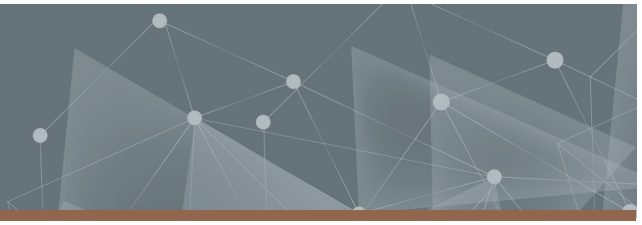
CHALMERS
UNIVERSITY OF TECHNOLOGY



LIDAR & camera reference system for generating ground truth data for lane detection

Master's thesis in Complex adaptive systems

KEVIN VU
ISMAIL GÜLEC

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

LIDAR & camera reference system for generating ground truth data for lane detection

KEVIN VU
ISMAIL GÜLEC

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

LIDAR & camera reference system for generating ground truth data for lane and road edge detection
KEVIN VU
ISMAIL GÜLEC

Supervisor: Oscar Pantzare, Aptiv
Examiner: Lars Hammarstrand, Electrical Engineering

Master's Thesis 2023
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

LIDAR & camera reference system for generating ground truth data for lane and road edge detection
Kevin Vu
Ismail Gülec
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The development of autonomous vehicles, also called self-driving cars, has the potential to revolutionize transportation. Advanced sensors and algorithms enable these vehicles to navigate and operate autonomously. To achieve high levels of safety and reliability, autonomous vehicles require massive amounts of well-labeled data. As a result, data annotation is crucial. As part of data annotation, various elements, such as objects, pedestrians, and road markings, are manually labeled and tagged. Annotating data is time-consuming, costly, and prone to human error. Hence, it is desirable to automate and improve the annotation processes.

This thesis proposes three main ideas: A pipeline for automatically annotating 3D lanes using LIDAR scans and 2D lane labels, a model for lane detection, and lastly, combining past and future inference to improve lane detection. The annotation pipeline considers the utilization of LIDAR scans before and after the current frame to strengthen the ground truth. Our model incorporated two machine learning frameworks: SuperFusion, and M^2-3DLaneNet to generate 3D lane predictions. To represent the 3D lanes, a grid representation of four classes (dashed, solid, other, and empty) was used. Combining the 3D predictions over time improved the performance toward ground truth. The evaluations demonstrate that the model utilizing more LIDAR scans for each image frame performs better, particularly for shorter distances (0-30m). The classified lane's root mean square error (RMSE) in the horizontal direction of the heading is approximately 5.5cm. Future improvements include longer training time and the use of a more complex grid representation to better capture the 3D lanes.

Keywords: Sensor fusion, Deep learning, LIDAR, lane detection, lane marker

Acknowledgements

First of all, we would like to express our sincere gratitude to Aptiv for giving us the opportunity to conduct our master thesis, which has been an enriching experience for our academic and professional growth.

We would like to extend a special thanks to Oscar Pantzare, Tommy Isebäck, and Rikard Hodzic for their invaluable contributions to our research. Our meetings have resulted in insightful inputs that have played a crucial role in shaping the outcomes of our master thesis.

Lastly, we would also like to express our gratitude to our examiner, Lars Hammarstrand, for his valuable critique and feedback throughout the thesis.

Kevin Vu & Ismail Gülec
Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ANN	Artificial neural network
BEV	Bird's-eye view
CNN	Convolutional neural network
GNSS	Global navigation satellite system
GPS	Global positioning system
IMU	Inertial measurement unit
LIDAR	Light detection and ranging
MAE	Mean absolute error
MSE	Mean square error
RMSE	Root mean square error
SGD	Stochastic gradient descent

Contents

List of Figures

List of Tables

1
Introduction

The development of autonomous vehicles has been started or is being started by several automotive manufacturers. A fully automated vehicle will enable safer and more environmentally friendly traffic. Automating certain tasks of a vehicle can reduce the chances of accidents caused by human error, which is one of the leading causes of accidents [1]. Additionally, automated cars can react faster than humans, which can help to avoid accidents or mitigate their severity. One of the main scopes of autonomous vehicle development is to improve sensors (camera, radar, LIDAR, etc.) and the algorithms associated with them. There are multiple sensors that different car manufacturers utilize in the development of autonomous vehicles. One type of sensor that is becoming more popular is called LIDAR, standing for light detection and ranging, with the capability to make an accurate 3D representation of its surroundings within a limited distance [2]. The 3D representation that the LIDAR provides has the potential to yield coordinates close to ground truth to support the detection of different kinds. Therefore it's possible to use the LIDAR as a reference sensor for automatically generating ground truth data for other sensors, reducing the time and cost of producing labels for road data. To further enhance the ability to capture the environment around the car, one can implement sensor fusion between different modalities like camera and LIDAR [3]. Sensor fusion is a technique for combining various sensor modalities to complement features that a particular modality is unable to capture. This can for example be colours, which a camera is able to provide, unlike LIDAR and radar. In addition, it may help reduce uncertainty for features that the fused modalities have in common. This means that depending on the objective, certain combinations of sensor fusion can be more appropriate than others.

The aim of this project is to develop and evaluate lane marker detection for a reference system generating ground truth data offline. The generated ground truth can be used to create a consistent 3D map to compare against a camera acting as an objective sensor. The modalities chosen to be part of the reference system were a LIDAR and a camera, as the LIDAR can retrieve 3D positions with high accuracy while the camera enables the vision of the road. As ground truth, the equivalent detections from the reference system will be used to evaluate the performance of the camera's lane marker and lane marker type detection. Performance of the reference system is dependent on the reference system's detections relative to the ground truth of test data and relative to the camera detections. To be able to use a LIDAR and a camera as reference sensors, the fused reference system detections need to be better than the camera detections and close to ground truth. By utilizing detections from

past and future frames, the ground truth generated by the reference system could be enhanced.

2
Related work

2.1 LIDAR & camera fusion

Three methods of fusion between LIDAR & camera that are commonly used by contemporary authors are: Data-level, feature-level, and decision-level [4]. The fusion methods mentioned are often carried out with the help of deep convolutional neural networks due to them being able to process raw data to extract features and to perform tasks such as object detection [5].

Data-level fusion is a method where data from multiple sensors are concatenated, resulting in a larger quantity of information. One example of data-level fusion is using the camera projection matrix to project LIDAR points to the image plane, a sparse depth image [6] is obtained. The sparse depth image contains a depth value for each pixel coordinate in the image, i.e. the distance from the point and the camera (as the points are projected into the camera frame). Using the depth information from the sparse depth image in combination with the camera image, it is possible to improve the performance of object detection [7].

Feature-level fusion is where multiple features derived from the unique modalities are fused to a higher-dimensional vector, creating a more rich semantic representation of the surroundings. There are different ways to perform feature-level fusion, more specifically, transformers (cross-attention) have proven themselves effective for the state-of-the-art models in the case of 3D object detection [8][9][10]. Before the transformer's cross-attention mechanism is leveraged, extraction of the LIDAR's features is done. Afterward, a query is performed between the extracted LIDAR features and image features using cross-attention to capture the correlating features between the camera and LIDAR.

Decision-level fusion is carried out by combining the results of several local decision processes or classifiers such that a global decision is reached. Since the project aims to generate ground-truth data for lane markers detection using LIDAR & camera, a possible common plane to produce the global decision could be in bird's-eye view (BEV). The procedure consists of extracting LIDAR features and image BEV features which are concatenated, composing a BEV feature map that later on can be decoded. The procedure to fuse LIDAR and camera features in the BEV space is called BEV-level fusion. The current state-of-the-art models take advantage of the BEV-level fusion to improve object detection of various classes including lanes

[11][12][13].

Data-level, feature-level, and decision-level fusion benefit object detection in their own way. SuperFusion is one of the most recent models which incorporate the three fusion methods for LIDAR and camera modalities, achieving performance that outperforms the current state-of-the-art baseline methods [14]. The SuperFusion model organizes the fusion techniques as follows: Data-level, feature-level and lastly BEV-level, resulting in a post-processed BEV map that can be used for path-planning. The post-processed map consists of lane markers, road edges, and pedestrian crossing detections. As the post-processed map generated by SuperFusion's model is at BEV-level it doesn't fully accomplish the desire to produce the map in 3D. However, the framework of SuperFusion can be re-purposed to fit into our aim. This is explained in the next section.

2.2 Generating 3D lanes

Having an accurate representation of the road a vehicle travels is one of the key ingredients for autonomous vehicles. In the previous section, we discussed the different sensor fusion techniques between LIDAR & camera for which they all showed improvement for a more accurate representation of the world. The authors of SuperFusion demonstrated that combining all three fusion methods would improve performance over models utilizing different fusion strategies. However, the SuperFusion method produces a BEV-level map that might not accurately represent roads with elevation. In addition, no investigation of the performance of SuperFusion on roads with elevation was done. To ensure that a correct representation of the surrounding road is formed, a 3D map would therefore be more beneficial.

There have been a number of models using LIDAR & camera that's been presented over the years which generate 3D road/lane detection [15][16][17]. The models mentioned were released between 2018 and 2021. However, none of these methods have similar fusion strategies as SuperFusion, unlike a model called M^2-3DLaneNet which was released at a similar time (late 2022) as SuperFusion [18]. Because of these factors, we believed that it would be more relevant to utilize the prediction head of the M^2-3DLaneNet model. The biggest difference between the SuperFusion and M^2-3DLaneNet model is that the latter has a 3D lane detection head. The idea is to make use of the SuperFusion framework for its fusion strategies and employ the 3D lane detection head from the M^2-3DLaneNet model. The output from the proposed framework would be the detections of our desired classes: lane markers represented as dots in 3D coordinates. The intricate details of how this is done will be explained in chapter 3.

3
Theory

3.1 Background theory

This section aims to provide the underlying knowledge of used sensors and machine learning theory before exploring the SuperFusion and M^2-3DLaneNet model in depth. It may also serve as a reminder for the more experienced reader regarding certain areas which the authors of the two models touch upon.

3.1.1 Sensors

The development of autonomous vehicles has required to implement different sensors over the years in order for the vehicle to gain certain knowledge of the surrounding environment [19]. Sensors utilized by autonomous vehicles can be divided into two categories: exteroceptive and proprioceptive. Exteroceptive sensors calculate the distance from the vehicle to an object while proprioceptive sensors measure internal values of the vehicle, such as velocity, battery charge, and joint angles.

3.1.1.1 LIDAR

LIDAR is an exteroceptive sensor that became popular within the autonomous vehicle industry due to the LIDAR's accurate precision for measuring distances. The way LIDAR works in principle is that it sends out a pulsed laser of light which is then reflected back. The time it takes for the laser light to be sent and reflected back is measured which enables the calculation of distance d, as can be observed in Equation 3.1.

$$d = \frac{t \cdot c}{2}, \tag{3.1}$$

where t is the time for the LIDAR to receive the reflected laser light and c is the speed of light. The division of 2 is because t accounts for the time the signal reaches the object and is reflected back. In addition to the LIDAR's precision, it also offers a visibility of 360° and a measuring frequency greater than 150 kHz. A LIDAR scan results in a point cloud, which is a set of data points that can represent the surrounding environment in 3D.

3.1.1.2 Inertial measurement unit

An inertial measurement unit (IMU) lies under the proprioceptive category and is an electronic device that measures a specific force, angular rate, and magnetic field.

IMU uses a combination of accelerometers, gyroscopes, and magnetometers, where one of each represent one of the orthogonal axis X, Y, and Z. Because IMUs are capable of position and heading relative to global reference they are commonly used to guide autonomous vehicles.

3.1.1.3 Global navigation satellite system

Global navigation satellite system (GNSS) is a proprioceptive system that utilizes several satellites to provide the coordinates of the receiver [20]. There exist four GNSS systems, with one of the most well-known being the global positioning system (GPS). Each satellite in a GNSS system transmits a signal that can be picked up by a receiver on the ground which measures the delay for the transmission and reception. Through this, the distance from each satellite to the receiver can be calculated, and by using several satellites one can determine the location (longitude/latitude) and altitude/elevation of the receiver. Its application can be found in various sectors, such as aircraft, agriculture, tectonic monitoring, etc. In the autonomous vehicle industry, GNSS has been used to identify a vehicle's location enabling tracking and navigation of the vehicle.

3.1.2 Machine learning & neural networks

Machine learning is a field within computer science where one builds algorithms that enables machines to learn through experience [21]. There are different types of machine learning models that affect the amount of human intervention required, these are primarily: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. In supervised learning, the data used to teach the model is labeled beforehand so that the algorithm can determine its accuracy. Unsupervised learning is when the data is not labeled and instead lets the algorithm attempt to find a pattern within the data. Semi-supervised learning is where the data is divided into ordered and non-ordered data that is meant to steer the algorithm to make its own conclusions, enabling the algorithm to learn to label the data. Reinforcement learning is where the algorithm makes decisions and is rewarded/punished according to a system. The goal for the algorithm is to maximize the potential reward it can gain and therefore learn to make more favorable choices.

Many machine learning algorithms have been developed over the years ranging from regression, Gaussian processes, and decision trees to artificial neural networks. In particular, artificial neural networks have gained immense popularity mostly due to the availability of data and processing power. Inspired by biological neural networks, artificial neural networks (ANN) is a computing system built with the intent to mimic the process of how biological neurons pass information to each other [22]. ANNs generally consist of three type of layers: input, hidden, and output. The input layer is where information is passed for it to be processed by algorithms in the hidden layer and finally, be observed in the output layer. Each of these layers contains a collection of neurons that are connected to each other through edges, with every edge having an associated weight and threshold value. The cornerstone

of ANN is that it learns by adjusting the connection strength between each neuron pair, in other words, the weight and threshold value for each edge. ANN performs most optimally when provided with enough time, "good" quality data, and also a sufficient amount of data. ANN can become effective in solving tasks in numerous fields, such as computer vision, classification, natural language processing, etc. There exist multiple architectures of artificial neural networks, each designed to solve certain problems. To remain within the scope of the thesis, only the necessary architectures will be presented in detail. The first architecture that will be presented is the feed-forward neural network, which is the first type of artificial neural network devised, illustrated in Figure 3.1.

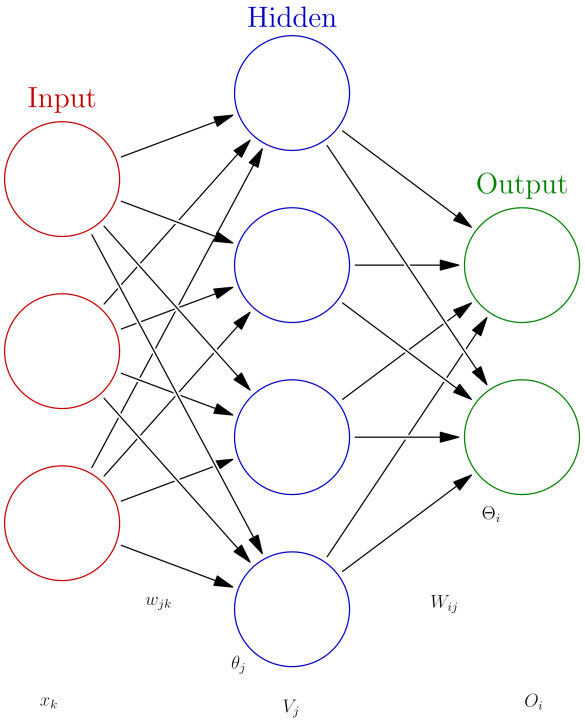


Figure 3.1: An illustration of a feed-forward neural network which is a type of artificial neural network [23]. It has three types of layers: input, hidden, and output. Each layer has a set of arbitrary numbers of neurons, with each circle representing a neuron. Each neuron has a connection (edge) to every neuron on their right side represented with an arrow. Note that the arrow points one way, indicating that information is sent in the same direction; hence the name feed-forward. The associated weights and thresholds for the edges between the input and hidden neurons are denoted w_{jk} and θ_j. The edges between hidden and output neurons correspond to W_{ij} and Θ_i. The variables x_k, V_j and O_i embody the input, hidden, and output values with their definition expressed in Equation 3.2, Equation 3.3,Equation 3.4 respectively.

The input of the network is defined in Equation 3.2.

$$\boldsymbol{x}^{(u)} = [x_1^{(u)}, x_2^{(u)}, ..., x_N^{(u)}], \quad u = 1, 2, 3, ..., p, \tag{3.2}$$

where u is the label for the different input patterns. N is the number of input terminals, which in the case of Figure 3.1 is $N = 3$. The hidden neurons are defined in Equation 3.3

$$V_j = g(b_j), \quad b_j = \sum_k w_{jk} x_k - \theta_j, \tag{3.3}$$

where $j = 1, 2, ..., H$ is the index for each hidden neuron, g is an activation function and b_j is called the local field, which is the sum of all weights associated with the j:th hidden neuron minus its threshold. The output layer's neurons perform similar computation as Equation 3.3, shown in Equation 3.4

$$O_i = g(B_i), \quad B_i = \sum_j W_{ij} V_j - \Theta_i, \tag{3.4}$$

where $i = 1, 2, ..., M$ is the index for each output neuron, each associated with weights W_{ij} along with their thresholds Θ_i. As mentioned previously, artificial neural networks learn by adjusting their weights and thresholds. Depending on the task, certain machine learning models can be more desirable. For feed-forward neural networks, the primary usage is for both regression and classification with supervised learning. Since the network learns by using supervised learning, it needs to have a target, which is defined as $\boldsymbol{t}^{(u)} = [t_1^{(u)}, t_2^{(u)}, ..., t_M^{(u)}]$. The goal here is to choose the weights and threshold such that the network produces an output that is identical to the targets, as shown in Equation 3.5

$$O_i^{(u)} = t_i^{(u)} \quad \text{for all } i \text{ and } u. \tag{3.5}$$

Having the outputs match the targets can be achieved in two ways. The first method involves applying small changes to the weights until the results match, in other words satisfy Equation 3.5. The second method is defining a loss function E with a global minimum that satisfies Equation 3.5, compute its gradient, and use them to update the weights with an optimization algorithm to minimize the loss function. The latter is called backpropagation or error correction and is used more commonly, especially for a network that contains multiple hidden layers due to more weights and threshold being introduced that needs adjustment.

3.1.2.1 Loss function

When optimizing, a loss function is used for achieving the same output as a target, corresponding to a loss function value equal to 0. For an artificial neural network that utilizes backpropagation, the loss function serves as a performance metric for the network which they will attempt to minimize using an optimizer (subsubsection 3.1.2.2) for adjusting its learnable parameters. There exists numerous loss functions which can be applied to artificial neural networks depending on the task at hand. As this thesis aims to classify and generate 3D points, the most appropriate type of loss functions is classification and localization.

For classification, the main loss function implemented is cross entropy, which can handle binary classification (sigmoid cross entropy) and multi classification (softmax cross entropy) [24]. The cross entropy is a measure of the distance between the

predicted output and the target, a zero cross entropy indicates that the output and target are the same. Sigmoid and softmax are activation functions that are applied to the output of the network. However, if the data provided has a class imbalance, the network will inherit this bias if binary cross entropy is used. A potential solution to this could be to use balanced cross entropy instead, but it cannot give proper feedback for the samples in which the network repeatedly makes mistakes during training, so-called hard examples. This is where focal loss comes in, which is based on cross entropy with a modification to enable the network to shift its focus more on classes that the network performs worse on [25]. The mathematical expression of focal loss is given in two forms: Equation 3.6 and Equation 3.7.

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

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{3.7}$$

γ is called the focusing parameter which was implemented to take care of hard examples. If $\gamma = 0$ the first form would result in the original form of cross entropy, while the second form would regain the form of balanced cross entropy. A value of $\gamma > 0$ reduces the relative loss for well-classified samples, with $\gamma > 0.5$ resulting in more focus on hard examples. As can be noted, the second form of focal loss contains a factor α_t called the weighting factor. It works similarly to balanced cross entropy where a higher α_t value makes the value of the loss function become higher if the network classifies the class wrong, which forces the network to pay more attention to it.

In the case of localization loss, the most commonly used functions in regression tasks are absolute loss and square loss functions. The absolute loss measures the absolute error between the prediction and target values, expressed in Equation 3.8

$$E(y, f(x)) = |y - f(x)|, \tag{3.8}$$

where y denotes the target and $f(x)$ the predicted value. The square loss function is very similar to the absolute loss, with the difference being that the whole term on the right-hand side is squared. This can be seen in Equation 3.9.

$$E(y, f(x)) = (y - f(x))^2 \tag{3.9}$$

As can be noted, the square loss provides a higher loss value for errors of higher magnitudes compared to the absolute loss. This means that the square loss emphasizes more on outliers, making absolute loss more robust if there exist outliers in the data the network will train on.

3.1.2.2 Optimizer

The choice of an optimization algorithm to update the weights through backpropagation affects the time it takes for the network to converge to the desired output of Equation 3.5. The quality of the model is in addition influenced by the optimization algorithm, such as its ability to generalize which is covered in subsubsection 3.1.2.5

and subsubsection 3.1.2.6. The optimization algorithm's general workflow is to minimize a loss function E accounts for the prediction $\hat{\boldsymbol{Y}}$ generated by a function $\boldsymbol{F}$, along with a set of points of coordinates $\boldsymbol{Y}$. Note that $\boldsymbol{F}$ represents the whole network with all its parameters, i.e the weights and thresholds while $\hat{\boldsymbol{Y}}$ represent the output of the network, i.e Equation 3.4. This means in order to minimize E, changes for the parameters of $\boldsymbol{F}$ have to be made. This is where the gradient comes in, which determines the change of the loss function at a certain point in all directions. Then depending on the optimization algorithm chooses a path that leads to a decrease in E that corresponds to certain weight adjustments. The magnitude of the direction taken is determined through a parameter called learning rate, denoted $\eta > 0$. This process is done iteratively until $E = 0$ or when a satisfactory output of $\boldsymbol{F}$ is given.

The optimization algorithm that makes the foundation of backpropagation is called gradient descent. Gradient descent is an optimization algorithm that always chooses the path of the gradient that is the steepest, meaning the path that decreases the value of E the most. Equation 3.10 expresses gradient descent applied for updating all the learnable parameters K in the network simultaneously by using all training samples, resulting in having trained the network for one epoch.

$$\boldsymbol{w} = \boldsymbol{w} - \eta \nabla \boldsymbol{E}, \quad \nabla \boldsymbol{E} = \Big[\frac{\partial E(O_i^{(u)}, t_i^{(u)})}{\partial w_1}, ..., \frac{\partial E(O_i^{(u)}, t_i^{(u)})}{\partial w_K} \Big], \text{for all } i \text{ and } u \quad (3.10)$$

An important thing to note is the learning rate η, as it determines the magnitude of adjusting the weight. Finding the optimal learning rate can be challenging, and having too low or high can create complications. A high learning rate could fasten the time to reach the desired output but also risk missing it. A low learning rate suffers from longer training time, and could potentially get stuck at a local minimum. Figure 3.2 illustrates the trajectory of minimizing a bowl-shaped loss function using three different learning rates for gradient descent. One can observe in Figure 3.2 that for this type of loss function, using optimal and low learning rates converges to the minimum, with the difference being that the latter requires more iterations. The one with too high a learning rate misses the minimum and diverges from it.

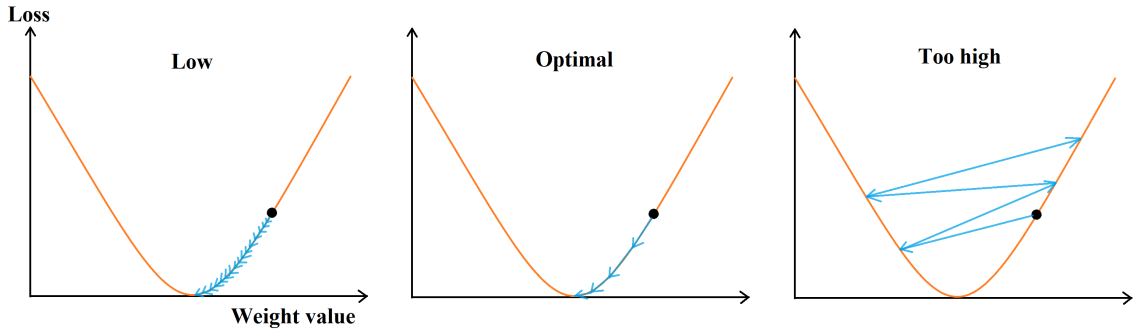


Figure 3.2: An illustration of three different values of learning rates for gradient descent, with the lowest on the left and ascending to the right. The blue arrows represent each iteration of Equation 3.10, with a lower learning rate resulting in a shorter arrow. For a low learning rate, the number of iterations will increase which results in a longer time to reach convergence compared to the optimal one. The plot on the far right has a learning rate that is too high which causes it to miss the point of convergence and results in divergent behavior. The optimal learning rate's value is high enough such that the time to converge is shorter than the lower learning rate, and does not diverge from the point of convergence.

One can note that as long as the learning rate is not larger than the optimal value, it will eventually converge to the minimum regardless of the starting position. However, this is only true when the loss function has no local minimum. If it does, then too low of a learning rate could result in the optimizer getting stuck at a local minimum depending on the starting point. Getting stuck in a local optimum due to a low learning rate is illustrated in Figure 3.3.

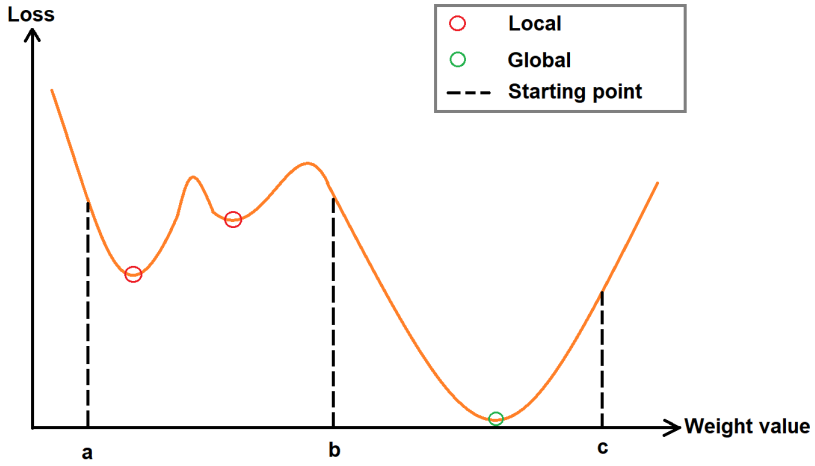


Figure 3.3: Illustration of a loss function as a function of the weight value. The image consists of two local minima and a global one, meant to show the vulnerability of choosing a value for the learning rate η with respect to the starting point. If the starting point is at the weight value a and η is small, the optimizer may get stuck at the local minima closest to the starting point. At points b and c however, it will converge to the global minima. This means that with a low learning rate coupled with the potential of receiving an unfavorable starting point when initializing the weights, the network's learning potential cannot be fully realized.

There exist various optimization algorithms for backpropagation, all of them based on gradient descent with slight modifications to combat the issue of converging at a local minimum. Stochastic gradient descent (SGD) is one of the most commonly used optimization algorithms due to it being able to help escape local minima and in some cases reduce the computation time [26]. SGD calculates the gradient for a single random sample of the training data to update all the weights, contrary to gradient descent which uses all training data. The weight update using SGD is expressed in Algorithm 1. One epoch here corresponds to when all the u training

Algorithm 1 Weight update using stochastic gradient descent

for $k = 1, 2, ..., u$ **do**
  $\boldsymbol{w} = \boldsymbol{w} - \eta \nabla \boldsymbol{E}_k$
end for

samples have been used to update the weights. SGD effectively approximates the true gradient of $\boldsymbol{E}$ which results in a stochastic trajectory in the weight space, making it less prone to get stuck in a local minima. However, it may also result in too much noise due to it using one sample of the training data to approximate $\boldsymbol{E}$. A way to alleviate this is to use a mini-batch-sized approach of SGD, which essentially means randomly choosing training samples of size $b < u$. This results in a trajectory that is not as stochastic as when using one sample, and still has the potential to escape a local minimum.

3.1.2.3 Deep learning

A question that may arise is why would one want to introduce more than one hidden layer. This is because a feed-forward network with multiple hidden layers can be trained to recognize and classify images more reliably. The area which considers networks with many hidden layers is called deep learning. Deep learning is a technique that is classified as a machine learning technique which is essentially an artificial neural network that utilizes more than two hidden layers. The idea behind this is to enable a network to learn more features from the data. For example, when distinguishing cats and dogs visually, humans can most of the time tell the difference based on their appearance which is composed of their features, such as the shape of their face, tail, paws, etc. Each hidden layer of the network may learn each of the features which the cats and dogs possess such that it can eventually distinguish between them. There are multiple deep learning architectures that were built to solve different tasks. Typically for vision tasks, one would most likely employ a convolutional neural network (subsection 3.1.3), while for natural language processing, it would instead be a transformer.

In order for the gradient to propagate through a general feed-forward neural network, i.e if it has multiple hidden layers, the chain rule has to evaluate for the loss function with respect to the weights as seen in Equation 3.11

$$\frac{\partial \boldsymbol{E}}{\partial W_{ij}^{(L)}}. \tag{3.11}$$

$L = L-1, L-2, ..., 1$ is the index for the hidden layer, note that its values are descending. L is in descending order because backpropagation propagates from the output neuron in the opposite direction of the network, therefore its name. Equation 3.11 needs to be expanded for weights that have a lower L index in order to propagate the gradients further back, as they are dependent on the weights with a higher L index.

3.1.2.4 Vanishing and exploding gradient

The application of the chain rule results in an expression that contains several multiplication factors that can give rise to two consequences that greatly affect the network's capability to learn, these two are exploding gradient and vanishing gradient. If the gradient is smaller than 1 for each layer, it will decrease exponentially for every layer it is propagated back to the initial layers, giving rise to the vanishing gradient problem. The exploding gradient problem occurs when the gradients of each layer are greater than 1, resulting in exponential growth of the gradient for each propagation. The reason why the vanishing gradient occurs is due to the values of the derivative of the activation function $g(b)$ being in the range $[0, 1)$. In contrast, the exploding gradient occurs when the initial loss becomes large such that a substantial change needs to be made for the weights, which is due to the values the weights were initialized with.

To combat vanishing and exploding gradients, there are a couple of methods that can be employed. Against vanishing gradient, a solution could be to employ a different activation function that doesn't cause small derivatives, such as ReLU and leaky ReLU. For the exploding gradient, the initialization of weight values could be small and with a low standard deviation. There exist certain weight initialization methods depending on the task which the network is used for, such as Uniform, Xavier, Kaiming, Normal, and One's initialization. An overall solution against both issues could be to reduce the number of layers in a network to avoid a potentially exponential increase or decrease in the gradients.

3.1.2.5 Overfitting

Another issue that may be encountered when adjusting the weights of the network is the phenomenon called overfitting. When using supervised learning to train a machine learning model, the goal is to ensure that the model is able to produce an output that satisfies the specified target. The way the model can tell how close it is to the target is through the loss function, which it attempts to minimize. While the model can become very good at producing results that satisfy the data it was trained on, it may not perform well on "unseen data". Unseen data is referred to data that is of similar content as the data the model was trained on, but its samples are unique. This means that the model is unable to generalize, giving rise to the overfitting phenomena. An illustration of this can be seen in Figure 3.4. Overfitting is more prevalent for artificial neural network models with more neurons due to them having more capacity to learn more specific features, resulting in them easier becoming fixated on the data. There are generally two solutions that are applied

to reduce the potential of overfitting in artificial neural networks, these are cross-validation and regularization schemes.

3.1.2.6 Cross-validation

Cross-validation is a method where one splits the data which a model uses to learn into two parts: training and validation. The objective is to use the training data solely for the model to learn from while the validation is for the user to check whether the model could be able to handle unseen data, but is not taken into account when adjusting the weights. An illustration of cross-validation can be observed in Figure 3.5. The idea of cross-validation is to use the validation part to give an indication of when the model may require more or less training time.

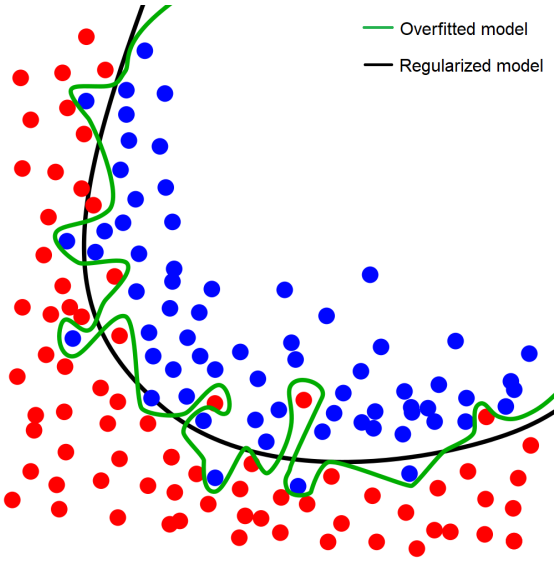


Figure 3.4: An illustration of a concept within machine learning known as overfitting [27]. In this image, there exist two types of dots in the training data, red and blue that are plotted in feature space. There are two types of curves, green and black, representing two models trained on the training data with an attempt to separate the blue and red dots. The model which generated the green curve perfectly separates the blue and red dots while the black does not. However, the trajectory of the green curve was formed to fit perfectly for this particular sample of the data. There may exist outliers that the model with the green curve has picked up. If a similar type of data were to be presented, the results may differ. The model represented by the black in contrast has a trajectory that is not too fixated on the training data when separating the red and blue dots, which can serve as a better model when applied to unseen data.

3.1.2.7 Regularization

The other solution that was mentioned in regard to overfitting was regularization schemes. There are multiple regularization schemes that have proven to combat

overfitting, especially for deeper networks, these are weight decay, drop out, expansion of training set, and batch normalization.

Weight decay works by assigning a constraint when minimizing the loss function E during backpropagation. The idea is to balance between small values of E while maintaining small weight values that help against exploding gradient. This is done by reducing the weights by a factor $0 < \varepsilon < 1$ during the model's training for each time backpropagation is performed or in regular intervals.

Drop out is a regularization scheme where a random sample of fraction q of the hidden neurons is ignored during training. For each backpropagation that is performed, the random samples' weights and thresholds are set to 0 while the remaining are updated and then restored to their state before being nullified. Once the training has been completed, all hidden neurons are activated but with their output multiplied with a factor $1-q$ in order to ensure that the local fields b on average are independent of q. The effect of this is that the values of the hidden neurons that are set to 0 during the backpropagation are temporarily lost, meaning their contribution of them when calculating the gradients is neglected. This prevents certain weights to specialize in specific features, i.e. reducing the potential for overfitting. However, this results in a longer training time as not all weights are considered for each backpropagation pass.

The expansion of the training set can help a model to become better at generalizing unseen data as more examples are provided. One could expand the training set by providing more data and also augment the data to produce more samples. The way in which one can augment data that would result in a more accurate model differs depending on the input one feeds into the network. As an example for image inputs, changing the color, adding noise, scaling, rotating, cropping, etc.

The last regularization scheme mentioned is batch normalization, which can help with reducing the training time for deep networks with backpropagation [28]. For each mini-batch, it performs re-centering and re-scaling of the input data and continues to do so for each hidden layer as the data propagates forward. The normalization of the data helps combat vanishing gradients as the values of the local fields and weights won't become larger than 1. The reason why batch normalization helps decrease the training time for a neural network is still under discussion. The notion that the author of the method had was that it reduces possible internal covariate shifts, which they defined as: "the change in the distribution of network activations due to the change in network parameters during training". However, some scholars argue that batch normalization does not reduce the internal covariate shift, but that it instead smooths the trajectory to reach convergence for the most optimal model with respect to the loss function E [29]. Regardless of the theories surrounding the reason why batch normalization accelerates the training, it has been proven empirically.

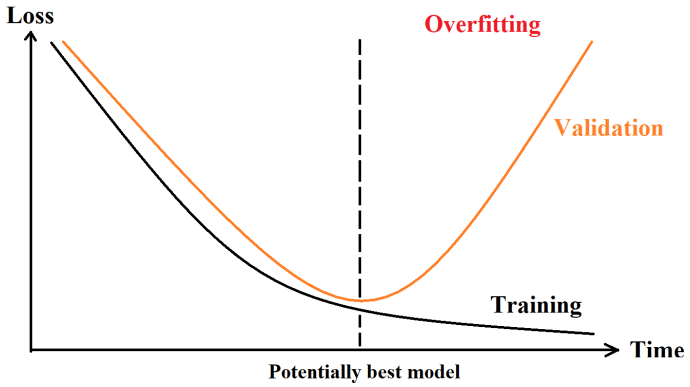


Figure 3.5: In this figure the concept of cross-validation is showcased through the loss as a function of time for the two graphs denoted as training (black) and validation (orange). Both graphs represent the same model at each time step, but with the black one showing the loss for the training data which it repeatedly learns from, and the orange one showing the loss when feeding the set of unseen data. As can be observed in this case, the training loss and validation loss initially decrease until a certain point (marked with a dashed line) when the model starts to become fixated on its training data such that the validation loss instead increases, entering the overfitting region. The usage of cross-validation can give an indication of when a model may or may not require more training.

3.1.3 Convolutional neural networks

Convolutional neural networks (CNNs) utilize convolutions to perform forward propagation, as opposed to feed-forward neural networks. Convolutions consist of a kernel, which can be viewed as a filter that is moved across the image. The kernel is applied to all image patches while being moved around with a certain stride (the number of "steps" between image patches). The padding parameter of convolutional layers allows the image to be extended by pixels (usually zeros) to extract edge information. In a convolutional layer, multiple kernels are used to extract different attributes. Figure 3.6 illustrates a kernel applied to an input image.

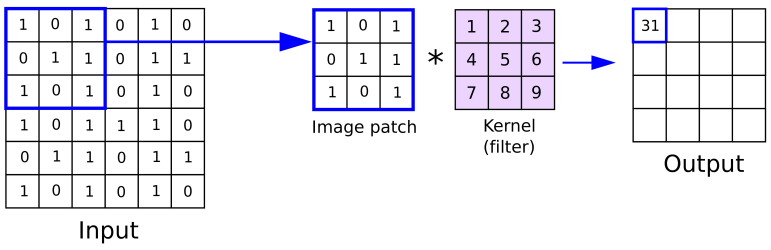


Figure 3.6: Illustration of how a kernel/filter is applied to an image patch of the input [30].

In addition to convolutional layers, max pool layers are another significant layer in CNNs. Max pool layers reduce the input to a lower dimension. The kernels of a max pool layer select the maximum value in each image patch, resulting in an

approximation of the image.

The convolutional neural network essentially extracts features from its input with it a feature map whose spatial resolution is smaller than the input. However, this can result in sparse feature extraction, meaning the feature map cannot represent the input's features well. A possible solution to this is atrous convolution, which changes the way the kernel weights are applied for the input [31]. The area in which the kernel is applied on the input is called a receptive field, which for standard convolution is the size of the kernel. Atrous convolution on the other hand has the capability to change the size of the receptive field through a parameter denoted as r, where a larger r results in the expansion of the receptive field. Specifically, r is an integer that adds space between each kernel weight in the horizontal and vertical direction. An illustration of this can be observed in Figure 3.7. The introduction

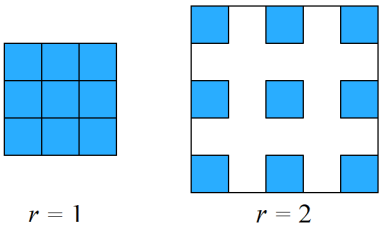


Figure 3.7: Illustration of a 3x3 kernel application when using atrous convolution for two different spacing rates r, with the blue boxes representing the kernel weights. The receptive field is the same as the standard convolution when having $r = 1$, i.e. the spacing between each weight is 1. An increase of r leads to a larger receptive field despite having the same kernel size, as seen for $r = 2$ where the receptive field is 5x5.

of r allows control of the output size of the convolutional layer without having to change the kernel size, effectively reducing the number of parameters or the amount of computation. In short, atrous convolution can be useful for extracting contextual information.

3.1.4 Transformer

Transformer is a deep learning model that was developed for learning contextual information, such that it can for example translate a sentence between two languages with correct grammar [32]. Its application today can be seen in areas such as natural language processing and computer vision, with ChatGPT being arguably the most known product using the transformer model. What separates transformer from other deep learning architectures is the mechanism known as attention, specifically scaled dot-product attention. Its formal definition can be observed in Equation 3.12

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V, \tag{3.12}$$

where d_k is the key dimension, Q, K and V are matrices containing queries, keys and values respectively. The term QK^T calculates all the attention weights, with the division of $\sqrt{d_k}$ to prevent vanishing gradient, followed by the softmax to normalize the weights, and lastly multiply with V to obtain the new embedding. The attention weights can be trained during backpropagation such that it allows the network to emphasize features of higher importance for certain tasks, signified by a higher weight value.

3.1.5 Depth completion

Depth completion is a technique that aims to provide an approximation of the three-dimensional appearance and shape of objects in a scene. The problem formulation of depth completion can be described as follows: find $\hat{f}$ that approximates f, where $f(I, D_{\text{sparse}}) = D_{\text{dense}}$ with I and D_{sparse} being an image and a sparse depth image respectively. A sparse depth image is the result of projecting LIDAR point clouds to the image plane. The mathematical formulation of the depth completion problem can be observed in Equation 3.13.

$$min||\hat{f}(I, D_{\text{sparse}}) - f(I, D_{\text{sparse}})||_F^2 = 0, \tag{3.13}$$

where $I \in \mathbb{R}^{W \text{ x } H}$, $D_{\text{sparse}} \in \mathbb{R}^{W \text{ x } H}$ and D_{dense} having the same size as I and D_{sparse}. Equation 3.13 is written in a general form that takes into account the different categories of depth completion. Depending on the category of the chosen depth completion method, the equation can be independent of either I or D_{sparse}. Depth completion methods can be divided into two categories: guided and non-guided [33].

Guided depth completion is a category for which the methods belonging to use color images as input to obtain a dense depth image. These methods utilize various approaches ranging from classical image processing algorithms such as median filters to the most recent ones using deep learning. The current state-of-the-art models in this category operate with deep learning models, which have proven to perform better compared to non-guided depth completion models as can be observed in the Kitti benchmark website [34].

Non-guided depth completion in contrast to the guided one uses a sparse depth image instead of a colored image, meaning they are only dependent on the LIDAR data. However, the sparse depth image may contain missing values (points) for the majority of all pixels, hence its name. To tackle this issue, classical techniques and deep learning models have been employed to interpolate the sparse depth image.

3.2 Evaluation metrics

As this thesis deals with classification and regression tasks, the ideal evaluation metrics would be those that regard such tasks. The classification metrics are those that deal with four quantities: true positive, true negative, false positive, and false negative. These quantities stem from the combination between the predicted class and

the ground truth class. An illustration of the combination can be seen in Figure 3.8. The four quantities enable the use of multiple classification scores where each report how well a model performs in various aspects. It would therefore be of interest to use numerous classification metrics to better grasp a model's performance. Such metrics would be recall, accuracy, precision, and F1 score, with each score defined in Equation 3.14, Equation 3.15, Equation 3.16, and Equation 3.17 respectively.

The accuracy score considers all the predicted samples that match the ground truth. The recall is a measure for the model based on its ability to find all positive examples. The precision score on the other hand measures the model based on its ability not to predict a sample as positive when the ground truth is negative. The F1 score is the harmonic mean between recall and precision, meaning it weighs the recall and precision scores evenly. All the mentioned classification scores have a value that resides in the range $[0, 1]$, with 1 being the best possible value for the model.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{TN}} \tag{3.14}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.15}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.16}$$

$$\text{F1} = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.17}$$

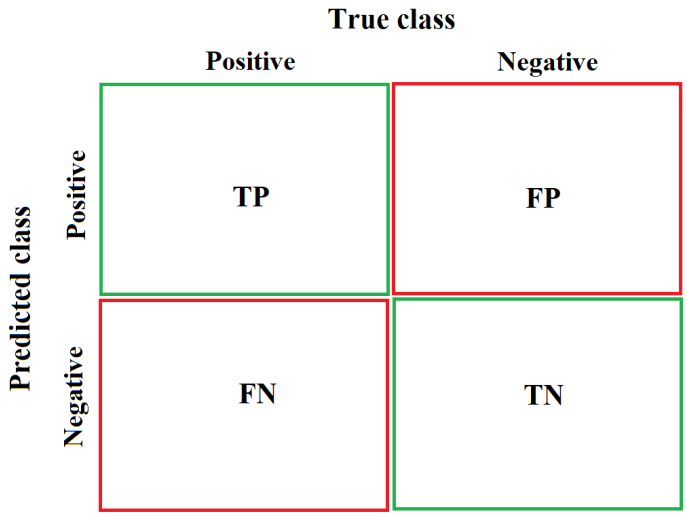


Figure 3.8: An illustration of the combination between the predicted class and ground truth (true) class. TP, FP, FN, and TN stand for true positive, false positive, false negative, and true negative respectively.

In the case of the evaluation metrics regarding regression, one of the most common methods is the root mean square error (RMSE) [35]. The mean squared error (MSE) is calculated by squaring the errors between each prediction and the corresponding ground truth. Afterward, each squared error is summed into one total value and

then divided by the number of squared error samples. The RMSE is calculated by taking the root of the MSE, resulting in the output being of the same unit as the original data. The value can be interpreted as the average distance between the predicted values and the ground truth. The definition of RMSE can be observed in Equation 3.18

$$\text{RMSE} = \sqrt{\text{MSE}}, \quad \text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i' - y_i)^2. \tag{3.18}$$

Another noteworthy evaluation metric for regression is the mean absolute error (MAE). As the name suggests, it takes the mean of the sum of all the absolute errors. RMSE and MAE are similar in the aspect of calculating the variation in the errors. If the individual errors vary greatly, then RMSE will be larger than MAE. If the individual errors however have similar magnitude, then RMSE will be equal to MAE. The expression of MAE can be observed in Equation 3.19

$$\text{MAE} = \frac{1}{N}\sum_{i}^{N}|y_i' - y_i|. \tag{3.19}$$

3.3 SuperFusion framework

As mentioned in chapter 2, SuperFusion is a model that executes sensor fusion between LIDAR & camera at three levels, these are: Data, feature, and BEV. This section will explain the intricate details for each level of fusion the SuperFusion model performs in chronological order. Before delving into the details, the reader is suggested to refer to Figure 3.9 when attempting to envision the whole process. Note also that letters written in bold texts are referred to as tensors.

3.3.1 Data-level fusion

The first fusion is done at the data-level where a LIDAR point cloud $\mathbf{P}$ is projected onto the image plane. This creates a sparse depth image $\mathbf{D}_{\text{sparse}}$ with the dimensions $\mathbb{R}^{W \times H}$, where H and W correspond to height and width. The sparse depth image contains LIDAR points in various height and width combinations on the image plane with each point having a distance associated with them, the distance being from the LIDAR to a point. This sparse depth image is concatenated with the corresponding RGB image $\mathbf{I}$, i.e. the image frame in which the LIDAR shares the same field of view and roughly the same timestamp. The reason for mentioning the timestamp is that depending on what LIDAR and camera one chooses to use, they may have a different frequency that affects the frame which the LIDAR and camera capture as the vehicle is moving. The concatenation between the two images will result in an RGB image containing LIDAR points and their depth information. We denote this resulting image as $\mathbf{I}_{conc}$ with the dimensions $\mathbb{R}^{W \times H \times C_S}$, where $C_S = 4$.

The last step in the data-level fusion involves interpolation through a process called depth completion. As mentioned in subsection 3.1.5, the output of depth completion is called a dense depth image $\mathbf{D}_{\text{dense}}$, an estimation of the three-dimensional

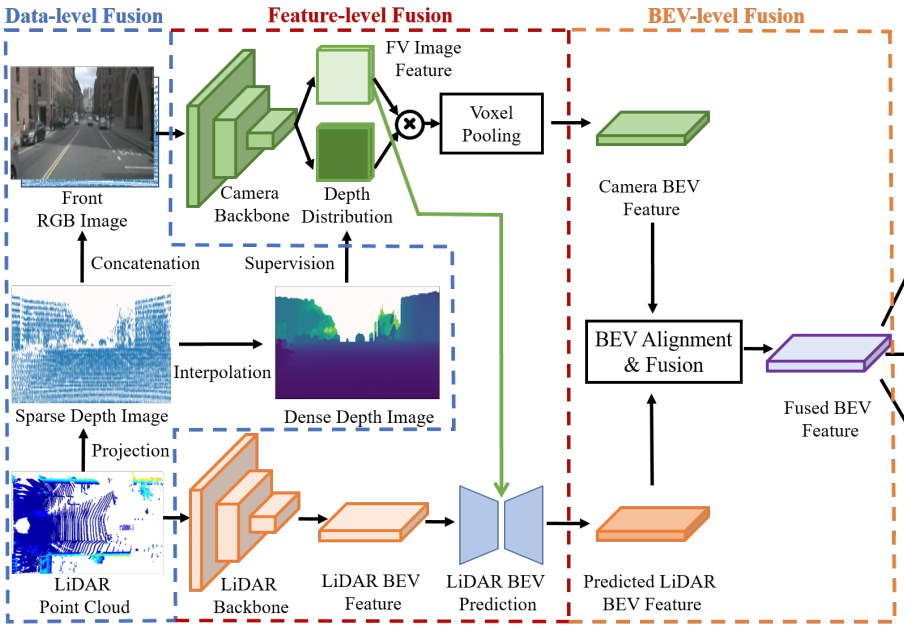


Figure 3.9: An overview of the body of the SuperFusion framework [14]. The input to the framework is a LIDAR point cloud and a front RGB image, the output is the fused BEV feature.

appearance and shape of objects in a scene. The depth completion method which the authors of SuperFusion utilized is called IP-basic and is classified as a non-guided depth completion method [36]. IP-basic uses classical image processing techniques which obtain performance close to other methods of the same category with a speed that rivals among the fastest depth completion methods. While IP-basic is not on par with methods of the opposite category, it serves as a stepping stone for the next fusion level, which is covered in the next subsection. Note that since IP-basic is a non-guided method, it needs only $\mathbf{D}_{\text{sparse}}$ as the input.

3.3.2 Feature-level fusion

In the second fusion level, feature-level fusion, we obtain three inputs from the data-level fusion's output: Firstly the raw LIDAR points $\mathbf{P}$, secondly, the sparse depth images along with their corresponding RGB images concatenated $\mathbf{I}_{conc}$, and lastly the dense depth image produced from the non-guided depth completion method IP-basic $\mathbf{D}_{\text{dense}}$. The feature-level fusion layout consists of two pipelines: the LIDAR which utilizes the LIDAR points $\mathbf{P}$, and the camera that uses the remaining inputs. Each of the pipeline's purpose is to extract their respective input's features to form BEV features for the LIDAR and camera modalities. In addition to the two pipelines, cross-attention and pillar pooling will be performed which will be explained further in the subsection.

The camera pipeline is made up of the backbone which has two branches: feature extractor and depth distribution. The choice of camera backbone for the SuperFusion model was ResNet-101, consisting of four convolution blocks with a total of

101 layers [37]. Initially the backbone takes as input $\mathbf{I}_{conc}$ and is processed through the first convolutional block resulting in a feature map denoted as $\tilde{\mathbf{F}}$. The feature map is then inserted into the two branches with the goal to create frustum features that enable 3D detection, a process similar to categorical depth distribution network (CaDDN) [38].

The depth distribution branch works by inputting a feature map $\mathbf{F}$ where all its features are used to estimate the depth distribution $\mathbf{D} \in \mathbb{R}^{W_F \times H_F \times D}$, where D is the number of discretized depth bins. The depth discretization method of choice was linear-increasing discretization, as it could provide depth estimation in a balanced way compared to other methods [39]. To compute the categorical depth distribution the authors of SuperFusion follow CaDDN by modifying the semantic segmentation network DeepLabV3, enabling it to produce pixel-wise probability scores of belonging depth bins [40]. Specifically, an atrous spatial pyramid pooling technique is employed to capture multi-scale features, its output is then upsampled to the original input size using bilinear interpolation with the application of a softmax function to each pixel in order to normalize the D depth bins. To increase the accuracy of the depth distribution, the dense depth image $\mathbf{D}_{\text{dense}}$ is utilized by converting its depth bins for each pixel to a one-hot encoding vector to supervise the depth prediction network.

In the feature extractor branch the feature map $\tilde{\mathbf{F}}$ is processed through the remaining convolutional blocks of ResNet-101 with the resulting output dimensions $\hat{\mathbf{F}} \in \mathbb{R}^{W_{\hat{\mathbf{F}}} \times H_{\hat{\mathbf{F}}} \times C_I}$, where C_I is the number of channels specified by the last convolutional layer. Afterward, an image channel reduction is performed to reduce the memory load using 1x1 convolutional, followed by a BatchNorm and ReLU, reducing the number of channels from C_I to C_{final}. We denote the final output of the branch as $\mathbf{F}_{final}$, which will be used in the LIDAR pipeline to perform the feature level-fusion.

To complete the second last step for the camera pipeline, we denote (u, v, c) as the coordinates of image features $\mathbf{F}$, with the corresponding coordinates for categorical depth distribution $\mathbf{D}$ as (u, v, d_i), where d_i is the depth bin index. The frustum feature grid $\mathbf{M}$ can be obtained by taking the outer product between $\mathbf{D}(u, v)$ and $\mathbf{F}(u, v)$, as shown in Equation 3.20

$$\mathbf{M}(u, v) = \mathbf{D}(u, v) \otimes \mathbf{F}(u, v), \tag{3.20}$$

where $\mathbf{M}(u, v) \in \mathbb{R}^{W_F \times H_F \times D \times C_{final}}$. The frustum feature grid contains each pixel of the image features $\mathbf{F}(u, v)$ and its weighted depth bin probabilities $\mathbf{D}(u, v)$.

The last step of the camera pipeline is where the frustum feature grid $\mathbf{M}$ is converted into the camera BEV features $\mathbf{C}$. This was done by assigning each voxel of the frustum feature grid $\mathbf{M}$ to its nearest pillar to then apply a sum pooling using the camera's intrinsics and extrinsic parameters, similar to the step called "Pillar pooling" in LSS [41]. The resulting dimensions of the camera BEV features $\mathbf{C}$ is $\mathbb{R}^{W \times H \times C_{final}}$.

The LIDAR pipeline consists of two parts: the backbone and the BEV prediction module. The LIDAR backbone's function is to generate BEV features from the raw point cloud $\mathbf{P}$, which then will serve as an input for the BEV prediction module. The backbone of choice was a combination of dynamic voxelization and PointPillars [42][43]. The initial process of the backbone involves dynamic voxelization, which converts the raw point cloud into stacked vertical columns (pillar) tensors. Note that PointPillars is able to perform the initial step, however, it uses hard voxelization which has been proven by the authors of dynamic voxelization to be less efficient. Once the initial process has been performed, the stacked pillars will serve as a learning objective for the feature encoder which then creates a 2D pseudo image to be processed by a 2D convolutional neural network. The 2D convolutional network processes the pseudo image and produces LIDAR BEV features $\mathbf{L} \in \mathbb{R}^{W \times H \times C_L}$.

The last step of the feature level fusion in the LIDAR pipeline is the LIDAR BEV prediction, with the architecture presented in Figure 3.10. The reason for including this module is to increase the range which the LIDAR can perceive by taking advantage of the camera which has a longer range, specifically its image features. Initially, the LIDAR BEV features $\mathbf{L}$ are sent into the BEV encoder, resulting in the bottleneck features $\mathbf{B} \in \mathbb{R}^{W/8 \times H/8 \times C_B}$, where C_B is the channel number specified by the last convolutional layer. The compressed LIDAR BEV features $\mathbf{B}$ is then applied to three fully-connected convolutional layers to query Q with the corresponding image features $\mathbf{F}$ to key K and value V, producing the aggregated feature $\mathbf{A}$ using the cross attention formula described in subsection 3.1.4. The procedure enables us to capture the relationship between bottleneck features $\mathbf{B}$ and the image features $\mathbf{F}$. Once $\mathbf{A}$ has been obtained, it proceeds through a convolutional layer to reduce the channel number in order to concatenate with the bottleneck features $\mathbf{B}$, further reducing its channel number by applying an additional convolutional layer where we end up with the final bottleneck features $\mathbf{B}'$. The final step in the LIDAR BEV prediction is where $\mathbf{B}'$ is sent through the decoder resulting in the predicted LIDAR BEV features $\mathbf{L}' \in \mathbb{R}^{W \times H \times C_D}$, where C_D is the channel specified by the last convolutional layer.

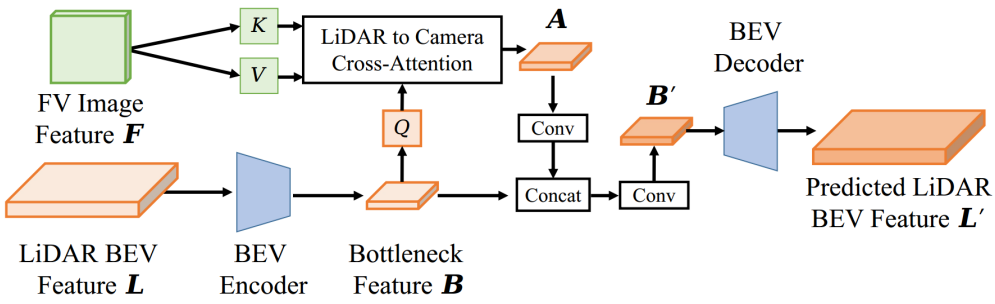


Figure 3.10: An illustration of the LIDAR BEV prediction module [14].

3.3.3 BEV-level fusion

In the third and last fusion level is where the camera BEV features $\mathbf{C}$ and the predicted LIDAR BEV features $\mathbf{L}'$ are, with its process illustrated in Figure 3.11. As can be observed in the figure, the BEV-level fusion does not simply concatenate the BEV representations of the LIDAR and camera branches. There are two reasons: The first is the possibility of depth estimation errors and inaccurate extrinsic parameters of the camera branch, and the second is the risk of losing boundary information due to the downsampling operations in convolutional neural networks. To mitigate these errors, a BEV alignment module was introduced by utilizing the flow field and warp/alignment function, following the works of AlignSeg and semantic flow [44][45].

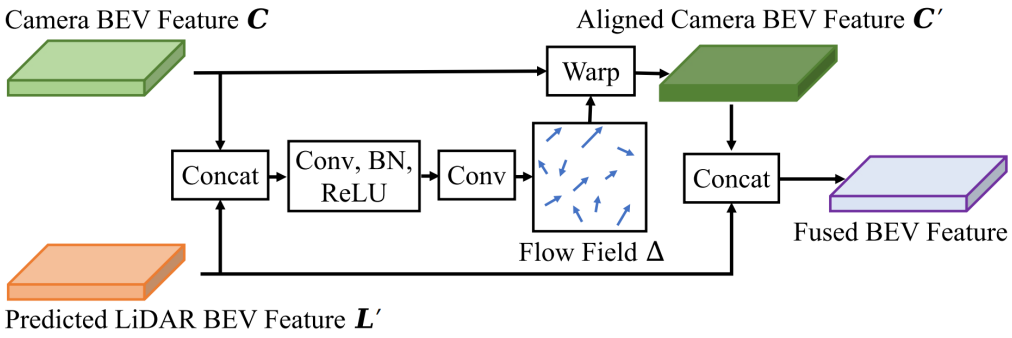


Figure 3.11: An illustration of SuperFusion's BEV-level fusion [14].

The flow field is a collection of offsets which are the discrepancy needed to guide the alignment of feature maps. The flow field is obtained through prediction, using the BEV alignment module. The process BEV of the alignment module starts with the concatenation between $\mathbf{C}$ and $\mathbf{L}'$, which later is processed through four layers in the following order: Convolutional, Batch normalization, ReLU, and lastly a convolutional. This results in a flow field $\mathbf{\Delta} \in \mathbb{R}^{W \times H \times 2}$ that will be used to warp the camera BEV features to align with the predicted LIDAR BEV features. The last step needed to correct the possible errors in $\mathbf{C}$ is to apply the warp function, which is defined as:

$$\mathbf{C}'_{hw} = \sum_{w'=1}^{H} \sum_{h'=1}^{W} \mathbf{C}_{w'h'} \cdot \max(0, 1 - |h + \Delta_{1wh} - h') \cdot \max(0, 1 - |w + \Delta_{2wh} - w'), \quad (3.21)$$

where $(w + \Delta_{1+wh}, h + \Delta_{2wh})$ are positions for which a bilinear interpolation kernel takes place when sampling features of $\mathbf{C}$. $\Delta_{1wh}, \Delta_{2wh}$ indicate the learned 2D transformation offsets (flow field) for the position (w, h). Note that if $\Delta_{1wh}, \Delta_{2wh}$ are both 0, then $\mathbf{C}'$ would be identical to $\mathbf{C}$. Once $\mathbf{C}'$ has been obtained, it will be concatenated with the predicted LIDAR BEV features $\mathbf{L}'$ that will form the fused BEV features $\mathbf{Z} \in \mathbb{R}^{W \times H \times C_Z}$, where C_Z is the sum of C_{final} and C_D.

3.4 M^2-3DLaneNet

The M^2-3DLaneNet model uses all the fusion strategies as SuperFusion, with slight differences in the way they fuse. An illustration of the model can be observed in Figure 3.12, where the inputs consist of raw point cloud data and images with the final output being 3D lane detection. The content of the M^2-3DLaneNet model is presented in three subsections: Data-level fusion, feature-level fusion in BEV lastly 3D lane prediction.

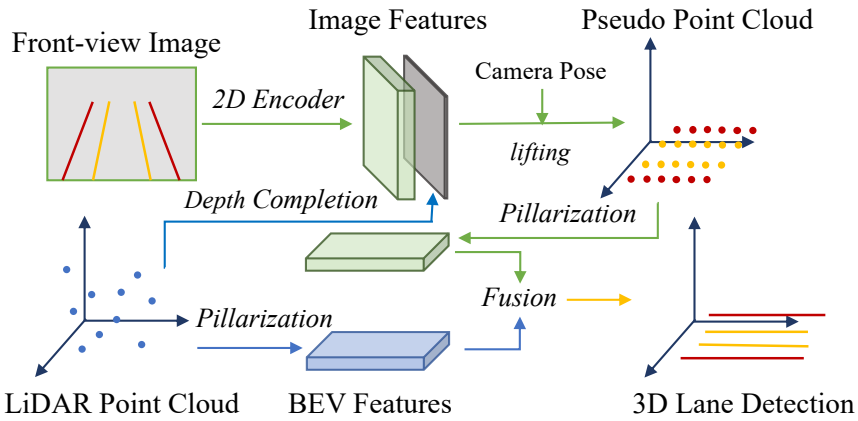


Figure 3.12: An overview of the M^2-3DLaneNet framework [18]. The input to the framework consists of a front RGB image coupled with a LIDAR point cloud, with the output of the model being a 3D lane detection.

3.4.1 Data-level fusion

Similar to SuperFusion's data-level fusion, the raw LIDAR point cloud $\mathbf{P}$ is projected to the image plane to create a sparse depth image $\mathbf{D_{sparse}} \in \mathbb{R}^{W \times H}$. Depth completion is then performed using $\mathbf{D}_{\text{sparse}}$ using a method called SFD to produce $\mathbf{D}_{\text{dense}}$ [46]. In contrast to SuperFusion, M^2-3DLaneNet does not concatenate $\mathbf{D}_{\text{sparse}}$ with the corresponding RGB image $\mathbf{I}$.

3.4.2 Feature-level fusion in BEV

The feature-level fusion is divided into two streams: camera and LIDAR which later are fused in BEV space.

Starting off with the camera stream, the input is an RGB image that is encoded into multiple feature maps, each of different resolutions. The encoding was done by using EfficientNet-B5 as the backbone, resulting in four feature maps of sizes: (W x H), ($W/2$ x $H/2$), ($W/4$ x $H/4$) and ($W/8$ x $H/8$) [47]. The four feature maps are then applied with a variant of the attention mechanism used in transformers, which the author dubbed as line-restricted deform-attention (LRDA). The idea of LRDA stemmed from deformable attention (DA), proposed in Deformable-DETR enabled

the capture of objects on different scales in any positions [48]. LRDA was proposed to solely focus on capturing lanes on different scales by limiting DA to the trajectory that a lane can have. LRDA, similar to DA, was used to increase the performance of object detection of smaller objects.

Once LRDA has been performed, the next step of the camera stream is to lift the image features to create a pseudo point cloud. Given the input image with a size (W, H), utilize $\mathbf{D}_{\text{dense}}$ to generate image coordinates (u, v, d) with $u \in [1, W]$, $v \in [1, H]$ and $d \in [1, D]$. Afterwards, by using the camera intrinsic and extrinsic matrices $K \in \mathbb{R}^{4\times4}$ and $T \in \mathbb{R}^{4\times4}$, the i:th image coordinate (u_i, v_i, d_i) can be transformed to 3D coordinates following Equation 3.22, yielding image features in 3D coordinates.

$$[x_i, y_i, z_i, 1]^T = T^{-1} \cdot K^{-1} \cdot [u_i \times d_i \times, v_i \times d_i, d_i, 1]^T \tag{3.22}$$

Once a pseudo point cloud has been created from the image features, pillarization is performed on it to obtain BEV features, which follows the steps of PointPillars's architecture. The final step of the camera stream involves the use of nearest BEV completion, where it attempts to tackle the issue of occlusion and a limited field of view which results in blank areas in BEV space. The procedure is done by generating an occupancy map that records whether a grid is occupied, which then interpolates the grids that are empty using their nearest neighbor.

The LIDAR stream is very similar to SuperFusion's LIDAR pipeline. They use the same backbone (PointPillar), with the exception that M^2-3DLaneNet does not perform dynamic voxelization. Once the LIDAR point cloud has been processed by the backbone it results in LIDAR BEV features of four resolutions, with the sizes being the same as the ones produced from the camera stream.

Once the camera and LIDAR stream have each produced four feature maps of different resolutions, they are concatenated. Then channel-wise attention is applied to focus on channels that are of more importance, increasing the representational power to improve classification-related tasks, similar to Squeeze-and-Excitation block [49]. Lastly, the four fused BEV feature maps are fused into one single representation by utilizing the feature fusion strategy of PersFormer [50].

3.4.3 3D lane prediction

The 3D lane prediction of M^2-3DLaneNet used the same prediction head presented by the authors of PersFormer. The 3D detection is based on anchor design, where an image is represented by a grid in BEV perspective. The anchors are defined as equally spaced lines in the vertical direction in the x-position $\{X^i_{\text{bev}}\}^N_{i=1}$ with an incline angle φ, along with a predefined y-position $\{y_k\}^K_{k=1}$ in the horizontal direction. Each anchor X^i_{bev} has 7 available values of the incline angle $\varphi \in \{\pi/2, \arctan(\pm0.5), \arctan(\pm1), \arctan(\pm2)\}$, which was implemented to easier represent lanes with large curvatures or perpendicular lanes. A 3D lane can be repre-

sented using an anchor $\{X^i_{\text{bev}}\}$, expressed in Equation 3.23

$$(\boldsymbol{x}^i, \boldsymbol{z}^i, \mathbf{vis}^i_{\text{bev}}) = \{(x^{(i)}_k, z^{(i)}_k, \text{vis}^{(i)}_{\text{bev}_k})\}^N_{k=1}, \tag{3.23}$$

where $\boldsymbol{x}^i$ consists of the horizontal offset to the closest vertical line X^i_{bev}. $\boldsymbol{z}^i$ contains the lane height in 3D, $\text{vis}^{(i)}_{\text{bev}_k}$ represents the visibility of each horizontal location k in anchor i, indicating the length of a lane with values of 0 and 1. Note that the y-position is not included, as all the attributes are represented for each fixed y-value. In addition to the 3D lane representation, 2D is included using a similar structure presented in Equation 3.23. The difference between the two representations is that the 2D case has the corresponding image in the image plane (u, v) with another incline angle θ, and with no height information. An illustration of the unification of the anchor design in 3D and 2D can be observed in Figure 3.13.

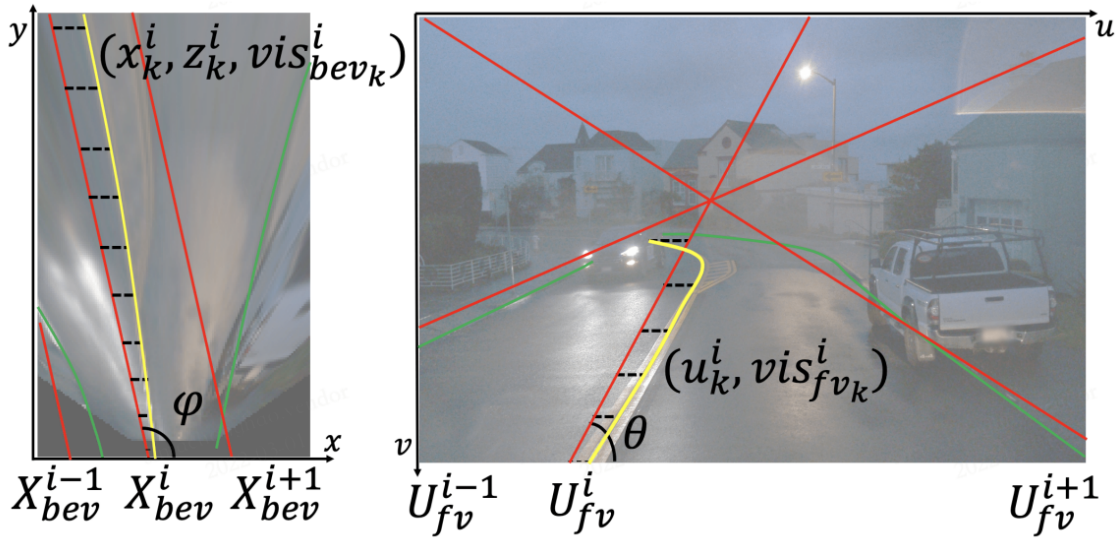


Figure 3.13: Reused figure from the authors of PersFormer where they illustrated their anchor design [50]. Both images in the figure represent the same scene with the left being in BEV and the right in the image plane. The dashed line in both images represents the offsets between the predicted (red) and ground truth (yellow), denoted as x^i_k and u^i_k respectively. Note that the green line also represents the ground truth lanes, but the offsets are not displayed.

The last component that the 3D lane prediction made use of was a semantic segmentation loss for the BEV feature produced at the end of the feature-level fusion in BEV (subsection 3.4.2). This was supervised by the projection of 3D annotations in order to support the network to learn the semantic information of the lanes. To summarize, the loss function which the network optimizes can be seen in Equation 3.24

$$E = \lambda_1 E_{\text{3d}}(\mathbf{Y}_{\text{3d}}, \mathbf{Y}^{\text{gt}}_{\text{3d}}) + \lambda_2 E_{\text{2d}}(\mathbf{Y}_{\text{2d}}, \mathbf{Y}^{\text{gt}}_{\text{2d}}) + \lambda_3 E_{\text{seg}}(\mathbf{Y}_{\text{bev}}, \mathbf{Y}^{\text{gt}}_{\text{bev}}), \tag{3.24}$$

where $\mathbf{Y}^{gt}_{(.)}$ denotes the ground truth, $\mathbf{Y}_{(.)}$ the prediction and $\lambda_{(.)}$ the weighting factor in case there is a need to emphasize more on a certain component.

4

Methods

4.1 Model architecture

In our work, we use both SuperFusion and M^2-3DLaneNet to construct our model. A variant of M^2-3DLaneNet's detection head and SuperFusion's body with three levels of fusion (data-, feature-, BEV-level) were used. The detection head of M^2-3DLaneNet's was split up into two segments, one for the regression and one for the classification of lanes. Figure 4.1 illustrates the regression head and Figure 4.2 illustrates the classification head, with both heads having the fused BEV features as input.

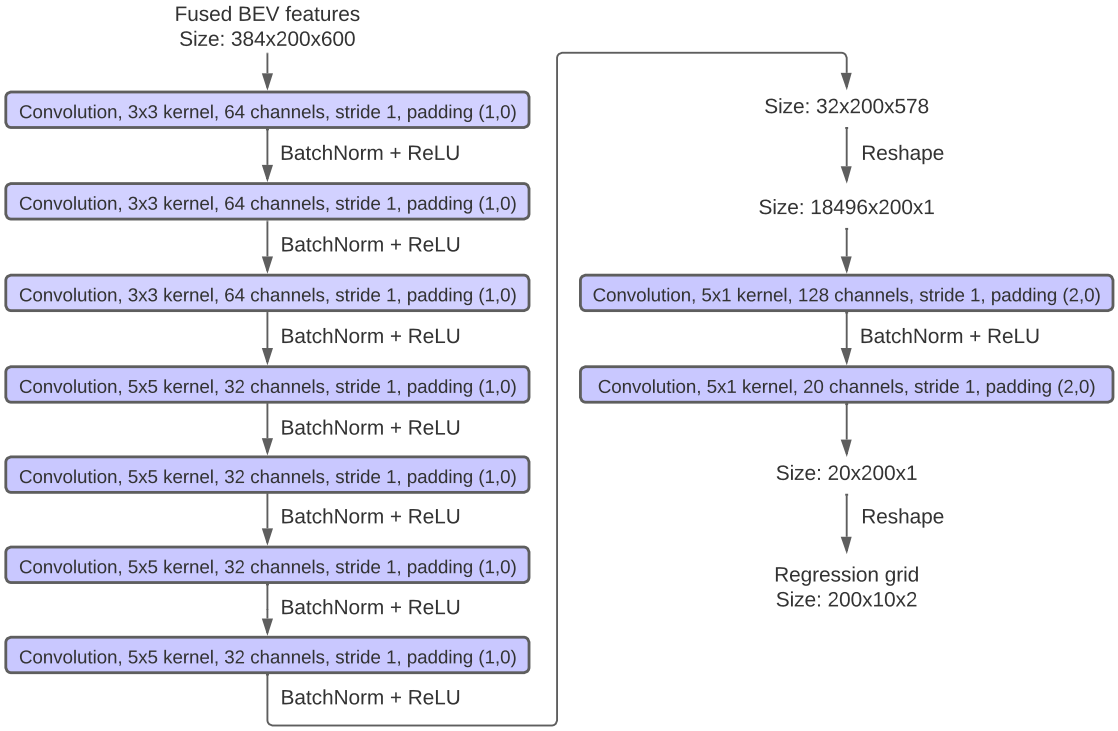


Figure 4.1: Regression head architecture. Taking the fused BEV features as input, the regression head outputs a grid with regression values (2 dimensions) for each grid point.

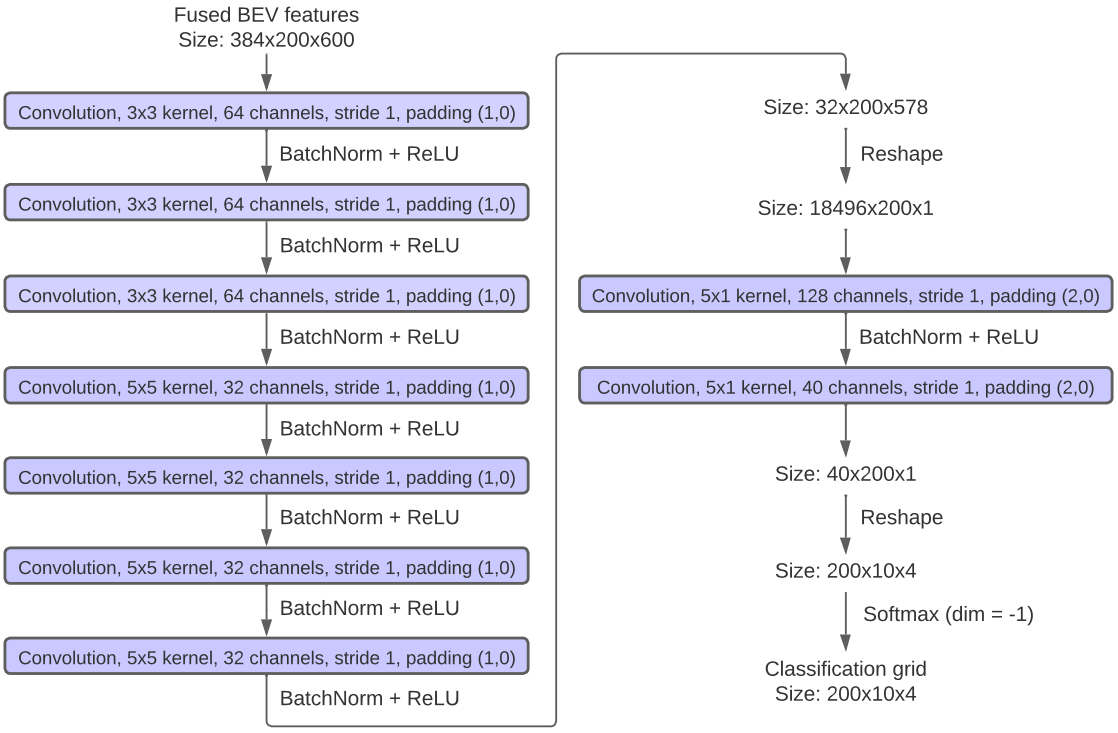


Figure 4.2: Classification head architecture. Taking the fused BEV features as input, the classification head outputs a grid with class probabilities for each grid point.

4.2 Dataset

The dataset used for this project was The Zenseact Open Dataset [51], consisting of 100k image frames with two seconds of sensor data (±1 second around the image frame). Images were taken with a 120° RGB camera with 3848x2168 resolution. The sensors consist of, among others, a LIDAR and high-precision GNSS/IMU data. The LIDAR is of type Velodyne VLS-128 and provides a point cloud with each point represented with a timestamp, 3D coordinates (x, y, z), and intensity at a speed of 9Hz. The GNSS/IMU data is logged at 100Hz with timestamp, latitude, longitude, altitude, heading, pitch, roll, velocities, accelerations, angular rates, and pose relative to the first pose for each frame. The lane annotations consist of lane markings and road paintings. Lane markings can be further divided into solid, dashed, botts dots (raised pavement markers) and shaded areas (areas where lane markings split or merge). The road paintings are divided into arrows, pictograms, text, traffic signs, crosswalks, markers, and others. An example of an image frame with its corresponding lane annotations can be seen in Figure 4.3.

4.3 Generating 3D annotations

The generation of 3D lane annotations can be divided into three steps: Projecting lidar point clouds to image timestamp, projecting lidar to image, and filtering lidar

Figure 4.3: An image frame with lane marking (blue), shaded area (green) and road paintings (red) annotations.

points to annotations. The 3D lane generation is created for multiple configurations.

4.3.1 Projecting lidar point clouds to image timestamp

By using the information from the GNSS/IMU sensors we transformed the LIDAR points cloud coordinate systems w.r.t. time. As we have the relative pose (position and orientation of the car) for each GNSS/IMU timestamp (at 200 Hz), all lidar points clouds (± 1 second) around the image frame were transformed to the image timestamp. This was done by first estimating (since the sensor readings are not fully synchronized) the poses for the car when the image was taken and when each lidar scan was done. The pose for a lidar/image scan (X) is estimated using the closest available pose before (P_{-1}) and after the scan (P_{+1}) with respective timestamps T_x (when the lidar/image scan was done), T_{x-1} and T_{x+1} (when poses (GNSS/IMU reading) were recorded). The pose estimation is shown in Equation 4.1 and Equation 4.2, where Δ_t is the fraction of time covered between the closest pose readings.

$$X \approx P_{-1} + \Delta_t(P_{+1} - P_{-1}) \tag{4.1}$$

$$\Delta_t = \frac{T_x - T_{x-1}}{T_{x+1} - T_{x-1}} \tag{4.2}$$

To further illustrate Equation 4.1 and Equation 4.2, Figure 4.4 is available. In Figure 4.4 we have the timestamps for poses (GNSS/IMU readings) denoted with X_1 to X_9, the timestamps for each lidar scan (denoted with L_{-3} to L_3) and the timestamp for the camera scan denoted with C. To get the pose for L_{-3}, we utilize the closest GNSS/IMU readings before and after the scan (X_1 and X_2 respectively). Following Figure 4.4, Δ_t approximates to ≈ 0.5 ($\mathrm{T}_{X_1} = 1$, $\mathrm{T}_{X_2} = 2$, $\mathrm{T}_{L_{-3}} \approx 1.5$). The pose for L_{-3} can therefore be estimated with $\approx X_1 + 0.5(X_2 - X_1)$.
Finally, when the pose was estimated for each lidar and image scan, all lidar scans were transformed to the coordinate system where the camera scan was done using

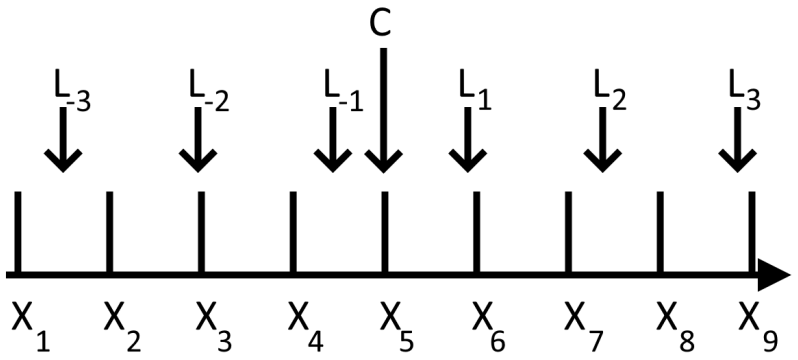


Figure 4.4: Time arrow with timestamps for pose (GNSS/IMU readings), lidar, and camera scans. The poses are denoted with X_i (where i increases with time), the camera scan is denoted with C and the lidar scans are denoted with L_j (where j describes the index relative to the camera timestamp).

the estimated pose with linear transformations. First, we compute the relative transformation for the car between the lidar scan to the camera scan, $\Delta_{pose} = {P_C}^{-1} P_L$ where P_L is the estimated pose for the car when the lidar scan was done and P_C is the estimated pose for the car when the camera scan was taken. The transformation process is as follows: Transform the lidar scan to the car frame using the extrinsic lidar calibration matrix, further transform the scan to the car frame for when the image was taken using Δ_{pose} and then transform the lidar scan back to the lidar frame using the inverse of the extrinsic lidar calibration matrix. The transformation results in (denoting the lidar extrinsic matrix with E) $E^{-1}\Delta_{pose}E = E^{-1}{P_C}^{-1}P_L E$ applied on the lidar scans.

4.3.2 Projecting lidar point clouds to image frame

When the lidar point clouds scanned before and after the image frame has been transformed to the car pose in which the image frame was taken in, we further project the lidar point clouds to the image frame. The projection is done using the extrinsic calibration matrices of both the lidar and the camera. First, we transform the lidar point cloud to the ego frame using the inverse of the lidar extrinsic calibration matrix and then we transform the point cloud from the ego frame to the camera frame using the camera extrinsic calibration. Denoting the lidar extrinsic calibration matrix with L and the camera extrinsic calibration matrix with C, the following transformation is applied on the point clouds: $L^{-1}C$. The last dimension represents the distance from the camera to the lidar point (depth). Using the intrinsic of the camera and the Kannala-Brandt model we can further transform the points in the camera frame to the image frame. Each lidar point thus has a pixel coordinate with a depth (i,j,d) with a corresponding lidar coordinate (x,y,z).

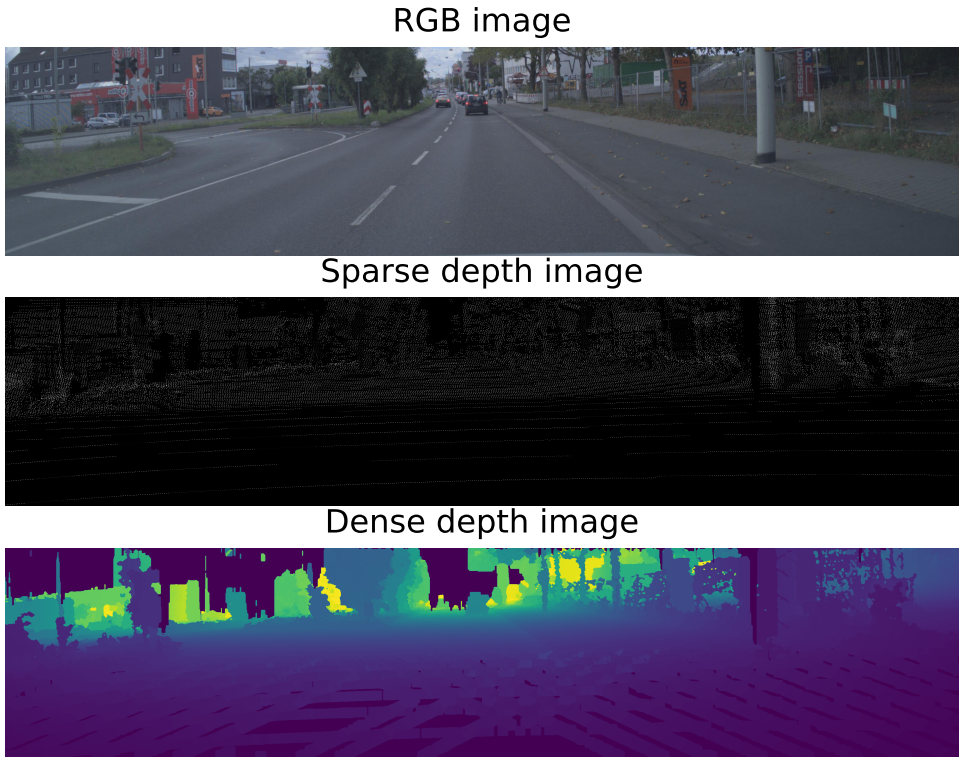


Figure 4.5: Sparse and depth image for the same frame. The depth image was generated using the sparse depth image with the IP-basic method.

4.3.3 Generating dense depth image

Using the lidar points in the image with their corresponding depth (i, j, d), the sparse depth image, and following subsection 3.1.5 we generated dense depth images. In Figure 4.5 we can see an image frame with corresponding sparse and dense depth images.

4.3.4 Filtering lidar points to annotations

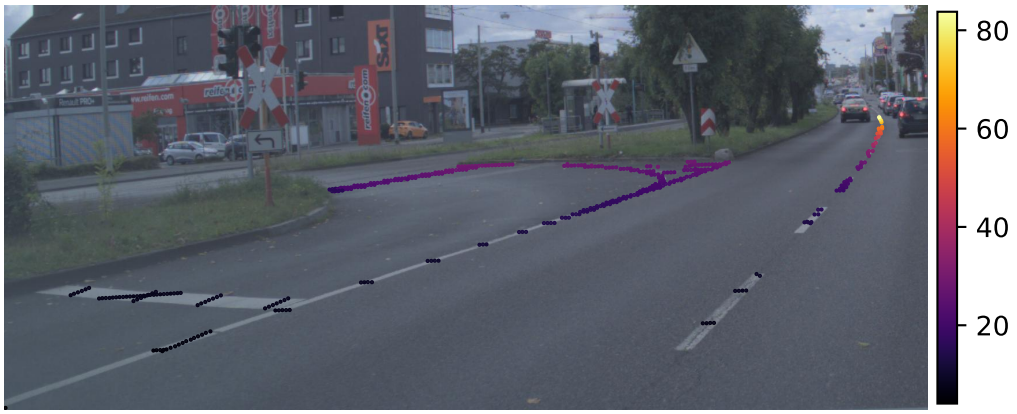


Figure 4.6: Lidar points that are kept for annotations on an image frame. The depth is encoded with the gradient color and a unit of meters.

Knowing the location of the lidar points on the image, we filtered out lidar points that were outside the already provided 2D lane annotations on the image. Keeping all lidar points that lie inside the annotations, we have a set of 3D lidar points for each annotated lane on the image. Figure 4.6 illustrates the lidar points that are kept for the lane annotations on an image frame.

4.3.5 3D lanes to grid

Using the 3D lidar points assigned to each annotation, a grid with the lane information was created for each frame (as the network will output a fixed size array). Each grid point contains the lane position in the Y-dimension ("left and right" from heading) and Z-dimension ("up and down" from heading) and a class. The classes are solid lane marker, dashed lane marker, botts dot lane marker, shaded lane marker, road painting, and empty. An illustration of the grid from BEV can be seen in Figure 4.7. The grid sizes in the Y-dimension are set to $L_y = 0.15$m and range from -15m to 15m, thus we have 200 grid points in the Y-dimension. The grid sizes in the X-dimension (L_x) are not fix and ranges from/to [0m, 5m], [5m, 10m], [10m, 15m], [15m, 20m], [20m, 30m], [30m, 40m], [40m, 50m], [50m, 60m], [60m, 70m], [70m, 90m], resulting in 10 grid points in the X-dimension. In total, the grid consists of $200 \cdot 10 = 2000$ grid points.

4.3.6 Annotation configurations

The annotation was created for 4 different annotation configurations (shown in Table 4.1), based on the number of lidar scans used before and after the closest lidar scan to the camera frame in time. Referring to Table 4.1, the first annotation configuration (Configuration 1) only uses the closest lidar scan to the image frame and the second configuration (Configuration 2) uses the closest lidar scan to the image frame, and the next 3 lidar scans as well.

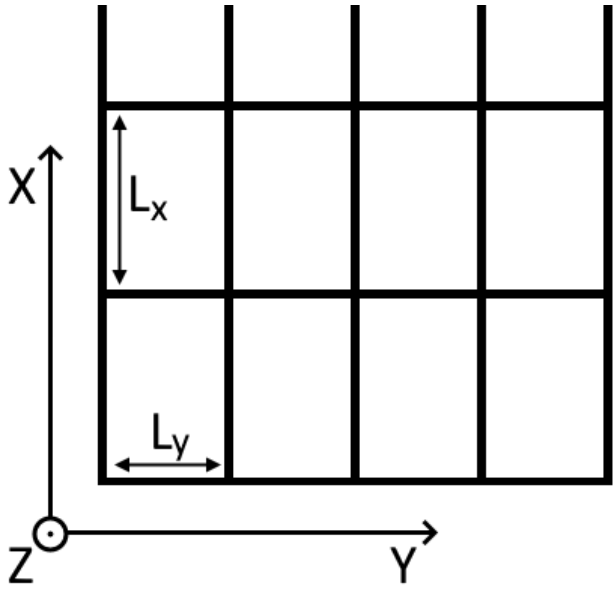


Figure 4.7: Visualization of the grid from a bird's-eye view with the used coordinate system. Each grid segment has a size of L_x x L_y.

Table 4.1: Annotation configurations

	Lidar scans before	Lidar scans after
Configuration 1	0	0
Configuration 2	0	3
Configuration 3	3	0
Configuration 4	3	3

4.4 Training

For a start, the network was trained on each annotation configuration for 3 epochs resulting in 4 different networks. All 3 epochs for each annotation configuration were trained with the same training setup. The optimizer used was Stochastic Gradient Descent with momentum = 0.9 and a learning rate with a step scheduler. The initial learning rate was set to 1×10^{-2} and was stepped down by a factor of 10 for each epoch (thus the learning rate was 1×10^{-3} and 1×10^{-4} for the second and third epoch respectively). Configuration 4 (3 before / 3 after) was further trained with 11 epochs, resulting in 14 total epochs. The first 9 epochs were trained with the mentioned step learning rate (1×10^{-2}, 1×10^{-3}, 1×10^{-4}), resetting for each 3 epochs. The last 5 epochs were trained with a constant learning rate of 1x10.

4.4.1 Loss

The loss used was a combination of three different losses resulting from our network: regression loss $\mathcal{L}_r$, classification loss $\mathcal{L}_c$ and depth loss $\mathcal{L}_d$. Using fixed parameters $(\lambda_r, \lambda_c, \lambda_d)$ to balance out each individual loss, we can define the total loss as seen in Equation 4.3.

$$\mathcal{L} = \lambda_r \mathcal{L}_r + \lambda_c \mathcal{L}_c + \lambda_d \mathcal{L}_d \tag{4.3}$$

The loss weighing factors were set to $\lambda_r = 12$, $\lambda_c = 40$ and $\lambda_d = 1$.

4.4.1.1 Regression loss

For non-empty grids, we define the lane regression loss $\mathcal{L}_r$ as the average L_1-norm of the difference between the predicted lane position $\hat{\boldsymbol{X}}_{reg}$ and the ground-truth lane position $\boldsymbol{X}_{reg}$ for all grids.

$$\mathcal{L}_r = \frac{1}{N} \sum_{i=1}^{N} \| \hat{\boldsymbol{X}}_{reg}^i - \boldsymbol{X}_{reg}^i \|_1 = \frac{1}{N} \sum_{i=1}^{N} \left| \hat{Y}_i - Y_i \right| + \left| \hat{Z}_i - Z_i \right| \tag{4.4}$$

The expression for the lane regression loss for a grid with N non-empty points can be seen in Equation 4.4, where Y_i and $\hat{Y}_i$ / Z_i and $\hat{Z}_i$ is the predicted and ground-truth lane position in the Y / Z dimension. In the case of no lanes in the frame ($N = 0$), the lane regression loss is set to zero $\mathcal{L}_r = 0$.

4.4.1.2 Classification loss

The lane classification loss $\mathcal{L}_c$ is defined as the average focal loss (FL)($\gamma = 2$) between all classes for the predicted and ground-truth grid. Denoting the predicted grid point classes with $\hat{\boldsymbol{G}}$ and ground-truth grid point classes with $\boldsymbol{G}$, the lane classification loss is described with Equation 4.5 for a grid with N_x/N_y grid points in the X/Y dimension.

$$\mathcal{L}_c = \frac{1}{N_x \cdot N_y} \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \mathrm{FL}(G(i,j), \hat{G}(i,j)) \tag{4.5}$$

4.4.1.3 Depth loss

Using the focal loss again, the depth loss is defined as the average focal loss ($\gamma = 2$) for each predicted depth bin. With a predicted depth distribution $\hat{\boldsymbol{D}}$ and ground-truth depth distribution $\boldsymbol{D}$ with width W_d and height H_d, we express the depth loss as in Equation 4.6.

$$\mathcal{L}_d = \frac{1}{W_d \cdot H_d} \sum_{i=1}^{W_d} \sum_{j=1}^{H_d} \mathrm{FL}(D(i,j), \hat{D}(i,j)) \tag{4.6}$$

4.5 Determining threshold

Using a subset of the test set, a classification threshold for empty grids was determined. If a grid point classifies an empty lane (highest softmax probability for the empty class) then the softmax probability needs to be higher than the threshold, otherwise, the class with the second highest softmax probability is the classified lane-type for the particular grid point. By changing the threshold from 0.2 to 0.8, with steps of 0.03, the F1-mean across all classes was recorded for the subset of the test set. The threshold that maximized the F1-mean across the classes was set as the threshold. A threshold was set for all 5 models trained.

4.6 Evaluation metrics

To evaluate the models trained, classification and regression metrics were calculated. The classification metrics used were accuracy, precision, recall, and F1 per class. To further analyze the classification results, the metrics were also evaluated based on the distance from the heading of the car for the range 0-90m (the whole grid), 0-30m, and 60-90m. The regression metrics included were RMSE and MAE for the Y and Z-directions on two sets of grid points, the ground truth, and the prediction mask. The ground truth mask defines all grid points where there exists a lane (non-empty class) as per ground truth while the prediction mask defines all grid points where there is a predicted lane (non-empty class) from the network output.

4.7 Past and future inference to improve lane detection

For this part of the method, sequences from the dataset were used. The sequences include 20 second sensor data with one annotated frame in the middle. By using the inference from past and future frames, the combined grid was evaluated with classification and regression scores for 120 different sequences. The classification scores used were F1-mean and the regression scores were RSME and MAE in the Y- and Z-directions. The contribution of the grids from the past and future frames was added to the current frame by first transforming the grid points to 3D coordinates. The transformation from grid points to 3D points was done by assuming that each 3D point lies in the middle of the grid in the X-direction (heading direction of the car). The 3D coordinates from the future/past coordinate frame were transformed to the current coordinate frame and then the 3D coordinates were transformed back to a grid representation. For each grid point, an average of the predicted Y, Z, and class probabilities were taken. The combinations of the number of future and past frames used to improve the lane detection for a given frame can be seen in Table 4.2.

4.8 Post-processing

To create contiguous lanes out of the detected grids, each detected grid was grouped into an instance. An instance represents one contiguous lane that contains unique grids, meaning one particular grid cannot exist in two instances. The grids were grouped into different instances based on two criteria: Firstly, the grids in the same instance need to be of the same class, and secondly, the horizontal distance (Y) from the neighboring grids needs to be within 1 m in order to avoid combining two separate lanes. By neighboring grids, it means the closest grids that the model predicts to be a lane in the vertical direction. This is illustrated in Figure 4.8. In Figure 4.8 detected grid points 1, 2, and 3 were clustered into one instance as they are neighbors in the vertical direction, have the same class, and fall into the range of 1m in the horizontal direction. Furthermore, the detected grid points 4 and 5 in

Table 4.2: Combinations of future and past frames used

Past frames	Future frames
5	5
4	4
3	3
2	2
1	1
5	0
4	0
3	0
2	0
1	0
0	5
0	4
0	3
0	2
0	1
0	0

Figure 4.8 are clustered together without detected grid point 6 as it is outside the range of 1m in the horizontal direction.

Once each grid had been uniquely associated with an instance, linear interpolation was applied for each instance. Specifically, 100 horizontal (Y) and height (Z) values were generated through linear interpolation with a sample rate of 0.1, with their endpoints being the minimum and maximum of the respective detected directions. Using the respective points in the ground truth, a corresponding interpolation was made. The interpolated lanes were then compared to their corresponding interpolated ground truth lane to calculate the MAE in the Y and Z dimensions for each interpolation point. Additionally, the average euclidean distance between each interpolation point was calculated.

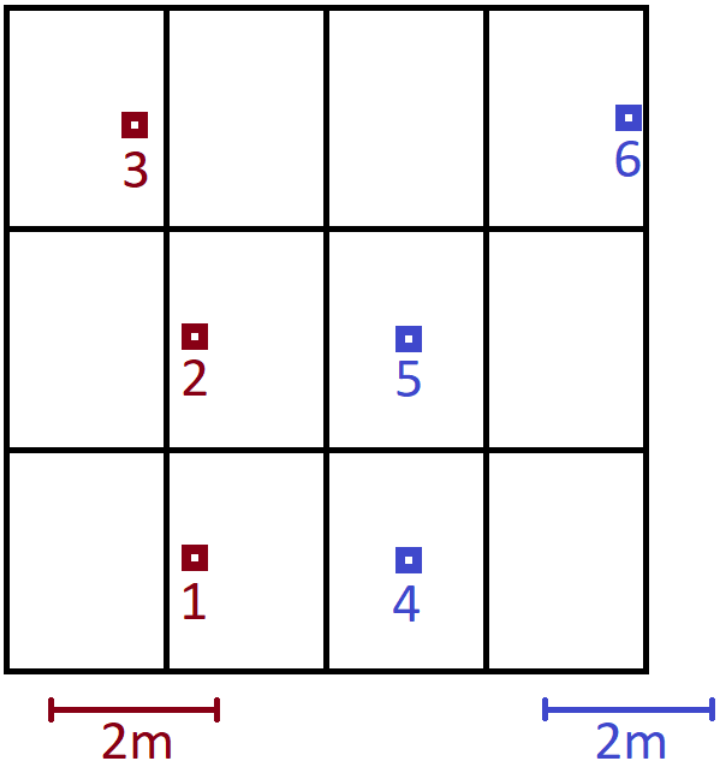


Figure 4.8: Visualization of lane instance clustering. Grid points 1, 2, and 3 are counted as a lane instance & grid points 4 and 5 are counted as another lane instance. Grid point 6 does not belong to the other instances.

5

Results

5.1 Annotation configurations

5.1.1 Loss

The resulting loss from training using the annotation configurations on 3 epochs can be seen in Figure 5.1. The regression, classification, and depth loss can be seen separately under Appendix A. The loss figures show the loss as a function of the training iterations, where one iteration corresponds to training a batch. The training loss versus the validation loss for the training can be seen in Figure 5.2.

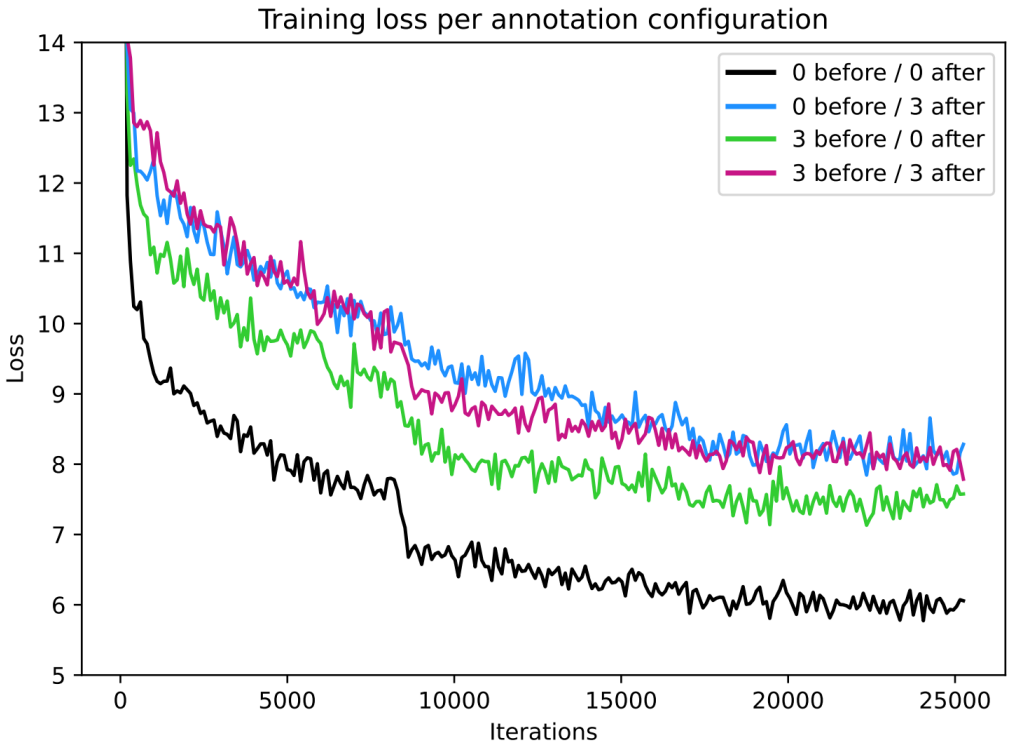


Figure 5.1: Training loss for all annotation configurations trained for 3 epochs.

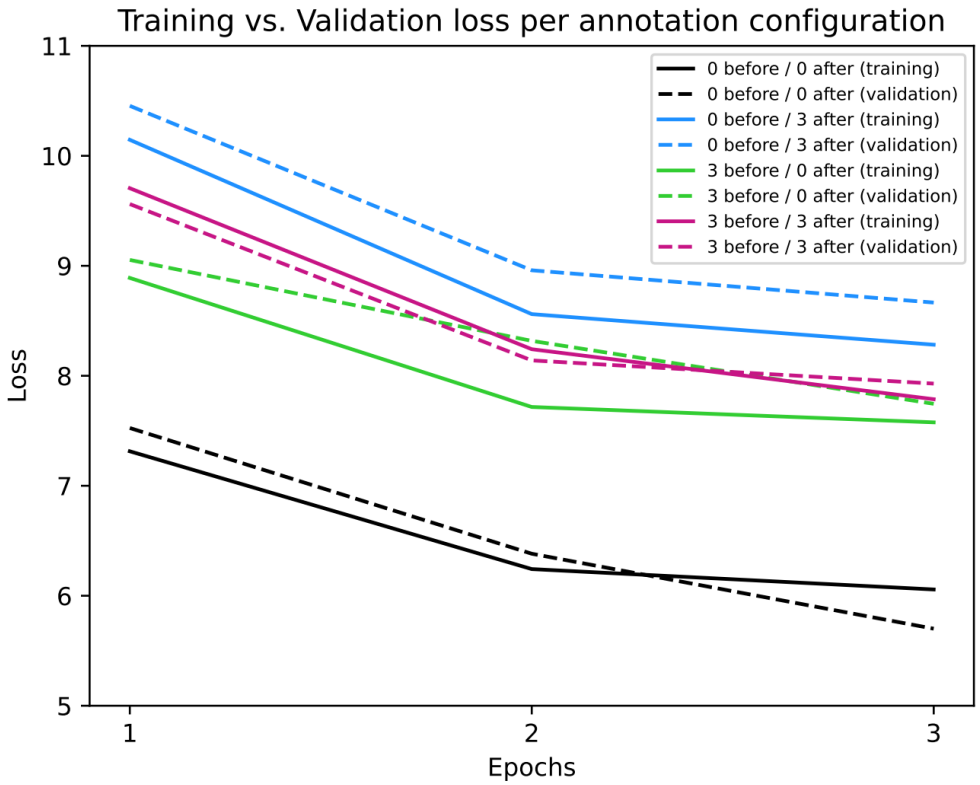


Figure 5.2: Training against validation loss for all annotation configurations trained 3 epochs.

5.1.2 Threshold

The thresholds for the annotation configurations can be seen in Table 5.1, where using each threshold maximizes the F1-mean for a subset of the test set. The gain of using a threshold can also be seen in Table 5.1. Table 5.1 displays the F1-mean for the non-empty classes and all classes when using a threshold and not using a threshold for all annotation configurations. Visualizations of the relationship between the F1-mean and threshold for all annotation configurations can be found under subsubsection A.1.1.2 in the Appendix.

Table 5.1: Threshold set for models for all annotation configurations & F1-mean (for non-empty/all classes) difference of using a threshold for all annotation configurations.

		Without threshold	With threshold
	Threshold	F1-mean	F1-mean
Configuration 1	0.625	0.408	0.540
Configuration 2	0.581	0.424	0.512
Configuration 3	0.610	0.397	0.515
Configuration 4	0.559	0.474	0.534

Table 5.2: Regression error (RMSE & MAE) for the Y- and Z-direction on grid points where lanes are present in the ground truth.

	Mask: ground truth			
	RMSE in Y	RMSE in Z	MAE in Y	MAE in Z
Configuration 1	0.031	0.399	0.023	0.188
Configuration 2	0.141	0.463	0.033	0.199
Configuration 3	0.031	0.382	0.023	0.182
Configuration 4	0.031	0.353	0.023	0.147

Table 5.3: Regression error (RMSE & MAE) for the Y- and Z-direction on grid points where lanes are detected by the model.

	Mask: prediction			
	RMSE in Y	RMSE in Z	MAE in Y	MAE in Z
Configuration 1	0.057	1.452	0.049	1.073
Configuration 2	0.186	1.527	0.063	1.180
Configuration 3	0.061	1.552	0.053	1.220
Configuration 4	0.059	1.483	0.051	1.130

5.1.3 Evaluation metrics

The regression error for the annotation configurations evaluated on the same ground truth is displayed in Table 5.2 and Table 5.3. Both figures show the RMSE and MAE error in both the Y- and Z-direction for each annotation configuration. In Table 5.3, all ground-truth grid points with a non-empty class are considered. In Table 5.3, all grid points with a detected lane (a non-empty class) are considered.

Table 5.4 displays classification scores (accuracy, precision, recall and F1) per class and ranges 0-90m, 0-30m and 60-90m. The distribution of the absolute errors in the Y- and Z-direction for the ground-truth and prediction mask can be seen in Figure 5.3 and Figure 5.4. The classification scores seen in Table 5.4 and the distributions in Figure 5.3 and Figure 5.4 are evaluated on the model trained on annotation configuration 4 (3 before / 3 after).

Table 5.4: Classification scores per class for ranges 0-90m, 0-30m, and 30-90m and for annotation configuration 4 (3 before / 3 after).

	0-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.955	0.972	0.975	0.918	0.955
Precision	0.368	0.365	0.319	0.966	0.505
Recall	0.471	0.386	0.513	0.945	0.579
F1	0.413	0.376	0.393	0.955	0.534

	0-30m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.948	0.968	0.971	0.909	0.949
Precision	0.400	0.394	0.367	0.970	0.533
Recall	0.579	0.512	0.609	0.929	0.657
F1	0.473	0.446	0.458	0.949	0.582

	30-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.961	0.976	0.978	0.927	0.961
Precision	0.303	0.294	0.234	0.963	0.449
Recall	0.316	0.213	0.359	0.960	0.462
F1	0.309	0.247	0.284	0.961	0.450

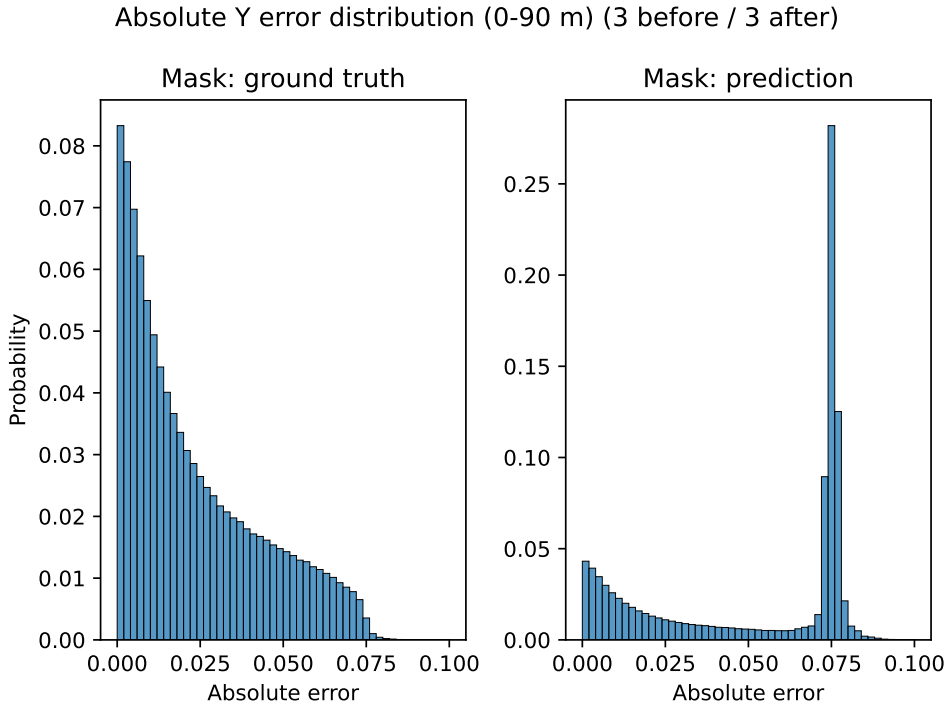


Figure 5.3: Absolute Y error distribution for annotation configuration 4 (3 before / 3 after). The absolute error has a unit of meters.

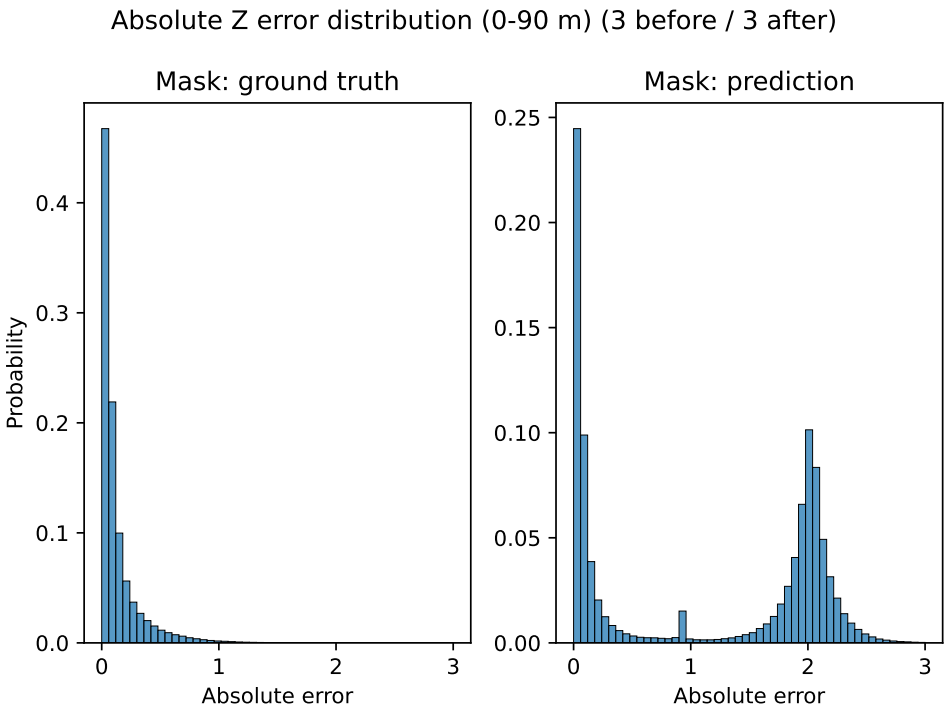


Figure 5.4: Absolute Z error distribution for annotation configuration 4 (3 before / 3 after). The absolute error has a unit of meters.

5.2 Further training on configuration 4 (3 before / 3 after)

The training loss for annotation configuration 4 (with 3 LIDAR scans before and after) can be seen in Figure 5.5, trained for a total of 14 epochs. The regression, classification and depth loss can be seen separately under subsubsection A.1.2.1 in the Appendix. The training loss versus the validation loss for the training can be seen in Figure 5.6.

5.2.1 Loss

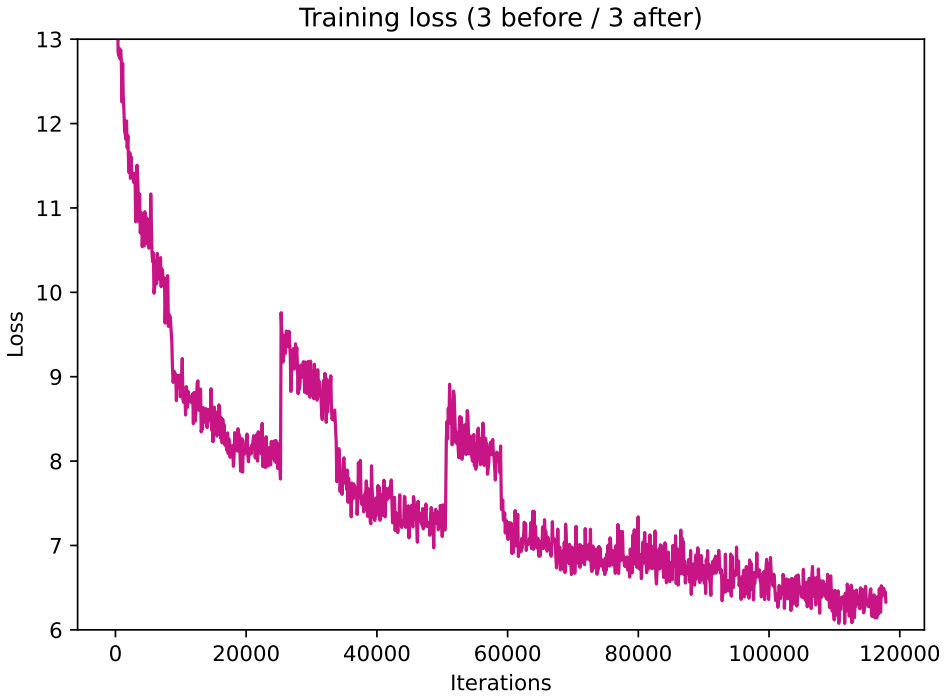


Figure 5.5: Training loss for annotation configuration 4 (3 before / 3 after) trained for 14 epochs.

5.2.2 Threshold

The threshold for the model further trained with annotation configuration 4 for a total of 14 epochs is determined to be 0.562. The gain of using the threshold can be seen in Table 5.5. Table 5.5 displays the F1-mean for the non-empty classes and all classes when using the threshold and not using the threshold. The relationship between the F1-mean and threshold for the model trained on configuration 4 (3 before / 3 after) for 14 epochs can be seen under subsubsection A.1.2.2 in the Appendix.

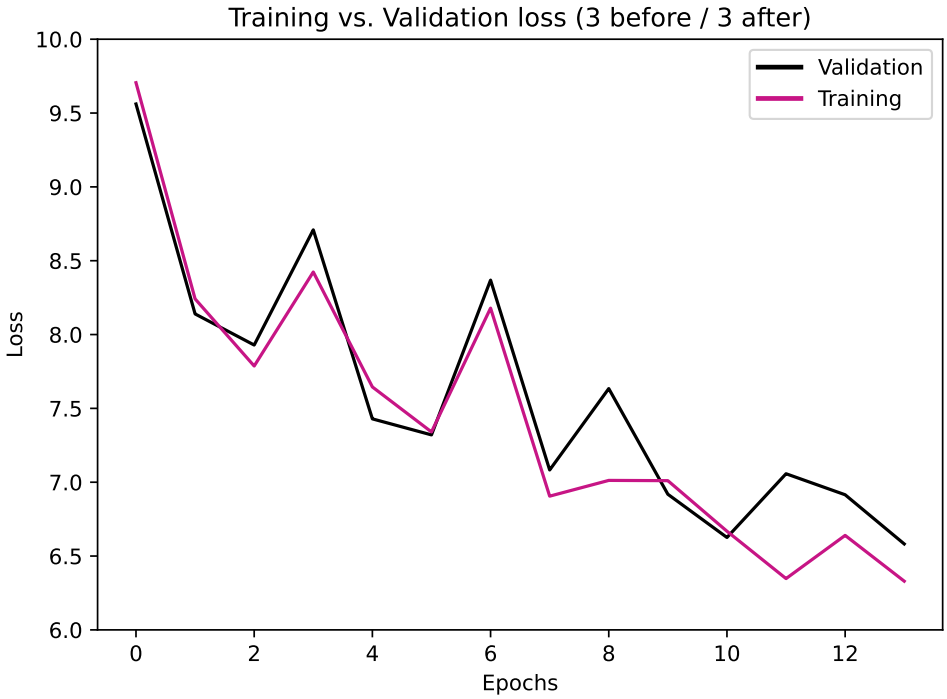


Figure 5.6: Training against validation loss for annotation configuration 4 (3 before / 3 after) trained for 14 epochs.

Table 5.5: F1-mean (for non-empty/all classes) difference of using a threshold for configuration 4 (3 before / 3 after) trained for 14 epochs.

No threshold		With threshold	
F1-mean (non-empty)	F1-mean	F1-mean (non-empty)	F1-mean
0.441	0.573	0.480	0.601

5.2.3 Evaluation metrics

The regression error for the model trained on annotation configuration 4 (3 before/3 after) for 14 epochs is displayed in Table 5.6 and Table 5.7. Both figures show the RMSE and MAE error in both the Y- and Z-direction for the ranges 0-90m (whole grid), 0-30m, and 30-90m. In Table 5.6, all ground-truth grid points with a non-empty class are considered. In Table 5.7, all grid points with a detected lane (a non-empty class) are considered.

Table 5.8 displays classification scores (accuracy, precision, recall, and F1) per class and ranges from 0-90m, 0-30m, and 60-90m. The distribution of the absolute errors in the Y- and Z-direction for the ground truth and prediction mask can be seen in Figure 5.7 and Figure 5.8. The classification scores seen in Table 5.8 and the distributions in Figure 5.7 and Figure 5.8 is evaluated on the model trained on annotation configuration 4 (3 before / 3 after) for 14 epochs. The same statistics for the other annotation configurations are available under subsubsection A.1.1.3 in the Appendix.

Table 5.6: Regression error (RMSE & MAE) for the Y- and Z-direction on grid points where lanes are present in ground truth for the model trained on annotation configuration 4 (3 before / 3 after) for 14 epochs. The regression errors are shown for ranges 0-90m (whole grid), 0-30m, and 30-90m.

	Mask: ground truth			
Range	RMSE in Y	RMSE in Z	MAE in Y	MAE in Z
0-90m	0.030	0.278	0.023	0.113
0-30m	0.030	0.206	0.023	0.102
30-90m	0.030	0.359	0.023	0.130

Table 5.7: Regression error (RMSE & MAE) for the Y- and Z-direction on grid points where lanes are detected by the model trained on annotation configuration 4 (3 before / 3 after) for 14 epochs. The regression errors are shown for ranges 0-90m (whole grid), 0-30m, and 30-90m.

	Mask: prediction			
Range	RMSE in Y	RMSE in Z	MAE in Y	MAE in Z
0-90m	0.056	1.369	0.047	0.959
0-30m	0.054	1.274	0.044	0.852
30-90m	0.059	1.518	0.051	1.142

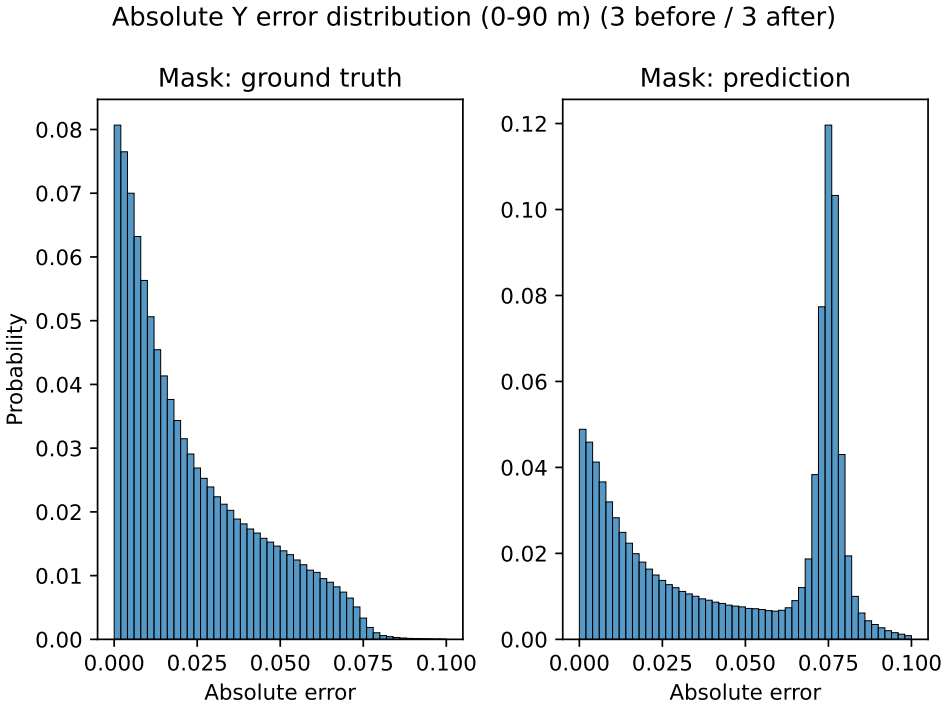


Figure 5.7: Absolute Y error distribution for the model trained on annotation configuration 4 (3 before / 3 after) for 14 epochs. The absolute error has a unit of meters.

Table 5.8: Classification scores per class for ranges 0-90m, 0-30m, and 30-90m and for the model trained on annotation configuration 4 (3 before / 3 after) for 14 epochs.

	0-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.963	0.976	0.979	0.933	0.963
Precision	0.452	0.455	0.413	0.971	0.573
Recall	0.549	0.477	0.571	0.956	0.638
F1	0.496	0.466	0.479	0.964	0.601

	0-30m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.961	0.975	0.978	0.932	0.962
Precision	0.502	0.504	0.479	0.975	0.615
Recall	0.651	0.592	0.654	0.950	0.712
F1	0.567	0.545	0.553	0.962	0.657

	30-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.965	0.977	0.981	0.934	0.964
Precision	0.367	0.362	0.309	0.968	0.502
Recall	0.402	0.314	0.438	0.962	0.529
F1	0.384	0.337	0.362	0.965	0.512

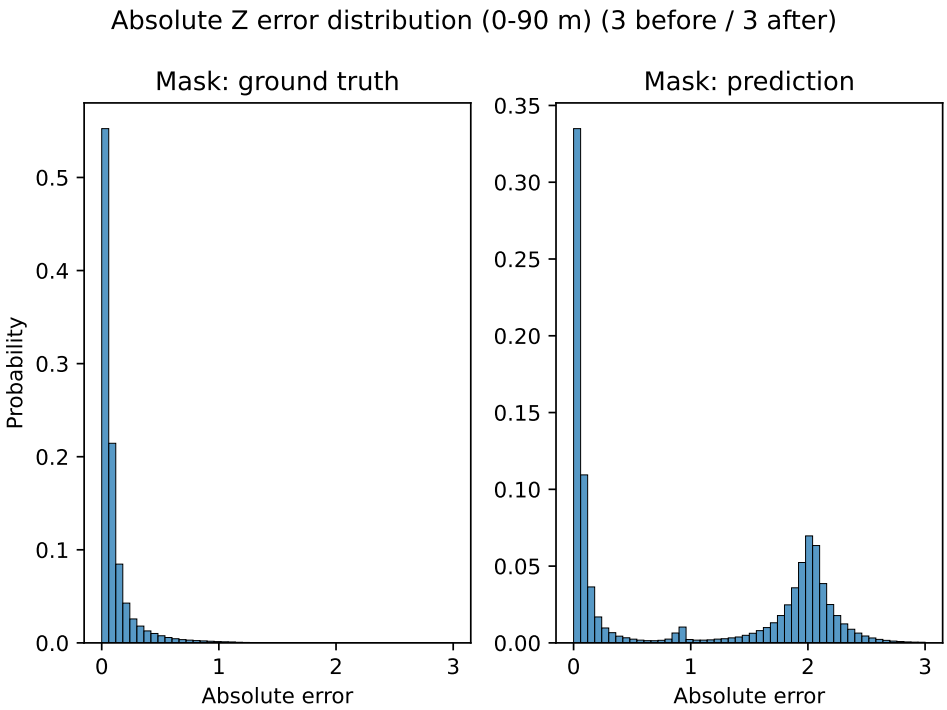


Figure 5.8: Absolute Z error distribution for the model trained on annotation configuration 4 (3 before / 3 after) for 14 epochs. The absolute error has a unit of meters.

5.3 Past and future inference to improve lane detection

The effect of using past and future grids can be seen in Table 5.9, Table 5.10 and Table 5.11 where the F1-mean for all classes and non-empty classes, as well as the RSME and MAE in the Y and Z-directions, is displayed. Table 5.9, Table 5.10 and Table 5.11 displays the effect of using both future & past, only past and only future frames respectively. In Table 5.9, Table 5.10 and Table 5.11 the regression scores are calculated over all grid points with a detected lane (prediction mask), the same statistics for the ground truth mask in available under subsection A.1.3 in the Appendix. Examples of a scene and corresponding detections viewed from BEV with and without using past/future inference are visualized in Figure 5.9 and Figure 5.10.

Table 5.9: Classification and regression scores when combining inference from the current frame with past and future frames.

			Mask: prediction			
Number of past/future frames	F1-mean	F1-mean (non-empty)	RSME in Y	RSME in Z	MAE in Y	MAE in Z
5 past / 5 future	0.507	0.359	0.057	1.450	0.048	1.067
4 past / 4 future	0.508	0.360	0.058	1.457	0.049	1.077
3 past / 3 future	0.510	0.364	0.058	1.466	0.049	1.088
2 past / 2 future	0.508	0.362	0.059	1.477	0.050	1.103
1 past / 1 future	0.509	0.363	0.060	1.501	0.051	1.130
0 past / 0 future	0.500	0.352	0.059	1.524	0.051	1.160

Table 5.10: Classification and regression scores when combining inference from the current frame with past frames.

			Mask: prediction			
Number of past/future frames	F1-mean	F1-mean (non-empty)	RSME in Y	RSME in Z	MAE in Y	MAE in Z
5 past / 0 future	0.495	0.345	0.062	1.545	0.053	1.189
4 past / 0 future	0.500	0.351	0.062	1.533	0.053	1.174
3 past / 0 future	0.501	0.353	0.063	1.520	0.053	1.158
2 past / 0 future	0.504	0.356	0.063	1.504	0.054	1.141
1 past / 0 future	0.505	0.358	0.065	1.510	0.056	1.145
0 past / 0 future	0.500	0.352	0.059	1.524	0.051	1.160

Table 5.11: Classification and regression scores when combining inference from the current frame with future frames.

			Mask: prediction			
Number of past/future frames	F1-mean	F1-mean (non-empty)	RSME in Y	RSME in Z	MAE in Y	MAE in Z
0 past / 5 future	0.504	0.355	0.057	1.428	0.048	1.044
0 past / 4 future	0.503	0.355	0.058	1.439	0.049	1.060
0 past / 3 future	0.504	0.356	0.058	1.457	0.049	1.079
0 past / 2 future	0.504	0.357	0.058	1.481	0.050	1.106
0 past / 1 future	0.506	0.360	0.059	1.512	0.051	1.140
0 past / 0 future	0.500	0.352	0.059	1.524	0.051	1.160

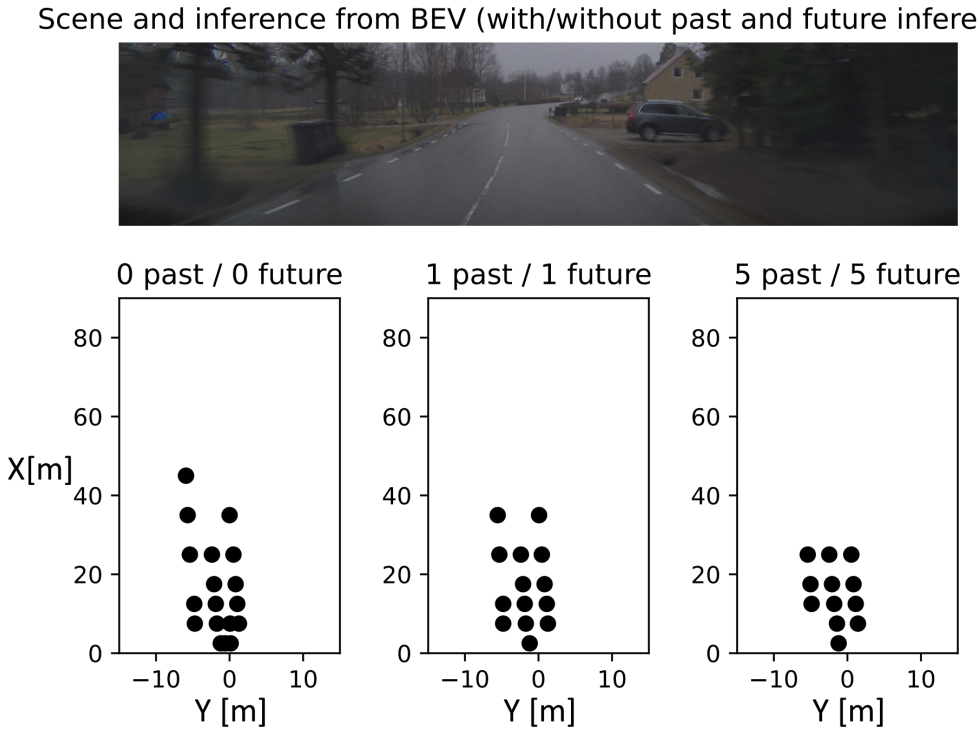


Figure 5.9: Example scene and inference from BEV with/without past and future inference. The shown configurations for the number of past/future inferences are 0 past & 0 future, 1 past & 1 future, and 5 past & 5 future.

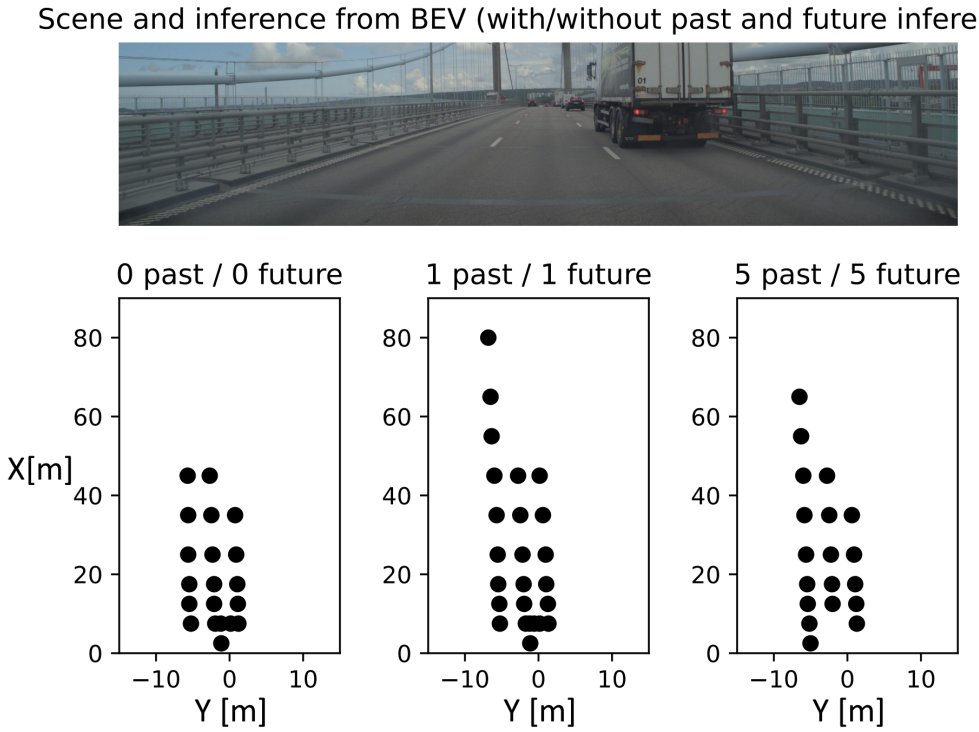


Figure 5.10: Example scene and inference from BEV with/without past and future inference. The shown configurations for the number of past/future inferences are 0 past & 0 future, 1 past & 1 future, and 5 past & 5 future.

5.4 Post-processing

The MAE in the Y- & Z-direction and the euclidean distance between the interpolated lanes compared to ground truth for different number of past/future frames can be seen in Table 5.13, Table 5.14 and Table 5.12. Table 5.13, Table 5.14 and Table 5.12 displays the errors for the interpolated lanes using both future & past, only past and only future frames respectively. Examples of interpolated lanes from BEV for two lane scenarios with and without using past/future inference are visualized in Figure 5.11 and Figure 5.12. The scenes used in Figure 5.11 and Figure 5.12 are the same as in Figure 5.9 and Figure 5.10 respectively.

Table 5.12: MAE in Y- and Z-directions & Mean euclidean distance for interpolated lanes using future frames.

Number of past/future frames	MAE in Y	MAE in Z	Mean euclidean distance
0 past / 5 future	0.212	0.151	0.295
0 past / 4 future	0.224	0.150	0.304
0 past / 3 future	0.221	0.147	0.302
0 past / 2 future	0.220	0.151	0.302
0 past / 1 future	0.213	0.149	0.295
0 past / 0 future	0.210	0.158	0.301

Table 5.13: MAE in Y- and Z-directions & Mean euclidean distance for interpolated lanes using past and future frames.

Number of past/future frames	MAE in Y	MAE in Z	Mean euclidean distance
5 past / 5 future	0.185	0.141	0.268
4 past / 4 future	0.194	0.145	0.280
3 past / 3 future	0.201	0.149	0.287
2 past / 2 future	0.212	0.155	0.300
1 past / 1 future	0.210	0.153	0.295
0 past / 0 future	0.210	0.158	0.301

Table 5.14: MAE in Y- and Z-directions & Mean euclidean distance for interpolated lanes using past frames.

Number of past/future frames	MAE in Y	MAE in Z	Mean euclidean distance
5 past / 0 future	0.204	0.159	0.297
4 past / 0 future	0.216	0.167	0.314
3 past / 0 future	0.221	0.162	0.313
2 past / 0 future	0.214	0.157	0.302
1 past / 0 future	0.226	0.161	0.313
0 past / 0 future	0.210	0.158	0.301

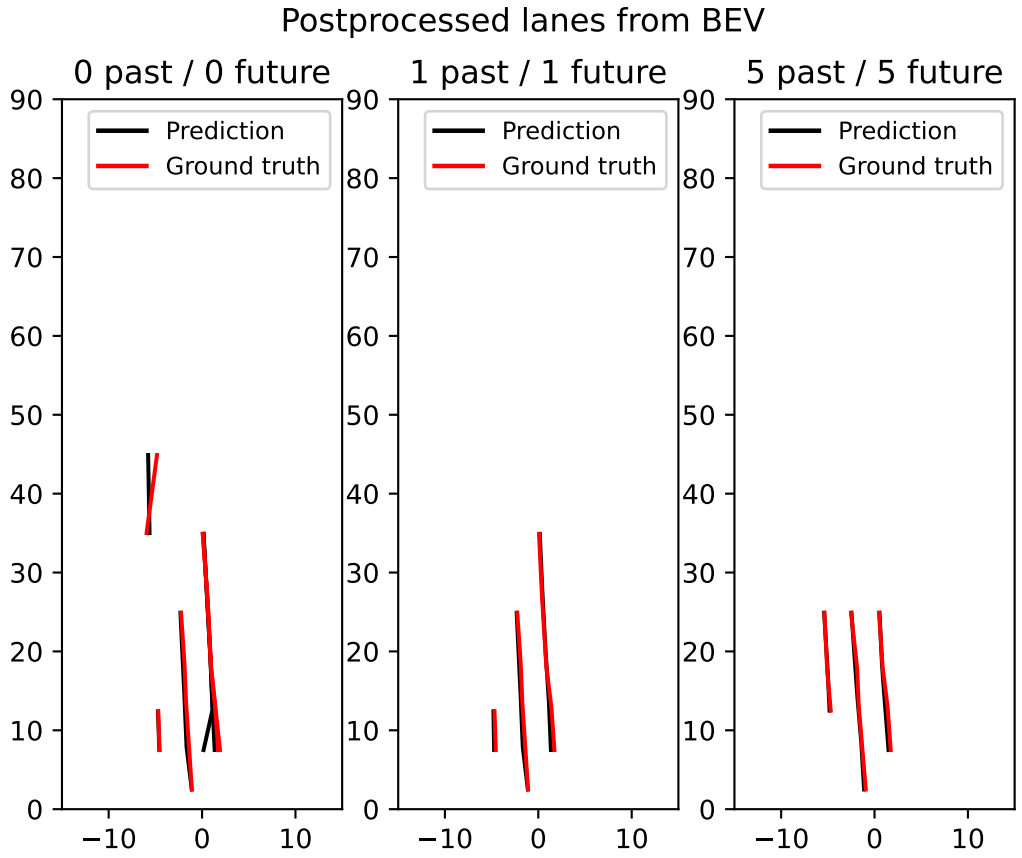


Figure 5.11: Example scene and inference from BEV with/without past and future inference. The shown configurations for the number of past/future inferences are 0 past & 0 future, 1 past & 1 future, and 5 past & 5 future. The scene is the same as in Figure 5.9.

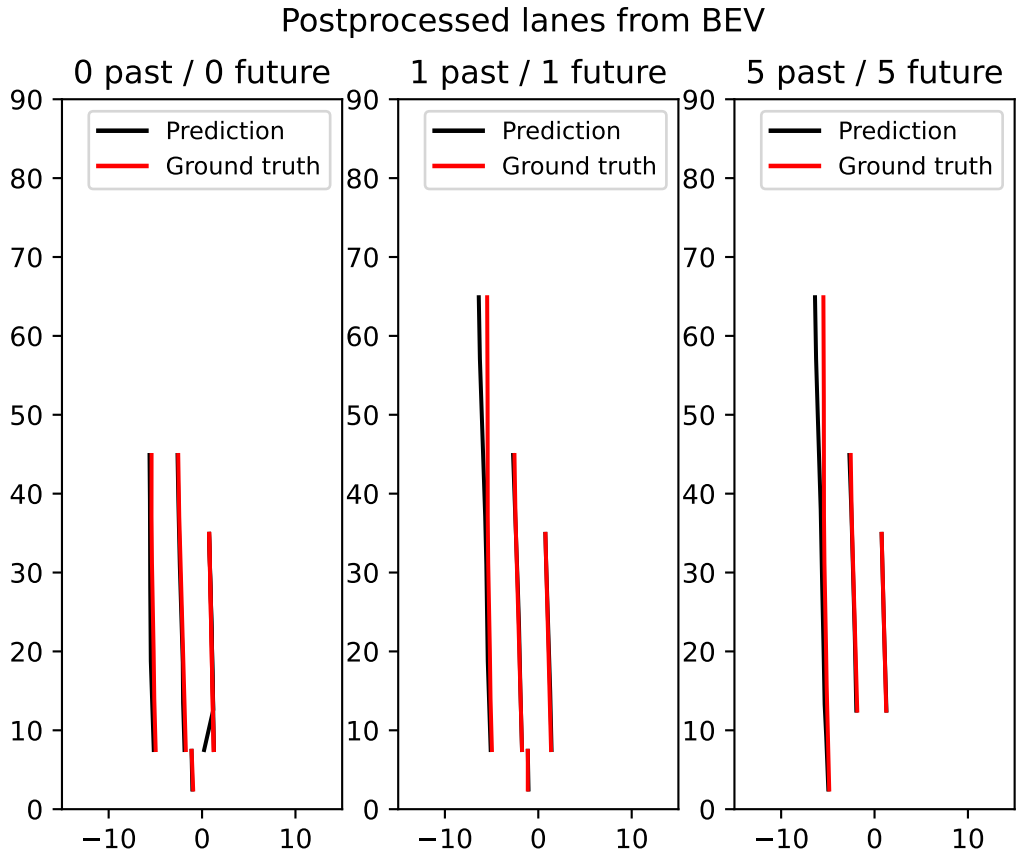


Figure 5.12: Example scene and inference from BEV with/without past and future inference. The shown configurations for the number of past/future inferences are 0 past & 0 future, 1 past & 1 future, and 5 past & 5 future. The scene is the same as in Figure 5.10.

6
Discussion

6.1 Choice of configuration

The combined loss for the models of different annotation configurations can be observed in Figure 5.1, with the model using only the closest LIDAR scan for its camera frame having the largest loss decrease over time. A possible reason for this is that the ground truth for the models using before/after LIDAR scans is more strict, due to more 3D annotations for each frame. Each configuration had the potential to be further improved, indicated by the validation loss that can be observed in Figure 5.2. The time it took to train 3 whole epochs for each model was roughly 21 hours, meaning to train the models using all the unique samples once took 7 hours.

We evaluated the models using classification and regression scores. The classification scores were done using a subset of the test data for two cases: with threshold and no threshold (subsection 5.1.2). The reason for introducing the threshold here was due to the class imbalance we had when constructing our grid design (subsection 4.3.5). Each grid must be assigned a class in order for the network to learn to separate from background and foreground classes, with the "empty" class denoting the background. This resulted in the majority of the grids being "empty" which skewed the class distribution heavily, which was why focal loss (subsubsection 3.1.2.1) was used. The definition of threshold stemmed from the softmax probability the models produced for the "empty" class for a grid, where the idea was to determine a threshold that the model had to exceed in order to associate the grid as "empty". The threshold was determined by the F1 mean score for all the classes, including "empty", with their values displayed in Table 5.1. Regardless of the configurations, the thresholds were over $\frac{1}{2}$, indicating that all the models were biased. It can also be observed in the same table that using a threshold helped increase the F1-mean score for all configurations by at least 0.06. When not utilizing the threshold, configuration 4 had the highest F1-mean score, which was expected as it was trained with the most 3D annotations. The application of the threshold resulted in configuration 1 having the highest F1-mean score, slightly surpassing configuration 4 with the second highest score of F1-mean by 0.006. However, this may haven been contributed by the lack of training time for the configurations with stronger annotations, i.e. 2, 3, and 4.

Once the thresholds had been determined for each configuration, another subset of the test data was used to obtain regression scores with the thresholds applied on the models for two cases: ground truth mask and prediction mask (subsection 5.1.3).

The first case considers the grids where the ground truth class has a lane (non-"empty"), while the second considers grids that the model has classified as a lane. The regression scores for the two cases can be observed in Table 5.2 and Table 5.3 respectively. The predicted mask is of interest as that's where the model predicts a lane and performs regression, regardless if there truly is a lane or not. This in turn will give an offset/error with respect to its grid's ground truth, if there is a lane the error will be the absolute difference between the predicted value and its ground truth. If there exists no lane yet the model predicts a lane the error will be its absolute predicted value itself. This is visible in the absolute error distribution for Y and Z direction in Figure 5.3 and Figure 5.4 respectively. One can notice that the absolute errors for the predicted mask in both Y and Z directions have peaks at roughly 0.075 and 2. These are values for which the model most likely has classified the grids as lanes without them being actual lanes, pushing the errors to the far right of the distribution.

Another thing that can be noted is the model's ability to classify with respect to distance. Table 5.4 showcases the classification performance for the model of the 4th configuration for three intervals: 0-90, 0-30, and 30-90 meters in front of the vehicle. The model's performance to classify grids decreases for longer distances, which may not be of surprise as the camera have fewer pixels to represent objects farther away and the LIDAR obtains more points at a "close" range. This was also true for the remaining models that used different configurations, which can be observed in subsubsection A.1.1.3.

As the time required to train the models for one epoch was long, we had to choose one of the models to be trained even further upon to see if any improvements could be made. We intuitively thought that the model of configuration 4 had the most potential as it was trained with the most 3D annotations. We, therefore, chose to research if further improvements to the model could be made (section 5.2). In Figure 5.5 the training loss for the model of the 4th configuration can be observed, where the model has been trained for a total of 14 epochs. One can see that the model's training loss has a downward trend, even at the last number of iterations. This can also be seen in the validation loss in Figure 5.6, indicating that the model could be further improved with a longer training time. The peaks at roughly 25-35 thousand (4th epoch) and 50-60 thousand (7th epoch) iterations, were attributed due to the learning scheduler (section 4.4). At the mentioned intervals the learning rate was at its highest, i.e. 1×10^{-2}, which indicates that the learning rate at that certain stage of the model was too high. In summary, the model can be further improved with longer a training time (more epochs) followed by a learning rate smaller than 1×10^{-2} after it has been trained for 3 epochs.

While the training loss and validation loss have been shown to decrease for a total of 14 epochs, has the model of the 4th configuration improved? Let's first compare the classification scores shown in Table 5.4 and Table 5.8. For every distance and class combination, the model that has been trained for the longest time has improved. The improvement of the model's classification ability is reflected in the absolute

error distribution for Y (Figure 5.7) and Z (Figure 5.8) directions. The peaks have a lower representation of the two distributions when compared to the model's less trained version (Figure 5.3 and Figure 5.4). The last comparison of the two models is their regression scores for the ground truth mask and prediction mask. The two models can be compared using their respective regression score tables, i.e. Table 5.2 and Table 5.3 against Table 5.6 and Table 5.7. Note that comparing the two latter tables should be done in the 0-90m range. The model trained for a longer time has a lower value or the same value for each regression metric and direction combination, meaning it has improved from its earlier iteration.

6.2 Past and future inference to improve lane detection

The result presented in section 5.3 showcases the impact of combining the current frame with different combinations of past and future predicted frames, according to section 4.7. Table 5.10 considers past,Table 5.11 future, and Table 5.9 both past and future. It can be observed in both F1-mean scores when using solely future frames, the classification performance improves for all cases. Using only past frames improved classification performance for 3/5 cases, with the exception being 4 and 5 past frames. This seems reasonable if one recalls from Table 5.8 that the classification performance becomes worse at a longer range. As the past frames' predictions with respect to the current frame's predictions are farther away, meaning detections at a farther range from the previous frames are the ones that fall within the current frame's field of view. When instead combining both past and future, the F1-mean scores improve for all cases. The magnitude of the F1-mean scores was at their highest when utilizing both past and future frames. In summary, combining inference from the current frame with past and future frames strengthens the model's classification performance.

In the regression performance, the RMSE and MAE in both Y and Z directions can be observed for the grids which the model predicted as lanes. There is a similar trend in regression as in classification, i.e. using solely past frames yields a regression performance that is worse than when utilizing past and/or future frames. The reason here is the same as that explained in the classification case; recall Table 5.7). When comparing the best RMSE values using past and future frames against only future frames, the latter performs better in the Z direction, but equally well in the Y direction. This seems reasonable as the model's overall performance decreases for longer distances, notably in the Z direction (recall Table 5.7). The future frames can with their more accurate prediction at closer range support the current frame's prediction at a longer range. In summary, using future frames helps to improve the model's regression performance the most.

The effect of combining previous and future frames can be observed in Figure 5.9 and Figure 5.10, where a unique scene is presented along with our model's lane marking

predictions represented by black dots in BEV space. One thing that can be noted is that using fewer past and future frames yield fewer predictions in the long-range, and also fewer predictions overall. This was most likely due to the effect of our inference procedure, where each grid point's class probability was averaged over all merged grids (section 4.7). As mentioned, using only 4 or 5 past frames for inference did not improve the classification performance, because their predictions at the longer range are the ones that are included in the current frame. Thereby when averaging the class probabilities with many past frames included, could potentially lead to increasing the probability of the "empty" class and passing beyond the threshold value.

The last result is presented in section 5.4 where the grids that our model have predicted as a lane have been processed into a coherent lane according to the methods described in section 4.8. The same combinations of past & future frames were utilized as in section 5.3 to observe the effects of the mean error in different directions between the interpolated prediction and ground truth lanes. Table 5.12 showcases the use of future, Table 5.14 past, and Table 5.13 past & future predictions. A similar trend as section 5.3 can be observed; when only incorporating future predictions the error in all directions reduces in general, with the opposite being true when only using past predictions. A combination of both past & future predictions had the most beneficial impact, with the best mean error being 18.5 cm in horizontal, 14.1 cm in height, and 26.8 cm in the euclidean distance for the interpolated lanes. One thing that can be noted is that for longer vertical distances, the distance between each detection become larger (larger grid size in vertical distance) resulting in a longer interpolation. This means in order to minimize the error, the predicted grids must be close to ground truth for larger distances.

6.3 Future work

For the 3D lane generation pipeline, improvements may be done in the choice of grid size and the number of LIDAR scans used before/after. The grid size used in this paper was static but could be changed to see how it affects the quality of the annotation and the following detections as well. Studying the result of varying both the longitudinal/latitudinal width of the whole grid and the longitudinal/latitudinal width of each grid point could be of interest. The dataset we used in this study included up to 10 LIDAR scans before and after each camera frame, but only 3 LIDAR scans before/after were used due to computational limitations (each LIDAR scan includes millions of points). Thus, more than 3 LIDAR scans before/after the annotation pipeline could be used.

We only trained the model for a maximum of 14 epochs due to a shortage of time, with no sight of a plateau in the training. Therefore, training the model for longer is recommended. To further improve the training of the model, more data could also be relevant. The choice of weighting factors for the loss could also be optimized, as they were set arbitrarily in this study.

When using past and future inference to improve the current grid, all past and future

frames are equally weighed. In other words, each grid from past inference, future inference, and current inference contributes equally to the current detected grid. To optimize the use of past and future inference to improve the current grid, weighing the past and future inference would be beneficial. Most naturally, a weighing factor would be correlated with the inverse of the time from the current timestamp (the more time from the current frame, the lower the contribution).

The post-processing technique employed in this paper is quite simplistic, lacking the ability to adapt effectively to various scenarios. As a result, we suggest that future work should consider incorporating an extra head (similar to the regression head and classification head employed in this study) specifically for instance embedding of each lane. By introducing a dedicated head for lane instance embedding, the post-processing step would be trivial and yield a more robust lane representation.

7
Conclusion

The growing need for precisely annotated data in the automotive sector has brought attention to the automation of the annotation process. To automate lane labeling, this thesis presents three main concepts: a lane detection model, a pipeline for automatically annotating 3D lanes with LIDAR scans and 2D lane labels, and integrating past and future inference to improve lane detection.

Our model, based on SuperFusion and M^2-3DLaneNet model, trained for 14 epochs is not sufficient for generating ground-truth data for lane detection. The RSME in the horizontal direction is 5.6 cm. The RSME in height is 1.369 m. The F1-mean across all classes is 0.601. The regression score in the horizontal direction might be adequate for ground-truthing camera systems, but the regression score in the height and the classification score is not sufficient.

We have constructed a 3D lane label pipeline, that has 2D image lane labels and LIDAR point clouds as input. For the pipeline, we recommend using as many LIDAR scans from before and after the frame as possible.

There is a gain in using past and future inference to improve the current frame inference. Though, inference from past frames negatively impacts the regression and classification ability. Some frames from the past, but a larger number of frames from the future, is the most beneficial when using past/future inference to improve the current frame inference.

Lastly, we recommend future research to train our model for more epochs and on more data. Investigating the impact of the grid size and the number of LIDAR scans used in the 3D lane label pipeline, changing the loss weighing factors, and adjusting the weighting of past and future inference could also be of interest. A head specifically for instance embedding of the lanes is also of recommendation.

Bibliography

[1] Ashleigh Rose-Harman. The environmental benefits of driverless cars, 2021. URL https://greenerideal.com/news/vehicles/driverless-cars-environmental-benefits/.

[2] Velodyne. What is lidar? learn how lidar works, 2022. URL https://velodynelidar.com/what-is-lidar/.

[3] Wilfried Elmenreich. An introduction to sensor fusion. *Vienna University of Technology, Austria*, 502:1–28, 2002.

[4] Ramon F. Brena, Antonio A. Aguileta, Luis A. Trejo, Erik Molino-Minero-Re, and Oscar Mayora. Choosing the best sensor fusion method: A machine-learning approach. *Sensors*, 20(8), 2020. ISSN 1424-8220. doi:10.3390/s20082350.

[5] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6), 2021. ISSN 1424-8220. doi:10.3390/s21062140.

[6] Fangchang Ma and Sertac Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4796–4803, 2018. doi:10.1109/ICRA.2018.8460184.

[7] G Ajay Kumar, Jin Hee Lee, Jongrak Hwang, Jaehyeong Park, Sung Hoon Youn, and Soon Kwon. Lidar and camera fusion approach for object distance estimation in self-driving vehicles. *Symmetry*, 12(2), 2020. ISSN 2073-8994. doi:10.3390/sym12020324.

[8] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers, 2022. URL https://arxiv.org/abs/2203.11496.

[9] Zehui Chen, Zhenyu Li, Shiquan Zhang, Liangji Fang, Qinhong Jiang, and Feng Zhao. Autoalignv2: Deformable feature aggregation for dynamic multi-modal 3d object detection, 2022. URL https://arxiv.org/abs/2207.10316.

[10] Yingwei Li, Adams Wei Yu, Tianjian Meng, Ben Caine, Jiquan Ngiam, Daiyi Peng, Junyang Shen, Bo Wu, Yifeng Lu, Denny Zhou, Quoc V. Le, Alan Yuille, and Mingxing Tan. Deepfusion: Lidar-camera deep fusion for multi-modal 3d object detection, 2022. URL https://arxiv.org/abs/2203.08195.

[11] Tingting Liang, Hongwei Xie, Kaicheng Yu, Zhongyu Xia, Zhiwei Lin, Yongtao Wang, Tao Tang, Bing Wang, and Zhi Tang. Bevfusion: A simple and robust lidar-camera fusion framework, 2022. URL https://arxiv.org/abs/2205.13790.

[12] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han. Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation, 2022. URL https://arxiv.org/abs/2205.13542.

[13] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Qiao Yu, and Jifeng Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers, 2022. URL https://arxiv.org/abs/2203.17270.

[14] Hao Dong, Xianjing Zhang, Xuan Jiang, Jun Zhang, Jintao Xu, Rui Ai, Weihao Gu, Huimin Lu, Juho Kannala, and Xieyuanli Chen. Superfusion: Multilevel lidar-camera fusion for long-range hd map generation and prediction, 2022. URL https://arxiv.org/abs/2211.15656.

[15] Xinyu Zhang, Zhiwei Li, Xin Gao, Dafeng Jin, and Jun Li. Channel attention in lidar-camera fusion for lane line segmentation. *Pattern Recognition*, 118:108020, 2021. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2021.108020.

[16] Min Bai, Gellert Mattyus, Namdar Homayounfar, Shenlong Wang, Shrinidhi Kowshika Lakshmikanth, and Raquel Urtasun. Deep multi-sensor lane detection, 2019. URL https://arxiv.org/abs/1905.01555.

[17] Luca Caltagirone, Mauro Bellone, Lennart Svensson, and Mattias Wahde. Lidar-camera fusion for road detection using fully convolutional neural networks, 2018. URL https://arxiv.org/abs/1809.07941.

[18] Yueru Luo, Xu Yan, Chaoda Zheng, Chao Zheng, Shuqi Mei, Tang Kun, Shuguang Cui, and Zhen Li. M^2-3dlanenet: Multi-modal 3d lane detection, 2022. URL https://arxiv.org/abs/2209.05996.

[19] Sean Campbell, Niall O'Mahony, Lenka Krpalcova, Daniel Riordan, Joseph Walsh, Aidan Murphy, and Conor Ryan. Sensor technology in autonomous vehicles : A review. In *2018 29th Irish Signals and Systems Conference (ISSC)*, 2018. doi:10.1109/ISSC.2018.8585340.

[20] Richard B Langley, Peter JG Teunissen, and Oliver Montenbruck. Introduction to gnss. *Springer handbook of global navigation satellite systems*, pages 3–23, 2017.

[21] Andriy Burkov. *The hundred-page machine learning book*, volume 1. Andriy Burkov Quebec City, QC, Canada, 2019.

[22] Bernhard Mehlig. *Artificial neural networks*. University of Gothenburg. Department of Physics, 2019.

[23] Wikimedia Commons. Colored neural network, 2022. URL https://upload

.wikimedia.org/wikipedia/commons/4/46/Colored_neural_network.svg. Online; accessed 24-April-2023.

[24] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26, 2020. doi:https://doi.org/10.1007/s40745-020-00253-5.

[25] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018. URL https://arxiv.org/abs/1708.02002.

[26] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[27] Wikimedia Commons. Overfitting, 2022. URL https://upload.wikimedia.org/wikipedia/commons/1/19/Overfitting.svg. Online; accessed 29-April-2023.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

[29] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?, 2019. URL https://arxiv.org/abs/1805.11604.

[30] Anh H. Reynolds. Convolutional neural networks (cnns). Available at https://anhreynolds.com/blogs/cnn.html (accessed 2022/10/20).

[31] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017. URL https://arxiv.org/abs/1606.00915.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

[33] Muhammad Ahmed Ullah Khan, Danish Nazir, Alain Pagani, Hamam Mokayed, Marcus Liwicki, Didier Stricker, and Muhammad Zeshan Afzal. A comprehensive survey of depth completion approaches. *Sensors*, 22(18), 2022. ISSN 1424-8220. doi:10.3390/s22186969.

[34] Kitti vision benchmark. Depth completion evaluation, 2012. URL https://www.cvlibs.net/datasets/kitti/eval_depth.php?benchmark=depth_completion. Accessed: 2023-03-03.

[35] Max Kuhn and Kjell Johnson. *Measuring Performance in Regression Models*, pages 95–100. Springer New York, New York, NY, 2013. ISBN 978-1-4614-6849-3. doi:10.1007/978-1-4614-6849-3_5.

[36] Jason Ku, Ali Harakeh, and Steven L. Waslander. In defense of classical image

processing: Fast depth completion on the cpu, 2018. URL https://arxiv.org/abs/1802.00036.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

[38] Cody Reading, Ali Harakeh, Julia Chae, and Steven L. Waslander. Categorical depth distribution network for monocular 3d object detection, 2021. URL https://arxiv.org/abs/2103.01100.

[39] Yunlei Tang, Sebastian Dorn, and Chiragkumar Savani. Center3d: Center-based monocular 3d object detection with joint depth understanding, 2020. URL https://arxiv.org/abs/2005.13423.

[40] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017. URL https://arxiv.org/abs/1706.05587.

[41] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d, 2020. URL https://arxiv.org/abs/2008.05711.

[42] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds, 2019. URL https://arxiv.org/abs/1910.06528.

[43] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds, 2018. URL https://arxiv.org/abs/1812.05784.

[44] Xiangtai Li, Ansheng You, Zhen Zhu, Houlong Zhao, Maoke Yang, Kuiyuan Yang, and Yunhai Tong. Semantic flow for fast and accurate scene parsing, 2021. URL https://arxiv.org/abs/2002.10120.

[45] Zilong Huang, Yunchao Wei, Xinggang Wang, Wenyu Liu, Thomas S. Huang, and Humphrey Shi. Alignseg: Feature-aligned segmentation networks, 2021. URL https://arxiv.org/abs/2003.00872.

[46] Xiaopei Wu, Liang Peng, Honghui Yang, Liang Xie, Chenxi Huang, Chengqi Deng, Haifeng Liu, and Deng Cai. Sparse fuse dense: Towards high quality 3d detection with depth completion, 2022. URL https://arxiv.org/abs/2203.09780.

[47] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. URL https://arxiv.org/abs/1905.11946.

[48] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021. URL https://arxiv.org/abs/2010.04159.

[49] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2019. URL https://arxiv.org/abs/1709.01507.

[50] Li Chen, Chonghao Sima, Yang Li, Zehan Zheng, Jiajie Xu, Xiangwei Geng, Hongyang Li, Conghui He, Jianping Shi, Yu Qiao, and Junchi Yan. Persformer: 3d lane detection via perspective transformer and the openlane benchmark, 2022. URL https://arxiv.org/abs/2203.11089.

[51] Mina Alibeigi, William Ljungbergh, Adam Tonderski, Georg Hess, Adam Lilja, Carl Lindstrom, Daria Motorniuk, Junsheng Fu, Jenny Widahl, and Christoffer Petersson. Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving, 2023. URL https://arxiv.org/abs/2305.02008.

A

Appendix 1

A.1 Additional results

A.1.1 Annotation configurations

A.1.1.1 Loss

The resulting regression, classification and depth losses from training using the annotation configurations on 3 epochs can be seen in Figure A.1, Figure A.2 and Figure A.3 respectively.

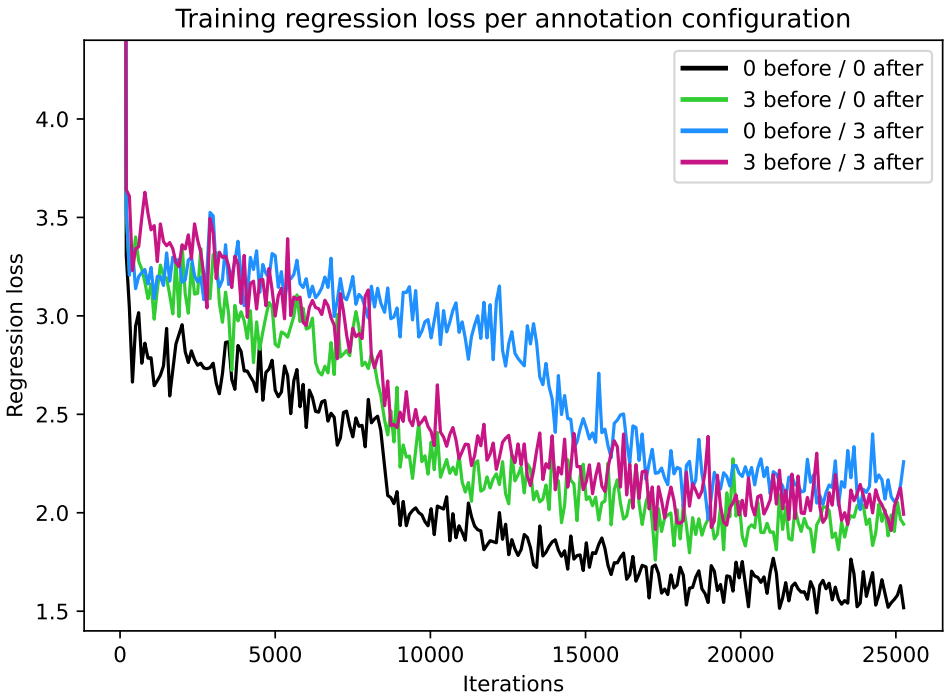


Figure A.1: Training regression loss for all annotation configurations trained for 3 epochs.

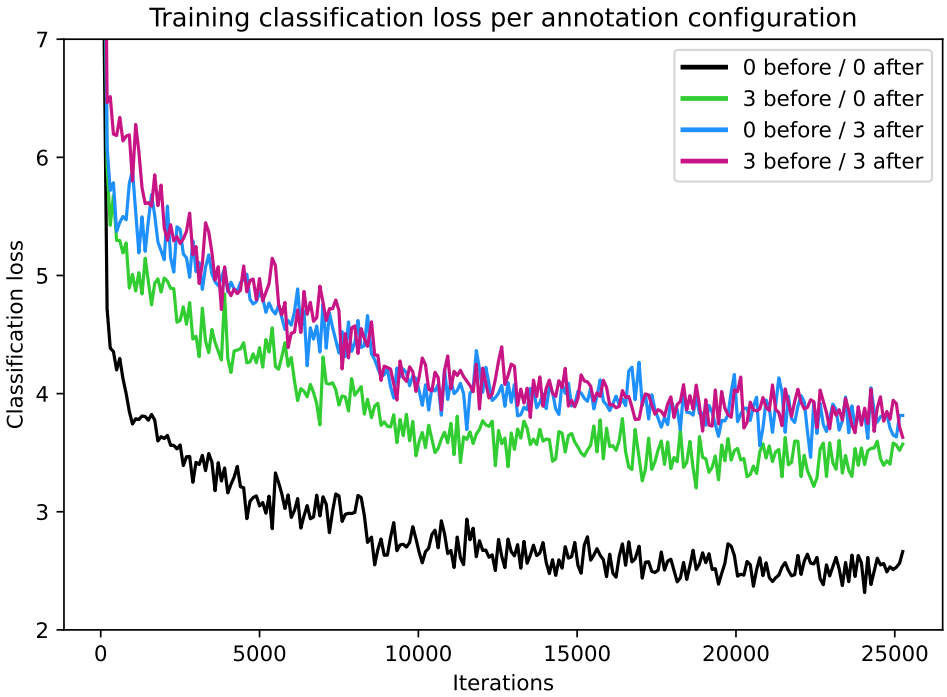


Figure A.2: Training classification loss for all annotation configurations trained for 3 epochs.

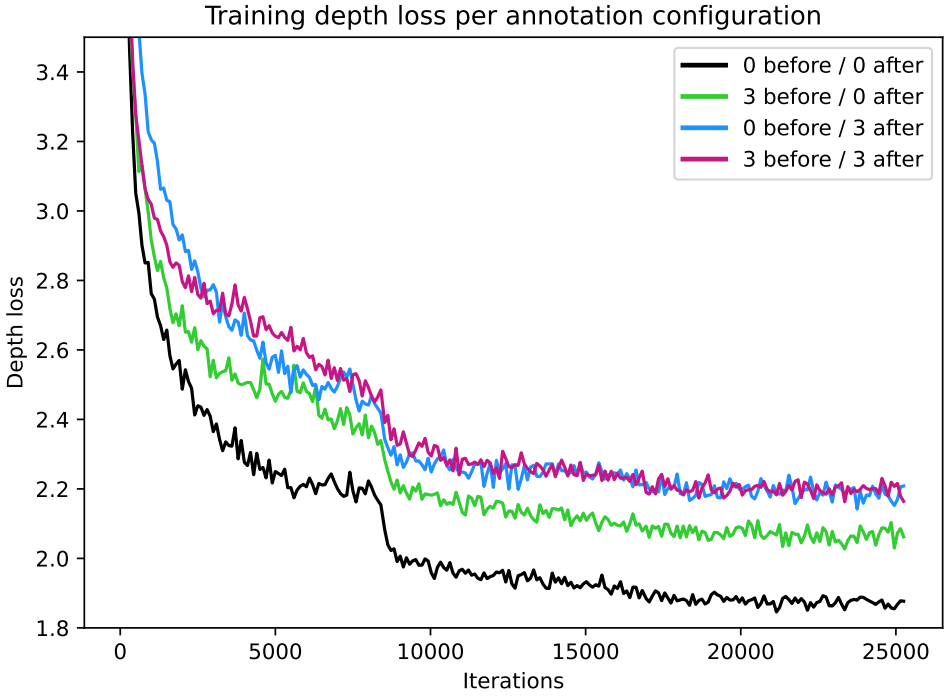


Figure A.3: Training depth loss for all annotation configurations trained for 3 epochs.

A.1.1.2 Threshold

The relationship between the F1-mean and threshold for configuration 1, configuration 2, configuration 3 and configuration 4 is visualized in Figure A.4, Figure A.5, Figure A.6 and Figure A.7 respectively, where the F1-mean is displayed for threshold values from 0.2 to 0.8.

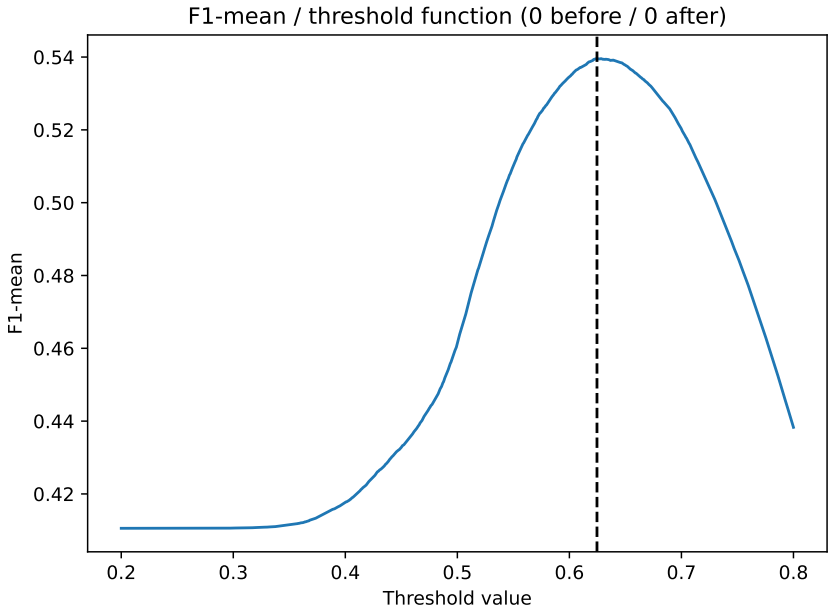


Figure A.4: F1-mean as a function of threshold for empty class for annotation configuration 1 (0 before / 0 after)

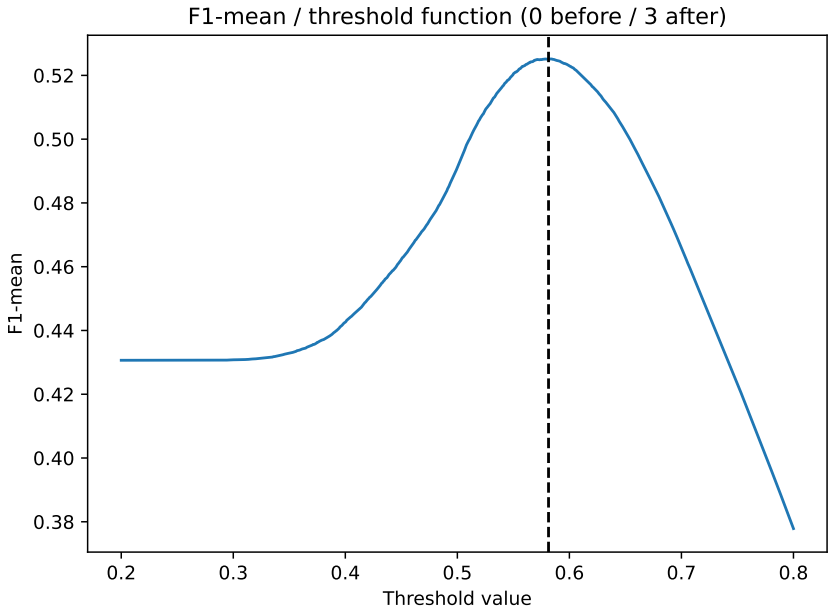


Figure A.5: F1-mean as a function of threshold for empty class for annotation configuration 2 (0 before / 3 after)

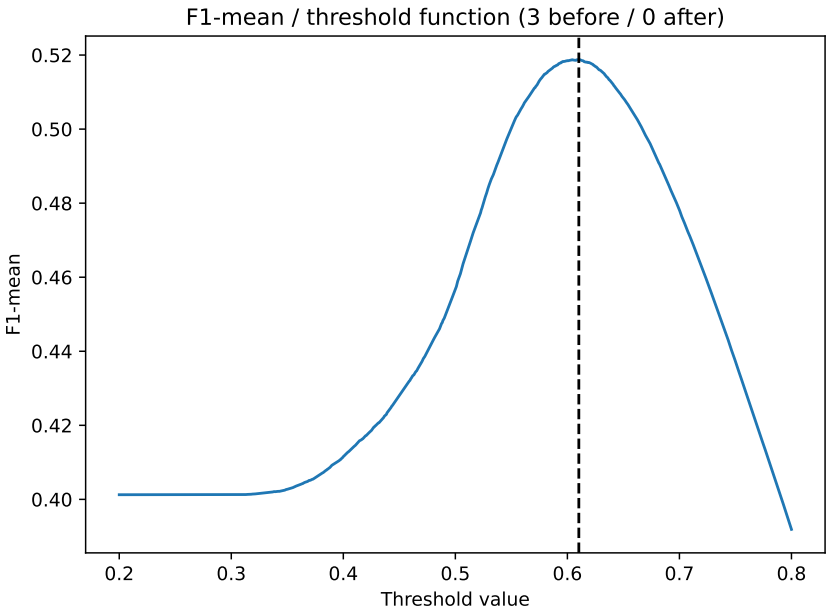


Figure A.6: F1-mean as a function of threshold for empty class for annotation configuration 3 (3 before / 0 after)

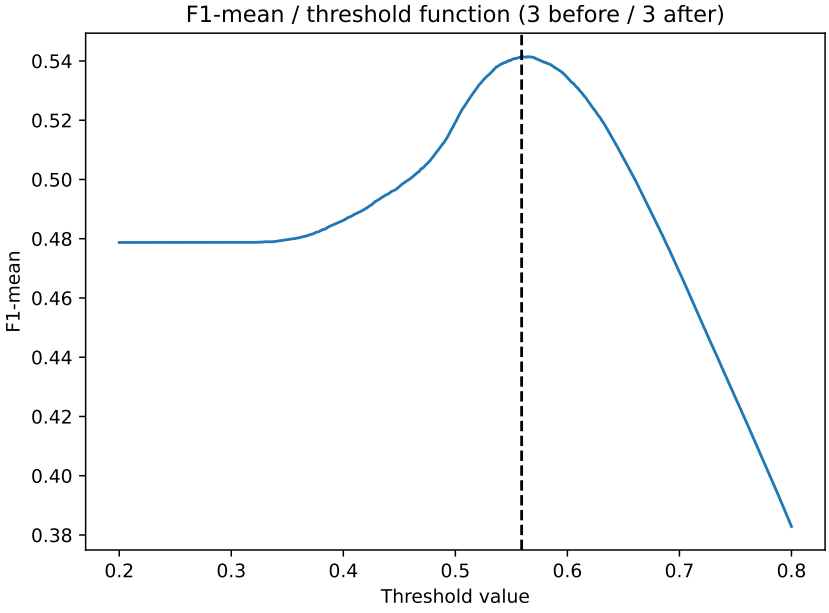


Figure A.7: F1-mean as a function of threshold for empty class for annotation configuration 4 (3 before / 3 after)

A.1.1.3 Evaluation metrics

Table A.1, Table A.2 and Table A.3 displays classification scores (accuracy, precision, recall and F1) per class and ranges 0-90m, 0-30m and 60-90m for models trained on annotation configuration 1, 2 and 3. The distribution of the absolute errors in the Y- and Z-direction for the ground-truth and prediction mask can be seen in Figure A.8 & Figure A.11, Figure A.9 & Figure A.12 and Figure A.10 & Figure A.13 for annotation configurations 1, 2 and 3.

Table A.1: Classification scores per class for ranges 0-90m, 0-30m and 60-90m and for annotation configuration 1 (0 before / 0 after).

	0-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.961	0.976	0.976	0.927	0.960
Precision	0.408	0.434	0.340	0.962	0.536
Recall	0.399	0.366	0.478	0.959	0.551
F1	0.404	0.397	0.397	0.961	0.540

	0-30m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.957	0.973	0.974	0.921	0.956
Precision	0.453	0.470	0.399	0.962	0.571
Recall	0.474	0.479	0.550	0.951	0.614
F1	0.463	0.475	0.462	0.957	0.589

	60-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.965	0.979	0.979	0.933	0.964
Precision	0.330	0.348	0.244	0.962	0.471
Recall	0.289	0.208	0.355	0.967	0.455
F1	0.308	0.260	0.289	0.965	0.456

Table A.2: Classification scores per class for ranges 0-90m, 0-30m and 60-90m and for annotation configuration 2 (0 before / 3 after).

	0-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.954	0.971	0.971	0.915	0.953
Precision	0.358	0.352	0.264	0.965	0.485
Recall	0.448	0.373	0.456	0.943	0.555
F1	0.398	0.362	0.334	0.954	0.512

	0-30m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.950	0.966	0.965	0.907	0.947
Precision	0.404	0.372	0.301	0.968	0.511
Recall	0.526	0.505	0.543	0.929	0.623
F1	0.457	0.428	0.387	0.948	0.555

	60-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.958	0.977	0.976	0.924	0.958
Precision	0.284	0.294	0.197	0.962	0.434
Recall	0.336	0.193	0.314	0.956	0.450
F1	0.308	0.233	0.242	0.959	0.436

Table A.3: Classification scores per class for ranges 0-90m, 0-30m and 60-90m and for annotation configuration 3 (3 before / 0 after).

	0-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.953	0.970	0.969	0.911	0.951
Precision	0.337	0.345	0.273	0.966	0.480
Recall	0.423	0.414	0.520	0.937	0.574
F1	0.375	0.377	0.358	0.951	0.515

	0-30m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.948	0.964	0.966	0.902	0.945
Precision	0.381	0.369	0.320	0.968	0.510
Recall	0.496	0.547	0.589	0.924	0.639
F1	0.431	0.440	0.414	0.945	0.558

	60-90m				
	SOLID	DASHED	OTHER	EMPTY	Mean
Accuracy	0.958	0.976	0.973	0.920	0.957
Precision	0.267	0.284	0.203	0.965	0.430
Recall	0.316	0.226	0.407	0.950	0.475
F1	0.290	0.252	0.271	0.957	0.443

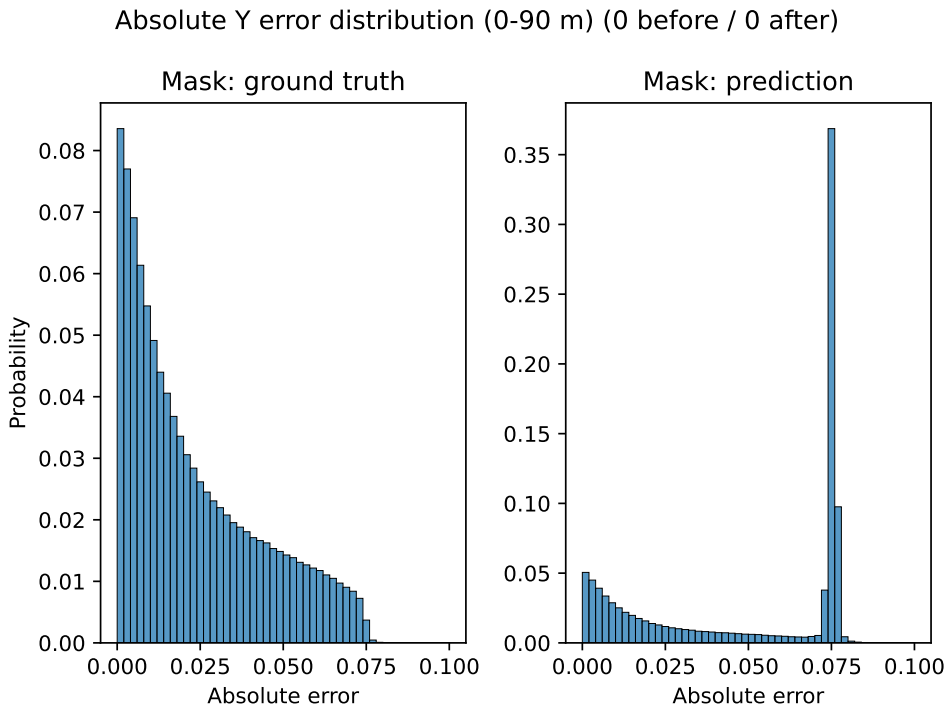


Figure A.8: Absolute Y error distribution for annotation configuration 1 (0 before / 0 after). The absolute error has a unit of meters.

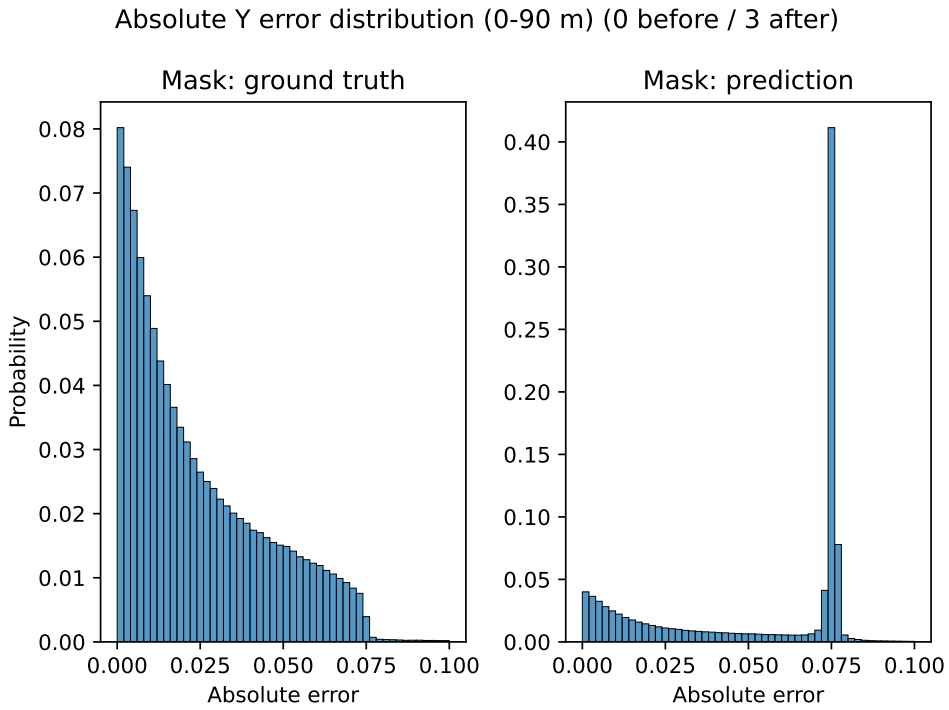


Figure A.9: Absolute Y error distribution for annotation configuration 2 (0 before / 3 after). The absolute error has a unit of meters.

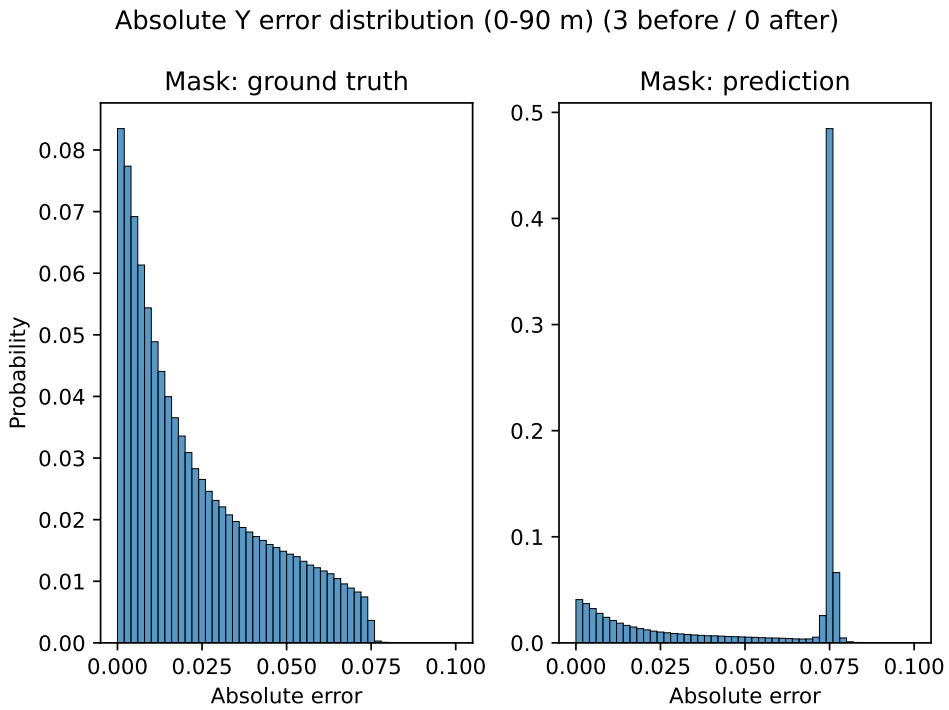


Figure A.10: Absolute Y error distribution for annotation configuration 3 (3 before / 0 after). The absolute error has a unit of meters.

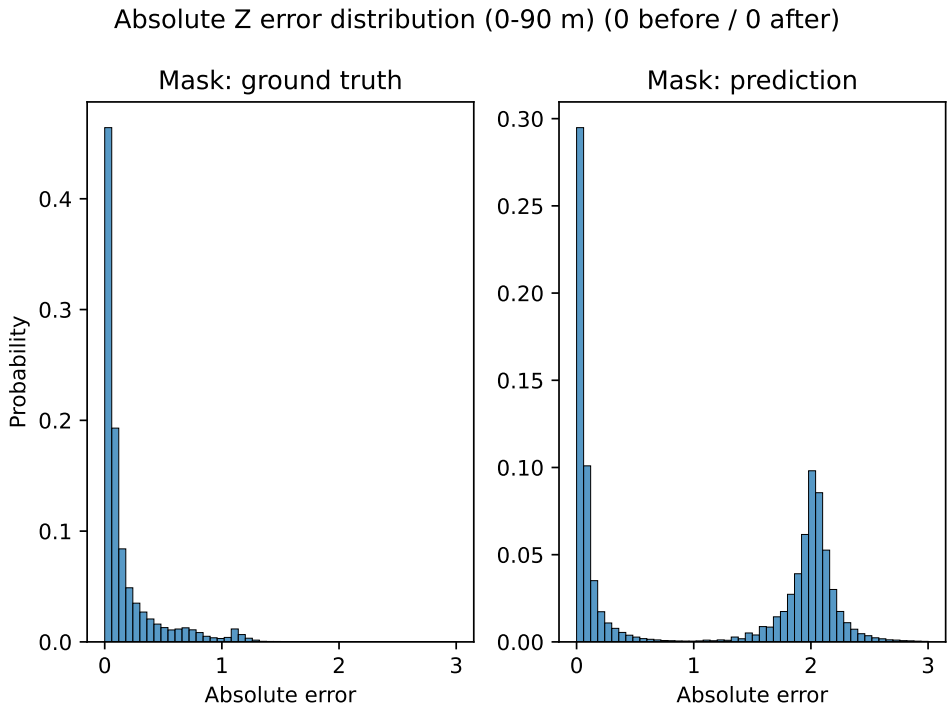


Figure A.11: Absolute Z error distribution for annotation configuration 1 (0 before / 0 after). The absolute error has a unit of meters.

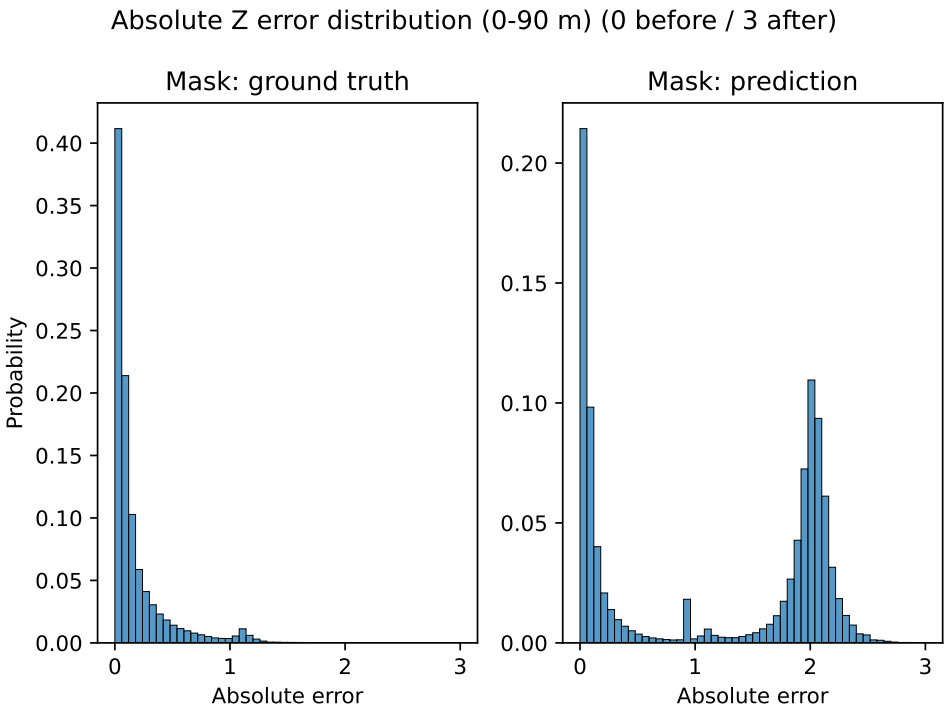


Figure A.12: Absolute Z error distribution for annotation configuration 1 (0 before / 3 after). The absolute error has a unit of meters.

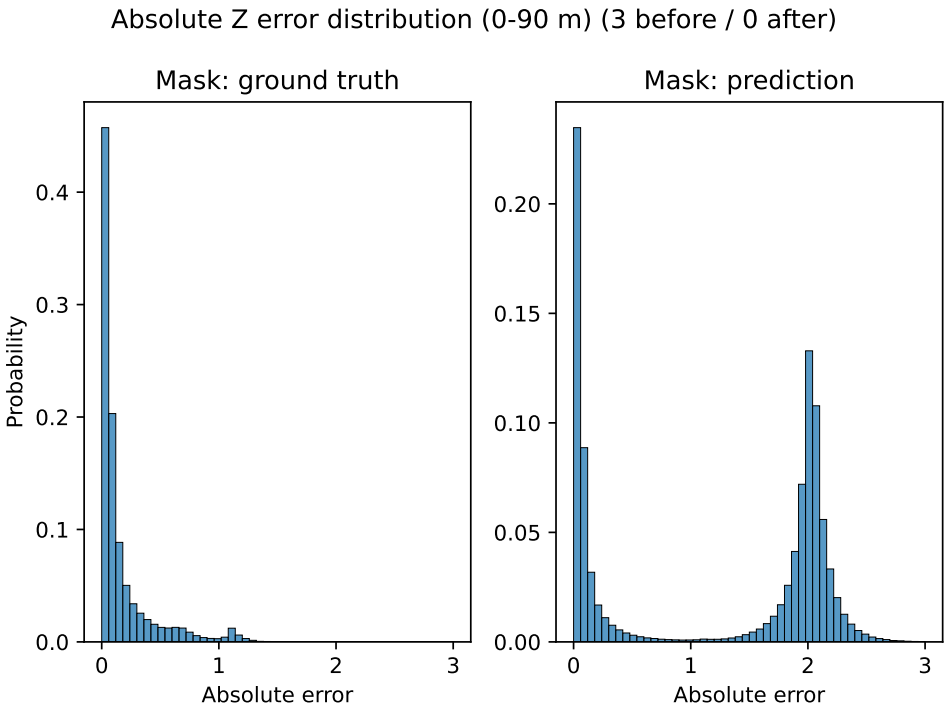


Figure A.13: Absolute Z error distribution for annotation configuration 1 (3 before / 0 after). The absolute error has a unit of meters.

A.1.2 Further training on configuration 4 (3 before / 3 after)

A.1.2.1 Loss

The resulting regression, classification and depth losses from training the model on annotation configuration 4 (3 before / 3 after) for 14 epochs can be seen in Figure A.14, Figure A.15 and Figure A.16 respectively.

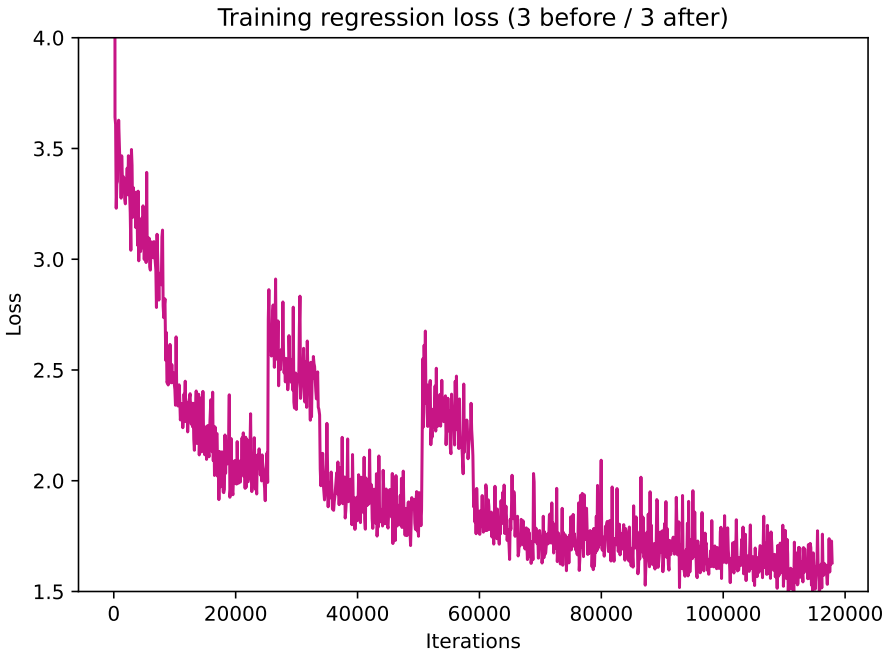


Figure A.14: Training regression loss for annotation configuration 4 (3 before / 3 after) trained for 14 epochs.

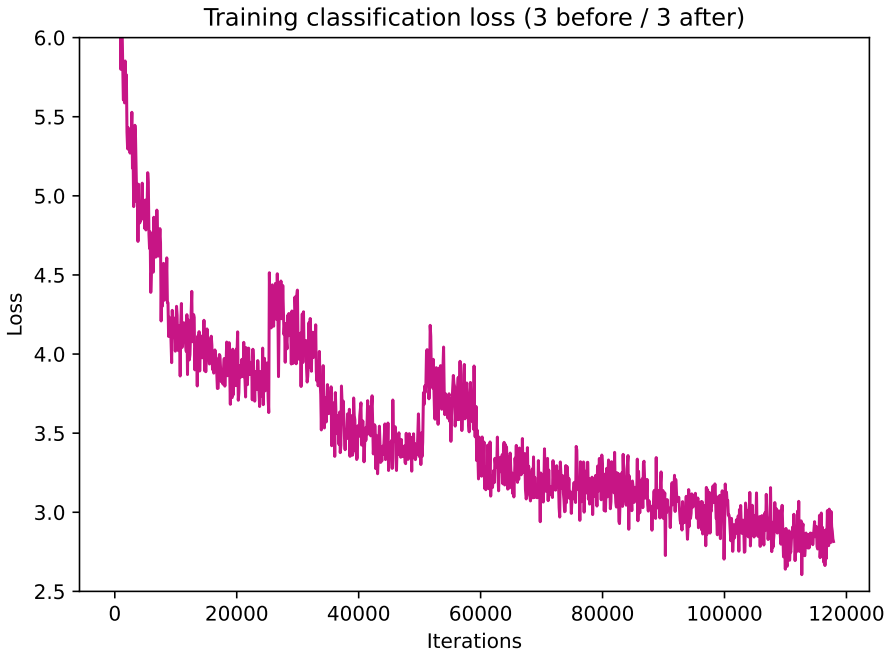


Figure A.15: Training classification loss for annotation configuration 4 (3 before / 3 after) trained for 14 epochs.

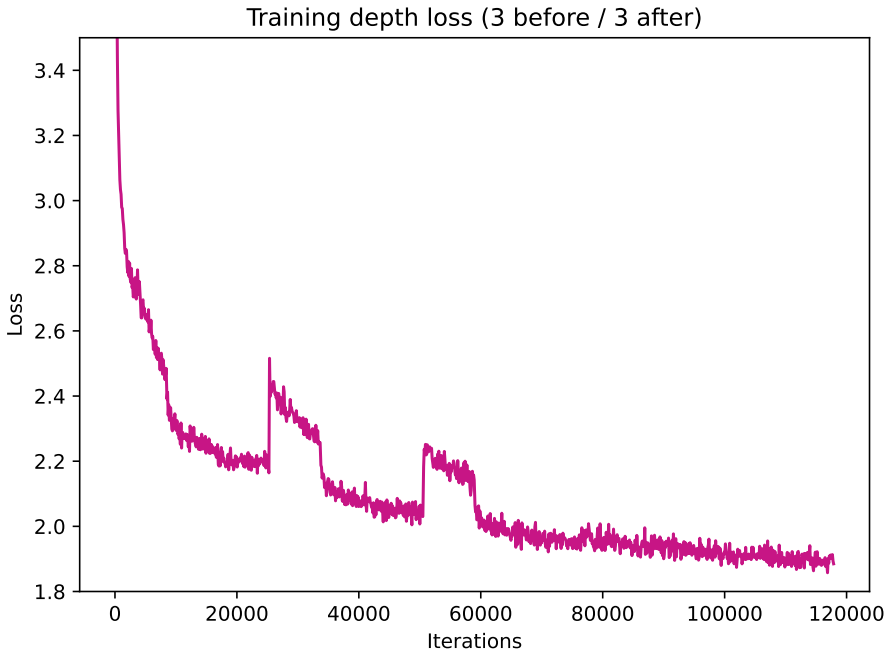


Figure A.16: Training depth loss for annotation configuration 4 (3 before / 3 after) trained for 14 epochs.

A.1.2.2 Threshold

The relationship between the F1-mean and threshold for configuration 4 trained for 14 epochs is visualized in Figure A.17, where the F1-mean is displayed for threshold values from 0.2 to 0.8.

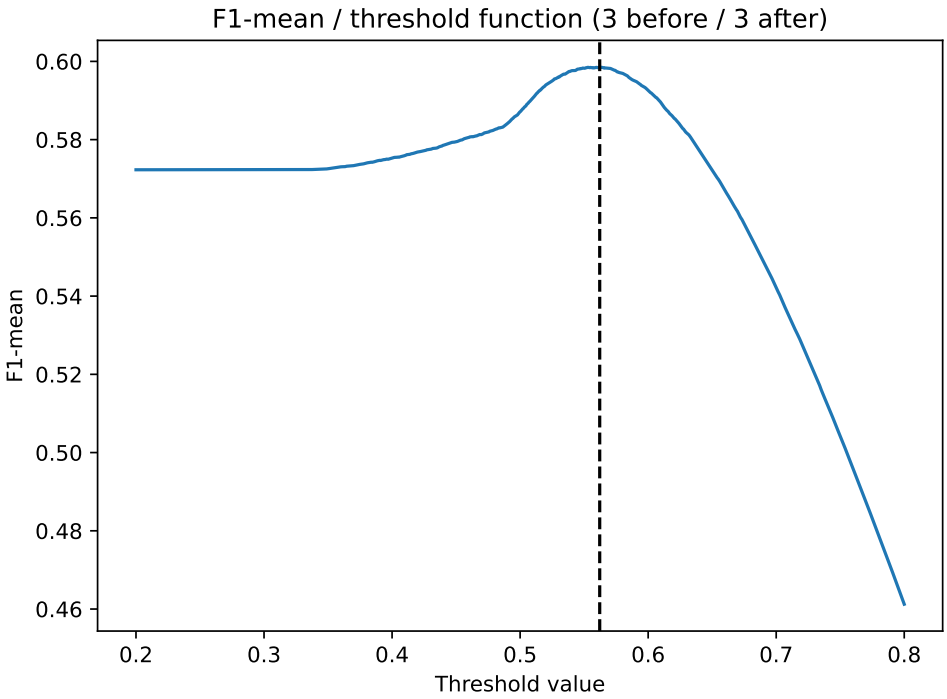


Figure A.17: F1-mean as a function of threshold for empty class for annotation configuration 4 (3 before / 3 after) (trained for 14 epochs).

A.1.3 Past and future inference to improve lane detection

The effect of using past and future grids can be seen in Table A.4, where the RSME and MAE in the Y and Z-directions is displayed. In Table A.4 the regression scores are calculated over all grid points with a lane according to ground truth (ground truth mask).

Table A.4: Change in classification and regression scores when combining inference from current frame with past and future frames.

	Mask: ground truth			
Number of past/future frames	RSME in Y	RSME in Z	MAE in Y	MAE in Z
5 past / 5 future	0.031	0.337	0.023	0.213
4 past / 4 future	0.031	0.342	0.024	0.215
3 past / 3 future	0.031	0.345	0.024	0.215
2 past / 2 future	0.031	0.347	0.024	0.215
1 past / 1 future	0.032	0.350	0.024	0.217
5 past / 0 future	0.033	0.364	0.025	0.234
4 past / 0 future	0.032	0.360	0.025	0.230
3 past / 0 future	0.034	0.357	0.027	0.226
2 past / 0 future	0.034	0.354	0.026	0.221
1 past / 0 future	0.036	0.350	0.029	0.219
0 past / 5 future	0.032	0.339	0.025	0.208
0 past / 4 future	0.032	0.346	0.025	0.211
0 past / 3 future	0.033	0.350	0.026	0.213
0 past / 2 future	0.034	0.350	0.027	0.214
0 past / 1 future	0.036	0.354	0.029	0.216
0 past / 0 future	0.029	0.355	0.021	0.219

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY