional neural network. As it is demonstrated in Figure (6.7), objects from different perspectives will look similar in front- as well as rear-view of the LiDAR scan. As this approach focuses to provide semantic lane segmentation, loss of orientation might possibly lead to segmentation errors. Therefore, the yaw-angle (ϕ) is included to the stack of feature channels, where every position of the 360° range map is encoded with a unique angular component for horizontal field of view.

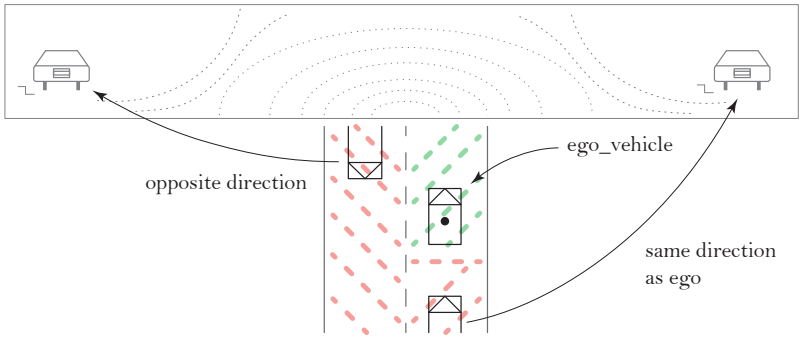


Figure 6.7: *Stress problem of perspective disorientation due to* 360° *surround view. Objects looking similar in front- and rear-view. Resolved by adding yaw-angle (ϕ) as additional feature channel to range-image.*

6.2.2 Results and Evaluation

After pre-processing the point cloud from sparse representation, data is now rendered to dense structure with spatial relationships indicating neighboured pixels. In Figure (6.8), the range-image is illustrated with the corresponding stack of provided feature-maps. Rendering the point cloud to a dense representation, it was expected to achieve a closed map without any gaps between pixels as every ping of the LiDAR sensor belongs to one unique pixel position. However, as it can be seen in Figure (6.8), the image contains plenty of black stains (noise-like), which actually don't belong to any LiDAR ping. This phenomenon is called *drop-out noise*, see Figure (6.9), which is caused due to mechanical error of sensor itself, environmental influences or if no signal reflection is registered. Further, *drop-out noise* has a significant influence on error of training [27, 19]. This is because parts of *drop-out noise* don't contain any information and need to be excluded to not influencing training of convolutional neural network. Excluding *drop-out noise* can be achieved with binary map mask, where 0 simply means no information is existing, 1 otherwise.

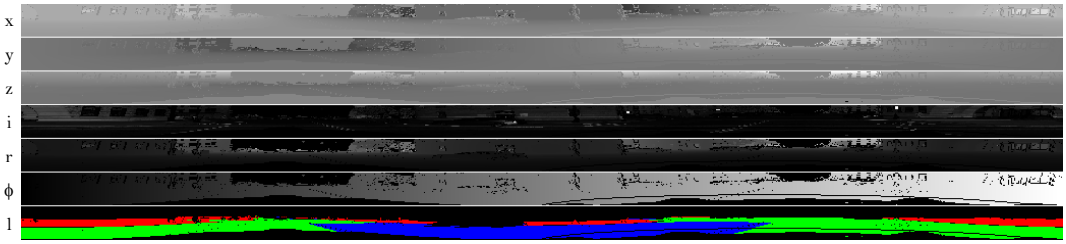


Figure 6.8: *Processed range-image from 3d point cloud with feature channels (x,y,z,i,r,ϕ,l) at a resolution of (1024x32) pixels.*

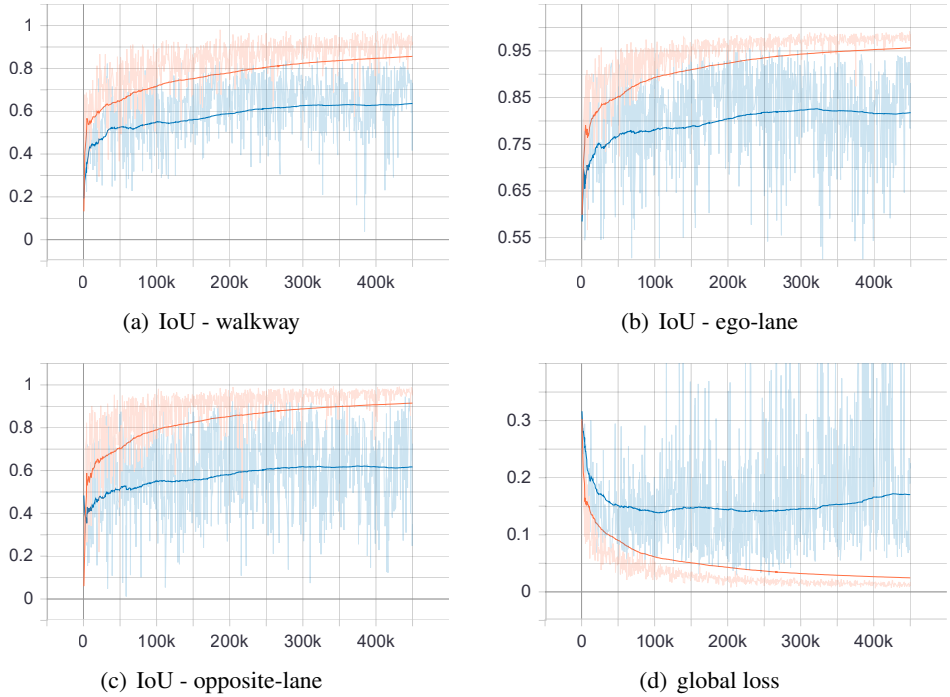
Figure 6.9: *Binary map mask example for corresponding range image from Figure (6.8). White areas indicate regions of drop-out noise, where no information is given from the scan device.*

6.3 Training - Results

U-Net has already been proven to perform well by processing projected point clouds in range image representation in [3]. The optimized network (U-Net) has been used without any further modifications to provide semantic segmentation in image domain.

Training is performed with single point cloud, where no temporal information was added by aggregating multiple LiDAR frames in time. Considering temporal aspects are important to maintain a continuous flow. This is especially helpful in estimating lane semantics, which could be demonstrated in [17]. As [3] did not included any temporal information, this will be left as a matter for future work.

The complete set of samples comprises approximately 34,000 annotated point clouds. Therefore, the *train/val* split contained *28,000/ 6,000 samples*. The overall training procedure counted 450,000 iterations with a batch size of **4**. Further, training has been stopped after 250,000 iterations (**35 epochs**), to prevent effect of overfitting from further increasing. Here, the global validation loss started to increase again and training loss continued to decrease. The checkpoint at 35 epochs showed training/ validation loss to be converged to a minimum of approximately **4%/ 14%**, Figure (6.10). Further network engineering, to better handle effects of overfitting, (e.g. including tuning of hyperparameters, regularization terms, dropout-layers, etc.) has not been considered within the scope of this approach. Therefore, this is left for future work to further improve network's performance.



(a) IoU - walkway

(b) IoU - ego-lane

(c) IoU - opposite-lane

(d) global loss

Figure 6.10: *Results of training procedure of LU-Net [3] performed on nuScenes data [5]. In orange, training loss and blue validation loss; class performance is measured as intersection over union (IoU): (a) walkway, (b) ego-lane and (c) opposite-lane; (d) global loss.*

6.4 Semantic Lane Segmentation Results

A quantitative evaluation of the provided model for semantic lane segmentation is considered. The validation split contains 6,000 LiDAR samples including various locations and environmental conditions.

The box-plot, which is based on scores of intersection over union (IoU, jaccard index) shows distributions in five-number summary. Considering dispersion for classes walkway, ego-lane and opposite-lane, it can be seen that the inter quartile range (IRQ) from ego-lane samples is much more compact and less varying (deviation from median value (orange solid line)) compared to classes walkway and opposite-lane. Further, range between minimum and maximum value of ego-lane distribution is almost the half if compared to ranges of classes walkway and opposite-lane, which shows that classification of ego-lane samples is more reliable.

Investigating the degree of modality of the underlying distributions, classification performance turned out to be unimodal with a negative skew (left skewed, as the mean values of the distributions (green dashed lines) are less than the median (orange solid lines)), Figure (6.11). It can be seen that the classification of ego-lane samples responds in clear unimodal distribution, whereas samples from walkway and opposite-lane classes show a trend towards smooth bimodal distribution. Hereby, the classifier is in the position to clearly distinguish between the defined semantic classes. Relative IoU score of ego-lane class is varying much less compared to classes of walkway and opposite-lane. That is because walkway and opposite-lane class suffers much more under larger ranges of dispersion if compared to ego-lane.

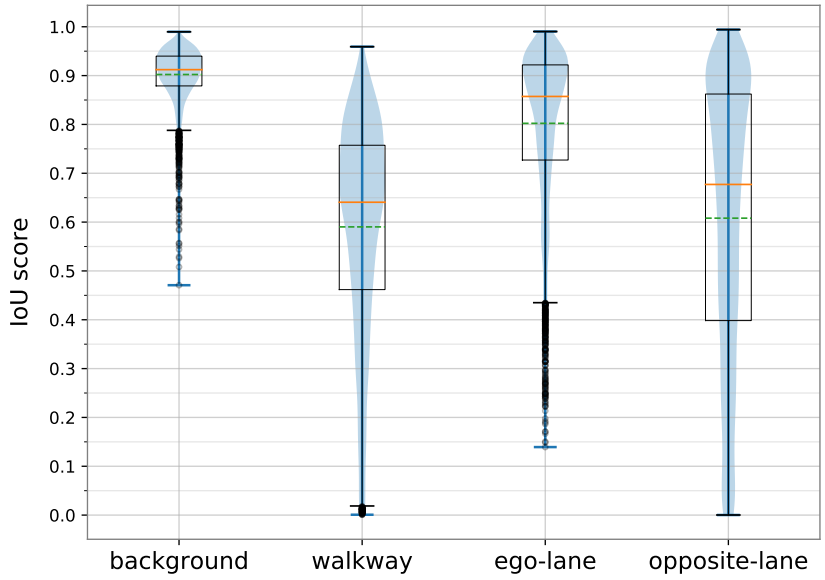


Figure 6.11: *Classification performance of proposed LU-Net. Illustration shows combination of box-plot and gaussian density function (blue), median (orange solid line), mean (green dashed line).*

IoU scores on the validation set result in a smooth-bimodal distribution for classes walkway and opposite-lane. This can be explained by the available number of sample data, where the number of ego-lane samples was four times the size compared to number of walkway

and opposite-lane samples (*1:4*). Further, the process of ground truth generation shows significant influence on the classification results of opposite-lane samples, which shows a very large dispersion with a range between minimum and maximum value of almost 1.0.

The generation of ground truth data was challenging and difficult to automate for the entire set of scene samples. Therefore, a perfect generation of ground-truth samples for the defined semantic ground classes could not be guaranteed. This error was generally caused by the provided set of polygonal semantic map layers, which were providing polygonal patches for several road parts, such as *road-segments* and *intersections*. Herein, a *road-segment* represents lane information only (either ego-lane or opposite-lane direction) whereas *intersections* can represent *road-segments* with a combined set of different lane directions (ego-lane direction and opposite-lane direction combined in one single polygonal patch). This led to the problem to either assign ego-lane label or opposite-lane label to intersection polygons, which inevitably reduces the number of ego-lane or opposite-lane samples. Focusing on the segmentation of lane samples, in case of label conflict, the ego-lane label has been assigned to intersection polygons. However, a perfect ground-truth, where polygonal layers are fully separated for each traffic direction, should result in much better detection scores for opposite-lane class. Due to the large size of the nuScenes dataset, and the amount of time for relabelling and retraining the segmentation network, this has been left for future work.

Further, rendering the prediction results in a multilabel confusion matrix directly offers class wise detection scores of defined semantic classes. In general, TP scores close to the value of 1.0 (main diagonal of confusion matrix, Figure (6.12)) show good classification of the model, where the trained network can distinguish well between the defined classes, as it has been explained in detail in (3.4) and (5.4).

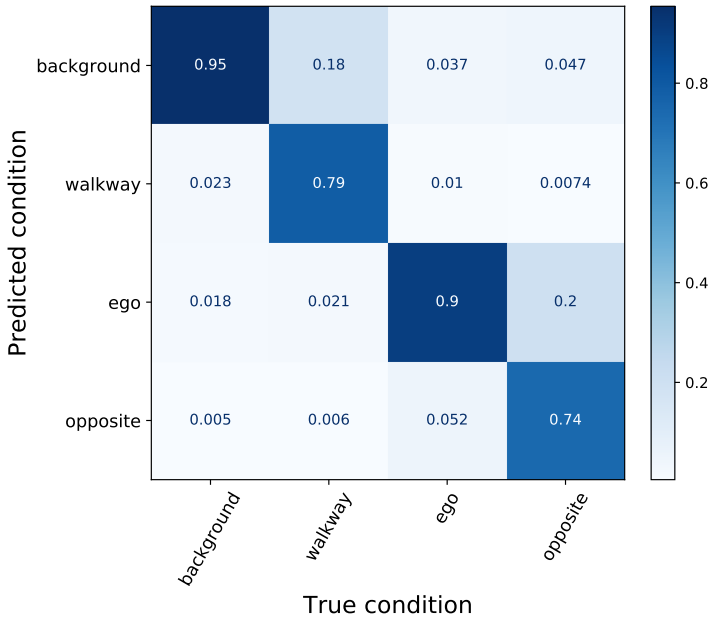


Figure 6.12: *Validation results of proposed LU-Net approach visualized by normalized confusion matrix.*

Performance for classification of the proposed model for semantic lane segmentation could achieve $\mathbf{IoU_c} = \mathbf{81.69}\%$ for ego-lane class. Samples which do not contain information of the provided semantic classes have not been excluded from testing procedures and could

lead to slightly better segmentation results for classes walkway and opposite-lane. Figure (6.13) shows the distribution of sample classification in TP, FP, FN and TN. Table (6.2) summarizes the results in class wise measures of IoU_c. Average IoU score remained at mIoU = 70.63%.

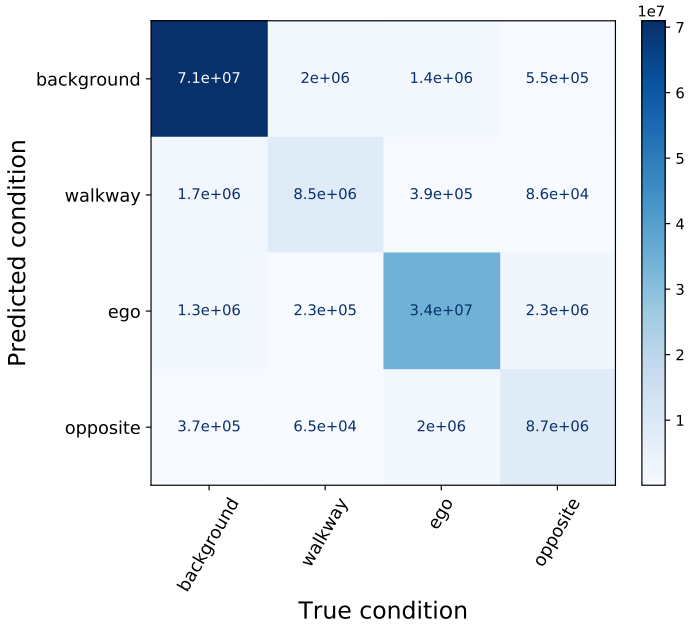


Figure 6.13: *Validation results of proposed LU-Net approach.*

	walkway	ego	opposite
LU-Net	67.77	81.69	62.43

Table 6.2: *Classification results of the proposed LU-Net model for semantic lane segmentation from nuScenes dataset, class wise score (IoU_c, %).*

Describing model performance by global metric or in a class wise manner, still lacks in level of detailed evaluation. No parameterized evaluations have been conducted nor model performance under different environmental conditions are considered. Sample wise comparison of segmentation results could allow to introduce a threshold value. This could be useful for real-world applications where a defined minimum level of quality is expected.

Therefore, a more detailed evaluation seems to be necessary to describe model performance more in detail.

6.5 Qualitative Evaluation

A selection of random samples showed good performance of the segmentation algorithm to well distinguish between the defined semantic ground classes. A detailed view of several scene samples and semantic map layers is provided in Appendix A. Here, the classification results of the introduced LU-Net approach have been projected to the camera front- and back-view for visualization purposes, to better investigate for quality of segmentation. Further, a top-view projection with additional map-layers directly points to false classifications of certain ground areas. The provided segmentation model appears to perform well under several environmental conditions (day, night, rain) as well as street scenes with varying degree of complexity, see Figure (6.14).

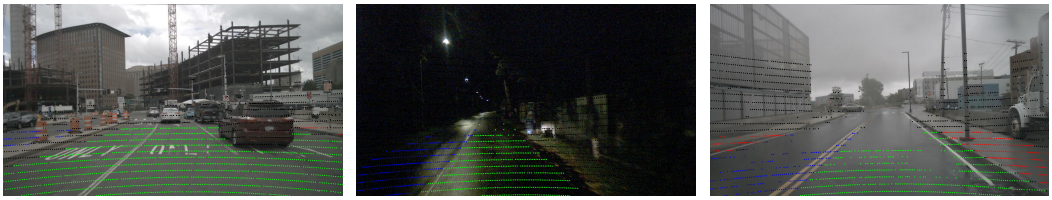


Figure 6.14: *Projection of LiDAR point cloud in camera image.*

day samples:	Figure (A.1) - Figure (A.8)
rain samples:	Figure (A.9) - Figure (A.12)
night samples:	Figure (A.13) - Figure (A.16)

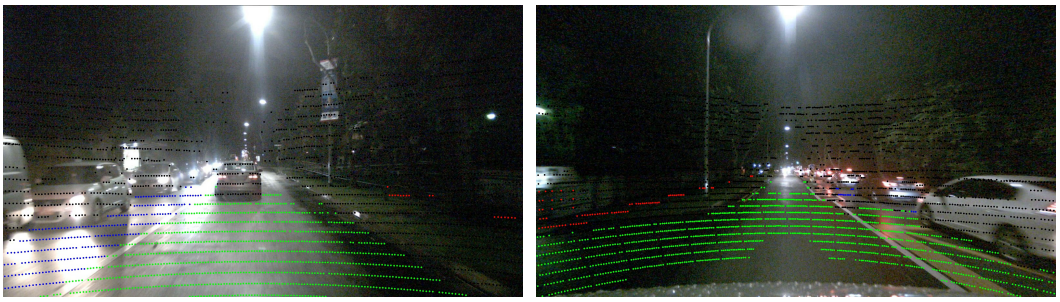
It can be seen that the appearance of geometrical features, e.g. curb stones or traffic-/ safety-islands, strongly supports the decision making process, where the segmentation feedback turned out to be almost perfect around such areas (see Figure (A.1), Figure (A.2), Figure (A.3)). In general, the separation between road and walkway is seen to be very well in almost all examples of the provided validation split, where confusion between classes of walkway and actual road-samples is very little.

Remarkable is the correct identification of road patches of the opposite-lane class. Hereby, even small parts of opposite-lane samples, of several meters in size, could successfully be distinguished from ego-lane samples (ego-lane as well as intersection patches), see Figure (6.15) and Figure (6.16). Figure (A.4) and Figure (A.15) with map-view.

(a) CAM_FRONT (b) CAM_BACK

Figure 6.15: *Projection of LiDAR point cloud in camera image (scene-0103).*

Further, it is assumed that intensity values originating from lane markings (e.g. stripes, arrows etc.) sufficiently separate lane areas which containing different traffic directions in



(a) CAM_FRONT (b) CAM_BACK

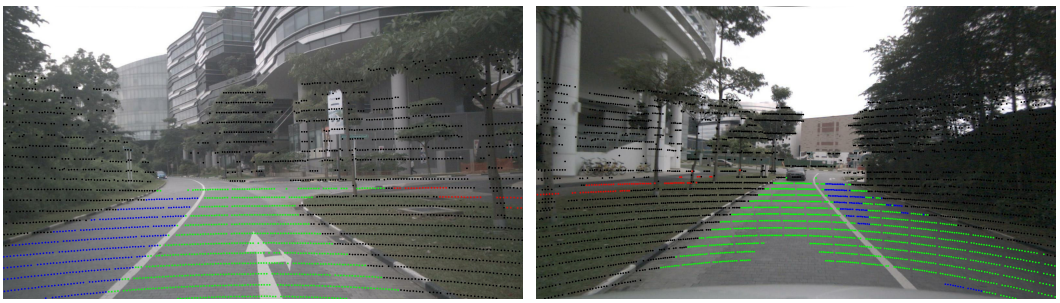
Figure 6.16: *Projection of LiDAR point cloud in camera image (scene-1070).*

several combinations of lane arrangements (ego-lane and opposite-lane, ego-lane, parallel-lane to ego and opposite-lane), see Figure (A.4), Figure (A.6) and Figure (A.13).

Intersection areas, in general, provide more contextual information to the network, where improved segmentation performance could be investigated, even due to increased degree of complexity within road scenario, see Figure (A.5) and Figure (A.10). In latter, in (a) parts of ego-lane segments are limited by the appearing of dominant areas of sections of walkways. Apart from traffic rules, which are not provided within the nuScenes dataset, the algorithm would have correctly classified ego-lane as well as opposite-lane segment. Further, Figure (A.15) shows ability of generalization of the network. Although, the ego-vehicle is directly passing an intersection area, due to oncoming traffic the network could identify the underlying road part as part of opposite-lane class.

The network could learn to also differentiate between fine grained details such as specific types of road-markings. In Figure (A.5) (b) area of ego-lane has been separated from lane markings very much in detail, whereas other types of lane markings and markings as arrows, from front-view perspective in (a), are also correctly classified as segment of ego-lane.

However, Figure (6.17) (Figure (A.2) with map-view) represents an example of false classification, where a larger area of the opposite-lane class has falsely been identified as part of ego-lane.



(a) CAM_FRONT (b) CAM_BACK

Figure 6.17: *Projection of LiDAR point cloud in camera image (scene-0269).*

Here, the surroundings of the scenery are simple without containing much contextual information. Although, road segments are clearly separated by curb stones from other parts of scene, the network could not manage to correctly identify segments from both classes. Figure (A.17) and Figure (A.18) show sample data from the same sequence in Figure (A.2). It is seen that errors mostly occur in back-view perspective, whereas front-view remains stable,

also see Figure (A.8). The segmentation quality shows appropriate results of opposite-lane segmentation during oncoming traffic and the existence of lane arrow markings.

In general, the provided segmentation approach showed several remarkable samples with good and very good segmentation quality. The network mostly fails in separating lane parts by refering semantic meaning to individual lane segments. It seems not to be fully clear for all scenarios, where to draw the boundary between ego and opposite direction of traffic. To further reduce confusion between lane classes, the definition of a separate class (for targeting intersection areas) is expected to increase performance of ego-lane and opposite-lane detection.

Further, from qualitative evaluation, it could be seen that confusion with semantic classes seems to appear more often in back-view ($-180°$) rather than front-view ($+180°$), where opposite-lane segments are often falsely identified as ego-lane sample. In Table (6.3), quantitative observation showed ego-lane segmentation to perform worse in rear-view by severeal percentages, whereas opposite-lane is slightly better in front-view perspective.

	walkway	ego	opposite
front	66.92	83.87	61.95
rear	64.14	79.43	61.37

Table 6.3: *Classification results of the proposed LU-Net model for semantic lane segmentation from nuScenes dataset, class wise detection score (IoU_c, %) for front- and rear-view (72° Field of View (FOV)).*

A separate training of the convolutional network could potentially improve segmentation results of lane classes by separately training front- and back-view. Hereby, it is expected to improve segmentation performance by dissolving effect of *perspective disorientation*, (6.2).

However, observing scene samples which could provide detailed contextual information, (e.g. vehicles, pedestrians, etc.), identification of lane classes has mostly been seen correctly, if compared to *empty* scenarios (samples containing very sparse point cloud information). Here, it is assumed that the neural network gained from detailed local knowledge to better identify actual class labels.

6.6 Quantitative Evaluation

6.6.1 Classwise Performance

Several aspects have been considered to investigate semantic lane segmentation in detail. Figure (B.1) shows the general detection performance of the classifier. Samples of semantic classes have mainly been recognized in the near-field of the sensor device (range $< 30m$). Fairly good detection performance can be seen for class walkway. Those samples could be separated well from lane space and are mostly false classified as background class, as it can be seen in Figure (6.12).

Figure (B.1) shows the overall distribution of classified sample data separated as number of true positive, false negative and false positive class samples. The sample space is scaled logarithmically due to visualization purposes. In the following, a sample is defined as a single LiDAR ping of a complete sensor frame. A sample frame respectively indicates a single LiDAR frame (point cloud) from the scan device.

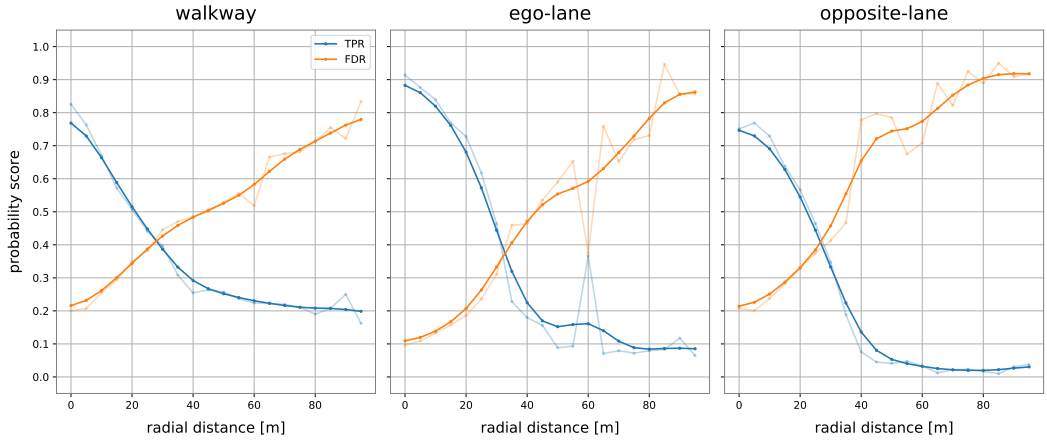
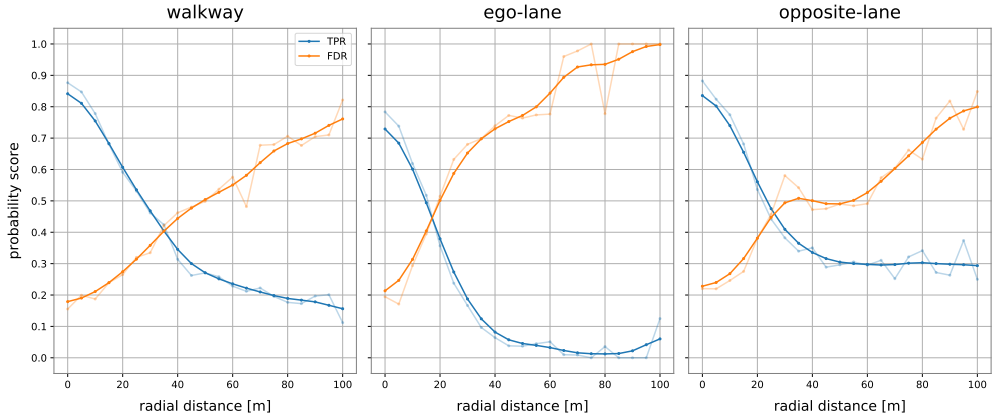


Figure 6.18: *TPR, FDR range measurements.*

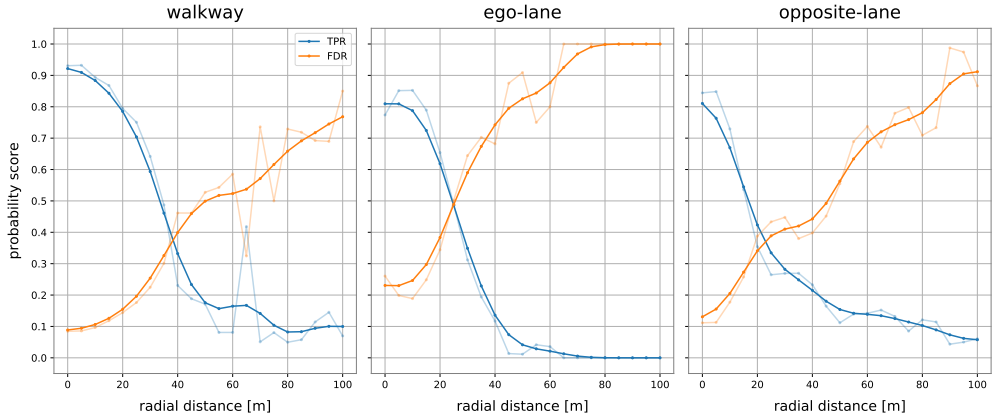
In general, Figure (B.1) shows samples identified as true positive gathered closer to the near-field of the sensor range ($< 30m$) for all classes. False negative as well as false positive classifications cover cells from farther distances. Considering lane classes only, it is interesting to notice that distribution of false positive and false negative samples counts a relatively large amount of false classifications along the rear-view ($180° - 360°$ range) of the scan view. In (6.2), the influence of *perspective disorientation* is emphasized which is assumed to have substantial effect to classification performance of 360° surround views. This is caused by a loss of orientation, where certain objects (e.g. vehicles, pedestrians, road segments) looking almost identical from several perspectives. Therefore, the integration of an additional feature channel as yaw-angle (ϕ) has been considered to dissolve conflicts of falsely assigning same class labels to multiple of orientations. Nonetheless, less accurate detection performance is investigated in parts of rear-view, which could indicate no appropriate compensation of the effect of *perspective disorientation* within training of network. Separating the point cloud in two clusters of front- and back-view segments might improve segmentation.

6.6.2 Environmental Conditions

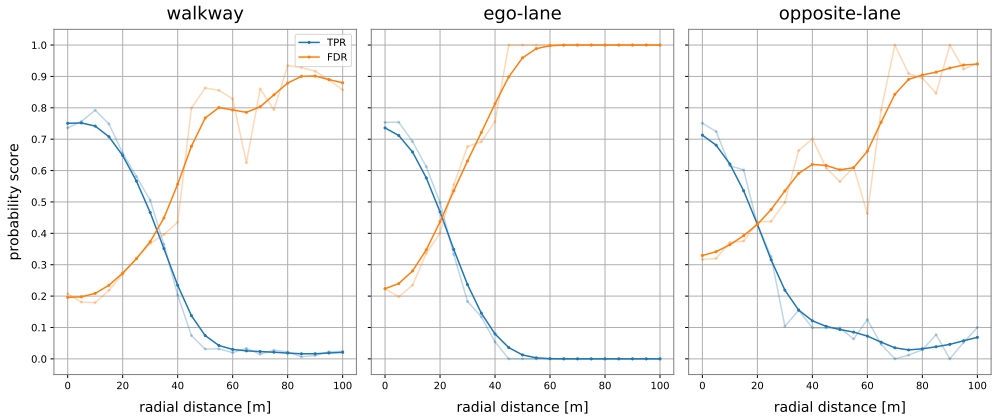
The influence of varying environmental conditions to the performance of segmentation has been investigated. In Figure (6.19) (a) - (c), it is seen that environmental conditions don't necessarily show direct influence on the segmentation results in the first place. Hereby, the *true positive rate* remains relatively stable if compared to day (a) and night (b) scenarios. Light detection and ranging defines its own light source, by definition, as it is emphasized in (3.3) and mostly suffers under the influence of direct sunlight, fog or rain. Considering the physical principles of range measurements by a LiDAR sensor, performance of segmentation shows to be better during night cycles by several percentages (3% - 8%). Further, in (A) Figure (A.9) - Figure (A.12) the performance of range measurements is demonstrated under rainy conditions. The response of measurements is clearly influenced by reflective properties of wet surfaces, where the point cloud information partly replies as coarse and sparse set of points. However, quality of segmentation seems to not suffer in the same way, where LiDAR pings still getting predicted correctly. Hereby, segmentation might depend on density of measurement points in general, rather than on quality of individual measurement pings. The overall probability of rain samples has been seen to be worse if compared to day and night scenarios by up to 12%.



(a) day sample frames



(b) night sample frames



(c) rain sample frames

Figure 6.19: *TPR, FDR wrt sensor distance and the influence of environmental conditions; (a) day, (b) night and (c) rain.*

6.6.3 Performance by Number of Class Samples

The number of LiDAR pings has been investigated wrt performance of segmentation. In the following, frames of validation set have been sorted by number of LiDAR pings per class per sample frame. Hereby, ranges have been defined to categorize frames with high and

low amounts of representative class samples. Detection scores, *true positive rate* (TPR) and *false discovery rate* (FDR), are used to quantitatively describe segmentation results.

In Figure (B.2) - Figure (B.4) the detection scores are visualized in box-plot representation with additional distribution of probability density in the shape of violin-plot.

Table (6.4) summarizes detection scores for groups of class samples. Hereby, rate of true positive remains relatively high (> 80%) by larger numbers of LiDAR pings, as it can be seen for all classes in Figure (B.2) - Figure (B.4) (a).

Class	TPR	# pings	TPR	# pings
Ego-Lane:	>50%	>4000	>90%	>11000
Opposite-Lane:	>50%	>2000	>90%	>5000
Walkway:	>50%	>2000	>80%	>6000

Table 6.4: *TPR by groups of class samples.*

In contrast to TPR, false discovery rate shows similar behaviour with results in reversed direction. In case of high amounts of representative class samples, FDR stays fairly low (< 20%) and increases by decreasing number of sample data.

Without considering any specific target, due to physical principles of the scan device, the number of overall measurement points on the target object is low for farther distances, on the other hand most measurement pings are registered if the target is close to the sensor. Segmentation in general could benefit by a sensor setup which is in the position to deliver high amounts of measurement values independent from radial distance. The static grid pattern cannot fulfil the demands from a dynamic environment, to dynamically change density of measurement values for several regions of interest. From above investigations, the segmentation result can be thought of to be much more reliable if there is a larger number of measurement values on the target object.

6.6.4 Performance by Radial Distance

Predicted class samples from (6.6.3) had been grouped by number of occurrence and showed to directly influence rate of true positive and falsely discovered samples. The scores showed to be increasing, resp. decreasing, by increasing number of measurement values.

In the following, performance degradation is considered wrt radial distance. In Figure (B.5) - Figure (B.7) classification scores are collected in bins of $5m$ in radial range which allows to categorize samples with labels located in near- and far-field. Generally, the detection probability in the near-field of the sensor was expected to be high and very good throughout all sample frames.

Here, the segmentation performance in the near-field of sensor space ($< 30m$) results in relatively small IQR, as it can be seen in Figure (B.5) (ego-lane) and Figure (B.7) (walkway). Segmentation performance for class opposite-lane, in Figure (B.6), mostly suffers from large ranges of dispersion in measurement values, which is assumed to be caused from class imbalance of the provided sample set (1:4). Hereby, the neural network most likely *could not derive* descriptive and *meaningful features* from sample space to appropriately identify opposite-lane samples.

In Figure (6.20) visualizes accuracy degradation in scatter plot representation. Scattering (blue) represents detection scores for single LiDAR frames, whereas second y-axis (orange line plot), shows the number of actual LiDAR pings, contained in one sample frame (arranged in descending order), see Figure (B.8) - Figure (B.13) for details. Especially for

sensor near-field ($< 30m$), TPR remains fairly high, > 0.8, for most of the sample frames. Further, segmentation performance for cells within close distance shows to be almost independent from actual number of LiDAR pings. This behaviour could introduce a certain **ping threshold**, when segmentation is always performing appropriately, if there is a minimum of required measurement values on the target object.

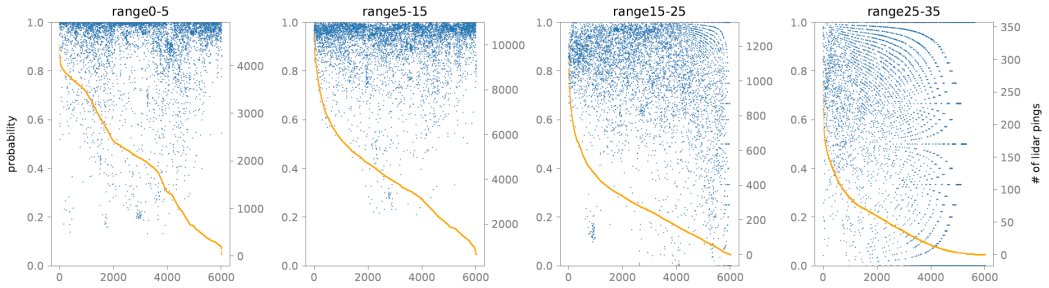


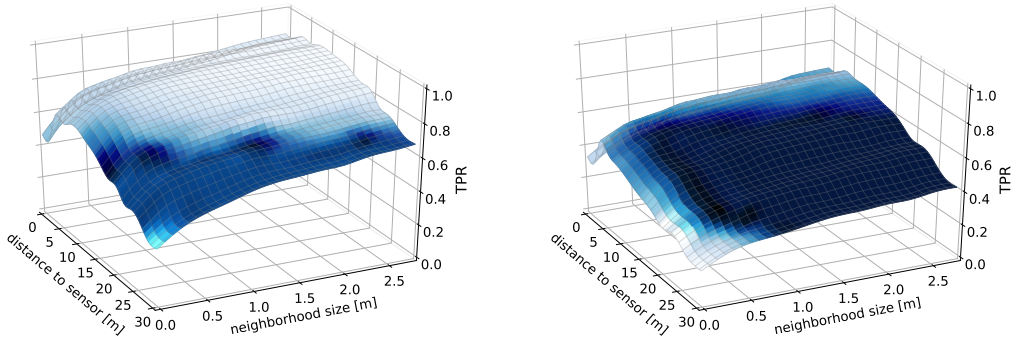
Figure 6.20: *TPR wrt sensor distance, Figure (B.8) for class* ***ego-lane****. Scattering (blue) shows TPR per LiDAR frame, line plot (orange) indicates number of LiDAR pings, in descending order.*

As it has been introduced in (3.3), range measurement depends on several factors such as: environmental conditions, radial distance, reflectivity properties of surface material and angle of target orientation (dependent on position of scan device). Furthermore, segmentation performance very much depends on the used scan-device. The scan pattern of the HDL-32E is static and limited to a predefined resolution, which makes it more difficult to detect certain objects along arbitrary distances (ROIs). Due to increasing distance, the resolution of point cloud points is decreasing. Moreover, the number of points on the target object decreases as well and the prediction of the category class is less reliable.

6.6.5 Distribution of Local Errors

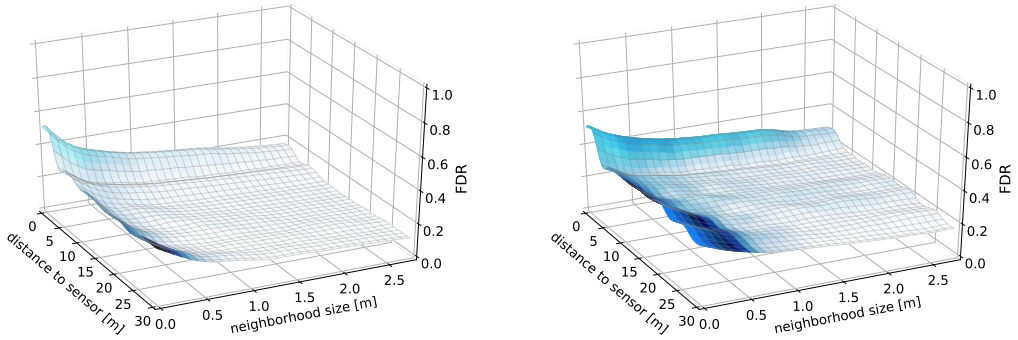
Investigations on radial distance and number of class samples in a LiDAR frame have been considered so far. The distribution and occurrence of local errors is observed to study behaviour of false classifications. Here the distribution of error samples is investigated for a **local neighbourhood size**, to determine whether the error occurs *point wise* or in *larger quantities of polygonal patches*. Therefore, the point cloud data is processed in such a way, that the local neighbourhood of falsely identified LiDAR pings is searched for the occurrence of other type I and II errors. Here the local neighbourhood size is limited to a radius of less than $3m$ in range. In Figure (6.21) the local neighbourhood of false classified samples is investigated for classes ego-lane and opposite-lane, represented as 3d box-plot (mesh layers represent the mean value of the underlying distribution whereas colour-shader shows IQR (dark colour represents large dispersion, white colour respectively no dispersion).

Furthermore, Figure (6.21) shows direct comparison of falsely identified sample data for lane classes ego-lane as well as opposite-lane. Herein it can be seen that size of IQR clearly decreases if distance to sensor decreases, as it was expected from previous investigations. Herein, the detection rate remains to be good, if the sample is located close to the sensor. Differences in both plots (ego-lane and opposite-lane samples) shows the size of local neighbourhood. Considering results of ego-lane only, occurrence of false classified samples remains relatively low with small range of dispersion. Error count especially peaks



(a) TPR, ego-lane samples

(b) TPR, opposite-lane samples



(c) FDR, ego-lane samples

(d) FDR, opposite-lane samples

Figure 6.21: *Neighbourhood size of* ***local errors*** *(TPR/ FDR). 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*

in the near-field of the local neighbourhood (radial distance <1m). This indicates the error to appear in smaller quantities, which could also suggest point wise appearance.

Comparing to opposite-lane samples, distribution of error behaves differently. The range of local neighbourhood errors differs and exceeds radial distances of 1m. Therefore, error distribution of opposite-lane samples can appear differently within patches containing larger quantities of false classified LiDAR pings. Observing several scene examples from (6.5), qualitative evaluation showed that the neural network is partly not in the position to well distinguish between ego-lane and opposite-lane samples. Further, false classifications for opposite-lane samples suffers from correctly identifying entire lane parts. In Figure (A.10), the segmentation model failed to successful predict opposite-lane samples. Due to the loss of contextual information such as global maps or traffic rules, it might be difficult to automatically exclude specific scenarios from evaluation process. Here, samples have been segmented incorrectly, where the interpretation of the environment from trained network was appropriate.

In the following, complex scenarios of urban sceneries are taken into account where the segmentation performance at *road intersections* has been investigated, see Figure (6.22) for general results. Appropriate segmentation performance is seen for ego-lane samples (**TPR >90%**). Hereby, TPR remains relatively high for a large radial distance (about 40m in range), if compared to general results from Figure (6.18).

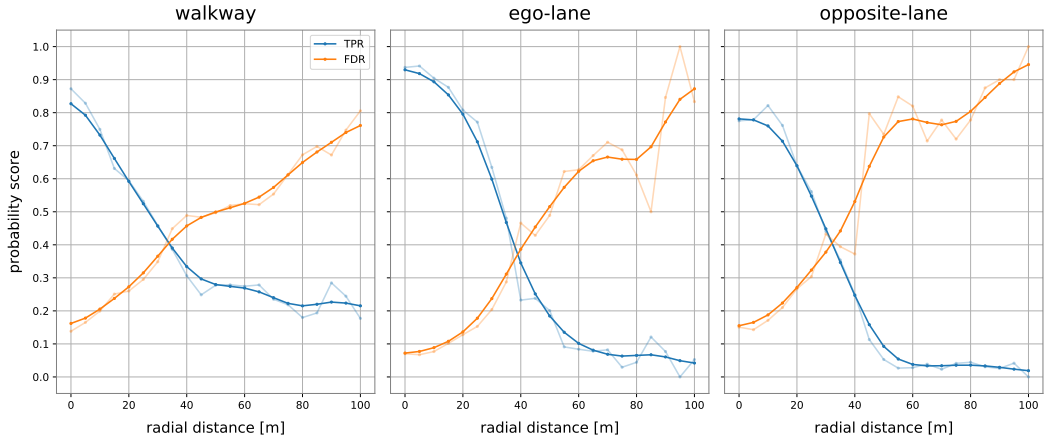


Figure 6.22: *TPR, FDR range measurements, intersections samples.*

Previous investigations on the segmentation of ego-lane samples, in Figure (B.2) (b), showed that FDR suffers a large range of outlier frames among groups containing 5000-9000 LiDAR pings per sample frame. On the other hand, groups containing more samples (9000-16000) show much less dispersion of errors. In Table (6.5) the distribution of sample frames containig intersections is shown, compared to number of frames without road junctions.

# pings	# frames (w/, w/o)
5000-9000	(1122, 1674)
9000-16000	(494, 700)

Table 6.5: *Distribution of sample frames containing road junctions.*

Due to the complexity of certain environments, it is assumed that segmentation performs worse within complex scenes. Here, a lot of contextual information might like to lead to consecutive confusion with samples from other category classes or background samples.

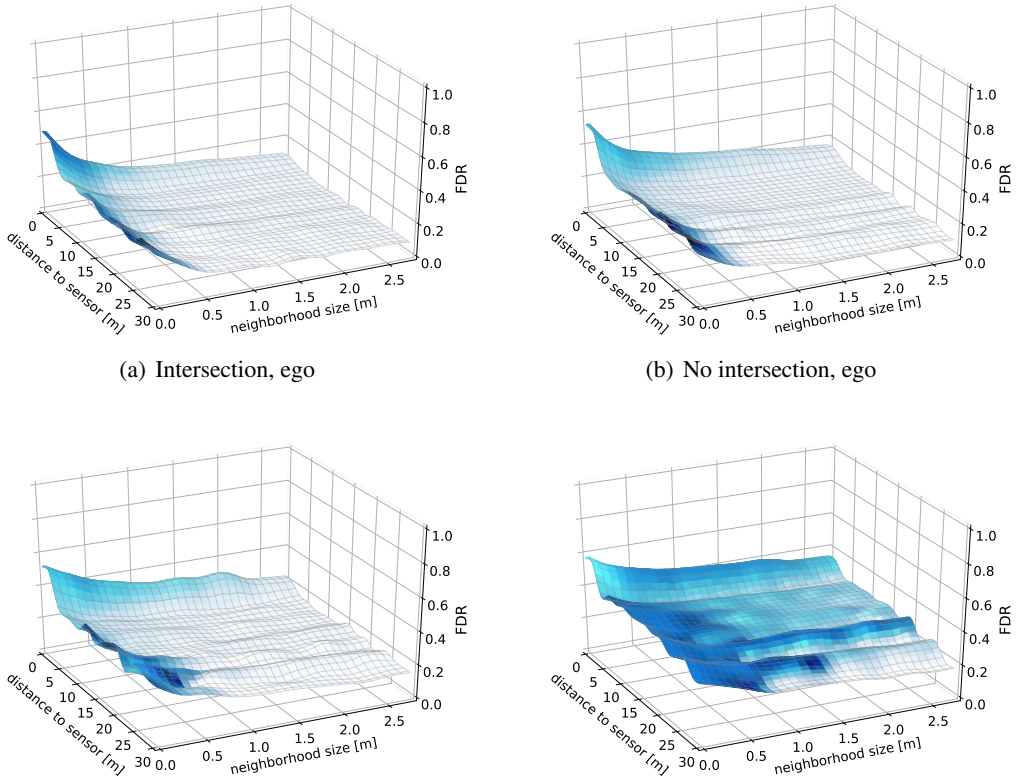
Therefore, the introduced group ranges of LiDAR pings, from Figure (B.2), are investigated separately for both intersections and areas which don't contain road junctions. Additionally, segmentation performance of ego-lane and opposite-lane samples is investigated separately, to further demonstrate ability of the segmentation model in differentiating lane classes within complex scenarios.

Grouped-Samples:

Considering FDR in Figure (6.23) shows the segmentation performance to not differ a lot from general results in Figure (6.21) (a). For a detailed illustration for both groups of LiDAR samples ($5000-9000$ and $9000-16000$) see Figure (B.14) - Figure (B.21). The most of falsely identified samples are concentrated among a smaller neighbourhood size, radius ($< 1m$). Small differences are seen within Figure (6.23) (a) and (b), where the segmentation of ego-lane samples turned out to by slightly better (by a few percentages) for sample frames which belong to areas of road junctions. Hereby, existence of contextual information potentially could have caused the neural network to form more complex features which enables the application to better differentiate in such situations.

On the contrary, performance of opposite-lane segmentation (FDR) is illustrated in Figure (6.23) (c) and (d). Hereby, the distributions of false classifications show significant

differences in comparison to results from average lane segmentation in Figure (6.21). Considering frames counting 5000 – 9000 pings (without road junctions), most errors does not occur in sensor near-field but is scattered across larger radial distances for both sensor distance and neighbourhood size. Distribution of errors across larger radial distances indicates the error to occur within larger quantities of LiDAR pings.



(a) Intersection, ego

(b) No intersection, ego

(c) Intersection, opposite

(d) No intersection, opposite

Figure 6.23: *Neighbourhood size of local errors, ego-/ opposite-lane. Grouped by number of LiDAR pings wrt local ranges* ***(5000-9000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*

By further considering sample frames containing 9000−16000 LiDAR pings, the amount of false classifications is significantly higher, especially in sensor near-field, see Figure (B.16) - Figure (B.17) and Figure (B.20) - Figure (B.21). Here, the segmentation model suffers in separating opposite-lane samples from other categories.

In general, semantic segmentation performance, within scenarios such as road junctions, has been seen to be better compared by classifying ordinary road segments.

Qualitative evaluation showed that the model partly failed to correctly identify entire road segments, as it can be seen in Figure (A.10). This might be attributed to the loss of semantic information (e.g. traffic rules), which is not explicitly encoded in the provided data. Furthermore, a profound quantitative evaluation would require to exclude such scene samples from validaiton set, where it is assumed that local error level should improve.

7 Conclusion and Future Work

In this thesis the approach for semantic lane segmentation of 360° LiDAR point clouds is addressed under use of machine learning applications.

This approach introduced the use of an extended multimodal dataset for the application of semantic point cloud segmentation. Semantic classes have been derived from 2d map annotations from the well-established nuScenes dataset. Due to encoded information of traffic direction, lane semantics could be provided to the segmentation process, where it was possible to successfully distinguish between lanes with different semantic meaning. However, the generation of appropriate ground truth samples from flat 2d maps, provided in top-view perspective (without information of z-level position) turned out to be suboptimal, where training and inference partly suffered from imperfect label generation.

Performing semantic lane segmentation, the model of *U-Net* architecture is used as a single-stage approach, which is simple but highly effective and performs in sensor real-time (*32fps*). Inference of the used convolutional neural network showed good performance of segmenting ground pings within a radius of approximately *30m* from position of ego-vehicle. Hereby, the detection score of TPR generally remained higher if compared to measure of false discoveries. Further, samples containing very low detection scores ($TPR \ll FDR$; exceeding specific distance from sensor, where it is most unlikely to process true positives) should be down weighted or excluded from processing as such samples don't provide meaningful and reliable information to the application of ground segmentation. Moreover, this treatment would allow the use of a threshold which can be useful in real-world applications.

Several ways of evaluating semantic segmentation have been considered. Qualitative evaluation has shown the overall performance to be good and reliable in most of the cases. Influence of several environmental conditions (e.g. rain, fog) clearly showed degrading performance of LiDAR range measurement but was not directly affecting quality of segmentation in the same way. Distinguishing ground classes has best seen between walkway and lane classes. Here, geometrical properties (e.g. curb stone) seems to have a clear influence on the segmentation result, where the neural network appeared to derive strong and meaningful features from. However, differentiating semantic lane classes seems to be much more complex, where spatial information only could not provide enough context to robustly separate lanes with different semantic meanings. Moreover, segmentation performance of the proposed model is most likely to suffer from class imbalance of provided annotations by a factor of almost *1:10*.

Nonetheless, very good results could be seen in frontal-view perspective of the laser scan, in front of the vehicle, whereas rear-view consequently suffered from false detections of opposite-lane samples. This could introduce the need of differently handling surround views to obtain appropriate results throughout the entire point cloud information. Capabilities of generalization have been seen throughout various scenarios presenting ego-lane samples. Due to contextual information (vehicles, pedestrians) the network was in the position to correctly predict class labels along opposite-lane samples, especially along areas of intersections, which had been generated as part of ego-lane in the first place.

The quantitative approach showed performance degradation due to decreasing density of point cloud information as well as increasing radial distance from ego-vehicle position. Where latter it was difficult to use as appropriate metric to describe quality of segmentation due to the fact of large ranges in dispersion of measurement values and number of outlier

samples. Hereby, point density on the target object much more replied to adequate and reliable measurement. Therefore, point density could give an appropriate metric for providing reliable results of semantic lane segmentation. Further the occurrence and density of locally distributed errors has been investigated. It could be seen that smaller patches of errors occurred for class ego-lane, where on the hand, segmenting opposite-lane samples suffered from confusion with ego-lane segments consisting out of larger patches of collections of falsely identified LiDAR pings.

Areas of intersections have been separated from other sample frames to quantitatively evaluate the influence of complex traffic scenarios under isolated circumstances. Herein, the existence of contextual information seems to have a greater influence on the segmentation process, where the neural network performs better by several percentages if compared to road segments containing no intersections.

The detailed quantitative evaluation rapidly starts to become complex and difficult to handle, especially by trying to investigate conditions of specific situations. The work with publicly available datasets usually is limiting the range of conditions, where only a specific subset of selected scenarios is provided. Future work should comprise integration of data samples which contain only specific traffic situations and environmental conditions in detail to explicitly investigate model performance throughout a set of several predefined scenarios.

Confusion in semantic lane segmentation has mostly been seen in ego-lane and opposite-lane samples. Considering lane semantics, the definition of additional category classes such as *intersection* could potentially improve the results of segmentation by resolving conflicts between ego-lane samples and intersection areas.

Splitting point cloud information into two clusters of front- and rear-view might dissolve issues of *perspective disorientation* where especially parts from rear-view suffered from consequent false detections. On the other hand, subdividing the point cloud by multiple *spherical cuts* potentially influences quality of segmentation differently.

Further, spatial information should not be the only source of information a network uses for the decision-making process. Temporal as well semantic aspects might significantly improve the semantic understanding of environment. Introducing time component by aggregating multiple LiDAR frames increases density of data which has been seen to proportionally react with increasing score of true positive samples. Further, semantic aspects (e.g. lamp posts, object tracking to provide information of driving direction) has previously seen to improve segmentation results which shows that lane semantics are not only embodied in road parts.

Processing predicted point labels, polygonal information can be extracted from segmentation masks, which allows efficient tracking of estimated free space areas. Hereby, problems of false classification and occlusion would need to be handled appropriately. Applying belief propagation (e.g. markov random fields or conditional random fields) have often been seen as method of postprocessing in semantic segmentation models. Segmentation patches which suffer from *blurry boundaries* and areas containing larger quantities of false detections, can be appropriately handled by applying *smoothing* to the segmentation results. Quantitative evaluation, for occurrence of locally distributed errors, showed false classifications to appear in smaller quantities of measurement values for class ego-lane, where postprocessing could have a substantial influence to improve classification results.

Fusing 3d point cloud as well as 2d camera image is a treatment which is often seen for camera applications. Here, depth-rich information from 3d data provides an accurate depth map, which often cannot be derived from camera sensors at the same level of quality. Processing multimodal information in data fusion applications requires accurate sensor calibration as well as detailed annotations for both sensors, camera and LiDAR.

This thesis has shown validation concepts to characterise model of semantic lane segmentation. Validation study further demonstrated scenarios which require retraining and the integration of additional classes or contextual information (e.g. spatial-, semantic- and temporal-information) to improve segmentation performance. For using the approach within real-world applications, the model must be proven to provide appropriate performance under variety of conditions. The suggested metric, measured by number of pings registered on the target object, also has the potential to provide real-time validation while the vehicle is operating on road. Furthermore, this validation workflow can be extended for the needs of sensor-fusion applications to include other sensor modalities to increase robustness of autonomous driving.

8 Bibliography

[1] Julie Stephany Berrio et al. “Semantic sensor fusion: from camera to sparse lidar information”. In: *ArXiv* abs/2003.01871 (2020) (cit. on p. 1).

[2] J. Behley et al. “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences”. In: *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*. 2019 (cit. on p. 38).

[3] Pierre Biasutti et al. “LU-Net: An Efficient Network for 3D LiDAR Point Cloud Semantic Segmentation Based on End-to-End-Learned 3D Features and U-Net”. In: *ArXiv* abs/1908.11656 (2019) (cit. on pp. 5, 34–36, 51, 61).

[4] Pierre Biasutti et al. “RIU-Net: Embarrassingly simple semantic segmentation of 3D LiDAR point cloud”. In: *CoRR* abs/1905.08748 (2019). arXiv: 1905.08748 (cit. on p. 5).

[5] Holger Caesar et al. “nuScenes: A multimodal dataset for autonomous driving”. In: *CoRR* abs/1903.11027 (2019). arXiv: 1903.11027 (cit. on pp. 37–43, 45, 56, 57, 61).

[6] Ming-Fang Chang et al. “Argoverse: 3D Tracking and Forecasting with Rich Maps”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on pp. 38–40, 54).

[7] Yilun Chen et al. “Fast Point R-CNN”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 9774–9783 (cit. on p. 4).

[8] R. Kesten et al. *Lyft Level 5 AV Dataset 2019*. 2019 (cit. on pp. 38–40).

[9] Loïc Landrieu and Mohamed Boussaha. “Point Cloud Oversegmentation with Graph-Structured Deep Metric Learning”. In: *CoRR* abs/1904.02113 (2019). arXiv: 1904.02113 (cit. on p. 36).

[10] A. Milioto et al. “RangeNet++: Fast and Accurate LiDAR Semantic Segmentation”. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. 2019 (cit. on pp. 5, 34, 35).

[11] Fabio Pizzati and Fernando García. “Enhanced free space detection in multiple lanes based on single CNN with scene identification”. In: *CoRR* abs/1905.00941 (2019). arXiv: 1905.00941 (cit. on pp. 1, 5).

[12] Pei Sun et al. *Scalability in Perception for Autonomous Driving: Waymo Open Dataset*. 2019. arXiv: 1912.04838 [cs.CV] (cit. on p. 38).

[13] Tom Bruls et al. “Mark Yourself: Road Marking Segmentation via Weakly-Supervised Annotations from Multimodal Data”. In: May 2018, pp. 1863–1870. DOI: 10.1109/ICRA.2018.8460952 (cit. on pp. 1, 5).

[14] Johann Laconte et al. “Lidar Measurement Bias Estimation via Return Waveform Modelling in a Context of 3D Mapping”. In: *CoRR* abs/ 1810.01619 (2018). arXiv: 1810.01619 (cit. on p. 21).

[15] Alex H. Lang et al. “PointPillars: Fast Encoders for Object Detection from Point Clouds”. In: *CoRR* abs/ 1812.05784 (2018). arXiv: 1812.05784 (cit. on pp. 5, 34, 35).

[16] Yangyan Li et al. “PointCNN”. In: *CoRR* abs/ 1801.07791 (2018). arXiv: 1801.07791 (cit. on pp. 32–34).

[17] Annika Meyer et al. “Deep Semantic Lane Segmentation for Mapless Driving”. In: Oct. 2018, pp. 869–875. DOI: 10.1109/IROS.2018.8594450 (cit. on pp. 1, 5, 61).

[18] Yuan Wang et al. “PointSeg: Real-Time Semantic Segmentation Based on 3D LiDAR Point Cloud”. In: *CoRR* abs/ 1807.06288 (2018). arXiv: 1807.06288 (cit. on pp. 5, 34, 35).

[19] Bichen Wu et al. “SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud”. In: *CoRR* abs/ 1809.08495 (2018). arXiv: 1809.08495 (cit. on pp. 5, 34–36, 60).

[20] Wei Zhou et al. “Automated Evaluation of Semantic Segmentation Robustness for Autonomous Driving”. In: *CoRR* abs/ 1810.10193 (2018). arXiv: 1810.10193 (cit. on p. 1).

[21] Luca Caltagirone et al. “Fast LIDAR-based Road Detection Using Fully Convolutional Neural Networks”. In: *CoRR* abs/ 1703.03613 (2017). arXiv: 1703.03613 (cit. on pp. 1, 5).

[22] Kaiming He et al. “Mask R-CNN”. In: *CoRR* abs/ 1703.06870 (2017). arXiv: 1703.06870 (cit. on pp. 4, 5).

[23] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-Excitation Networks”. In: *CoRR* abs/ 1709.01507 (2017). arXiv: 1709.01507 (cit. on p. 38).

[24] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *CoRR* abs/ 1708.02002 (2017). arXiv: 1708.02002 (cit. on pp. 5, 10, 51).

[25] Charles Ruizhongtai Qi et al. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *CoRR* abs/ 1706.02413 (2017). arXiv: 1706.02413 (cit. on pp. 5, 33, 34).

[26] Lyne P. Tchapmi et al. “SEGCloud: Semantic Segmentation of 3D Point Clouds”. In: *CoRR* abs/ 1710.07563 (2017). arXiv: 1710.07563 (cit. on pp. 5, 30, 31).

[27] Bichen Wu et al. “SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud”. In: *CoRR* abs/ 1710.07368 (2017). arXiv: 1710.07368 (cit. on pp. 5, 34–36, 60).

[28] Manzil Zaheer et al. “Deep Sets”. In: *CoRR* abs/ 1703.06114 (2017). arXiv: 1703.06114 (cit. on p. 32).

[29] Barret Zoph et al. “Learning Transferable Architectures for Scalable Image Recognition”. In: *CoRR* abs/ 1707.07012 (2017). arXiv: 1707.07012 (cit. on p. 4).

[30] Dan Barnes, William P. Maddern, and Ingmar Posner. “Find Your Own Way: Weakly-Supervised Segmentation of Path Proposals for Urban Autonomy”. In: *CoRR* abs/ 1610.01238 (2016). arXiv: 1610.01238 (cit. on pp. 1, 5).

[31] Marius Cordts 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. 5, 39).

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 6–13, 51).

[33] Charles Ruizhongtai Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *CoRR* abs/ 1612.00593 (2016). arXiv: 1612.00593 (cit. on pp. 5, 32–34).

[34] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. “OctNet: Learning Deep 3D Representations at High Resolutions”. In: *CoRR* abs/ 1611.05009 (2016). arXiv: 1611.05009 (cit. on pp. 5, 31).

[35] Evan Shelhamer, Jonathan Long, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *CoRR* abs/1605.06211 (2016). arXiv: 1605.06211 (cit. on pp. 4, 28, 29).

[36] Ross B. Girshick. “Fast R-CNN”. In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083 (cit. on p. 4).

[37] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/ 1512.03385 (2015). arXiv: 1512.03385 (cit. on pp. 4, 17, 18, 30, 38).

[38] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015 (cit. on pp. 12, 13).

[39] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning Deconvolution Network for Semantic Segmentation”. In: *CoRR* abs/ 1505.04366 (2015). arXiv: 1505.04366 (cit. on pp. 4, 15, 16, 29).

[40] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497 (cit. on p. 4).

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597 (cit. on pp. 4, 29, 30, 35, 51).

[42] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y (cit. on pp. 3, 4, 38).

[43] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015 (cit. on pp. 3, 38).

[44] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/ 1409.4842 (2014). arXiv: 1409.4842 (cit. on p. 3).

[45] Ross B. Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524 (cit. on p. 4).

[46] Michael Galetzka and Patrick O. Glauner. “A correct even-odd algorithm for the point-in-polygon (PIP) problem for complex polygons”. In: *CoRR* abs/ 1207.3502 (2012). arXiv: 1207.3502 (cit. on pp. 47, 55).

[47] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on pp. 38, 39).

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105 (cit. on pp. 3, 38).

[49] Velodyne lidar Inc. *Data-Sheet: Velodyne LiDAR HDL-32E, HIGH RESOLUTION REAL-TIME 3D LIDAR SENSOR*. Nov. 2010 (cit. on p. 19).

[50] Sylvie Soudarissanane et al. "Incidence angle influence on the quality of terrestrial laser scanning points". In: *Proceedings ISPRS Workshop Laserscanning 2009, 1-2 Sept 2009, Paris, France* 38 (Jan. 2009) (cit. on pp. 20, 22, 23).

[51] Ove Steinvall. "Effects of Target Shape and Reflection on Laser Radar Cross Sections". In: *Applied optics* 39 (Sept. 2000), pp. 4381–91. DOI: 10.1364/AO.39.004381 (cit. on p. 24).

[52] R.G.M Morris. "D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949". In: *Brain Research Bulletin* 50.5 (1999), p. 437. ISSN: 0361-9230. DOI: https://doi.org/10.1016/S0361-9230(99)00182-3 (cit. on p. 6).

[53] A.V. Jelalian. "Laser Radar Systems". In: Artech House radar library (1992) (cit. on p. 22).

[54] Horst D. Simon, ed. *Parallel Computational Fluid Dynamics: Implementations and Results*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0262193264 (cit. on p. 48).

[55] Williams Ronald J. Rumelhart David E. Hinton Geoffrey E. "Learning representations by back-propagating errors". In: *Nature* 323 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0 (cit. on pp. 7, 8).

[56] Gene M. Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS 67 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1967, pp. 483–485. ISBN: 9781450378956. DOI: 10.1145/1465482.1465560 (cit. on pp. 47, 48).

[57] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* (1958), pp. 65–386 (cit. on p. 6).

[58] Walter Pitts Warren S. McCulloch. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (Dec. 1943), pp. 113–115. DOI: 10.1007/BF02478259 (cit. on p. 6).

Declaration

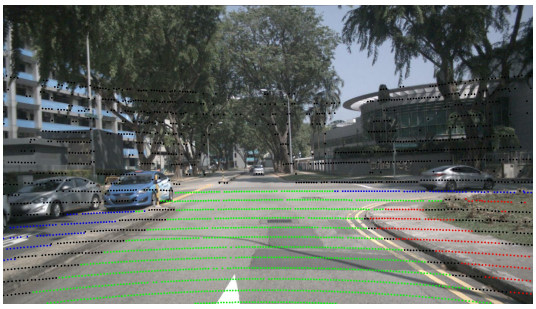
Herewith, I declare that this thesis with the following title:

Semantic Lane Segmentation of LiDAR Point Clouds using Convolutional Neural Networks

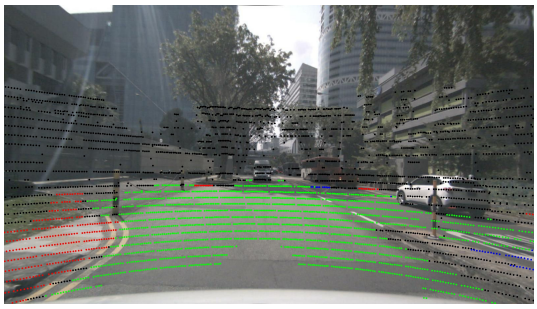
has fully been created by myself without the influence of other people. All of the used materials, source-code libraries as well as literature and research-papers, to create this work, are referenced in the documents text accordingly. This thesis has not been submitted before and is totally unique.

Place and Date	Signature

A Qualitative Evaluation



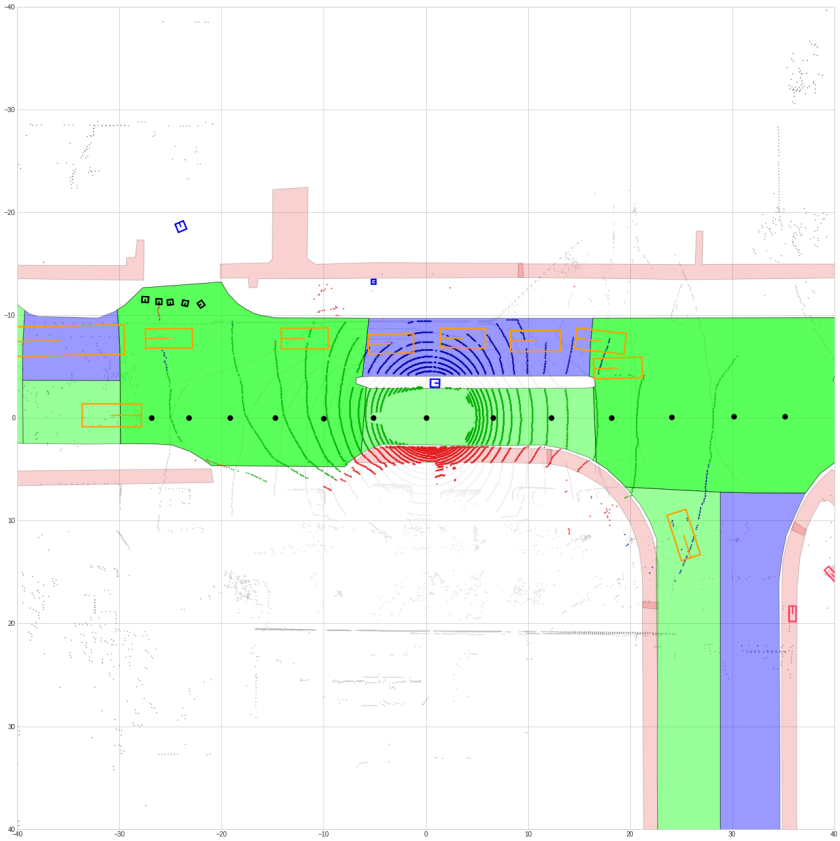
(a) CAM_FRONT



(b) CAM_BACK

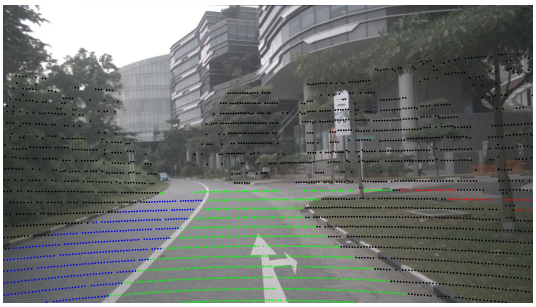
(c) range-image: ground truth

(d) range-image: prediction

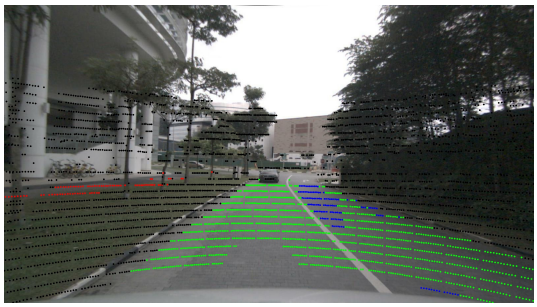


(e) Point cloud top-view: prediction + map layers

Figure A.1: *Projection of LiDAR point cloud in camera image of scene sample (scene-0039) with range images and point cloud top-view with additional map layers.*



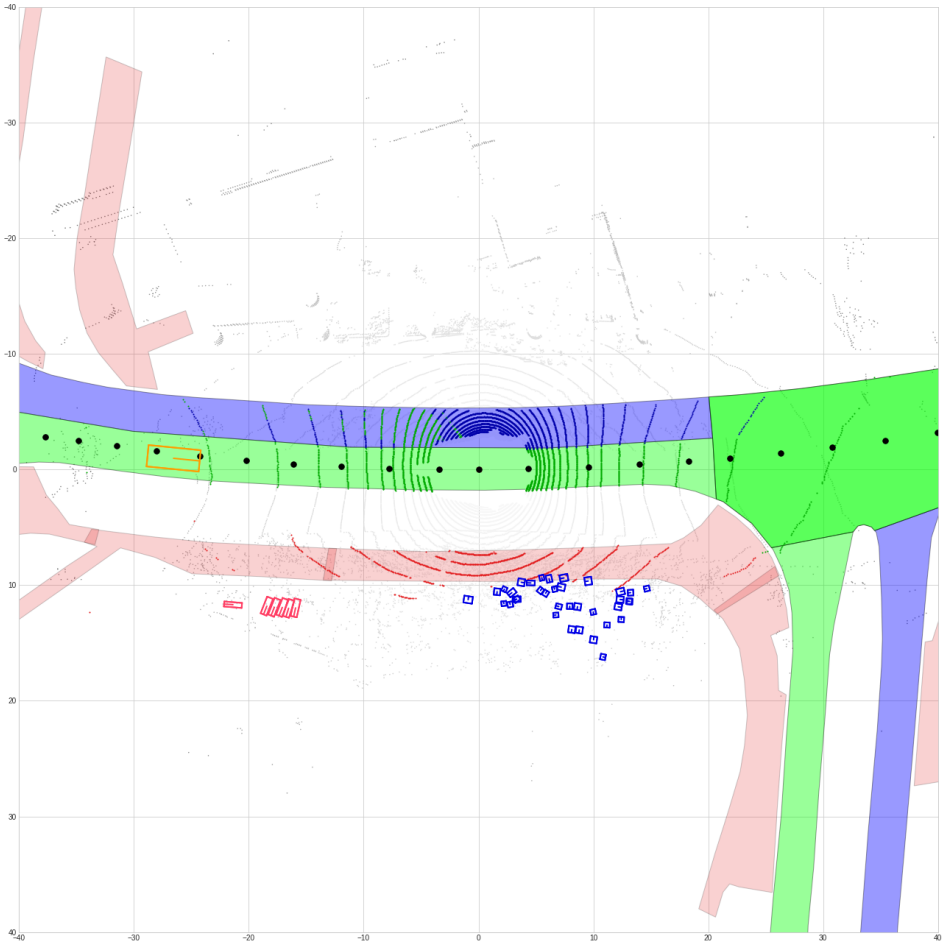
(a) CAM_FRONT



(b) CAM_BACK

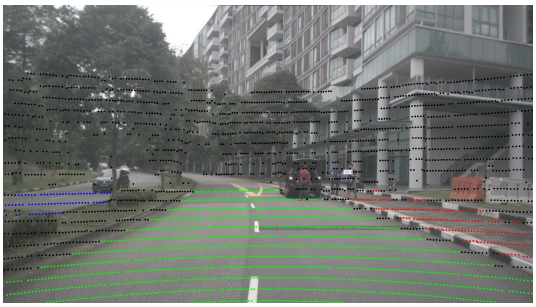
(c) range-image: ground truth

(d) range-image: prediction

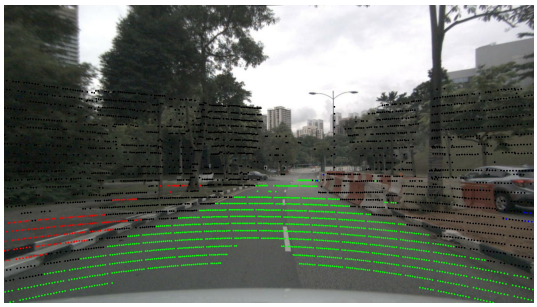


(e) Point cloud top-view: prediction + map layers

Figure A.2: *Projection of LiDAR point cloud in camera image of scene sample (scene-0269) with range images and point cloud top-view with additional map layers.*



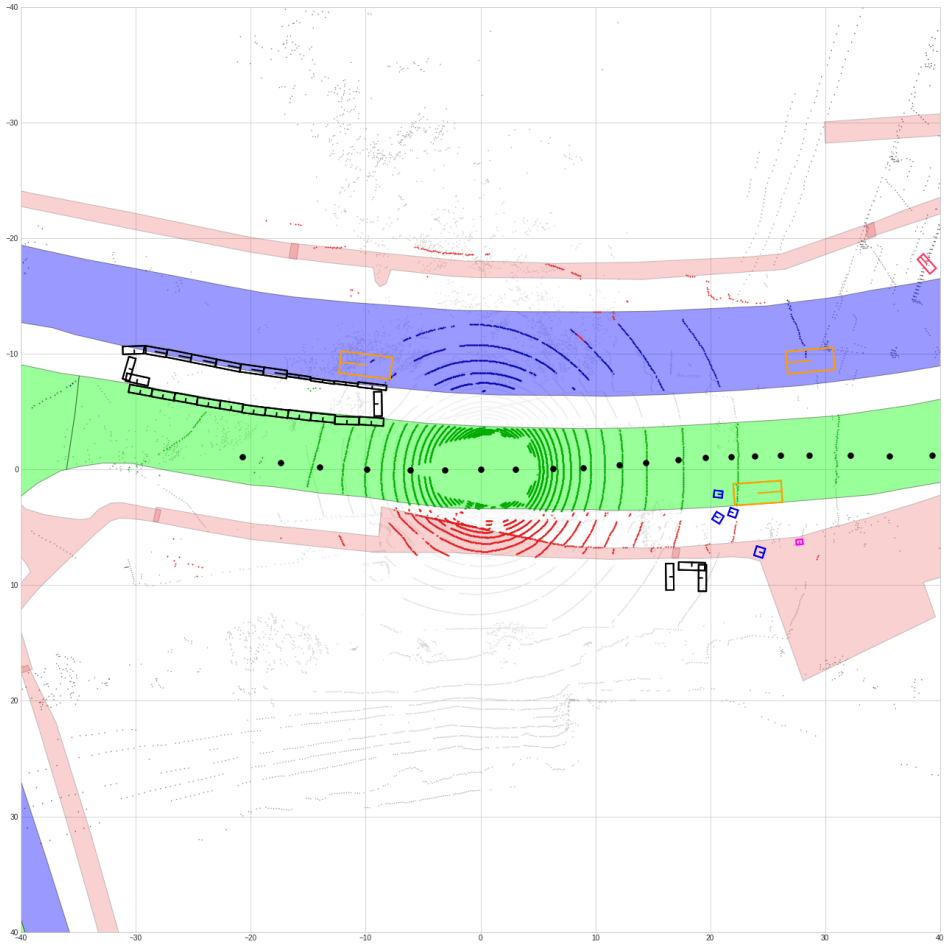
(a) CAM_FRONT



(b) CAM_BACK

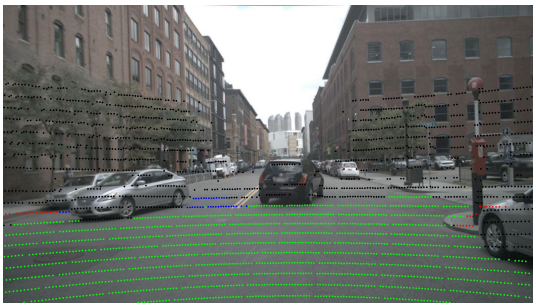
(c) range-image: ground truth

(d) range-image: prediction

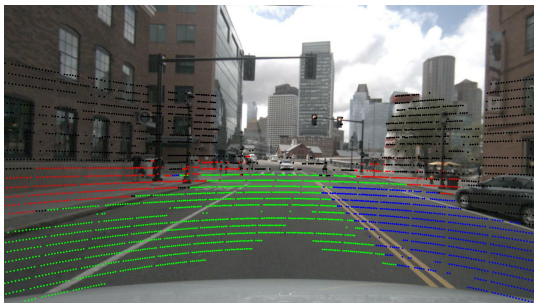


(e) Point cloud top-view: prediction + map layers

Figure A.3: *Projection of LiDAR point cloud in camera image of scene sample (scene-0273) with range images and point cloud top-view with additional map layers.*

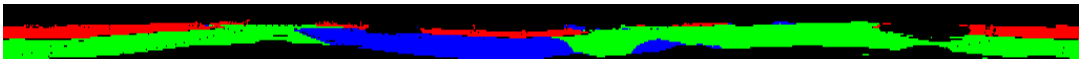


(a) CAM_FRONT

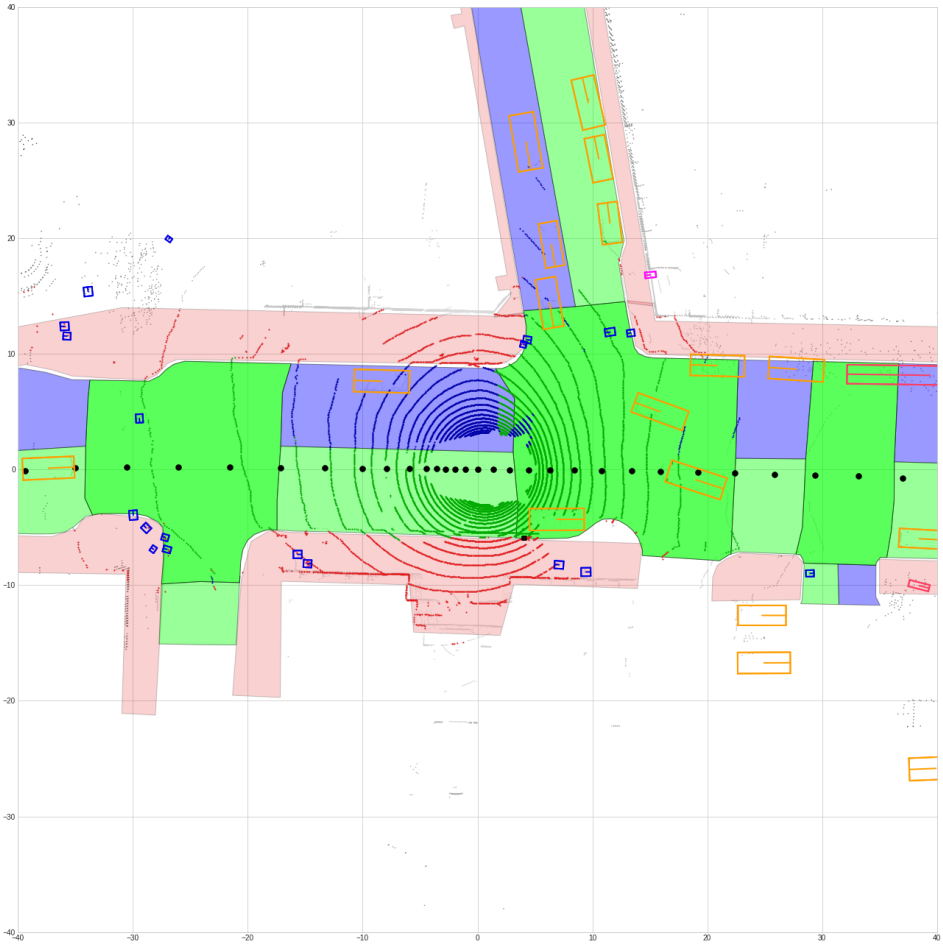


(b) CAM_BACK

(c) range-image: ground truth

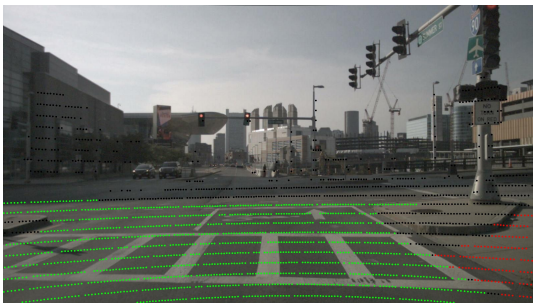


(d) range-image: prediction

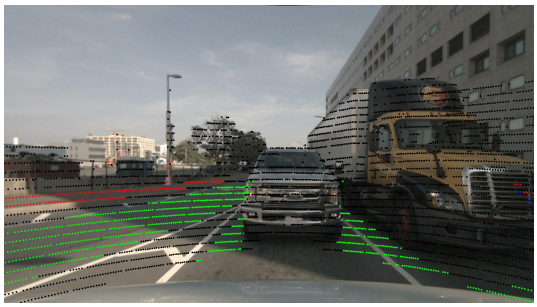


(e) Point cloud top-view: prediction + map layers

Figure A.4: *Projection of LiDAR point cloud in camera image of scene sample (scene-0103) with range images and point cloud top-view with additional map layers.*



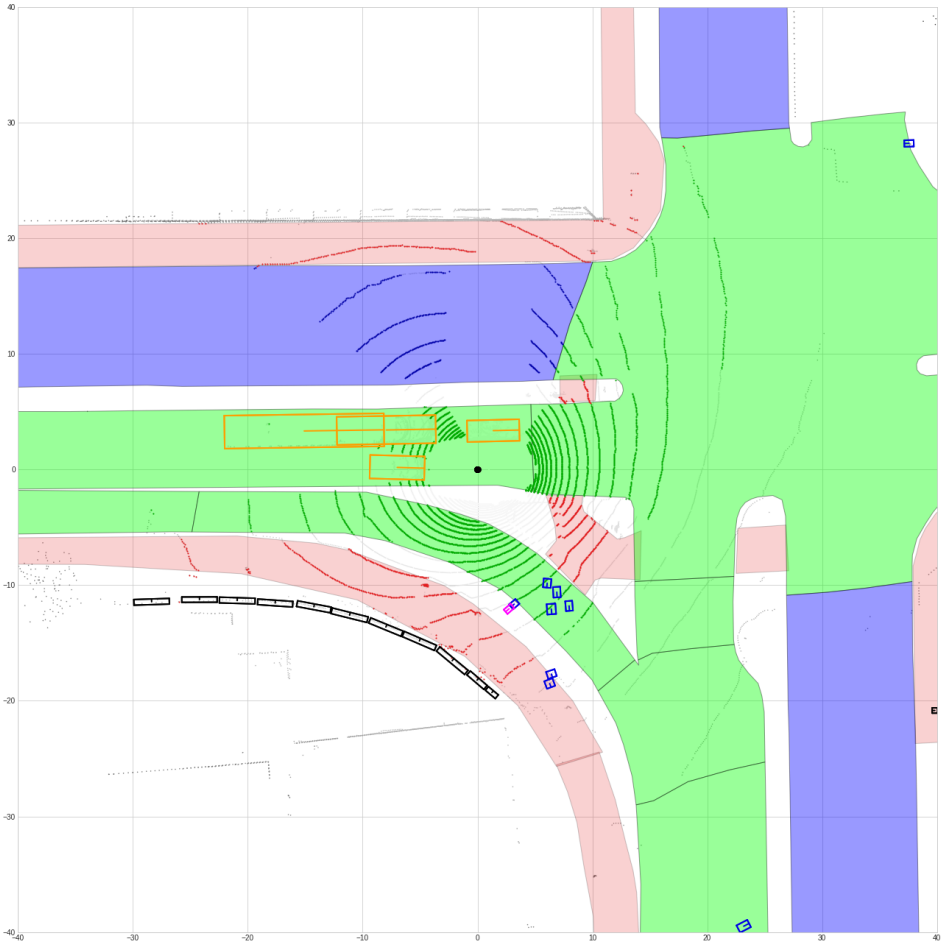
(a) CAM_FRONT



(b) CAM_BACK

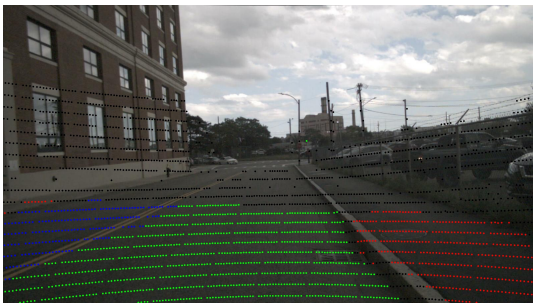
(c) range-image: ground truth

(d) range-image: prediction



(e) Point cloud top-view: prediction + map layers

Figure A.5: *Projection of LiDAR point cloud in camera image of scene sample (scene-0554) with range images and point cloud top-view with additional map layers.*

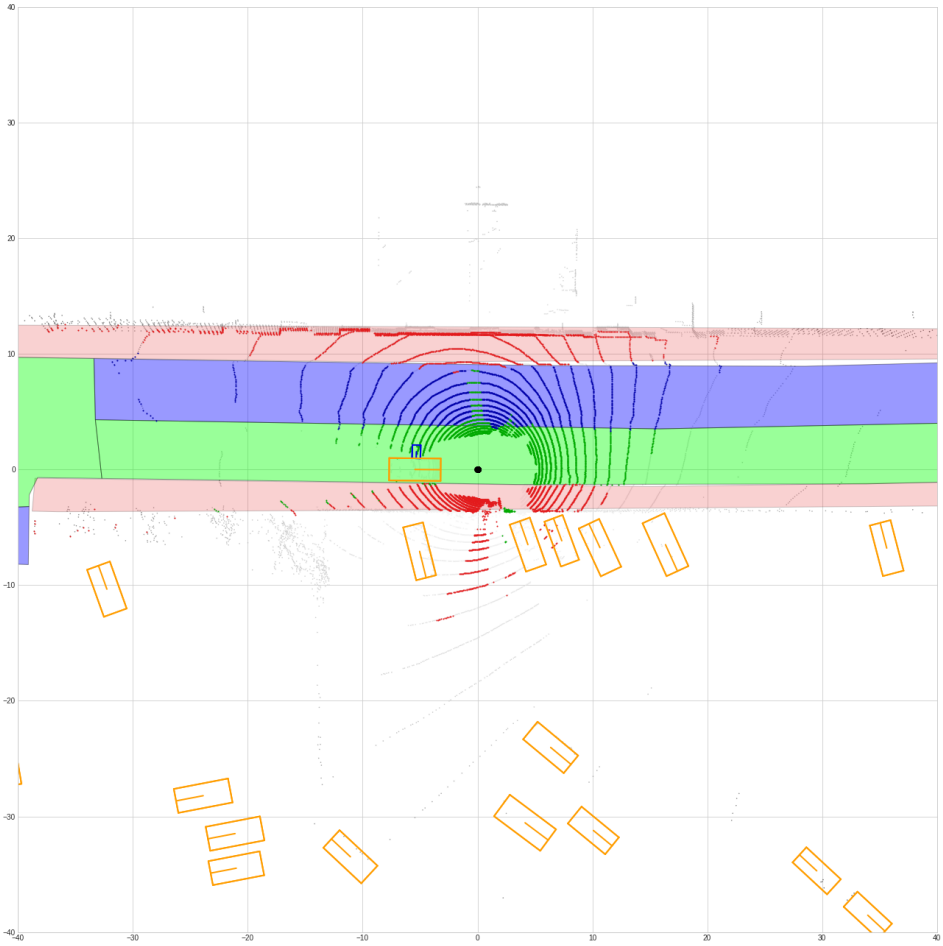


(a) CAM_FRONT

(b) CAM_BACK

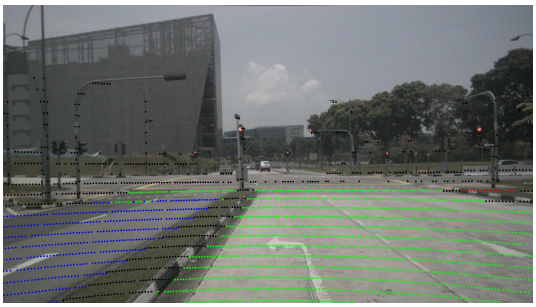
(c) range-image: ground truth

(d) range-image: prediction

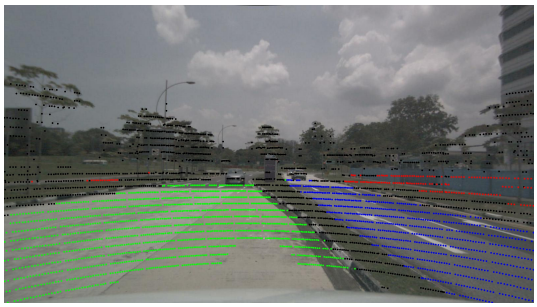


(e) Point cloud top-view: prediction + map layers

Figure A.6: *Projection of LiDAR point cloud in camera image of scene sample (scene-0771) with range images and point cloud top-view with additional map layers.*



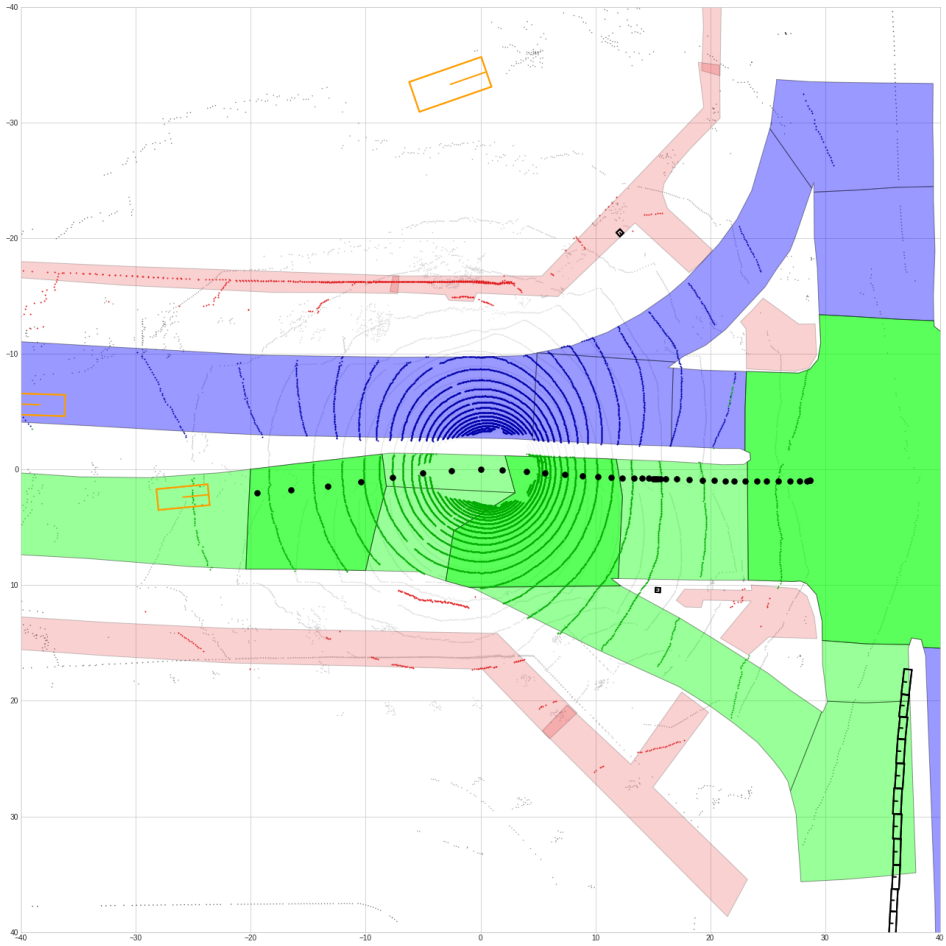
(a) CAM_FRONT



(b) CAM_BACK

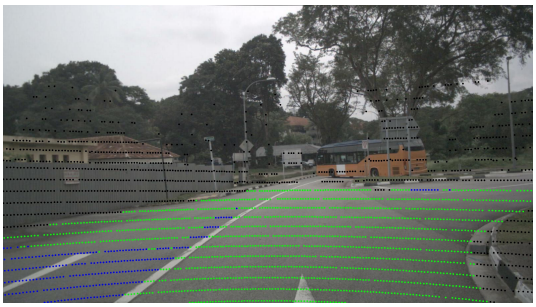
(c) range-image: ground truth

(d) range-image: prediction

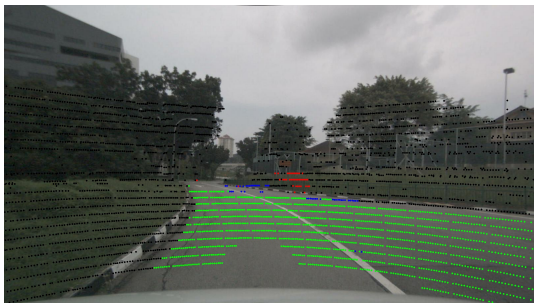


(e) Point cloud top-view: prediction + map layers

Figure A.7: *Projection of LiDAR point cloud in camera image of scene sample (scene-0963) with range images and point cloud top-view with additional map layers.*



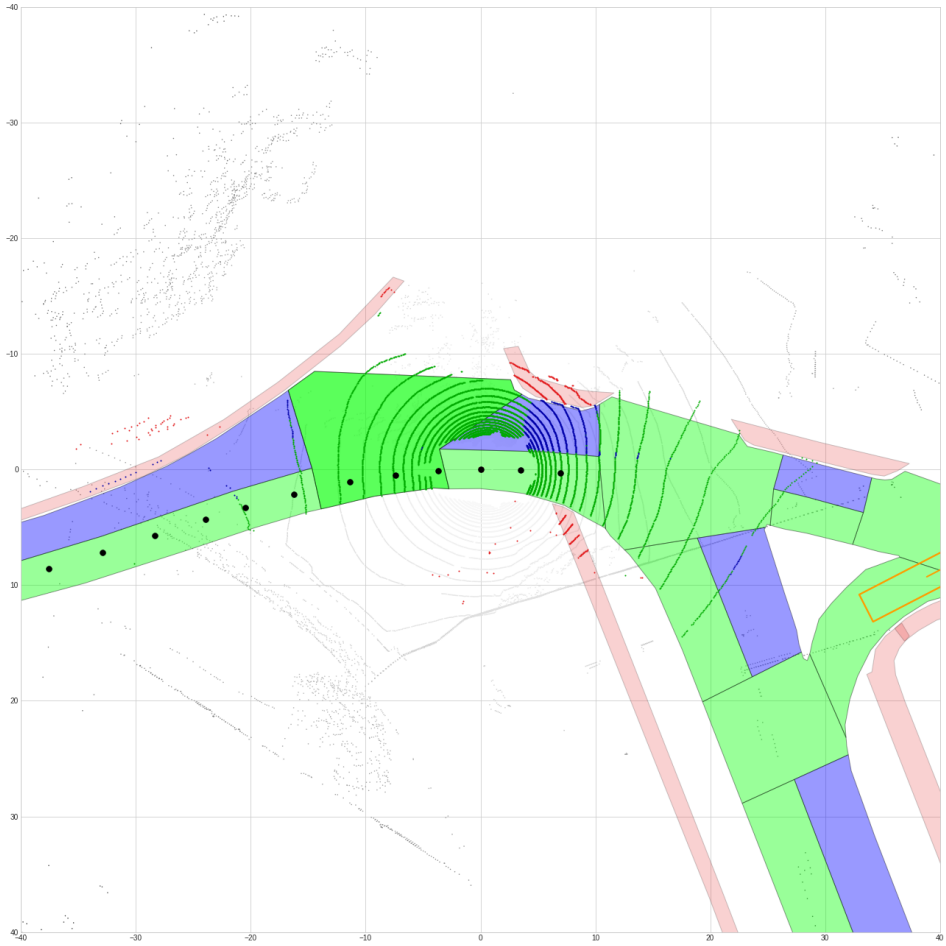
(a) CAM_FRONT



(b) CAM_BACK

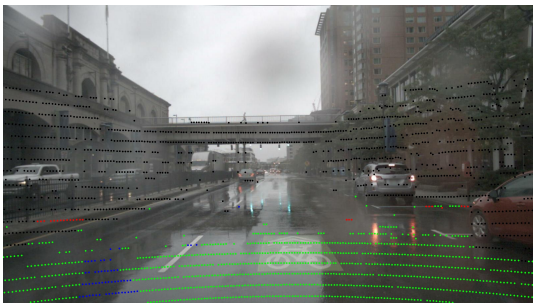
(c) range-image: ground truth

(d) range-image: prediction

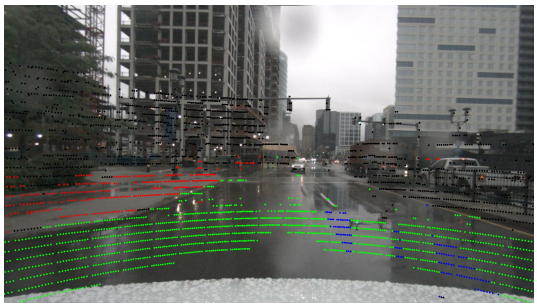


(e) Point cloud top-view: prediction + map layers

Figure A.8: *Projection of LiDAR point cloud in camera image of scene sample (scene-0012) with range images and point cloud top-view with additional map layers.*

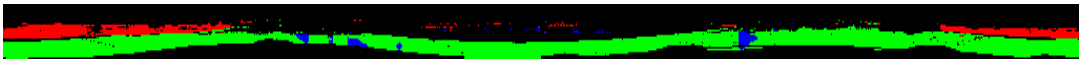


(a) CAM_FRONT

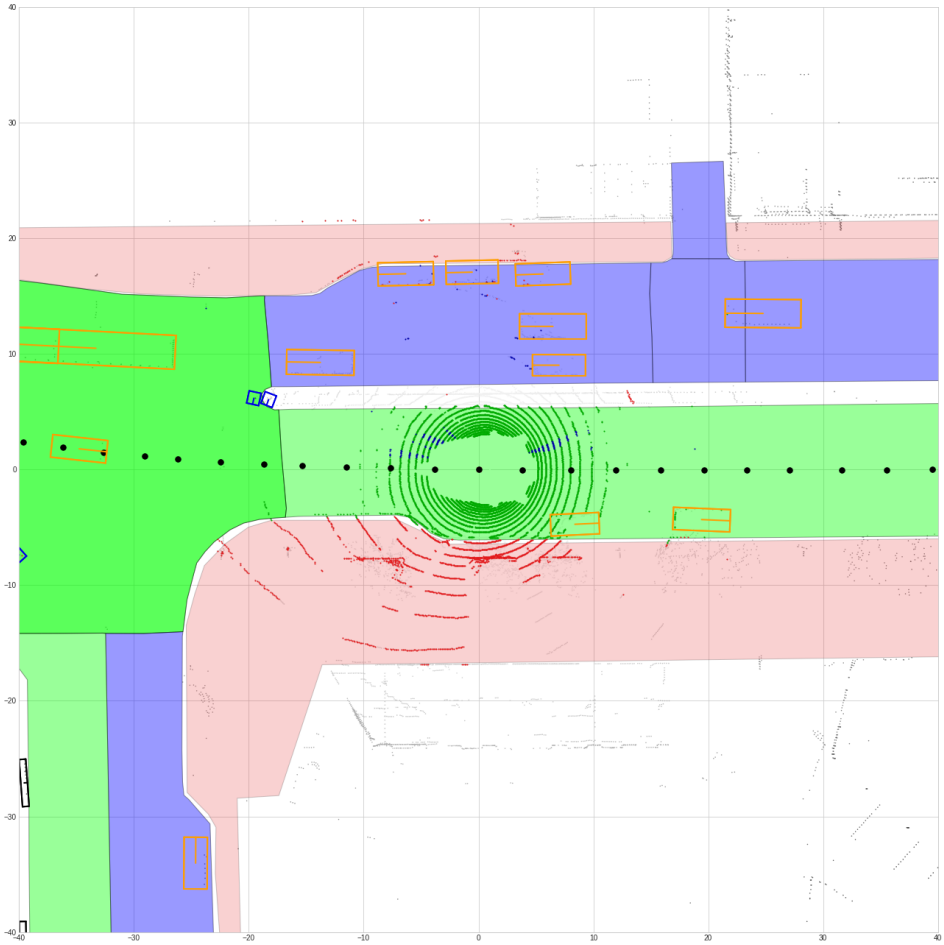


(b) CAM_BACK

(c) range-image: ground truth

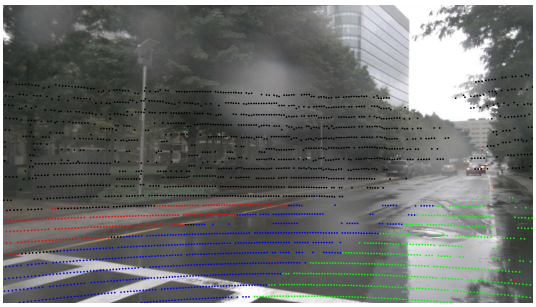


(d) range-image: prediction

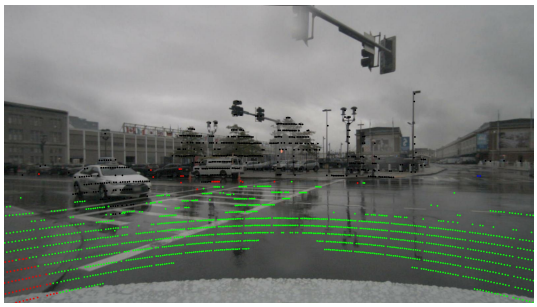


(e) Point cloud top-view: prediction + map layers

Figure A.9: *Projection of LiDAR point cloud in camera image of scene sample (scene-0905) with range images and point cloud top-view with additional map layers.*



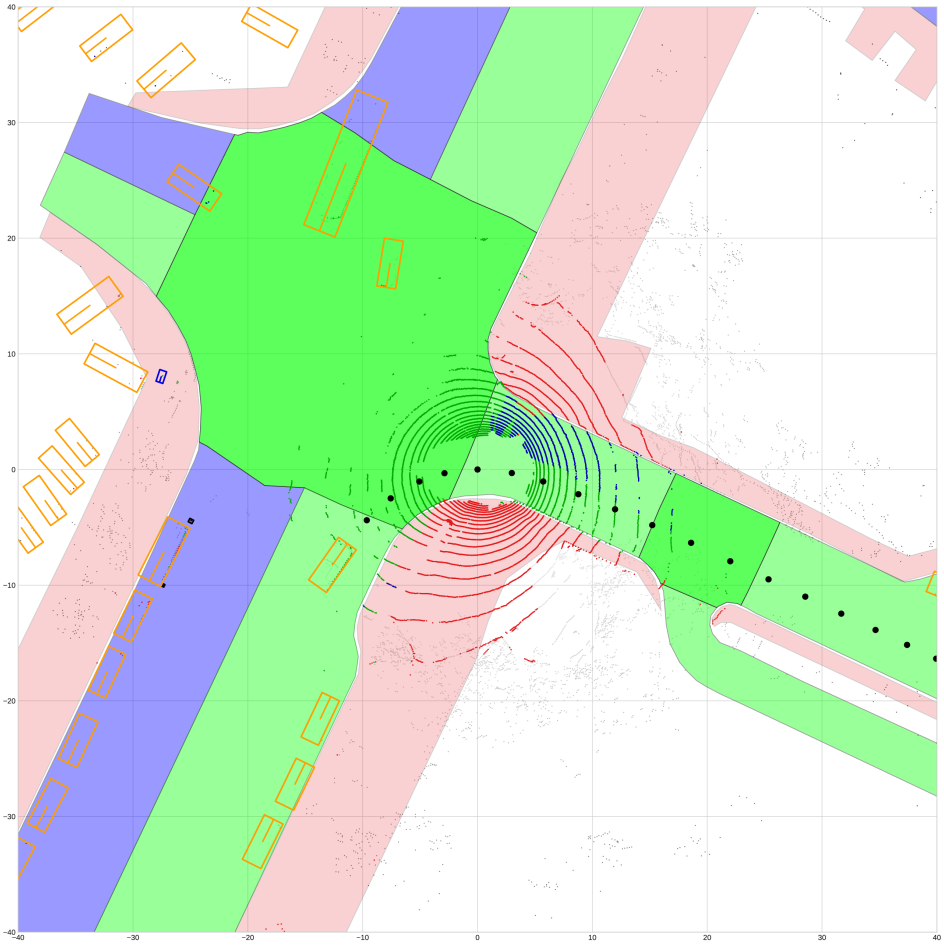
(a) CAM_FRONT



(b) CAM_BACK

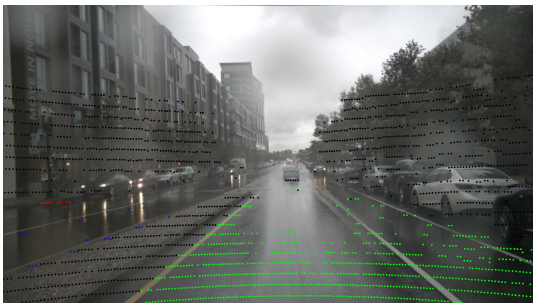
(c) range-image: ground truth

(d) range-image: prediction

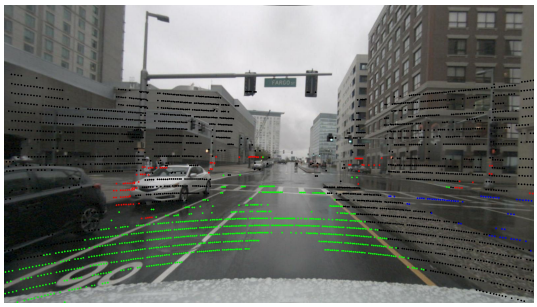


(e) Point cloud top-view: prediction + map layers

Figure A.10: *Projection of LiDAR point cloud in camera image of scene sample (scene-0907) with range images and point cloud top-view with additional map layers.*

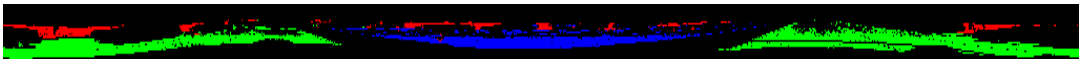


(a) CAM_FRONT

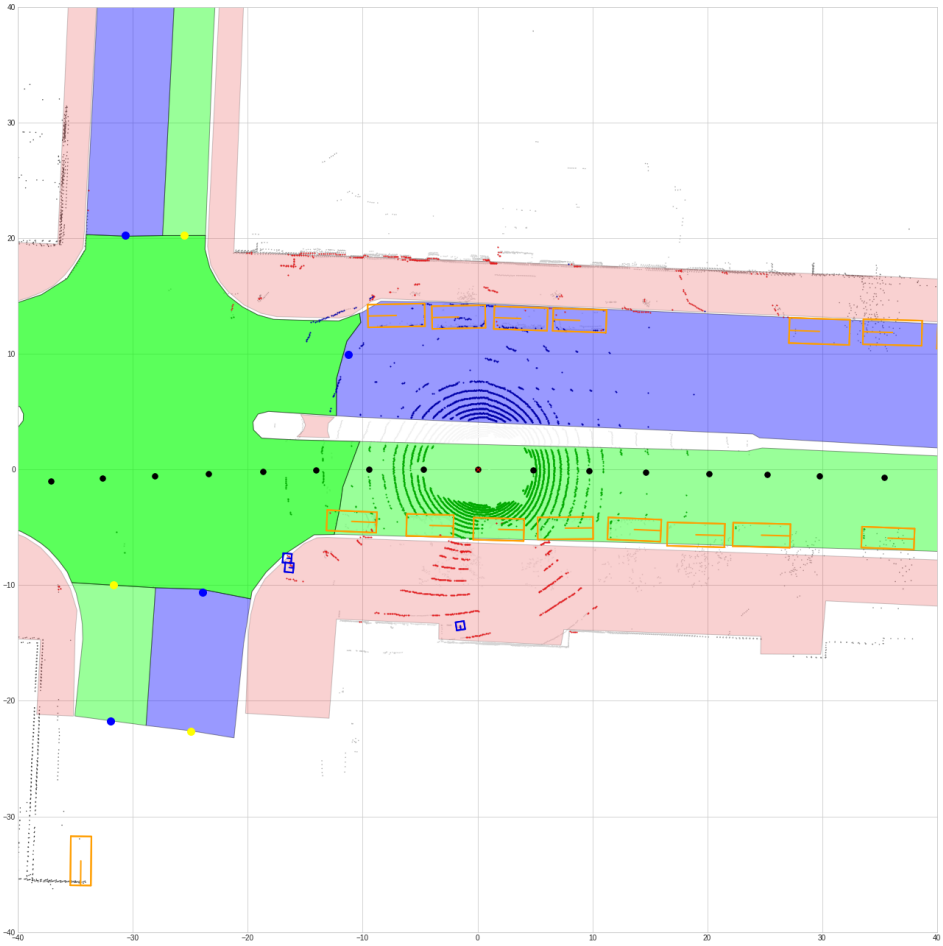


(b) CAM_BACK

(c) range-image: ground truth

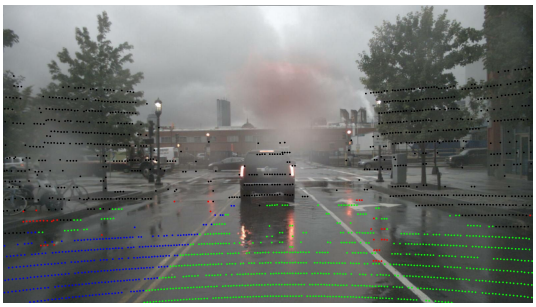


(d) range-image: prediction

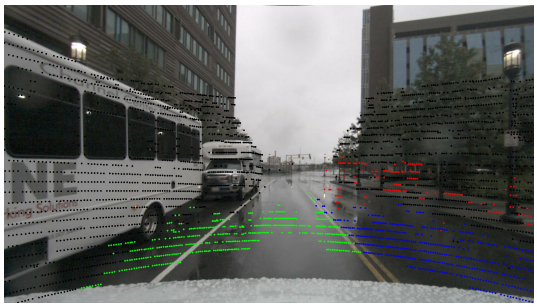


(e) Point cloud top-view: prediction + map layers

Figure A.11: *Projection of LiDAR point cloud in camera image of scene sample (scene-0911) with range images and point cloud top-view with additional map layers.*

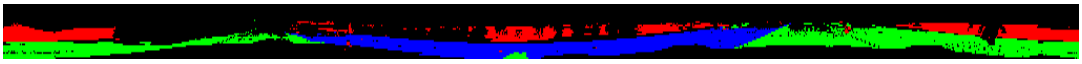


(a) CAM_FRONT

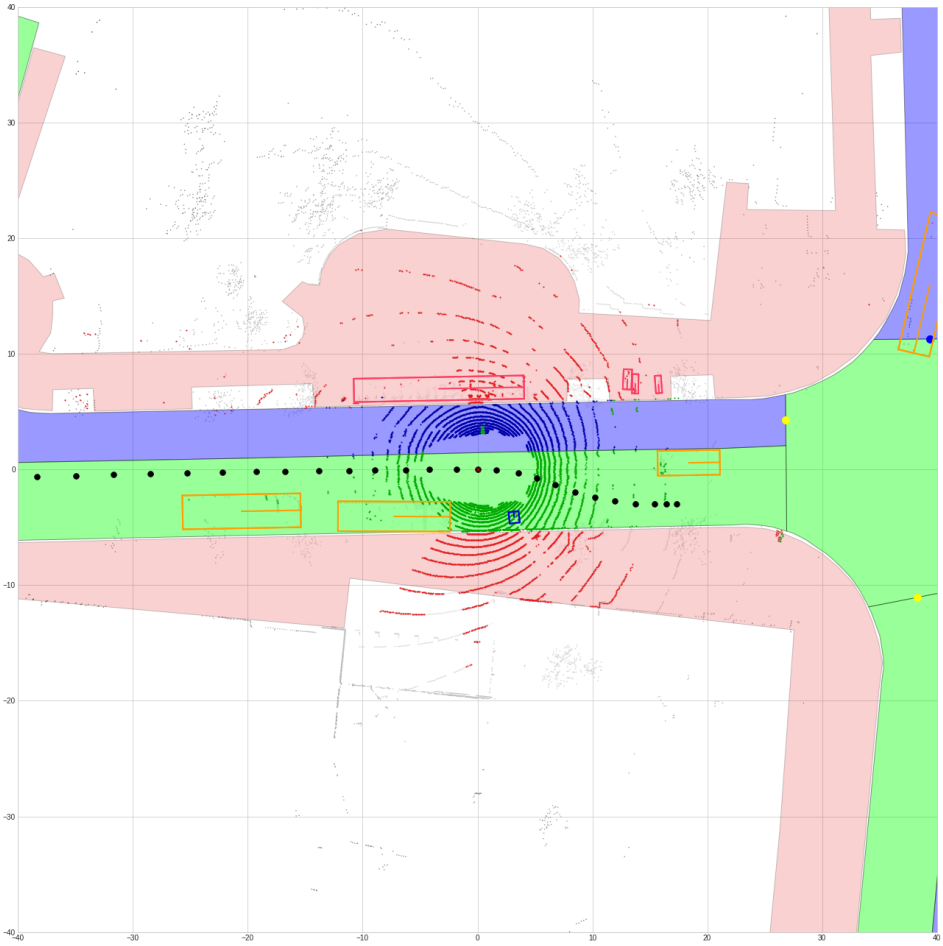


(b) CAM_BACK

(c) range-image: ground truth

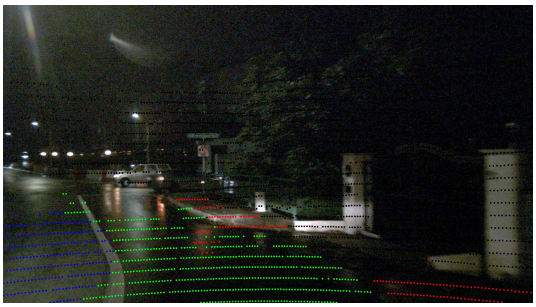


(d) range-image: prediction

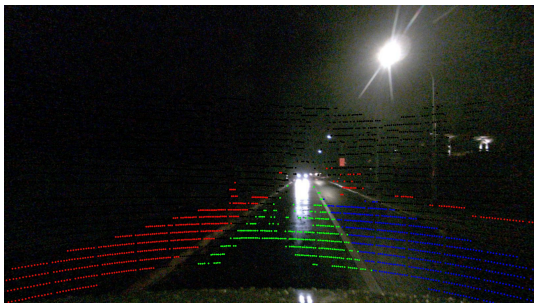


(e) Point cloud top-view: prediction + map layers

Figure A.12: *Projection of LiDAR point cloud in camera image of scene sample (scene-0914) with range images and point cloud top-view with additional map layers.*



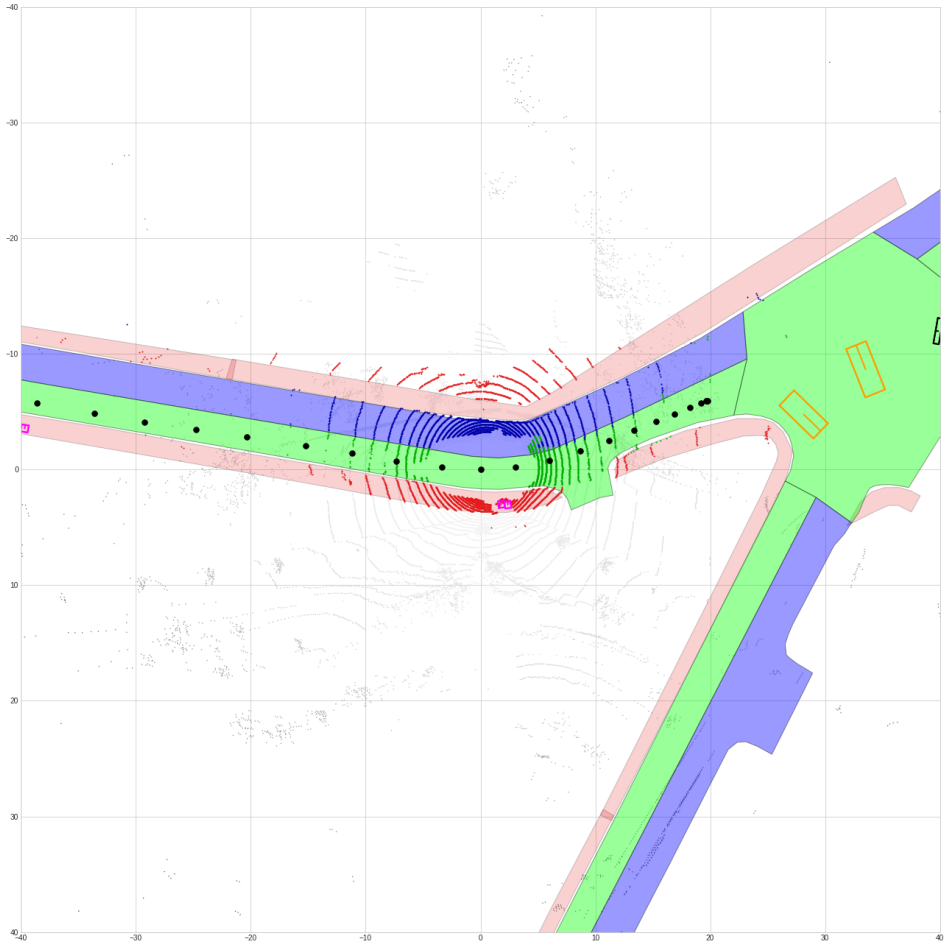
(a) CAM_FRONT



(b) CAM_BACK

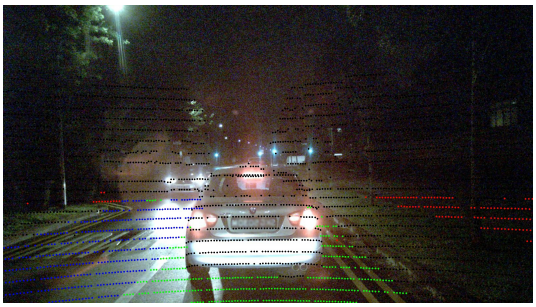
(c) range-image: ground truth

(d) range-image: prediction

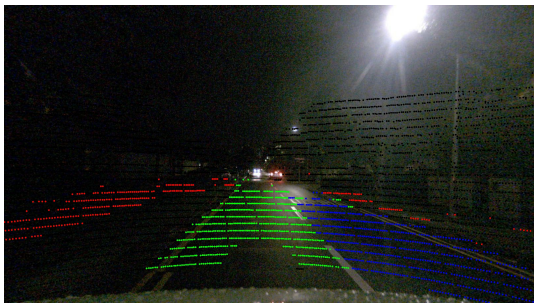


(e) Point cloud top-view: prediction + map layers

Figure A.13: *Projection of LiDAR point cloud in camera image of scene sample (scene-1062) with range images and point cloud top-view with additional map layers.*



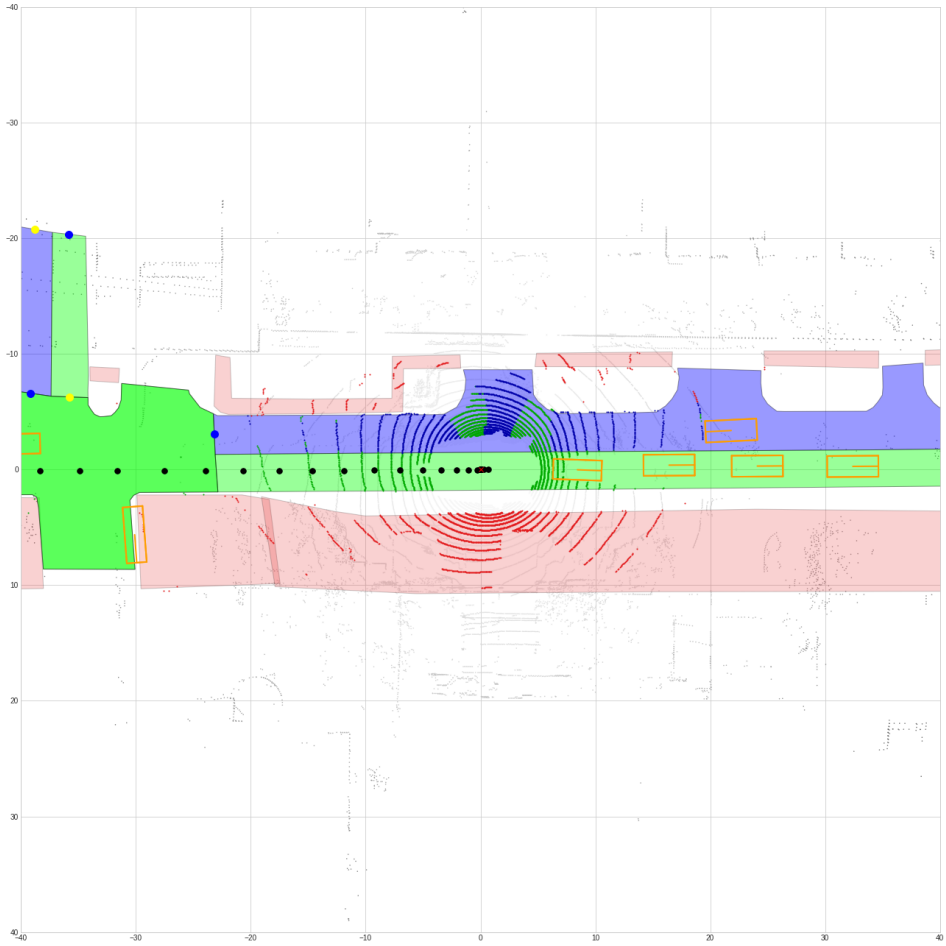
(a) CAM_FRONT



(b) CAM_BACK

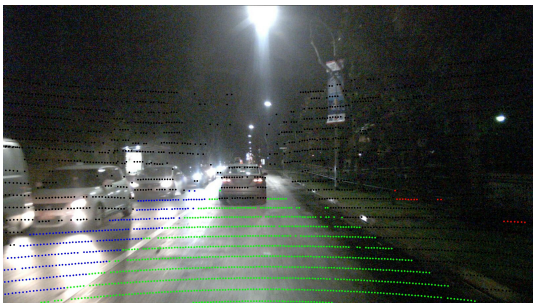
(c) range-image: ground truth

(d) range-image: prediction

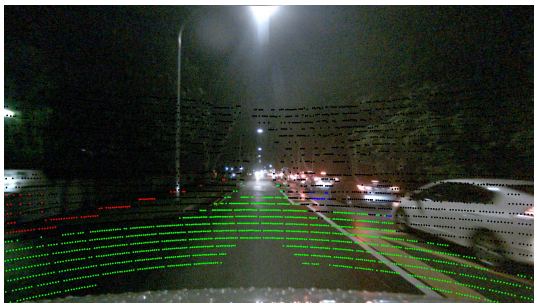


(e) Point cloud top-view: prediction + map layers

Figure A.14: *Projection of LiDAR point cloud in camera image of scene sample (scene-1067) with range images and point cloud top-view with additional map layers.*



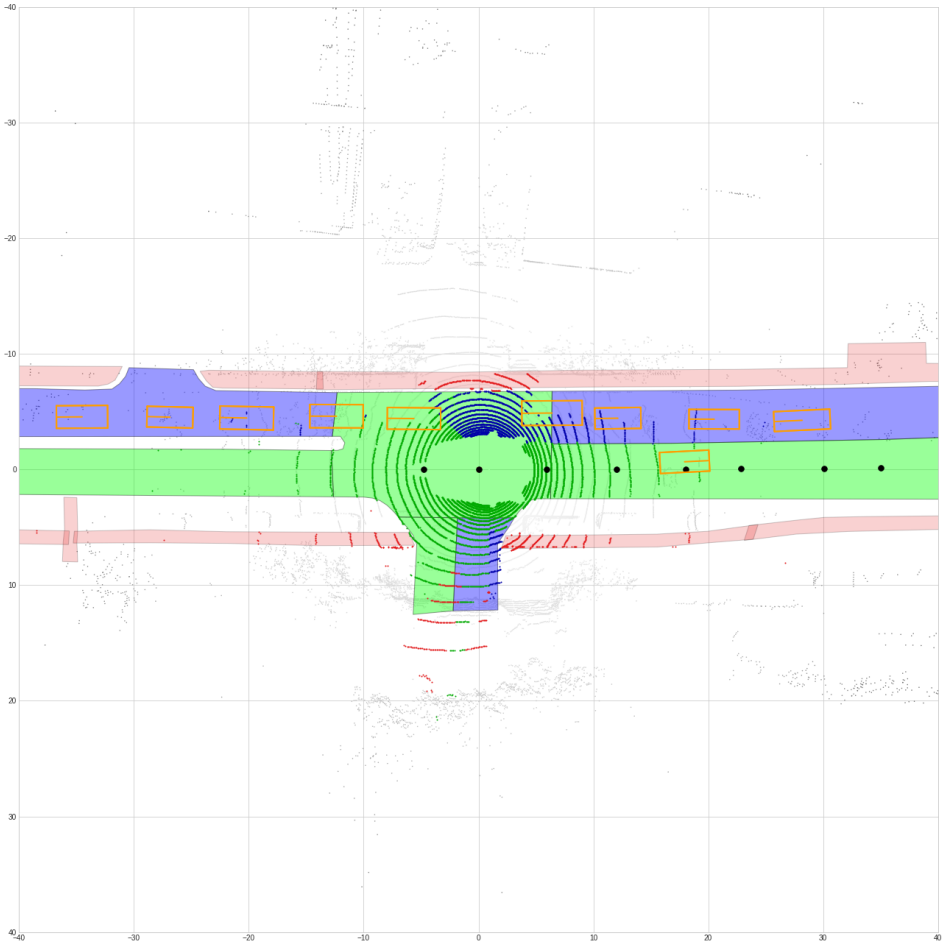
(a) CAM_FRONT



(b) CAM_BACK

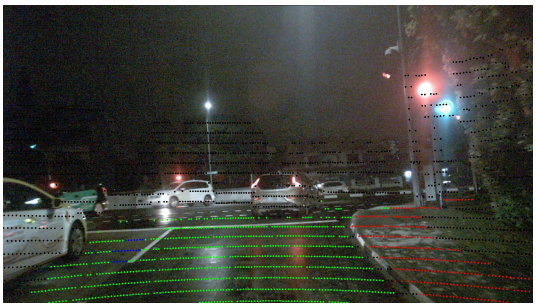
(c) range-image: ground truth

(d) range-image: prediction

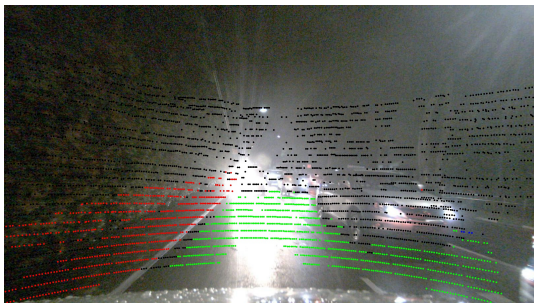


(e) Point cloud top-view: prediction + map layers

Figure A.15: *Projection of LiDAR point cloud in camera image of scene sample (scene-1070) with range images and point cloud top-view with additional map layers.*



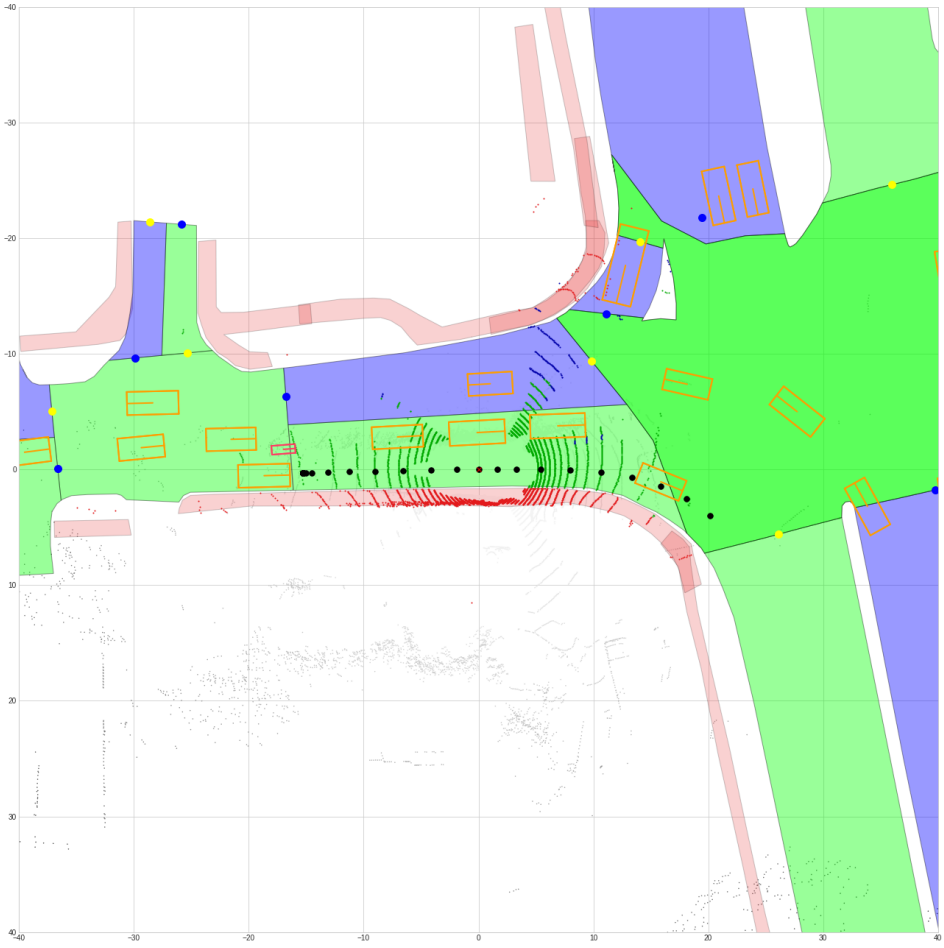
(a) CAM_FRONT



(b) CAM_BACK

(c) range-image: ground truth

(d) range-image: prediction



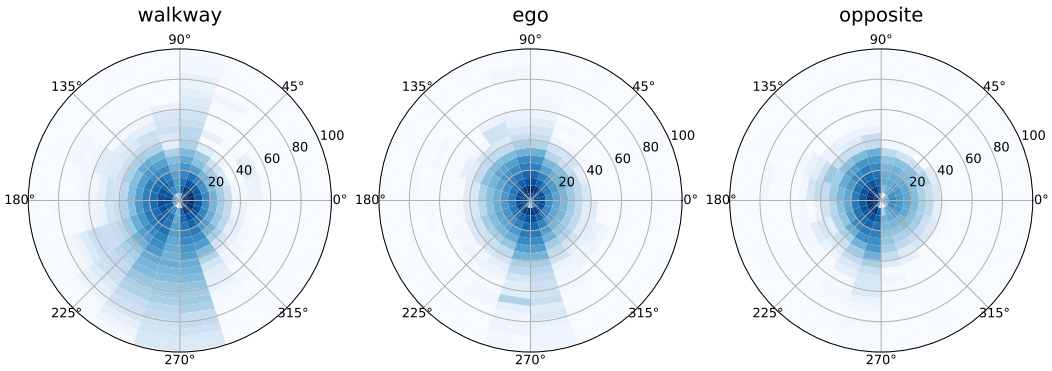
(e) Point cloud top-view: prediction + map layers

Figure A.16: *Projection of LiDAR point cloud in camera image of scene sample (scene-1073) with range images and point cloud top-view with additional map layers.*

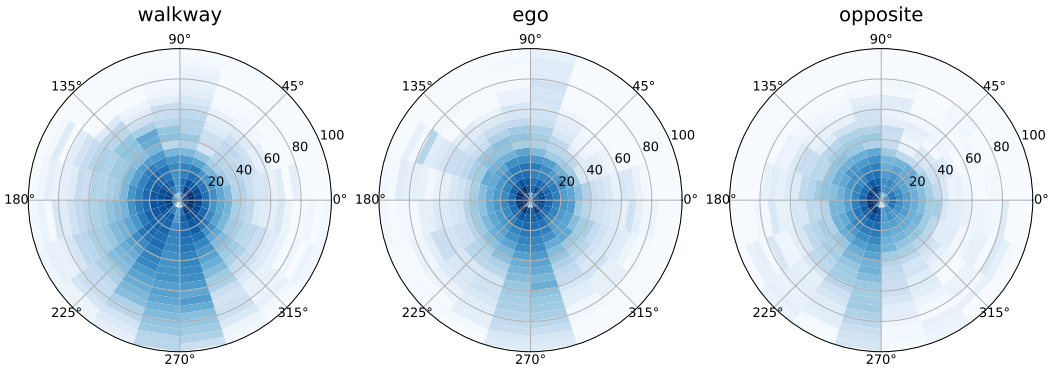
Figure A.17: *Projection of LiDAR point cloud in camera front- and back-view perspective. Providing sequence of scene samples (scene-0269). False detection mostly appears in back-view images, whereas front-view mostly remains stable.*

Figure A.18: *Projection of LiDAR point cloud in camera front- and back-view perspective. Providing sequence of scene samples (scene-0269). False detection mostly appears in back-view images, whereas front-view mostly remains stable.*

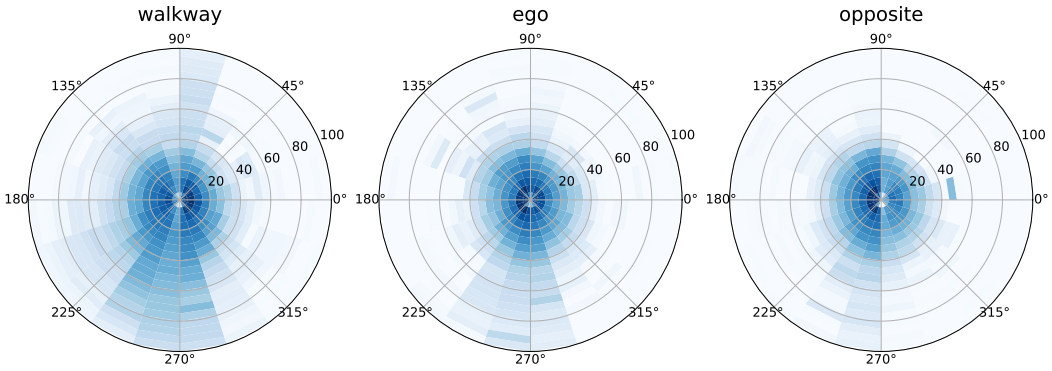
B Quantitative Evaluation



(a) true positive samples

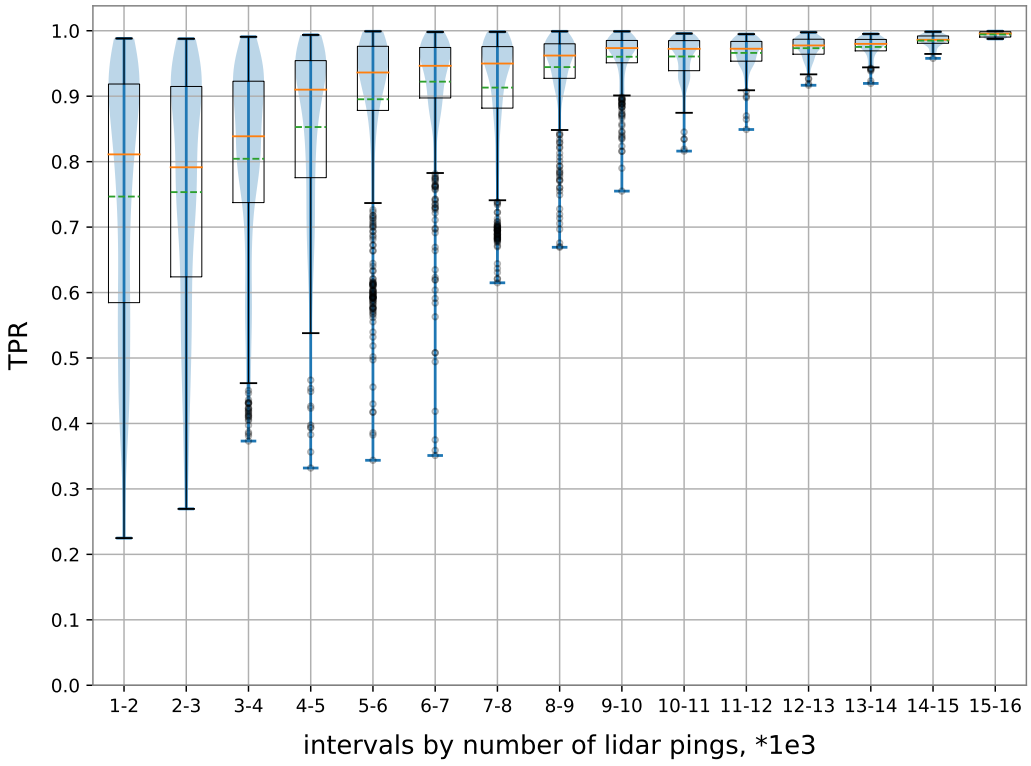


(b) false negative samples

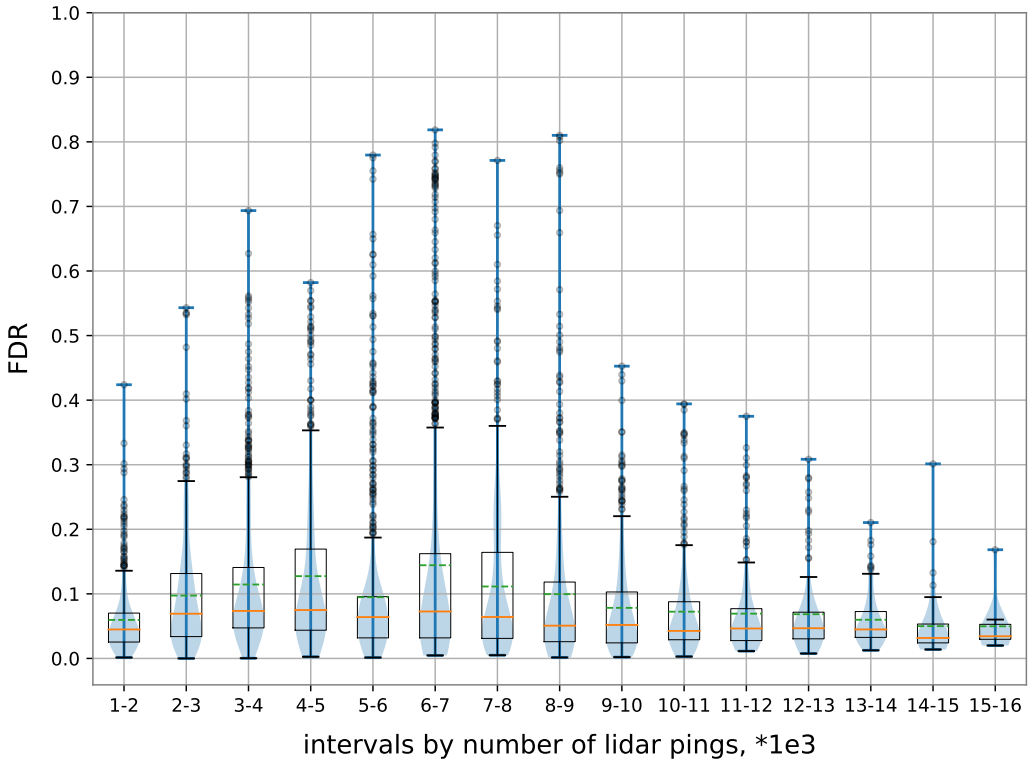


(c) false positive samples

Figure B.1: *Distribution of true positive, false negative and false positive class samples, wrt radial distance measured in [m], illustrated as log scaled polar plot.*

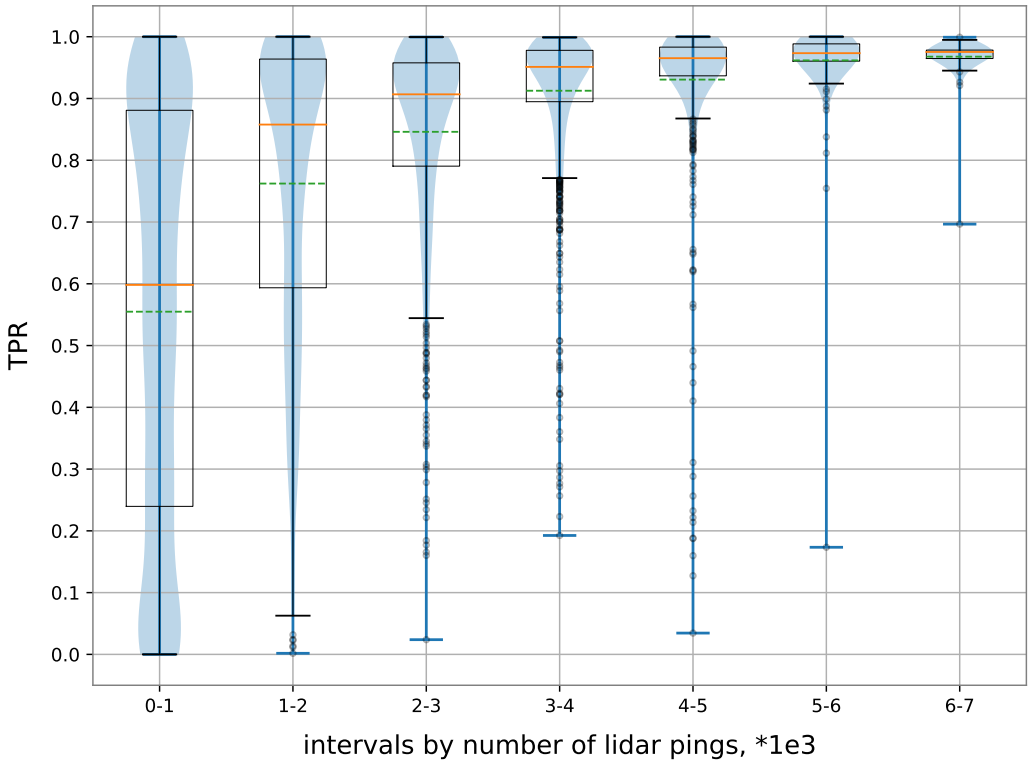


(a) true positive rate

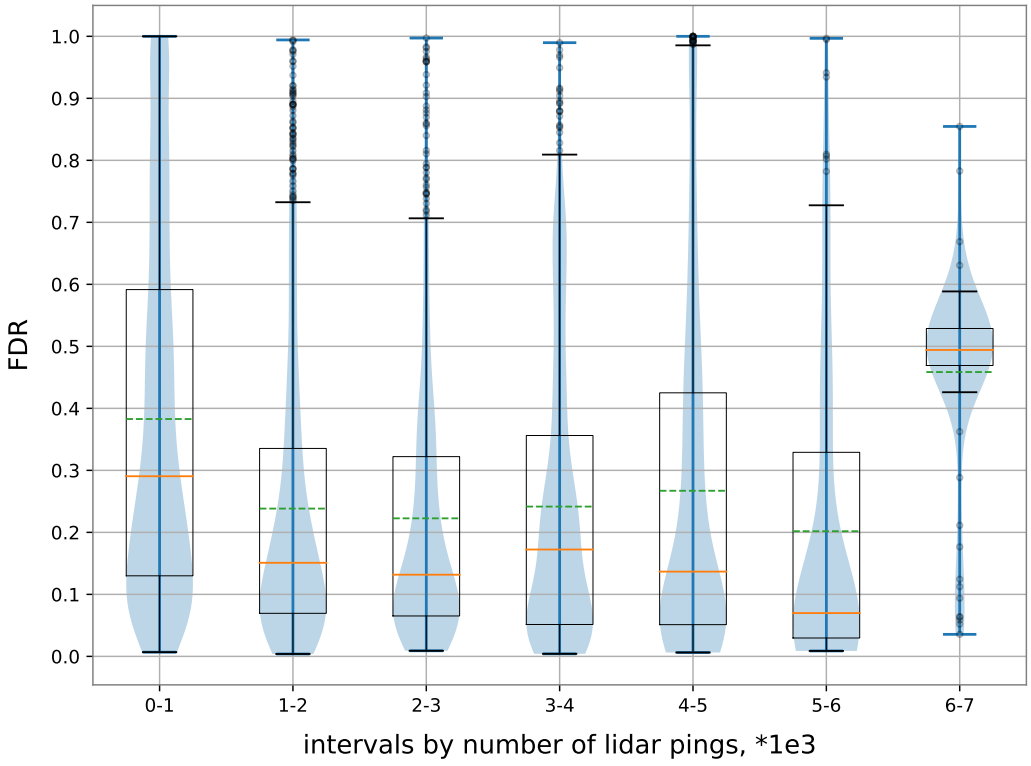


(b) false discovery rate

Figure B.2: *Detection scores for number of LiDAR pings (∗ 1e3), class* ***ego-lane****.*

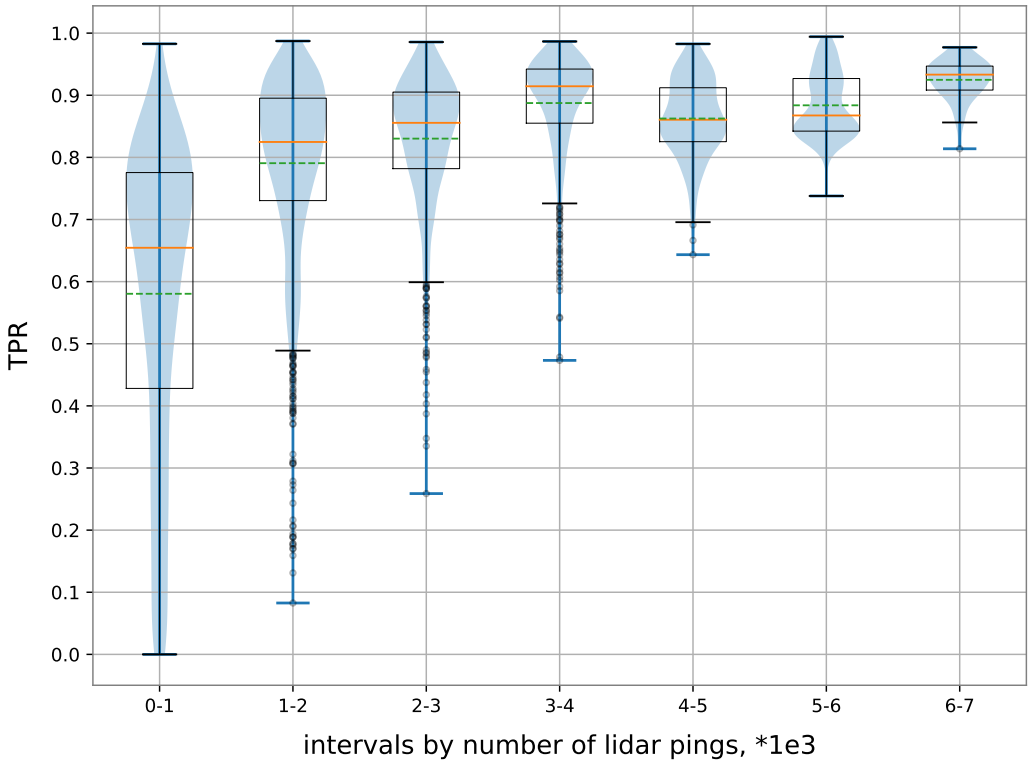


(a) true positive rate

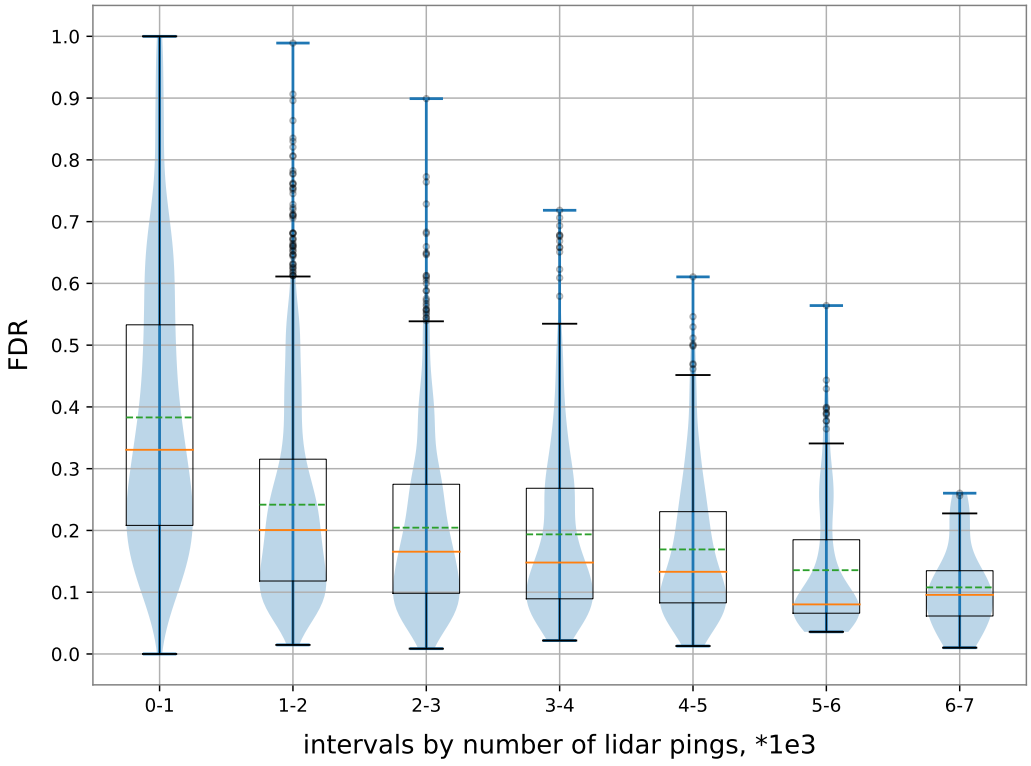


(b) false discovery rate

Figure B.3: *Detection scores for number of LiDAR pings (∗ 1e3), class* ***opposite-lane****.*

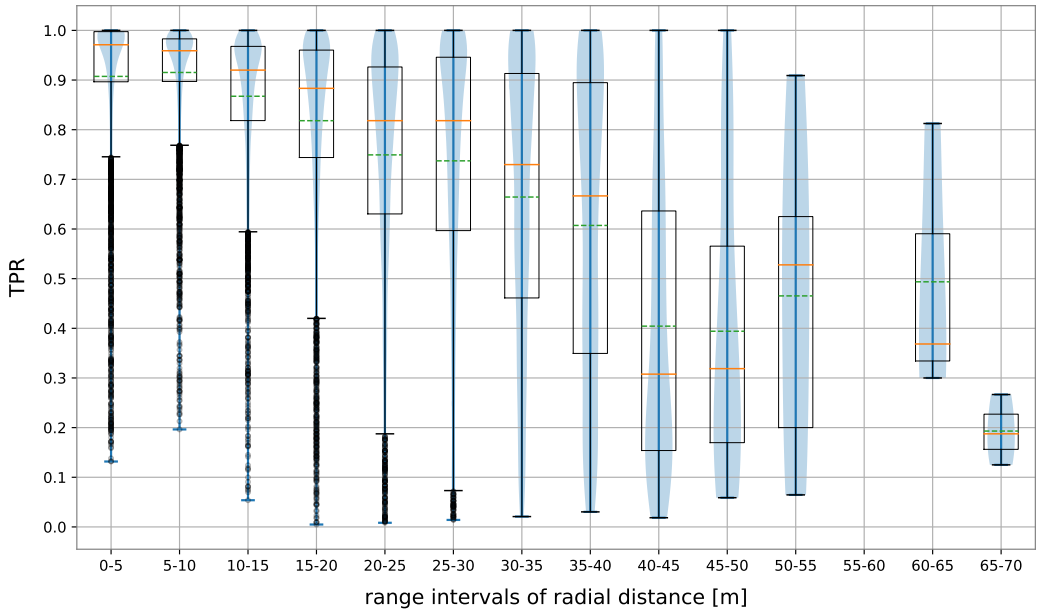


(a) true positive rate

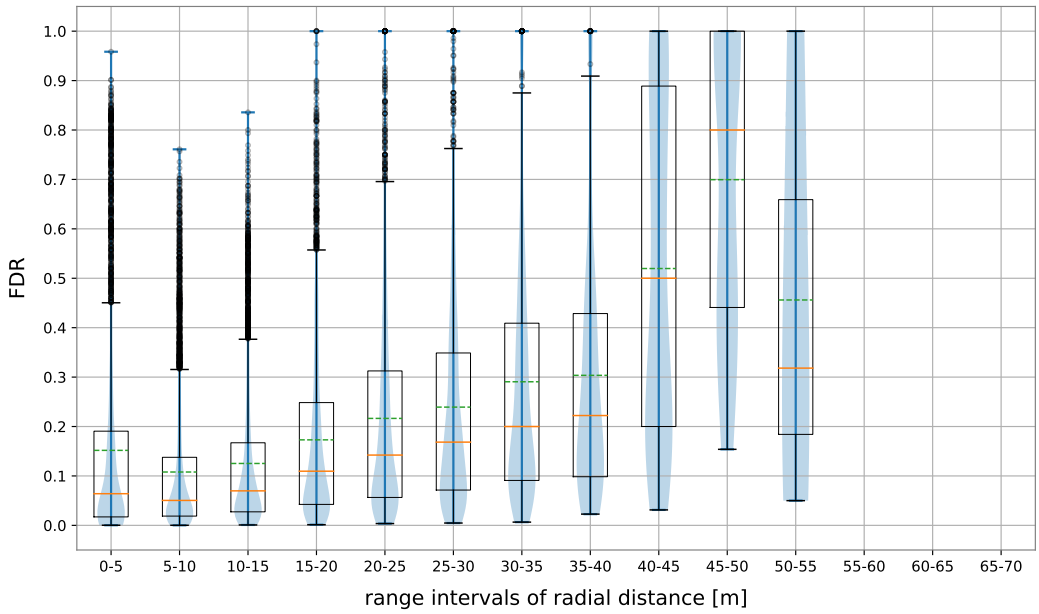


(b) false discovery rate

Figure B.4: *Detection scores for number of LiDAR pings (∗ 1e3), class* ***walkway****.*

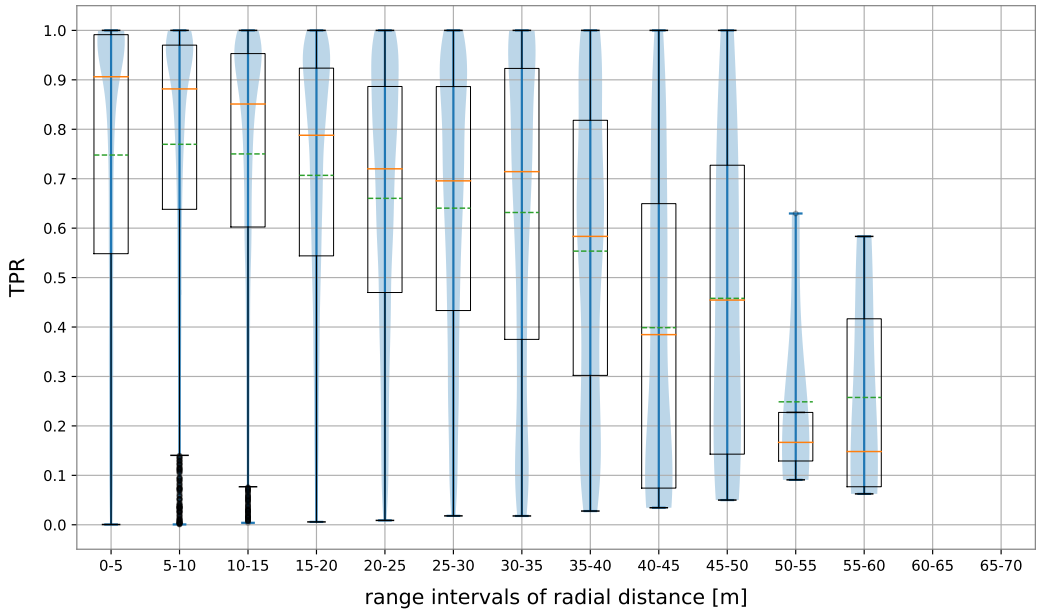


(a) true positive rate

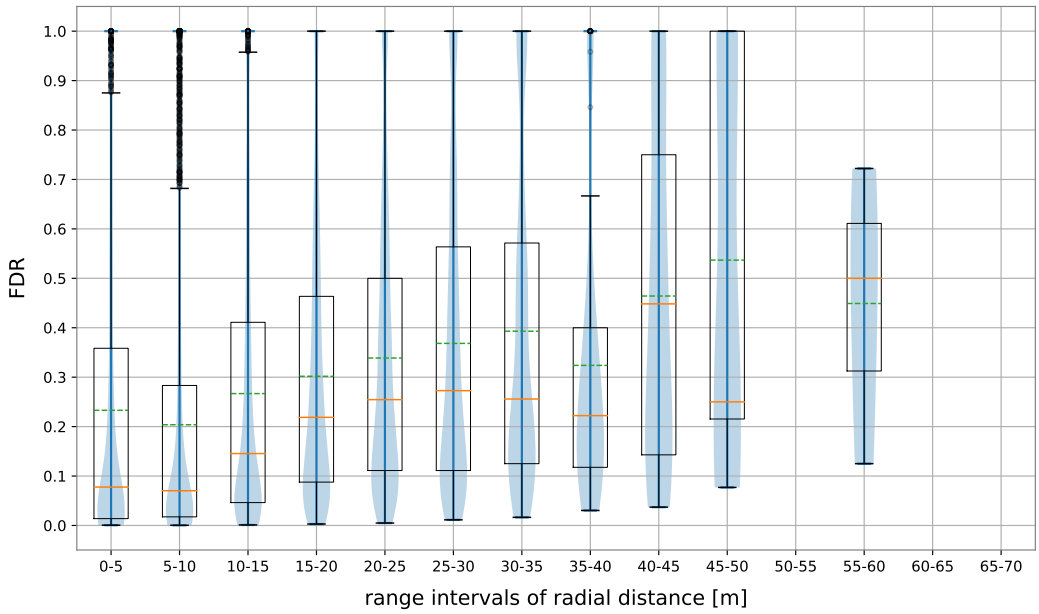


(b) false discovery rate

Figure B.5: *Detection scores for range intervals* [*m*]*, class* ***ego-lane****.*

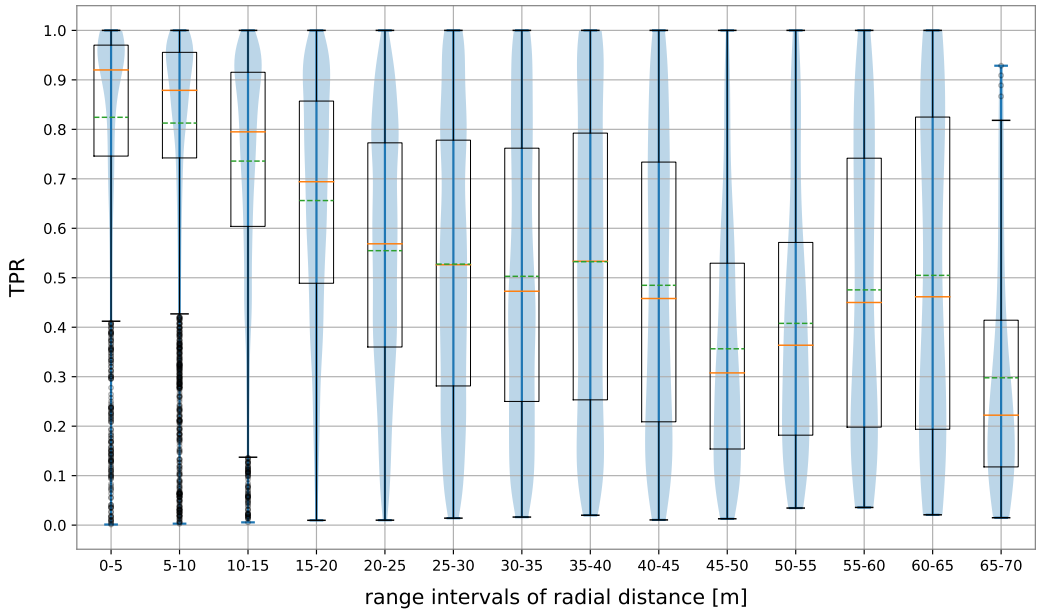


(a) true positive rate

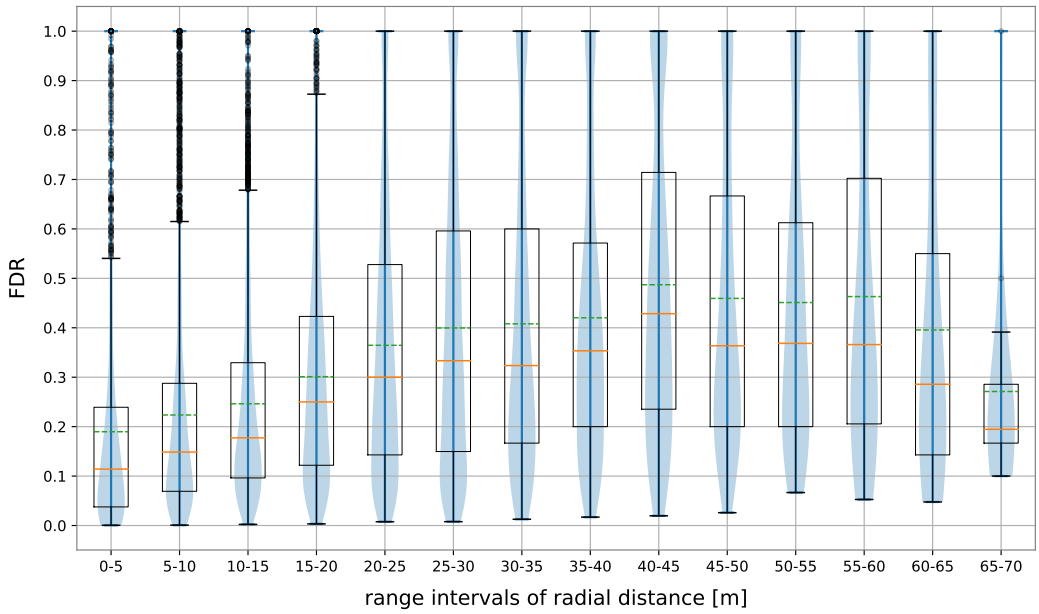


(b) false discovery rate

Figure B.6: *Detection scores for range intervals* [*m*]*, class* ***opposite-lane****.*



(a) true positive rate



(b) false discovery rate

Figure B.7: *Detection scores for range intervals* [*m*]*, class* ***walkway****.*

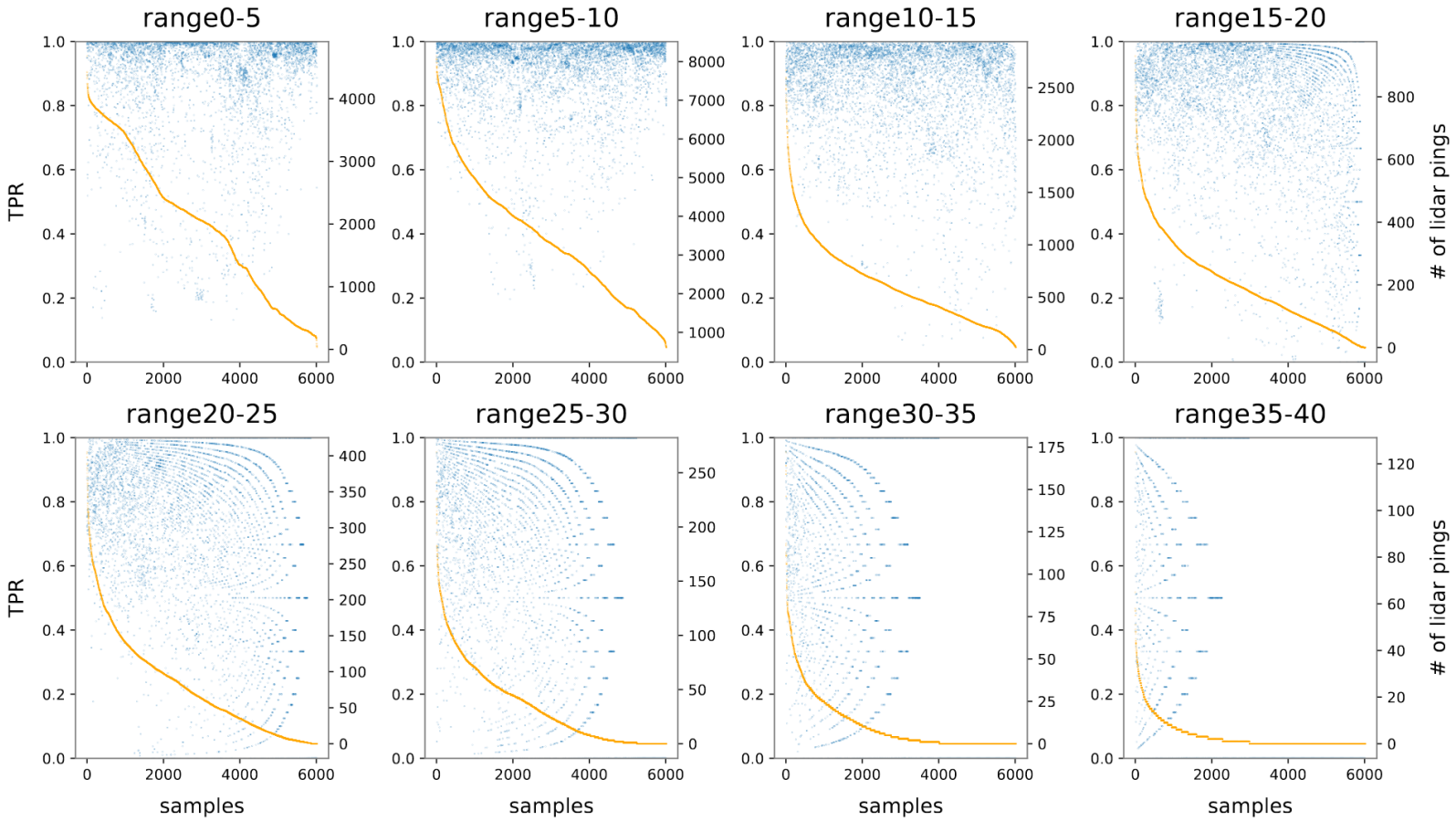


Figure B.8: True positive rate *of class* ***ego-lane*** *wrt range of radial distance from position of ego-vehicle. Blue - true positive scores of validation samples, orange - number of lidar pings per scene sample in descending order.*

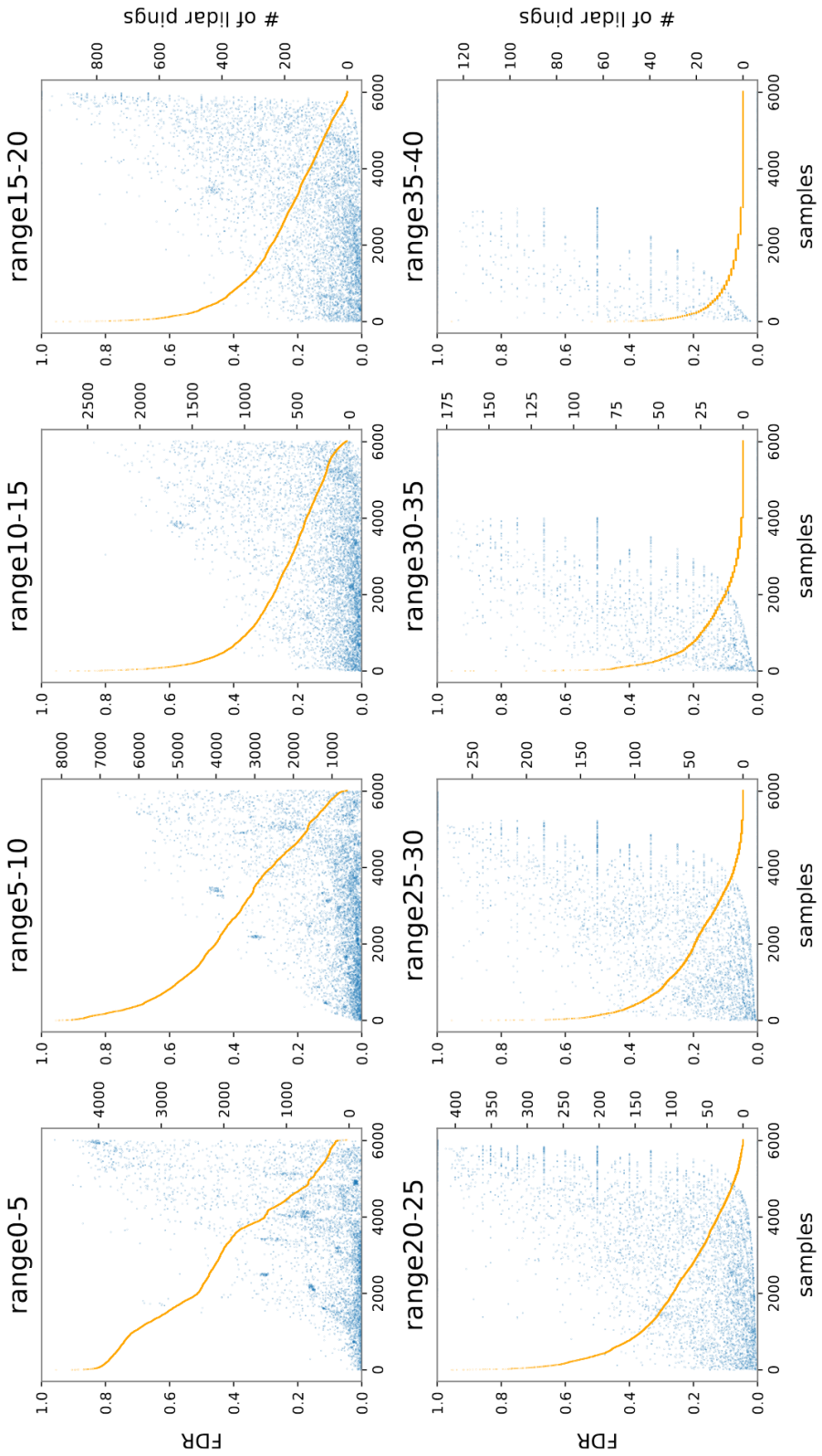


Figure B.9: ***False discovery rate*** *of class* ***ego-lane*** *wrt range of radial distance from position of ego-vehicle. Blue - false discovery rate of validation samples, orange - number of lidar pings per scene sample in descending order.*

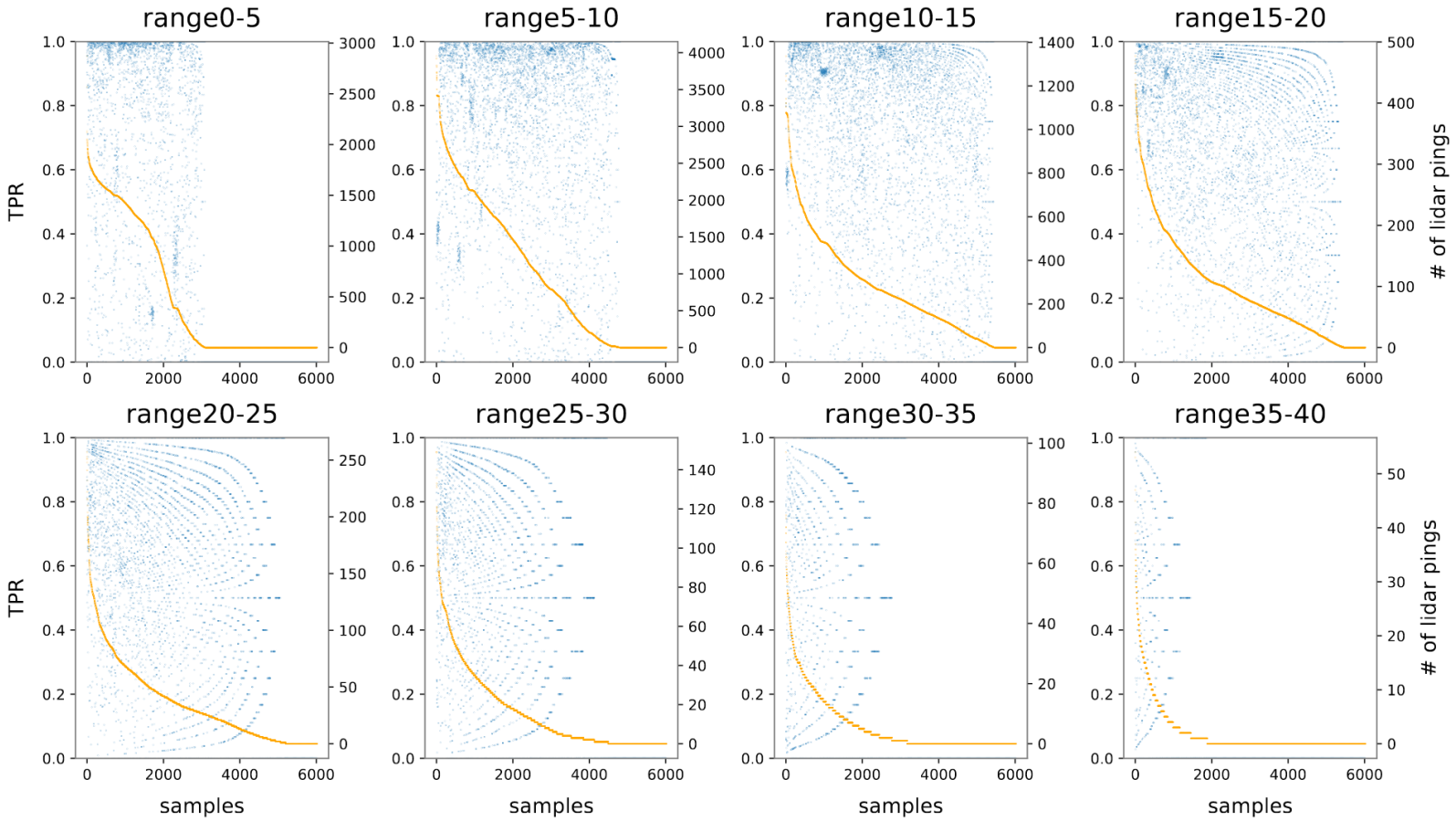


Figure B.10: ***True positive rate*** *of class* ***opposite-lane*** *wrt range of radial distance from position of ego-vehicle. Blue - true positive scores of validation samples, orange - number of lidar pings per scene sample in descending order.*

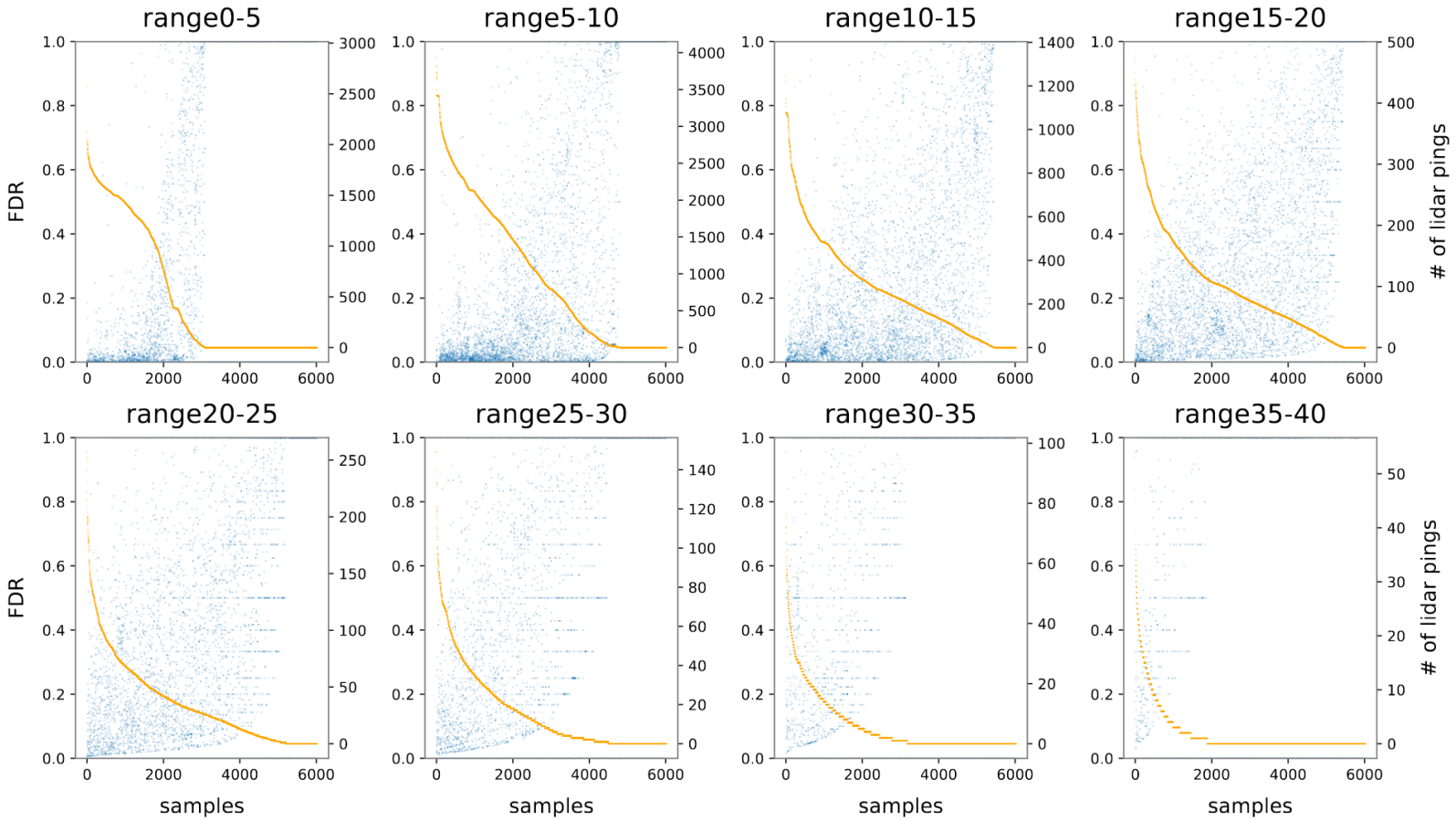


Figure B.11: False discovery rate *of class* ***opposite-lane*** *wrt range of radial distance from position of ego-vehicle. Blue - false discovery rate of validation samples, orange - number of lidar pings per scene sample in descending order.*

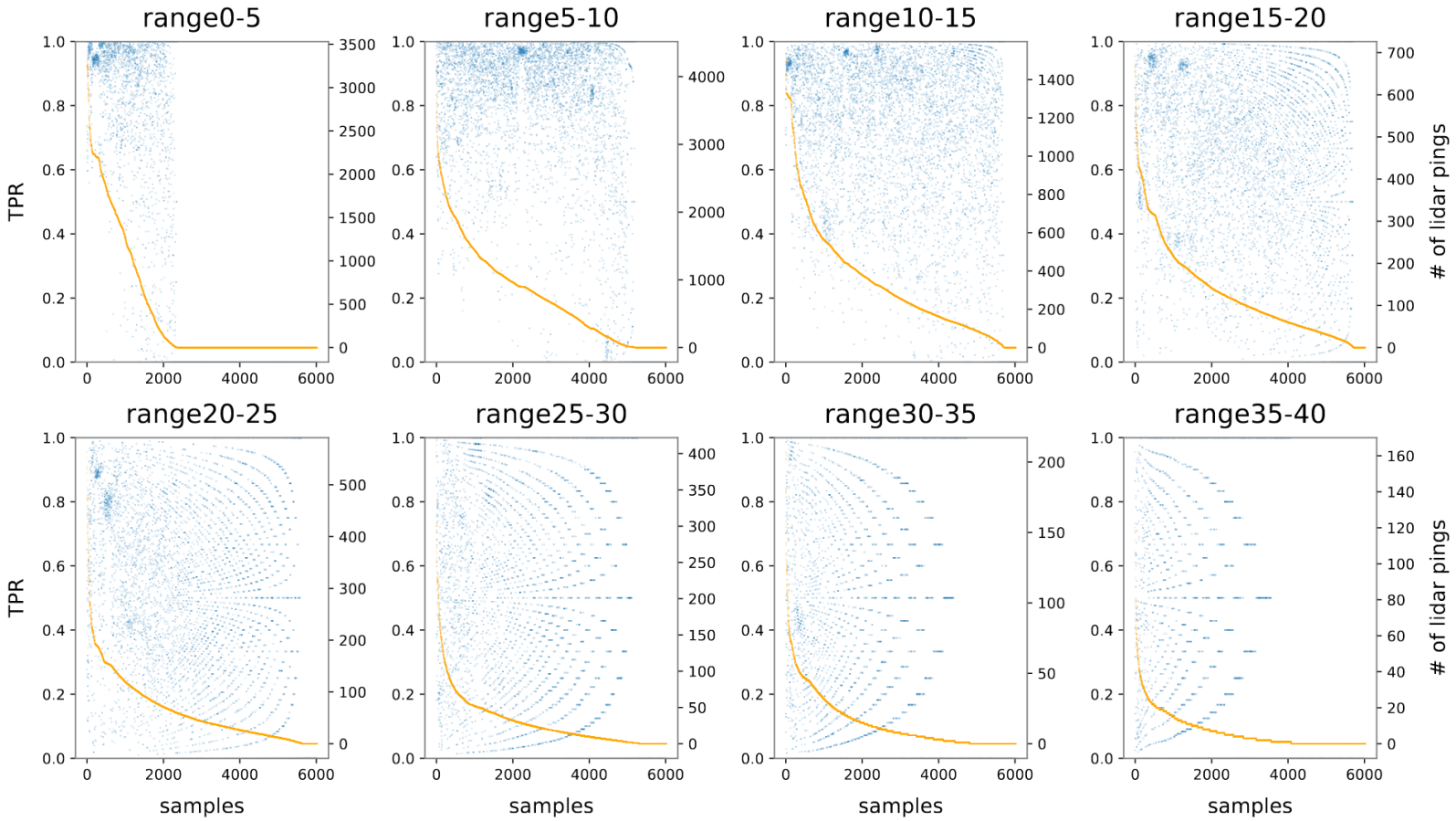


Figure B.12: True positive rate *of class* ***walkway*** *wrt range of radial distance from position of ego-vehicle. Blue - true positive scores of validation samples, orange - number of lidar pings per scene sample in descending order.*

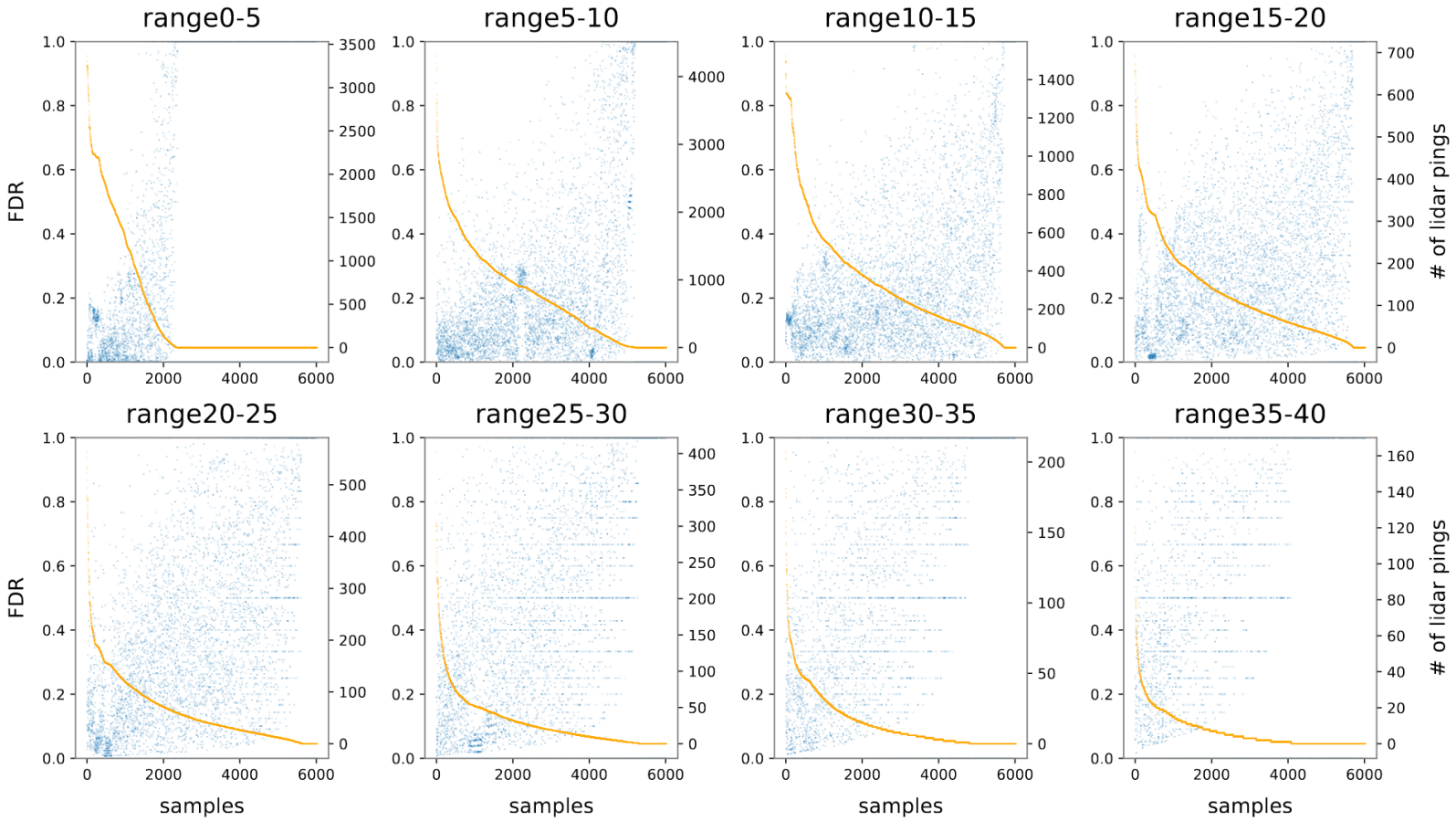
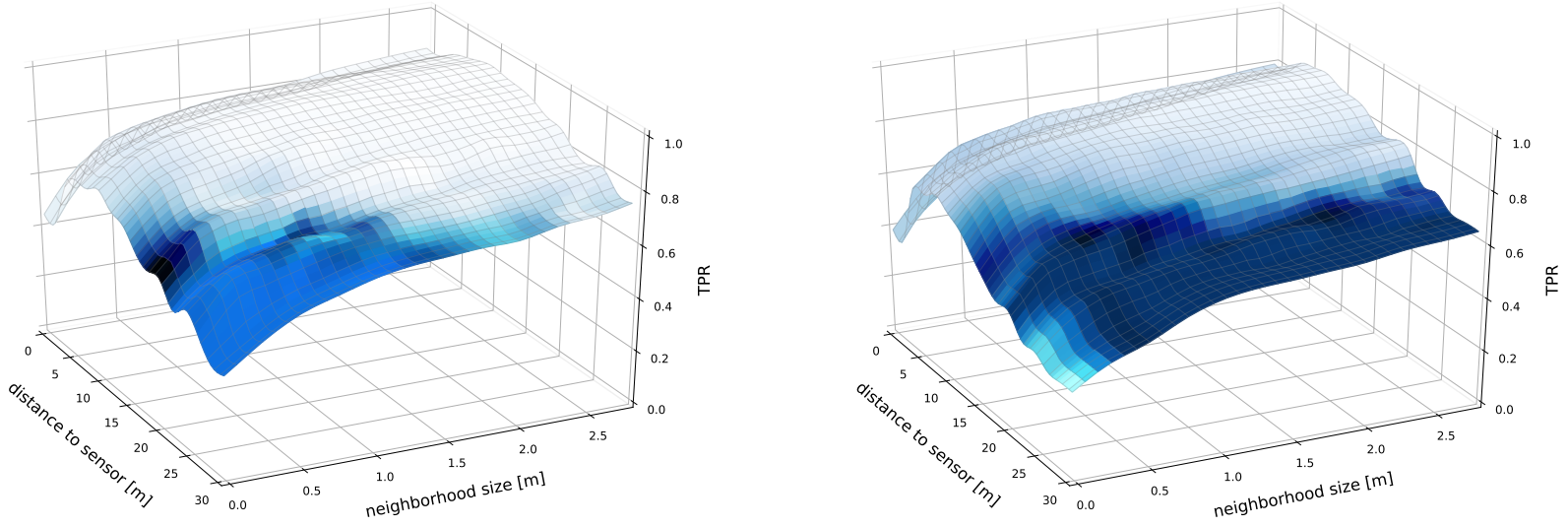


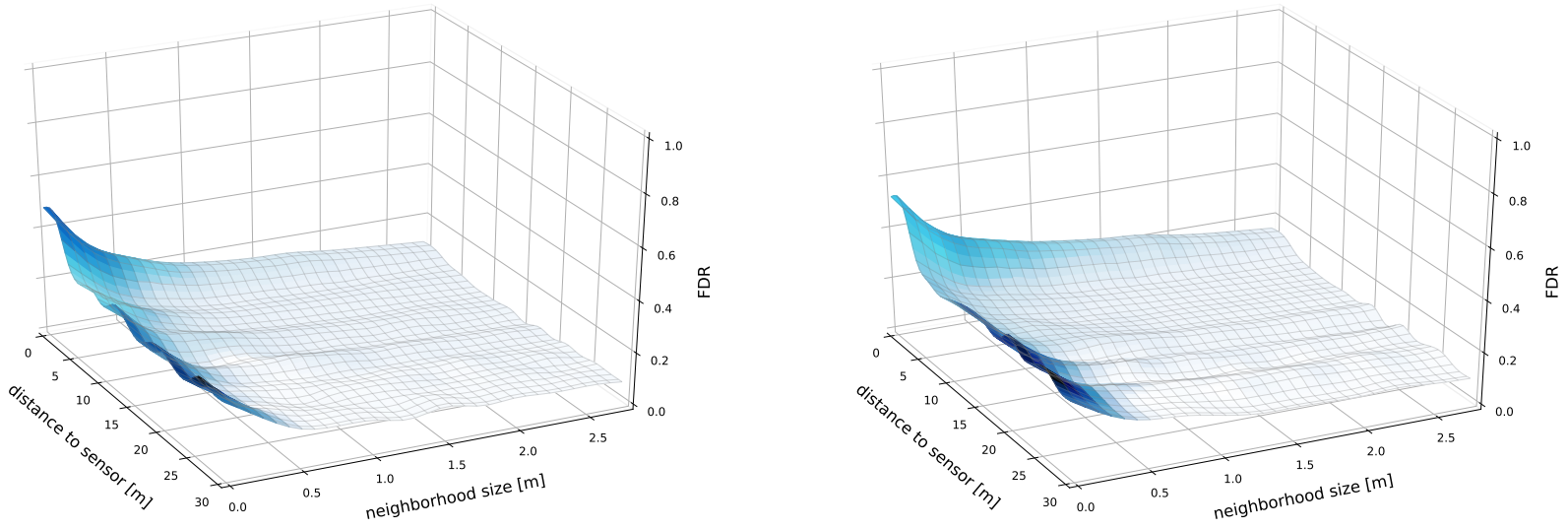
Figure B.13: False discovery rate *of class* ***walkway*** *wrt range of radial distance from position of ego-vehicle. Blue - false discovery rate of validation samples, orange - number of lidar pings per scene sample in descending order.*



(a) Intersection

(b) No intersection

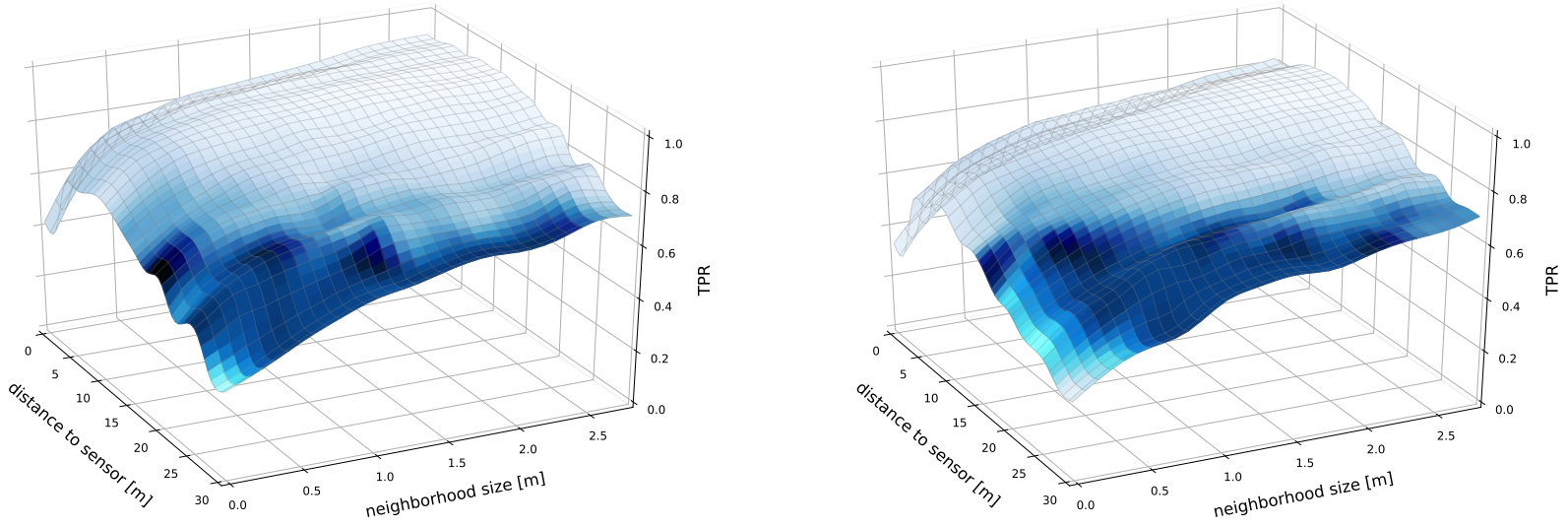
Figure B.14: *TPR of* ***ego-lane*** *samples, grouped by number of LiDAR pings wrt local ranges* ***(5000-9000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*



(a) Intersection

(b) No intersection

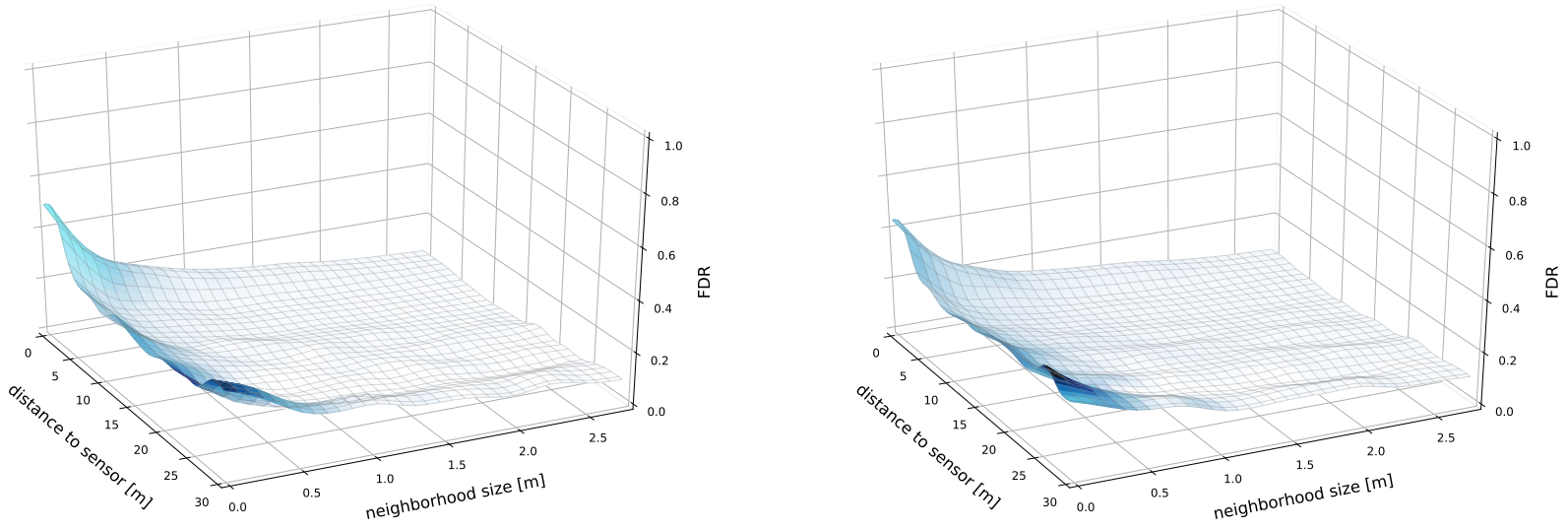
Figure B.15: *FDR of* ***ego-lane*** *samples, grouped by number of LiDAR pings wrt local ranges* ***(5000-9000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*



(a) Intersection

(b) No intersection

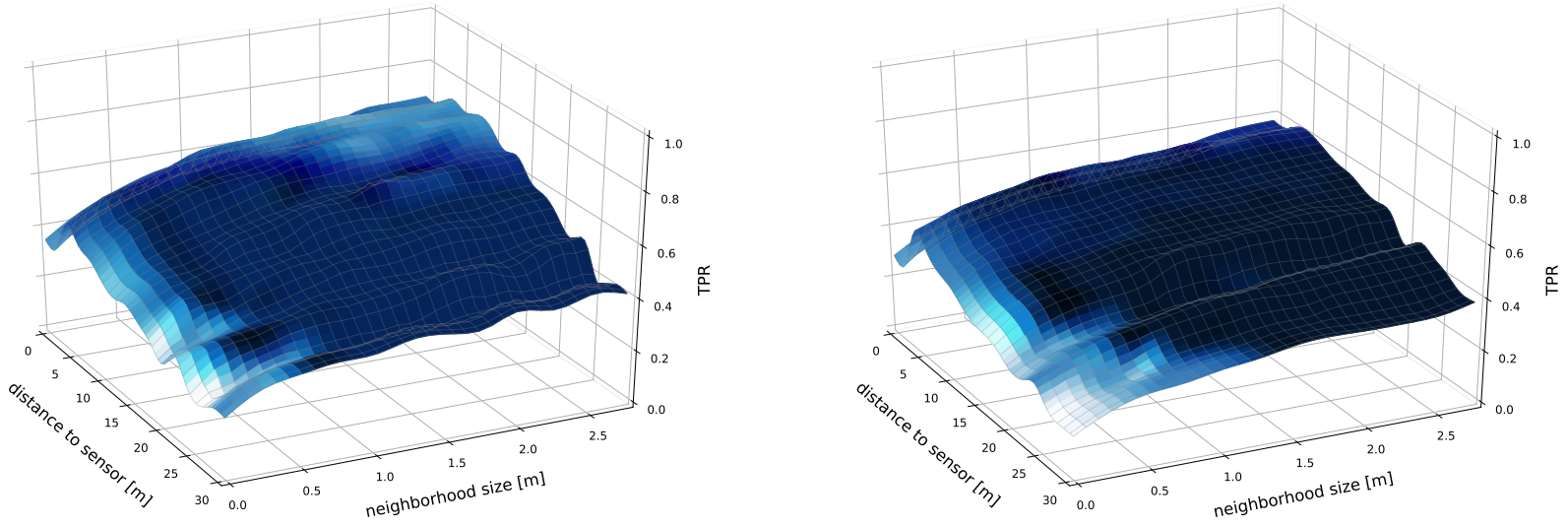
Figure B.16: *TPR of* ***ego-lane*** *samples, grouped by number of LiDAR pings wrt local ranges* ***(9000-16000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*



(a) Intersection

(b) No intersection

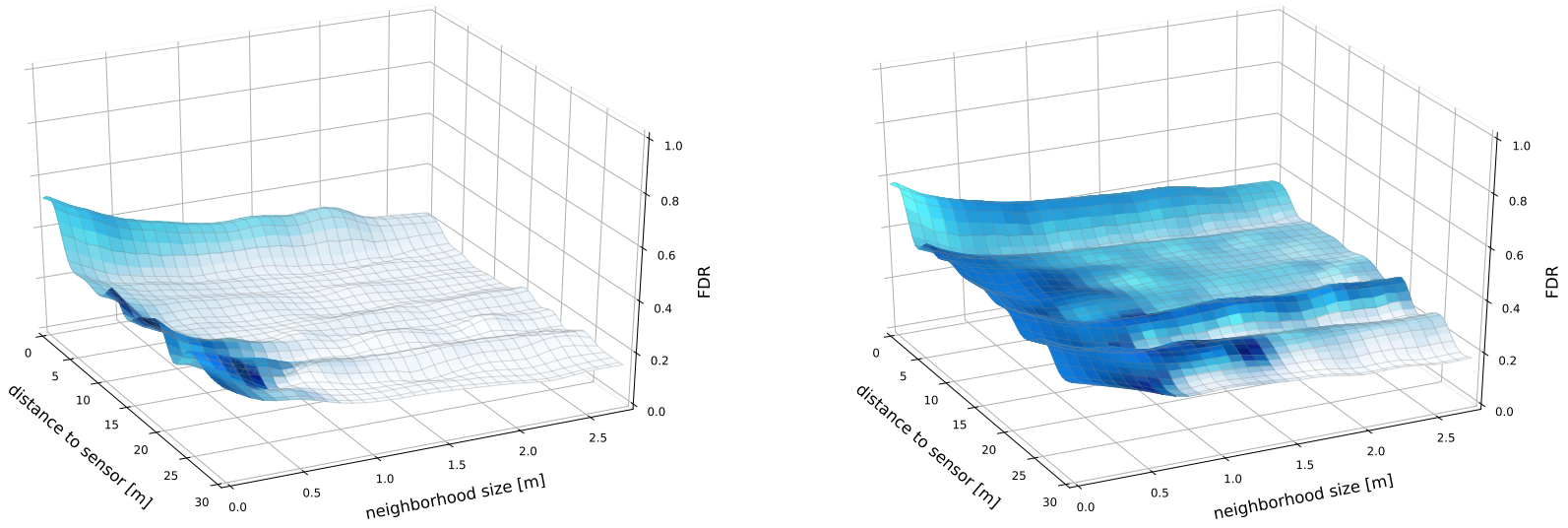
Figure B.17: *FDR of* ***ego-lane*** *samples, grouped by number of LiDAR pings wrt local ranges* ***(9000-16000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*



(a) Intersection

(b) No intersection

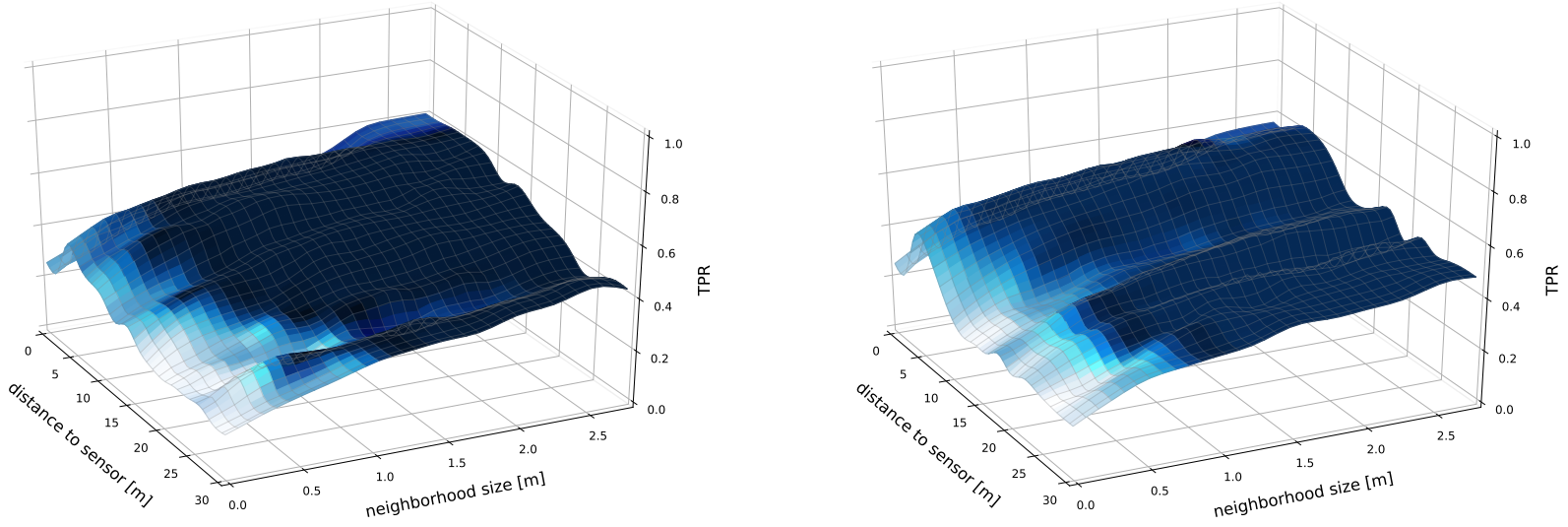
Figure B.18: *TPR of* ***opposite-lane*** *samples, grouped by number of LiDAR pings wrt local ranges* ***(5000-9000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*



(a) Intersection

(b) No intersection

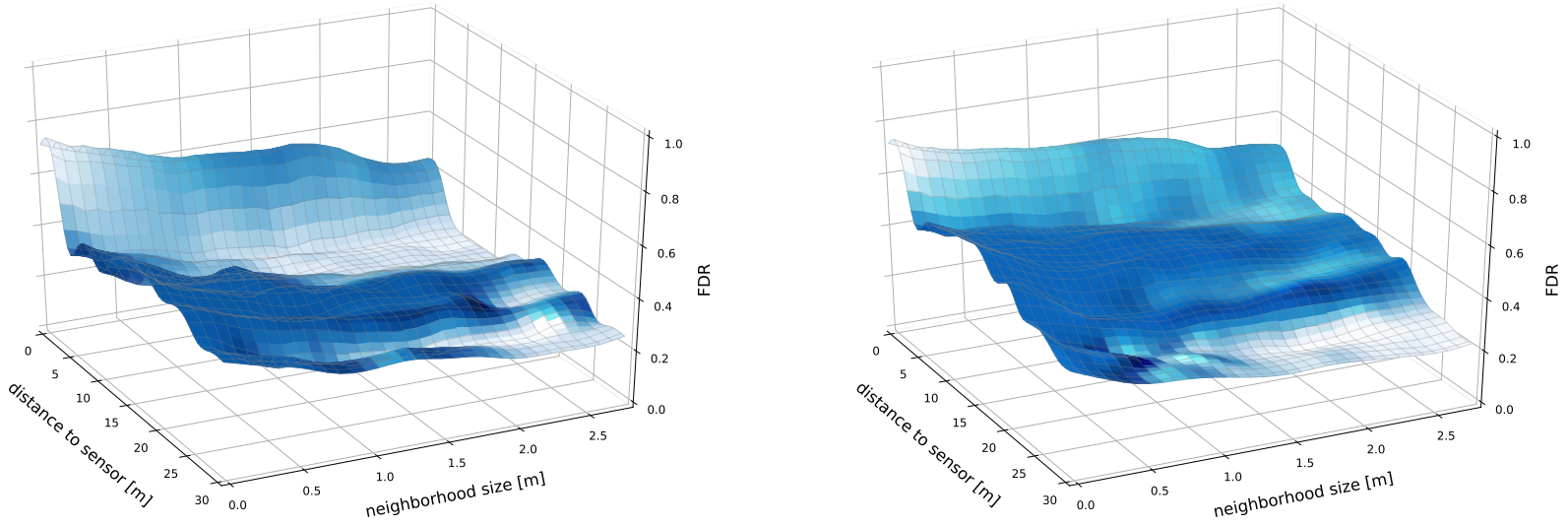
Figure B.19: *FDR of* ***opposite-lane*** *samples, grouped by number of LiDAR pings wrt local ranges* ***(5000-9000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*



(a) Intersection

(b) No intersection

Figure B.20: *TPR of* ***opposite-lane*** *samples, grouped by number of LiDAR pings wrt local ranges* ***(9000-16000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*



(a) Intersection

(b) No intersection

Figure B.21: *FDR of* ***opposite-lane*** *samples, grouped by number of LiDAR pings wrt local ranges* ***(9000-16000)****. 3d mesh plot represents box-plot statistics (mesh layer shows* ***mean*** *of distribution, combined with colour visualization of* ***inter quartile range*** *(IQR), white symbolizes no dispersion, dark blue maximum dispersion.*