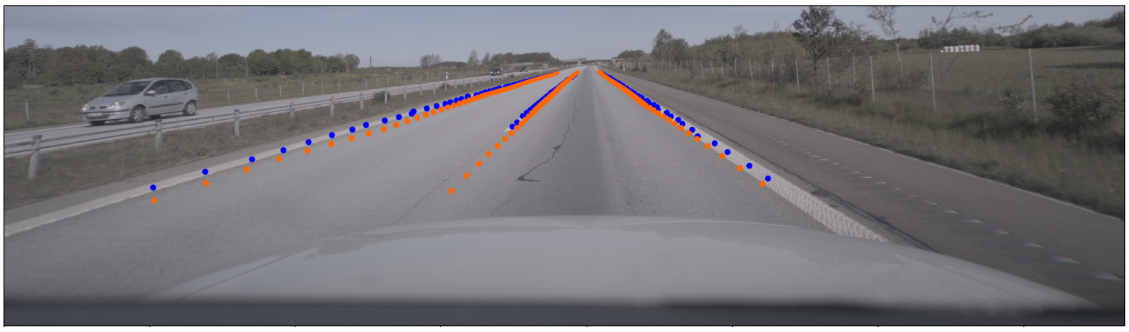


Semi-supervised 3D Lane Detection through Spatio-Temporal Consistency Learned from Videos

Master's thesis in Complex Adaptive Systems

Erik Brorsson & Silas Ulander

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Semi-supervised 3D Lane Detection through Spatio-Temporal Consistency Learned from Videos

ERIK BRORSSON & SILAS ULANDER

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Computer Vision Group
Chalmers University of Technology
Gothenburg, Sweden 2021

Semi-supervised Learning of 3D Lane Detection through Spatio-Temporal Consistency Learned from Videos
ERIK BRORSSON
SILAS ULANDER

Supervisor: Mina Alibeigi, Zenseact
Examiner: Fredrik Kahl, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An image from the test set where ground truth lanes are shown in blue and the predicted lanes are shown in orange.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Semi-supervised Learning of 3D Lane Detection through Spatio-Temporal Consistency Learned from Videos
ERIK BRORSSON
SILAS ULANDER
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The task of lane detection has lately been dominated by deep learning approaches for monocular vision which aim to predict the lanes in the image plane. These have recently been extended to deep learning models that instead predict 3D lanes directly, and are trained end-to-end for this task. These models overcome some of the shortcomings of previous methods, including the difficulties involved in lane model fitting and image-to-world correspondence. However, the 3D lane detection models need to be trained on 3D lanes data, which is difficult and costly to create. In this thesis project, we adopted a semi-automatic approach for creating a labeled 3D lanes dataset by combining manually annotated images with depth maps from aggregated LiDAR point clouds. Though semi-automatic, this method still entails manual labor of annotating the lanes in the images. In order to mitigate the need for annotating large datasets, our work investigates the possibility of training a 3D lane detection model on unlabeled data. We propose a novel spatio-temporal consistency loss together with a semi-supervised training scheme that allows for training the model both on available labeled and unlabeled data. In the conducted experiments, the size of the labeled dataset was varied between 512 and 3194 images while the unlabeled dataset always consisted of 5050 images. Our results show that the proposed method for leveraging unlabeled data increases the performance of the model when the available labeled dataset is small, and thus proves the feasibility of the approach. In particular, when training semi-supervised on 512 labeled and 5050 unlabeled images instead of training only on the 512 labeled images in a fully supervised manner, the average lateral error of the predictions in the far range (40-100 meters) decreases from 50.9 to 39.7 cm and the F-Score and Average Precision increase from 0.864 to 0.881 and from 0.924 to 0.950 respectively. However, little or no improvements are observed when the size of the labeled dataset is increased and therefore further research is needed if this method should replace substantial amounts of labeled data. We also generated synthetic data with the open-source simulator CARLA and used 1609 of these images to pre-train the model in an attempt to increase performance on real-world data. However, no significant improvements were observed, which was probably a result of both poor generalizability from synthetic to real world images as well as difficulties involved in creating such a dataset with CARLA since the simulator does not have support for lane instance annotations.

Keywords: 3D Lane Detection, Semi-automatic 3D Lane Annotations, Machine Learning, Deep Learning, Semi-supervised Learning, Spatio-temporal Consistency, Autonomous Vehicles.

Acknowledgements

We would like to thank both our supervisor Mina Alibeigi and Benjamin Waubert at Zenseact, not only for helping us with technical details regarding data collection and the model implementation, but also for many rewarding discussions, which have enabled the findings of this thesis.
We would also like to thank our examiner at Chalmers, Fredrik Kahl.

Erik Brorsson & Silas Ulander
Gothenburg, August 2021

Contents

List of Figures

List of Tables

1 Introduction

This chapter includes the background and aim of this thesis project as well as the relevant delimitations.

1.1 Background

In recent years, the dream of manufacturing fully autonomous vehicles has become less of a fiction and instead something many believe to be achievable. Much of the progress made within the field of autonomous driving can be attributed to the advances made in the regime of *machine learning*, which have resulted in efficient solutions to many fundamental image processing tasks. This has in turn lead to the development of increasingly complex perception systems which is viewed as a necessity for enabling autonomous vehicles to safely navigate the roads, including following traffic rules and showing consideration for other road users.

The development of autonomous vehicles is motivated by the fact that the critical reason for traffic accidents is assigned to the driver in 94($\pm$2.2)% of all cases [1]. Traffic accidents cause many deaths and injuries every year and are the leading cause of death for people aged between 5 and 29 years. The number of fatal accidents sums up to over 1.3 million and an additional 20-50 million people suffer non-fatal injuries, often resulting in long-term disabilities [2].

One of the most fundamental tasks for autonomous vehicles is to detect the lanes of the road. As described by [3], lane detection is involved in features such as Lane Departure Warning (LDW), Adaptive Cruise Control (ACC), Lane Change Assist (LCA) and is also a necessity for fully autonomous vehicles. Each of these features has different demands on the performance and reliability of the lane detection algorithm. For instance, in a LDW system, where a warning should be issued whenever the vehicle is close to exiting the ego-vehicle lane, it is only necessary to detect the ego-vehicle lane for a short distance ahead and neighboring lanes can be ignored completely. On the other hand, both ego-vehicle and neighboring lanes need to be handled in a LCA system and the detection range needs to be much larger. For a fully autonomous car it is also essential that the system can manage non-linear lane topologies such as merges and splits.

There exist two main approaches to lane detection. Firstly, the vehicle's position relative to the lanes can be computed by using maps created offline and monitoring the vehicle's position within the map. Secondly, the use of an on-board perception system can allow for detection of lanes during run time without relying on any such map. The offline approach to lane detection can be accomplished for example by combining GPS data, Inertial Measurement Unit (IMU) and high-resolution aerial images. This option has been explored extensively in the DARPA Grand Challenge [4] and Urban Challenge [5], where several vehicles successfully navigated the roads using only rudimentary on-board perception systems. However, the reliability and accuracy of this approach is not satisfactory according to [3]. It is also difficult and expensive to create the maps and keep them up to date. Furthermore, a perception system is in any case needed for fully autonomous vehicles, which among other things also need to take other road users into consideration. Most research towards lane detection has therefore aimed at utilizing the on-vehicle perception systems.

There also exists a number of perception modalities that have been extensively used for perception systems in vehicles. As described by [3], some of the most common ones are monocular vision (i.e. a single camera), LiDAR (Light Detection And Ranging), stereo vision (two cameras enabling perception of depth), radar, Geographic information systems (GIS), GPS, and inertial measurement unit (IMU). Each of these sensors has their pros and cons, both regarding price, reliability and the ability of detecting different objects and perceiving the road surface. Mainly due to the accessibility and low price of cameras, as well as the success of deep convolutional neural networks on tasks such as image classification, segmentation and object detection following their introduction by [6], the methods based on monocular cameras have achieved most attention in recent years. Methods based on monocular cameras have also had great success in the task of lane detection and is the most common perception-based solution according to [7].

Current lane detection methods relying on images from a monocular camera usually treat the problem as a 2D detection problem where the lanes are detected in the images. The lane features are usually extracted with deep convolutional neural networks. One popular approach is using a semantic segmentation network to find the pixels that correspond to a lane and then using some method of lane model fitting to create lane instances from the detected pixels [8], [9]. When lane detections have been made in the image, the lanes can be projected to 3D coordinates under the assumption of a flat ground. This method has the obvious drawback that it gives inaccurate estimates of the 3D position of the lanes when the road is hilly, even if lanes were correctly detected in the image.

Therefore, data driven methods that predict 3D lanes from images without using any flat ground assumption or constraints on the lane geometry have been investigated by for example [7], [10], [11]. Garnett *et al.* [7] published their model 3D-LaneNet in 2019 and [10], [11] are essentially follow-up papers on 3D-LaneNet, which marked the first step towards data driven 3D lane detection. These methods are all based on deep convolutional neural networks that are used to extract the lane features of the image and make predictions of the 3D lane geometry in a unified model. However, since these models aim to predict 3D lanes directly, rather than 2D lanes, the data needed to train these models consists of 3D lane ground truths.

Such data is expensive to create as it entails using complex multi-sensor setups such as IMU, LiDAR and camera, which is used by [7], and possibly also highly accurate maps which is used by [12]. Therefore, there is an interest in requiring as little labeled data as possible. In fact, both the models 3D-LaneNet [7] and Gen-LaneNet [10] were developed using only synthetic data. Although [7] also generated their own real-world dataset, they only used this for validating their approach, and preferred using the synthetic data for development since the ground truths of the real-world dataset were flawed in some aspects. Furthermore, they did not investigate whether the synthetic data could be used to increase the performance of the model on real-world data. However, [13] do investigate the possibility of leveraging synthetic data for the task of 2D lanes detection, and show that training on synthetic data indeed resulted in increased performance on their real-world dataset. Since synthetic data is much easier to collect than real-world data, they also request further research on this topic for 3D lanes detection. This was investigated by [11] and they found that training on synthetic data could increase the performance also of 3D lane detectors on real-world data. However, they still use a large labeled real-world dataset of around 300 000 images and do not establish how much labeled real-world data can be replaced by synthetic data (while maintaining good performance). Large labeled real-world datasets are therefore probably still required for any real-world application, although the improvements made from using synthetic data indicates that synthetic data may be used to replace some of the expensive real-world data.

Another way of replacing expensive labeled data is of course leveraging unsupervised learning. Works like [14]–[19] show the feasibility of unsupervised or semi-supervised learning for tasks such as image classification, object detection and semantic segmentation, which traditionally require vast amounts of labeled data. To the best of our knowledge, no approaches for unsupervised learning of 3D lane detection models have yet been proposed. Since 3D lanes are arguably even more expensive/difficult to annotate than for example the data used for image classification and 2D object detection, methods for unsupervised learning for 3D lanes would be of great value.

Although some methods for creating 3D lanes datasets exist, including the one used by [7] and the automatic approach introduces by [12], it is difficult to get the desired precision in the ground truths as well as acquiring a dataset that is diverse enough. Even with the automatic approach it is expensive and difficult to create a dataset that covers all possible scenarios of driving, e.g. different road topologies, weather conditions, lighting and so on. Since a large and diverse dataset is essential for reaching high performance with any deep learning model the value of using unsupervised learning may not only be to decrease the need for labeled data, but rather also improve the performance of the model. Any efforts in improving the lane detection models are well worth while since this enables more safe and reliable advanced driver-assistance systems as well as autonomous driving capabilities of future vehicles.

1.2 Aim

The recent papers [7], [10], [11] have shown the feasibility of data-driven end-to-end models for 3D lane detection. However, the difficulty of creating real-world 3D lanes datasets impairs both research and possible real-world application of these models. In an attempt of mitigating the need for large labeled 3D lanes dataset and possibly also improving the performance of the models, the aim of this thesis is to develop a method for unsupervised learning, as well as generating a synthetic dataset, suitable for training a 3D lane detector when the amount of labeled real-world data is limited.

Open source implementations are available for both 3D-LaneNet [7] and Gen-LaneNet [10], which make them suitable for further investigation. However, Gen-LaneNet utilizes a two-stage architecture where the first part of the network constitutes semantic segmentation of the image and the second part takes the semantic image as input and predicts 3D lanes only based on this. This makes Gen-LaneNet less suitable for a real-world application, since the information given by a semantic image is obviously quite limited. Because of this, it seems likely that 3D-LaneNet, which is truly trained end-to-end on the input images, is the best candidate for scalable 3D lane detection. Motivated by this, the following research question is formulated:

Can training on unlabeled data improve the performance of the 3D lane detection model 3D-LaneNet?

Since the benefit of training on unlabeled data can be assumed to decrease with increasing amounts of available labeled data a second research question is formulated as follows:

How does the (potential) performance gain from training on unlabeled data depend on the size of the labeled dataset?

Furthermore, to investigate whether synthetic data also can be used to improve the performance of 3D-LaneNet on real-world data, and thereby mitigate the need for large labeled real-world datasets further, the final research question is formulated as:

Can labeled synthetic data be used to improve the performance of 3D-LaneNet on real-world data?

1.3 Delimitations

One delimitation we make is to only use a single image from a monocular camera to predict the 3D lanes during inference. Using multiple cameras in a stereo vision setup or combining the image data with a LiDAR could be a way to increase the performance of the model, but any such approaches will not be considered in this thesis. Furthermore, tracking the lanes over several video frames and applying a method of temporal aggregation could also lead to increased performance, but since we only consider one image at a time this will not be investigated either.

A critical requirement of a fully autonomous car is the ability to handle all possible traffic situations, e.g. highways, multiple lanes, splits, merges, crossings and roundabouts. Moreover, different types of road conditions such as weather and different times of the day must also be managed. In this project however, the used datasets can not be expected to contain all possible scenarios and the performance of the implemented lane detection model is therefore not expected to meet the requirements of real-world applications. Furthermore, since the main goal of this thesis is to develop a method for unsupervised learning of 3D lanes, rather than optimizing the model's performance for a real-world application, we will not focus on collecting a dataset that is as diverse as possible, but rather a dataset that can be used to prove the feasibility of the proposed semi-supervised approach. Much work will be left before the implemented model can be put to the test in an autonomous vehicle.

The computational efficiency is also of great importance for any algorithms related to the perception system of autonomous vehicles. Since the vehicles need to perceive their surroundings in real time the algorithm for lane detection needs to be fast. This aspect is not investigated in this thesis and the main goal is to improve detection accuracy without any concern for the computational efficiency. In any case, the work done in this thesis only modifies the training process and therefore does not affect the run time during inference of the chosen model 3D-LaneNet.

2
Theory

This chapter includes the relevant theoretical framework used in the thesis. Since we aim to train the deep learning model 3D-LaneNet, which casts the lane detection problem as an object detection one, on unlabeled data, this chapter is divided into four main sections: deep learning, object detection with deep learning, unsupervised learning, and related work.

2.1 Deep Learning

This section aims to give the reader an introduction to deep learning and describe the fundamental building blocks and training procedure of artificial neural networks.

2.1.1 Introduction to Deep Learning

Artificial neural networks (ANNs) are computing systems with one or several layers of artificial neurons. Networks equipped with multiple layers between the input and output layers are called *deep neural networks.* The layers are usually ordered sending information from the first layer l_1 to the next layer l_2 and so on until reaching the last layer l_n, also know as the *output layer.* These forward passing networks are known as *feedforward neural networks.* Between each layer there exist a number of connections linking the neurons from layer l_i to layer l_{i+1}. These connections are associated with a weight and bias that scales and shifts the output of the previous neuron. Each neuron is equipped with what is called an activation function. The collection of these neurons and the weighted connections between them create trainable systems that have shown great potential for tasks such as image classification, image segmentation and object detection. The end goal when training a neural network is to make the model correctly map an input X to the corresponding output y. For example models trained to classify handwritten digits, e.g. correctly assigning the label 7 to a given input image of a handwritten 7. The design of the network architecture will depend on input and output as well as the desired properties of the model. For instance optimizing the balance between precision and recall. The topology of models can vary by e.g. changing the number of layers, changing the number of neurons in the layers and/or using different activation functions in different layers.

An artificial neuron receives the output from all n connected neurons in the previous layer as input. The inputs $\boldsymbol{x} = \{x_i\}_{i=1}^{n}$ are first scaled by the weights $\boldsymbol{w} = \{w_i\}_{i=1}^{n}$ associated with the corresponding connections, then summed up and added to the next neurons bias b. Finally, the neurons *activation function*, denoted as φ in this thesis, is applied and the resulting value is the output of the neuron. This process is visualized in Figure 2.1.

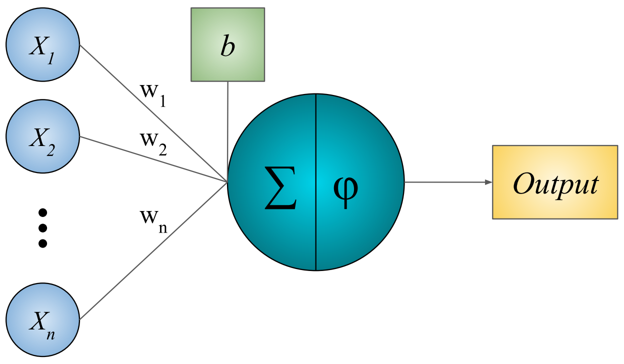


Figure 2.1: Illustration of an artificial neuron. Here $\{x_i\}_{i=1}^{n}$ are the inputs to the neuron, $\{w_i\}_{i=1}^{n}$ are the weights associated with each corresponding input and b is the neurons bias. The neuron's input is weighted and summed up, added to the neurons bias and passed through the activation function φ. The resulting value is the output of the neuron.

If the network is to solve nontrivial problems, the model must contain nonlinear activation functions. Two commonly used activation functions are the *ReLU*(Rectified Linear Unit) and the *Sigmoid* function [20]. The ReLU function is a piecewise linear function and is defined as $\max(0, \alpha)$, where α is the input, meaning that the function will output α if $\alpha > 0$ and zero otherwise. This makes the activation function computationally cheap. It also help to prevent issues such as *The vanishing gradient problem*, making the function well suited for hidden layers in larger neural networks. The *Sigmoid* function involves exponentials for both the the function and it's derivative, making it less computationally efficient. However, a common application is to use it in the final layer to force the output of the network to lie between 0 and 1. The output of a network can hence be interpreted as a probability and is used for tasks such as image classification and segmentation. The shape of these two functions can be seen in Figure 2.2 together with their derivatives, which are used when training the network.

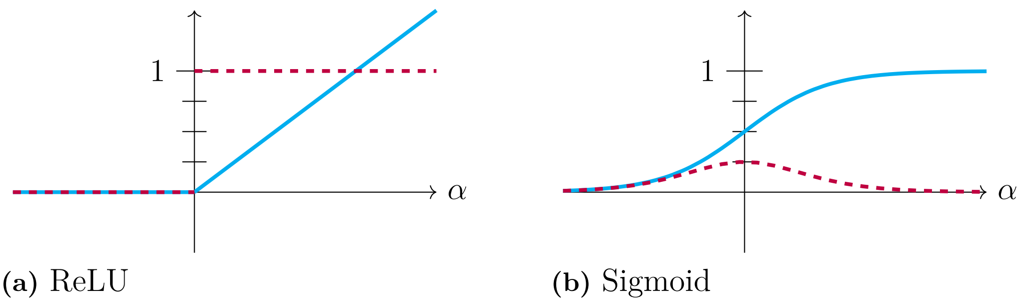


Figure 2.2: A schematic illustration of two activation functions $\varphi(\alpha)$ shown as solid cyan lines and their derivative $\varphi'(\alpha)$ shown as dashed purple lines. Figure 2.2a illustrates the ReLU function and Figure 2.2b illustrates the sigmoid function.

2.1.2 Training Neural Networks

The training of a neural network revolves around updating the weights and biases of the model. The most common form of training is referred to as *supervised learning*, where the model receives a training set containing input data together with their corresponding labels. During training, the model is fed the input data and tries to classify the data according to the corresponding label. The model is penalized based on the difference between the prediction and the true label (also known as the ground truth) of each training example. The magnitude of the penalty is determined by the *loss function*. Hence the training of the network entails minimizing the total error in the loss function. Even though the individual parts of a neural network often are convex, the composition of them is often non-convex. Optimization of a non-convex problem is known to be hard (*NP-hard*) to solve [21]. The optimal weights for a neural network is therefore approximated, often by the use of *Stochastic gradient descent*(SGD) and *Backpropagation* [21].

2.1.2.1 Loss Functions

One of the most important parts of the training of neural networks is choice of the loss function, since it determines the magnitude of inaccuracy of the predictions when compared to the ground truths. There exist many different categories of loss functions, designed for different prediction task. A few examples are:

- Regression.
- Binary classification.
- Multi-class classification.

For this thesis there are two main loss functions utilized; a binary classification loss called *cross entropy* and a regression loss called l^1-*norm* loss.

Cross entropy The cross entropy is a measurement used when comparing two probability distributions. When used in machine learning, cross entropy can be used as a signal when comparing the true probability distribution with the predicted one made by the model. For a set of predictions $\{\boldsymbol{y}\}_{j=1}^{N}$ and the corresponding true labels $\{\hat{\boldsymbol{y}}\}_{j=1}^{N}$, a cross entropy loss function can be formulated as:

$$Loss_{\mathbf{CE}} = -\sum_{i=1}^{N} \left[\hat{y}_i \log(y_i) + (1-\hat{y}_i)\log(1-y_i)\right]. \tag{2.1}$$

$\boldsymbol{l^1}$ The l^1-norm is the sum of the absolute distance in each dimension between two points. Therefore it can be used to penalize networks predictions of objects in 3D-space by comparing the predicted location to the true position. Formulated as a loss function the penalty becomes:

$$Loss_{l^1} = \sum_{i=1}^{N} |y_i - \hat{y}_i|, \tag{2.2}$$

where N is the dimension in which the object lies and y_i $(\hat{y}_i)$ is the coordinates of the predictions (ground truths).

2.1.2.2 Optimization

When training neural networks, the aim is to find the optimal values $\Theta_{\boldsymbol{opt}}$ of the networks weights, such that given an input training set X to the model, the loss function is minimized. *Gradient descent* is a first-order iterative optimization algorithm used for finding local minimum for a differentiable function such as neural networks equipped with a loss function. For each iteration of the algorithm the weights of the network are updated by following the opposite direction of the gradient, since this is the direction of the steepest descent [22].

The update of the networks weights are done by calculating gradient of the loss function $\mathcal{L}$ with respect to the networks weights Θ. The magnitude of this update is scaled by what is known as the *learning rate* η. In each iteration of the gradient descent method Θ is updated according to:

$$\Theta \longrightarrow \Theta - \eta \frac{\partial \mathcal{L}}{\partial \Theta}. \tag{2.3}$$

The calculation of the gradient of feed forward neural networks is done via *Back-propagation*. The algorithm is based on the chain rule and calculates the gradient of one layer at a time propagating backwards through the network [23].

There exist many different types of variations of the gradient descent algorithm, but a commonly used update rule is *Adam* (Adaptive Moment Estimation) [24].

2.1.2.3 Train/Validation/Test split and Overfitting

The training of neural networks is usually done by repeatedly presenting the training data to the model, decreasing the error in the loss function every iteration. Doing this will make the model increasingly adept at predicting the correct outputs for the given dataset. However, if this process is repeated too many times a problem known as *Overfitting* can occur. A model is considered overfitted if the model performs well on examples within the training dataset but performs considerably worse on examples outside the training set. This means that the model has only learned what to predict for the training set but is not able to generalize to new data. To remedy this problem the labeled data can be divided into a Train/Validation/Test split. The model is then presented with data from the training set and for each iteration trained to minimize the loss function with respect to this data. This is what is referred to as the training of a neural network. For each iteration, the performance is also measured on the validation set. Although the validation set is repeatedly presented to the model during training, the model's weights are not updated as to minimize the loss function for the validation set. Instead, this is only done for the training set. Therefore, the model is not trained explicitly to perform well on the validation set, but if it generalizes well to new data it should of course exhibit good performance also on this dataset if it performs well on the training set. To make sure that the model is able to generalize to new data, a model is considered optimized when the performance on the validation set is the highest. High performance on the validation set is a good indicator that the model is able to generalize to new data, and thus is not overfitted to the training set. However, since the optimal model is chosen such that the performance on the validation set is maximized, the performance on the validation set is inherently inflated and a test set is used as an unbiased final measure of the model's performance on unseen data.

2.1.3 Layers

Depending on the purpose of a neural network, the layout of the model can be altered by introducing different types of layers.

In this section we will describe the basic types of layers of the 3D-LaneNet [7] used in this thesis, together with a short description of their functionality. We will describe another type, the *Projective Transformation Layer* separately, see Section 3.2.

2.1.3.1 Fully-connected

One of the most common layers within neural networks is the fully-connected layer, also known as the dense layer. A fully-connected layer is defined such that all its associated neurons are connected to every neuron in the following layer [25]. This is visualized in Figure 2.3 with a small deep neural network with two dense layers.

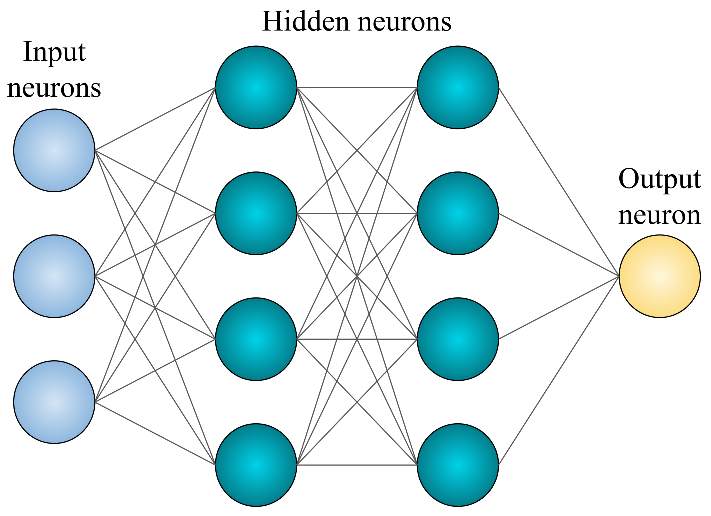


Figure 2.3: A schematic illustration of a small deep neural network with an input layer consisting of three input neurons, two fully connected layers each consisting of four neurons and an output layer consisting of one output neuron.

2.1.3.2 Convolutional and Pooling

Convolutional layers are commonly applied in neural networks to analyze images. By utilizing shared-weight architecture of the convolutional kernels, also known as filters, they become space invariant [26]. In a standard *Convolutional Neural Network* (CNN), the input is a tensor of shape:

$$(\#\texttt{inputs}) \times (\texttt{input height}) \times (\texttt{input width}) \times (\#\texttt{input channels}).$$

The input is passed through convolutional layers which contains one or several filters that transform the information from the image into so called *feature maps*. A standard convolutional filter is a $n \times m$ matrix, where each cell is associated with a trainable weight. Since features in images can occur at different locations the weights and thresholds of a filter is held constant regardless of where the filter is applied. The current part of the image the filter is applied on is referred to as the *receptive field* of the filter [27] (Section 8.1). A convolutional filter takes in the current receptive field and scale each element by the corresponding weight. This is then added to the filters threshold and passed through the filters activation function. The resulting value is the output of the filter at that specific location. By applying the filter to several positions of the input a new output feature map is created for each filter in the layer. The number of feature maps created by a convolutional layer is hence equal the number of filters. If the filter is a *MaxPooling*-filter the output of each receptive field is the maximum value of all cells. To illustrate the difference between a standard convolutional filter and a MaxPooling filter, the resulting output of both filters applied on the same receptive field is shown in Figure 2.4.

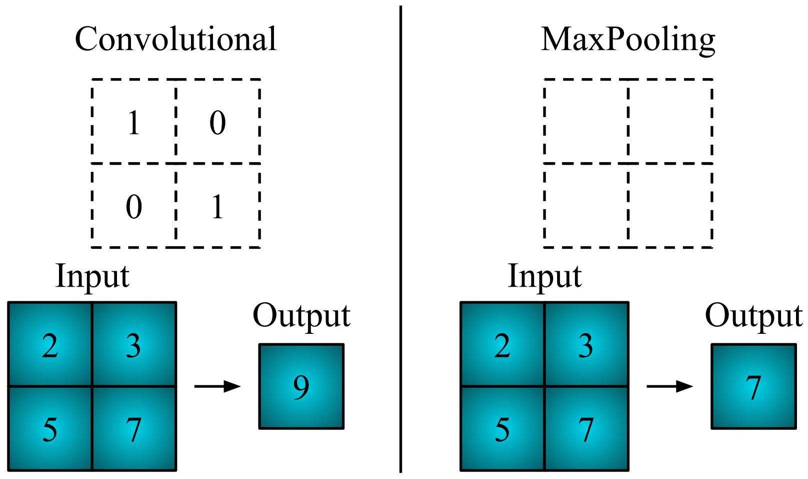


Figure 2.4: Two example of space invariant filters without activation functions. (left) A convolutional filter. The output becomes $(1\times2)+(0\times3)+(0\times5)+(1\times7) = 9$. (right) A MaxPooling filter. The output becomes $\max(\mathit{all\ cells}) = 7$.

The convolutional layers are also equipped with hyperparameters such as *padding* and *stride*. Padding changes the output dimension of the output feature map by adding a row (or rows) of zeros along the width and/or height of the input. The dimension of the output feature maps can also be changed by using different strides. A (s_x, s_y)-stride shifts the receptive field by s_x cells horizontally and s_y cells vertically. In Figure 2.5 a single 2×2 filter with no padding and a stride of $(2, 2)$ acts on a input of shape 4×4, producing an output with shape 2×2. The filter is applied on each of the 4 colored areas of the input resulting in a single value in the corresponding colored area in the output. In general the filter is swept over a much larger input, such as the width and height of a Full-HD image with 1920×1080 -pixels.

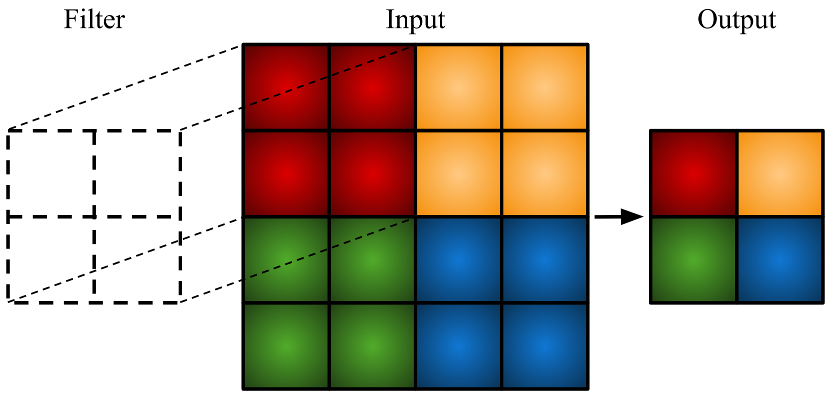


Figure 2.5: Demonstration of convolutional filters with a 2×2-filter with a 2×2-stride. The filter is applied on each of the 2×2 colored squares in the input layer, producing a single value to the corresponding colored output position.

2.1.3.3 Batch Normalization

Batch normalization is used to standardize the inputs to a layer by re-centering and re-scaling [28]. This method may in some cases, particularly for large networks, reduce the amount of training epochs required for training and stabilize the learning process, see Section 7.7.5 in [27].

2.1.3.4 Dropout

Dropout layers are used in neural networks to avoid overfitting. One type of dropout is to randomly set input units to 0 with a given frequency f [29]. To keep the sum over all inputs unchanged the units not set to 0 are scaled up by $\frac{1}{1-f}$. Randomly dropping out some nodes of a model is an effective regularization method commonly employed for larger and more complex networks, see Section 7.7.3 in [27].

2.2 Object Detection with Deep Learning

One of the most fundamental computer vision tasks is object detection, which entails finding the objects in an image and both specifying their position and class. Object detection does not only form the basis of other computer vision tasks such as object tracking and image captioning, but is also involved in many real world application including autonomous driving and video surveillance [30].

The field of object detection has been dominated by deep learning approaches since 2014, when R. Girshick *et al.* [31] introduced their method R-CNN for object detection. [31] and similar approaches such as [32]–[34] essentially treated the task in two steps, by first making region proposals in the image and then classifying the proposed regions as one of the object classes. The two-stage detection approach is perhaps the most straight-forward way of extending the simpler task of image classification to object detection, and these models have had great success and out-performed the previous (traditional) approaches for object detection by large margins. The main problem with the two-stage detection methods is that they are slow. Therefore, these models have largely been replaced by one-stage detectors, such as [35]–[37] in recent years. The one-stage detectors essentially skip the region proposal part and instead make predictions of the objects position and class directly, in a single forward pass through the network. These models have proven to be much faster than the two-stage detectors and still maintain relatively high performance. In the remainder of this section, the one-state detectors will be presented in more detail as well as the evaluation metrics commonly used for the object detection task.

2.2.1 One-stage Detectors

What allows the one-stage detectors to skip the region proposal step is essentially the use of predefined regions, also known as anchors or anchor boxes (bounding boxes). Instead of making explicit region proposals, the networks simply predicts offsets to these anchors to localize the objects in the image. All three papers [35]–[37] use a large set of predefined anchors with different sizes and aspect ratios that densely covers the input image. For each anchor box, the models predict how likely it is that an object in the image is covered by the box, what the object's offset to the predefined box is and what class the covered object belongs to. The offsets to the anchors are usually defined by lateral and vertical offsets x and y as well as width and height offsets w and h. Together with the corresponding anchor box they determine the position, size and aspect ratio of the final prediction. This is illustrated in Figure 2.6, where a set of anchor boxes are shown and the prediction in terms of offsets to one of the boxes is also illustrated (this is just an illustrative example and in reality the number of anchor boxes is much larger). The networks can of course output many object predictions for each image and the number of classes can be close to 100, which is the case of the famous COCO dataset [38].

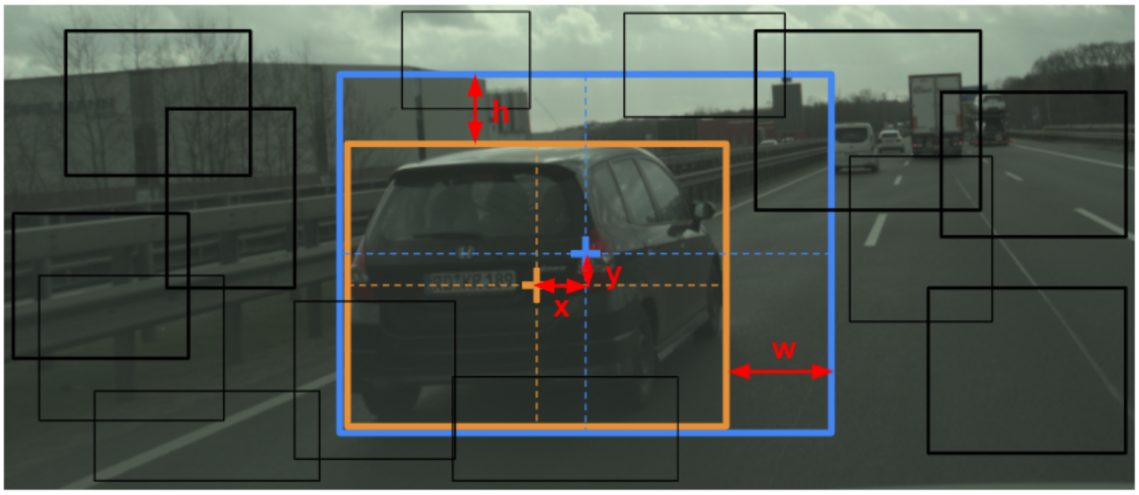


Figure 2.6: A depiction of how a predicted bounding box (orange) is derived from the predicted offsets (x, y, w and h) to a corresponding anchor box (blue). The black boxes represents all anchor boxes that densely cover the image (in reality there are many more anchor boxes than shown here).

During training, the objects in the image are assigned to one (or several) of the anchor boxes and the model is trained to predict which anchors have been assigned an object and what the corresponding ground truth offsets and classes are. At test time, the model's predictions of which anchors have been assigned a box is exposed to a threshold, and only those predictions that are confident enough are kept. The predicted offsets are then applied to the selected anchor boxes to arrive at the final object detections.

2.2.2 Evaluation Metrics

F_1-Score

A common metric used for measuring the performance of object detection models is F_1-*score*. This classification metric is based on two measures that can be calculated from the *confusion matrix*, namely the *precision* and *recall*. An overview of the confusion matrix is illustrated in Figure 2.7. Here the true labels are compared with the predicted labels. A *True Positive* (**TP**) is defined such that both the ground truth and predictions are positive. Besides **TP**, there are *False Positives* (**FP**), *True Negatives* (**TN**) and *False Negatives* (**FN**) as defined in the confusion matrix. This definition can be extended to a multi-class classification problem, but this is not applicable for our purpose.

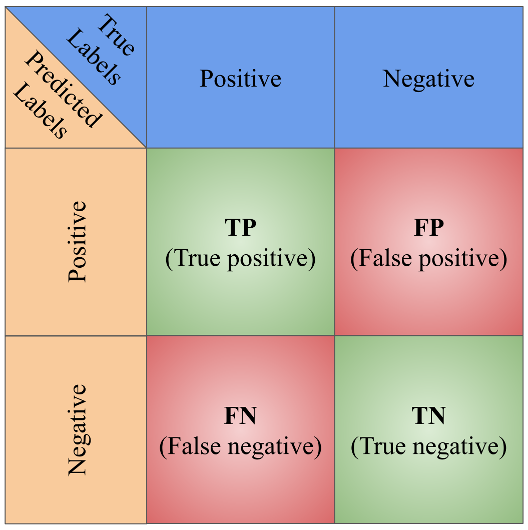


Figure 2.7: Confusion matrix for binary classification.

The precision of a test is the measure of how many percent of the positive predictions were accurate and is calculate as:

$$\texttt{Precision} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP}}. \tag{2.4}$$

Whereas the recall of the test is the measure of how many percent of the true positive labels were predicted correctly and is calculated as:

$$\texttt{Recall} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}}. \tag{2.5}$$

The $\texttt{F}_1$-score is defined as the harmonic mean of recall and precision, which is calculated as:

$$\texttt{F}_1 = 2 \cdot \frac{\texttt{Precision} \cdot \texttt{Recall}}{\texttt{Precision} + \texttt{Recall}}. \tag{2.6}$$

The definition of precision and recall requires that one can establish under which category in the confusion matrix the predictions fall into. In object detection, a common metric used for this purpose is the *IoU* (intersection over union) which takes into account the shape, size and location of the predicted bounding box. By the use of IoU, the quality of the predictions can be classified binary. The IoU is defined as the fraction between the intersection and the union of two bounding boxes as illustrated in Figure 2.8.

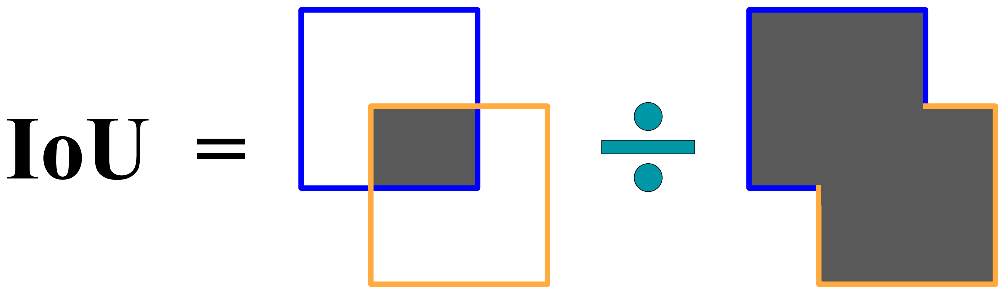


Figure 2.8: Visualization of the IoU-metric, defined as the fraction between the intersection and the union of the predicted (orange) and the ground truth (blue) bounding box.

Using the IoU, the predictions can be classified as accurate (TP) or inaccurate (FP). A (positive) predictions is considered a TP if the IoU is higher than a set threshold (between 0 and 1) and a FP otherwise. It also provides the false negatives (FN) as objects that the model was unable to identify or if the IoU score was too low.

Average Precision and maximum F_1-score

Another metric used for evaluating the performance of an object detection model is the average precision (AP). This metric is calculated by recording the precision as a function of recall($p(r)$) as the confidence threshold is varied and creating a precision-recall curve from this. The average precision computes the average value of $p(r)$ over the interval $r \in [0, 1]$, which in practice is replaced with a finite sum over every recorded positions of precision and recall [39]:

$$AP = \int_0^1 p(r)dr \quad \simeq \quad \frac{1}{n}\sum_{k=1}^{n} P_r(r(k)), \tag{2.7}$$

where k is the recorded positions, n is the number of recorded positions, $r(k)$ the recall at position k, $P_r(r(k))$ the precision at recall $r(k)$.

The highest F_1-score calculated at each of the recorded positions is what we call the *Maximum F_1-score.*

2.3 Unsupervised Learning

As discussed in the previous sections, supervised learning usually entails making some prediction (e.g. classification or regression) based on an observation. During training, the model should essentially learn the conditional probability density $Pr(Y|X)$ and thereby manage to predict the correct label y given a new input x at test time. On the other hand, the labels are unknown in the setting of unsupervised learning. Therefore, the desired output of the model may not be specified and the goal of unsupervised learning is usually to infer the properties of the probability density $Pr(X)$, rather than making any specific predictions [40].

This can be viewed as a type of data exploration and many methods have been adopted for this task. For example, clustering is used for creating groups of data based on the underlying probability density. The goal is usually to group the data such that samples within the same group are more ”similar” than samples from different groups. Different measures of similarity can be used depending on the task and there is usually no ”correct” way of grouping the data, but rather there may exist several distinct groupings of the data that are all meaningful. Sometimes the goal is also to impose some hierarchical ordering of the constructed clusters, or to investigate whether there exist different clusters at all or if the acquired data is in fact drawn from a common distribution. Among other clustering algorithms, K-means, Hierarchical clustering and DBSCAN have been used extensively. Another field of unsupervised learning that has been intensely studied is dimensionality reduction. In this case, the goal is to find a small set of important variables/dimensions that describe the original high-dimensional data as well as possible. Methods such as Principal Component Analysis, Non-negative Matrix Factorization and Kernel Principal Components have been used for this purpose [40].

Although the goal and purpose of unsupervised learning traditionally have been described as above, learning from unlabeled data has also proven useful for several tasks that are usually considered supervised learning tasks. This includes for example image classification, object detection, semantic segmentation and even monocular depth estimation [14]–[19]. Thanks to the extensive research on this topic, there exist many different methods for leveraging unlabeled data in the training process.

In some cases it is possible to train a network fully unsupervised even for complicated prediction tasks such as mono depth prediction. Although the task here is of the supervised nature (predict label y given input x) [19] solve it without any labeled data. However, what is perhaps more common is to use the unlabeled data in conjunction with a (small) labeled dataset, which is referred to as semi-supervised learning. One common approach to semi-supervised learning is self-training, which revolves around training the model on labeled data and then predicted pseudo-labels for the unlabeled data that can then be added to the training. Usually only the unlabeled data for which the model makes confident predictions are added to the training set, in an attempt to ensure good quality of the generated pseudo labels.

The purpose of semi-supervised learning is to increase the model’s performance and decrease the amount of labeled data required to excel at a certain task. Since unlabeled data is more accessible than labeled data, it is also a good means for collecting large and diversified datasets that is a necessity for generalizable and accurate deep learning models. The common belief is that semi-supervised learning is well suited for cases where little labeled data exist. Obviously, there is no need for training on unlabeled data in the (unrealistic) case of accessing unlimited amounts of labeled data. Therefore, one can expect that training on unlabeled data may not significantly increase the performance when the available labeled dataset is large.

2.4 Related Work

As described by [3], the typical lane detection pipeline consists of four steps:

1. Local lane feature extraction.
2. Lane model fitting.
3. Image-to-world correspondence.
4. Temporal aggregation.

The literature on lane detection is vast and a myriad of methods for lane feature extraction and lane model fitting have been proposed. However, the third and fourth steps of the pipeline have not been investigated as extensively, mainly due to the fact that most research towards lane detection has been dedicated to 2D lane detection. In this case, the lanes are only predicted in the image plane and therefore one does not care for the image-to-world correspondence that is necessary for 3D lane detection. The reason for little research towards 3D lane detection is probably the difficulty in creating a large-scale labeled real world 3D lanes dataset, and the first such dataset that was made publicly available was published by [12] as late as 2019.

Furthermore, although temporal aggregation can be assumed to make the predictions more accurate and robust, it is often left as future research in the literature. The general idea is that accurate lane predictions can be achieved by first making as accurate predictions as possible for any individual timestamp and then using temporal aggregation as a post-processing step to make the predictions more stable. Therefore, much of the research has focused on making accurate predictions from a single input frame and don't use temporal aggregation at all, which is also the case for 3D-LaneNet [7].

Up until recently, the feature extraction step has been done by using different heuristic methods. Various feature extraction methods based on gradients is utilized by [41]–[44] among others. [45] uses the well-known Canny edge detector (combined with a multi-resolutional Hough transform), and [46] instead extracts features in the frequency domain by utilizing the discrete cosine transformation. Furthermore, there exists a variety of filters that are handcrafted to be sensitive to edges, for example used by [47]–[50]. These traditional methods work quite well in simpler settings of lane detection, for example in a lane departure warning application where only the ego-lane needs to be detected for a relatively short distance ahead. However, the generalizability of these models is usually limited and they often struggle in different kind of lighting and weather conditions, especially in the far range where lane information is scarce in the image. Like in many other image processing tasks, neural networks have proven to be very efficient in extracting lane features. In 2015, [51] showed the feasibility of using deep learning methods for lane detection and thereafter these heuristic methods have largely been replaced by convolutional neural networks. One common deep learning approach is to treat the lane feature extraction step as a semantic segmentation task. This is done by [8], [9] among others. In these cases, the lane feature extraction step usually consists of binary segmentation (classifying each pixel as belonging to a lane or not) followed by a clustering step where lane instances are identified from the binary mask.

After extracting the relevant visual features of an image one needs to fit a lane model, which is often done using parametric or semi-parametric models [3]. For example, [45], [50] use the Hough transform (although slightly modified versions) to fit straight lines to the extracted features. Low order polynomials are also common parametric models used by [8], [9] among others. One downside of parametric models is that they assume global geometries of the lanes. Semi-parametric models such as splines or polylines are more adaptive and is used for example by [44], [52]. In general, lanes are only well approximated by straight lines close to the vehicle and a low order polynomial may not have the required complexity to correctly model the lane over far distances either. However, since the extracted visual features usually are noisy, any complex models will be prone to over-fitting. To alleviate this, RANSAC has been commonly used when fitting any type of lane models [3].

Furthermore, transforming the image to a virtual top-view (sometimes called bird's-eye view) generally decreases the complexity of the lanes in the image and has therefore widely been used as a pre-processing step to the lane model fitting. The top-view image is essentially created by warping the image such that it looks like it is viewed from above, which will be explained in more detail in Section 3.2. Extracting local lane features and fitting a lane model to these concludes the task of 2D lane detection.

Very few papers directly address the task of 3D lane detection and most that do use a flat earth assumption to project the 2D lane detections into the 3D world [3]. Under the flat earth assumption, the image-to-world correspondence is established by simply estimating the camera's position and orientation with respect to the local road surface. However, inaccuracy in both elevation and curvature of the 3D lanes is expected when the flat earth assumption is violated. [47] assumes a constant relation between camera coordinates and road coordinates and thus simply does a calibration before the start of a run. However, they found that this was problematic when the slope of the ground changes drastically. Others therefore predict the camera position and orientation to improve the image-to-world correspondence, but nevertheless still relying on a flat earth assumption. [53], [54] use a stereo setup to infer depth information and can therefore drop the flat earth assumption. [55] makes 3D lane predictions with a single camera setup using an extended Kalman filter and modeling the road curvature with third order polynomials. Although more flexible than using a flat earth, this parametric model of the road still imposes some constraints on the possible lane shapes.

Recently, a new type of model for 3D lane detection was proposed by [7] and further investigated by [10] that does not make any flat earth assumption nor any explicit modeling of the road surface. They are able to predict lanes accurately as far as 100 meters in front of the vehicle and can also handle complex lane topologies such as merges and splits, making them suitable for many real-world applications of lane detection. These data-driven models treat the 3D lane detection task in an end-to-end fashion and predict 3D lanes from a single input image. They are both based on convolutional neural networks that implicitly handle local lane feature extraction, lane model fitting and image-to-world correspondence in a single forward pass through the proposed networks.

Both networks in 3D-LaneNet [7] and Gen-LaneNet [10] make predictions in terms of confidences and geometric offsets to a set of predefined anchors. These models essentially work in the same way as one-stage object detectors such as SSD [35] and YOLO [36] and thus cast the lane detection task as an object detection one. 3D-LaneNet utilizes a dual pathway architecture were features are extracted both from the original image and from the virtual top-view of the image. The features from both pathways are then used to make predictions of the 3D lanes. Since 3D-LaneNet assumes zero yaw and roll of the camera the warping/transformation from image-view to top-view is uniquely defined by the camera pitch and height with respect to the local road surface, which are predicted by the network. While 3D-LaneNet train their feature extractor directly for the task of 3D lane prediction (as well as camera height and pitch prediction), Gen-LaneNet instead predicts a binary segmentation of the input image (classifying each pixel as belonging to a lane or not) as a first step in their feature extraction. They then use this predicted binary mask as input to the second part of the network that extracts further features by subsequent convolutional layers and then predicts 3D lanes solely based on this, without using any other information from the original input image than the extracted binary mask. The reason for their adoption of a two stage framework is that it lets them utilize large 2D lane detection datasets to make a good feature extractor (the part of the network that predicts the binary mask), while 3D-LaneNet is constrained to training their whole architecture only on annotated 3D lanes data. While this is an obvious advantage of Gen-LaneNet, it is also an apparent drawback since valuable information may be lost when converting the original image to a binary mask.

In summary, these data-driven 3D lane detection models have the possibility of overcoming many of the shortcomings of previous methods, including poor generalizability of heuristic methods for feature extraction, complex and error-prone clustering of the extracted features to create lane instances, as well as commonly used assumptions on road and lane geometry. However, as explained in Section 1.1 these models rely on expensive training data in the form of 3D lanes annotations.

While both 3D-LaneNet [7] and Gen-LaneNet [10] use synthetic data to develop their models, they don't investigate if the synthetic data can mitigate the need for labeled real-world data. As described in Section 1.1, Garnett *et al.* [13] show that training on synthetic data can increase the performance on real-world data for the task of 2D lane detection. The problem with training on synthetic data is that the difference in appearance between synthetic and real-world images makes it difficult for the model to generalize to real-world data. [13] therefore investigates several domain adaption techniques, including a novel autoencoder-based approach, that aim to adapt models trained on the unrealistic synthetic images to real-world images. Although their proposed domain adaption techniques indeed make the network generalize better to real-world data, they also show that training on the synthetic data in a naive way, using the regular supervised loss function, also improves the performance of the model, regardless of the domain differences. This basic approach of leveraging synthetic data is also used by [11] for 3D lane detection and they show that training on synthetic data increases the performance of their 3D lane detector. However, [11] have a large labeled real-world dataset of around 300 000 images and it is yet unclear how much of this data can effectively be replaced by the synthetic data.

To the best of our knowledge, improving the performance of 3D lane detection models by training on unlabeled data has not been investigated either. However, there has been extensive research towards leveraging unlabeled data to train deep learning models for other (supervised) tasks. For example, [19] leverage right-left consistency of two monocular cameras mounted at the front of a vehicle to learn monocular depth estimations from unlabeled data. Pasad *et al.* [56] also utilize a notion of consistency of objects between different frames, but instead of using two different cameras they take these frames from a video sequence captured by a single camera. The videos are taken from static indoor scenes with a moving camera that captures the scene from different viewpoints and they leverage these unlabeled video sequences to improve the performance of a semantic segmentation model. Their idea is to utilize spatio-temporal consistency of the objects seen in the video sequence and train the network to make consistent predictions over consecutive frames. For this purpose they formulate a measure of consistency between the predicted semantic mask of any two consecutive frames F_t and F_{t+1}. By predicted the camera's movement between the two frames they can compute the transformation T that transforms the pixels of frame F_t into frame F_{t+1}. Using this transformation, they compute $T(P_t)$ that is the predicted mask P_t of frame F_t transformed into frame F_{t+1}. Since the video is taken from static scenes one can expect $T(P_t)$ and P_{t+1} to be similar if both predictions are correct, and therefore the network is penalized by any difference between $T(P_t)$ and P_{t+1}.

This loss function formulation for unlabeled data is only dependent on the predictions of the network and therefore no ground truth labels are needed. [56] first train their model on a small labeled dataset until convergence and then add the unlabeled data and use the consistency loss as an additional supervision signal to train the model further. This is done at different levels of supervision (using different amounts of labeled data), while always using the full unlabeled dataset. The key result of their work is that a model trained with the consistency loss on unlabeled data in conjunction with supervised training on a small labeled dataset achieved comparable performance with a model trained on a labeled dataset of four times the size. This means that the semi-supervised approach effectively reduced the amount of required labeled data by a factor of four. However, an increase in performance when adding the unlabeled data was only observed when the size of the labeled dataset was relatively small. Meaning that no gain in performance was observed when the model had access to a much larger labeled dataset.

Although [56] consider the problem of semantic segmentation, the main ideas of their approach may be applicable also to other tasks. In particular, they leverage the spatio-temporal consistency of static objects filmed by a moving camera. Since lane markings on the roads are also static objects and the movement of a vehicle-mounted camera can be computed easily by using for example an Inertial Measurement Unit (IMU), it is not too far-stretched to believe that the same concept could be applied also to 3D lane detection. Using a measure of consistency of 3D lanes through video sequences could allow for training 3D lane detection models on unlabeled data in a similar fashion as [56] did for semantic segmentation, and thus mitigate the need for large labeled real world 3D lanes datasets.

3 Methods

Motivated by the recent success of 3D lane detection models, as well as semi-supervised learning methods, we introduce and investigate a novel spatio-temporal consistency loss for 3D lane detection that we use to train 3D-LaneNet [7] on unlabeled data in a semi-supervised fashion. While the consistency loss and method for semi-supervised training is new, we do not make any additions to the architecture of 3D-LaneNet. Instead, we use the unofficial Pytorch implementation of 3D-LaneNet, made by [10], as reference and reimplement 3D-LaneNet in Tensorflow without any major changes.

The layout of this chapter is as follows: Sections 3.1, 3.2 and 3.3 describe the relevant coordinate systems as well as the top-view projection and lane anchors used by the model. The architecture and implementation details of the model are then described in Section 3.4, followed by a description of the used evaluation metrics in Section 3.5. Thereafter, the semi-supervised approach is explained in Section 3.6 and finally the datasets and conducted experiments are described in Sections 3.7 and 3.8.

3.1 Coordinate Systems

There are two coordinate systems used in this project which are called C_{cam} and C_{road}. The coordinate system C_{cam} is simply defined as the system with the camera in the origin and orientated with the forward direction pointing in the direction of the camera. The coordinate system C_{road} lies straight beneath C_{cam} but is oriented such that the forward direction is aligned with the road surface. This means that the transformation between these coordinates systems is uniquely defined by the camera pitch (with respect to the local road surface) and the height of the camera above the ground. Estimating the transformation by predicting the camera pitch and height allows for warping the image to a virtual top-view, which is a key component in the network architecture. Furthermore, C_{road} is also used when representing both the ground truth and predicted lanes. In C_{road}, the x-axis points to the right (lateral direction), the y-axis points forward (longitudinal direction) and the z-axis points up (vertical direction), as shown in Figure 3.1.

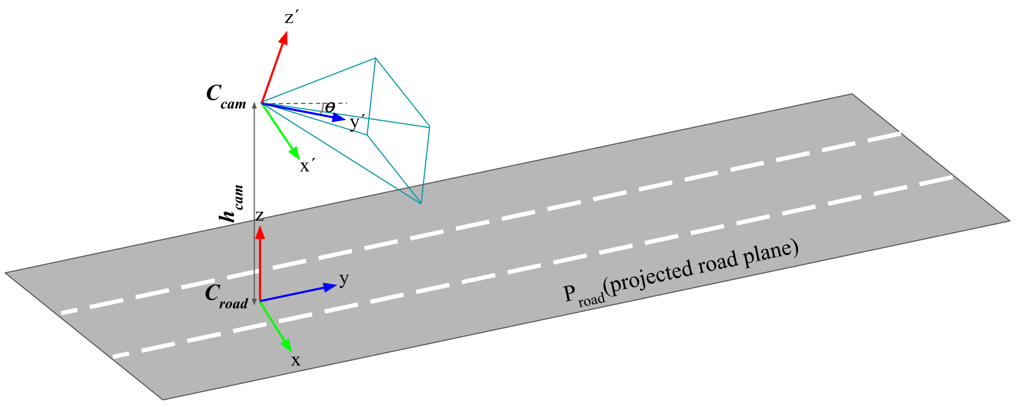


Figure 3.1: An illustration of the coordinate systems C_{cam} and C_{road}.

3.2 Top-view Projection

A core piece of the model is the representation and extraction of feature maps in both the image plane and the virtual top-view. This is done by transforming the image-view feature maps (feature maps extracted from the input image) to top-view feature maps by using the predicted camera pitch and height. To get a good understanding of what the top-view is, an example where the input image is transformed to top-view is shown in Figure 3.2. As will become clear after reading Section 3.4.1, it is only the extracted feature maps from convolutional layers that are transformed to top-view and not the input image itself, which is important to keep in mind when reading this section. However, since feature maps essentially can be viewed as images with some height, width and number of channels, the method for top-view transformation can be used analogously for both images and image-view feature maps. The top-view projection is done in the *Projective transformation layers*, which can be seen in the model architecture in Figure 3.4.

To transform the image to top-view a uniform, rectangular grid consisting of $\hat{w} \times \hat{h}$ points is defined in the coordinate system C_{road}. The width and height/length of the grid is 20 and 96 meters respectively and the points in the grid are evenly spaced between $x = -10$ and $x = 10$ meters and $y = 5$ and $y = 101$ meters in C_{road}, while the z-coordinate is zero for all points (meaning that the grid lies flat on the ground). In other words, the grid consists of regularly spaced rows with a distance of $20/\hat{w}$ meters between each row in the lateral direction and $96/\hat{h}$ meters in the longitudinal direction, with corners in $\{(x, y)\}_{i=1}^{4} = \{(-10, 5), (10, 5), (-10, 101), (10, 101)\}$. After the grid has been defined, it is transformed into the coordinate system C_{cam} by using the predicted height and pitch of the camera. It is thereafter projected into image by using the intrinsic calibration matrix of the camera, which depends on certain properties of the used camera such as the focal length of the lens. The intrinsic calibration is held constant for the experiments with data collected from the same dataset (taken with the same type of camera). As seen in Figure 3.2, the grid does not look regular when viewed in the image, which is of course due to the perspective transformation that makes distant objects appear smaller in the image than objects close to the camera.

When projected into the image, the grid simply defines which points to sample from the original image to create the top-view image. That is, each point in the grid determines the value of one pixel in the top-view image, which is computed by sampling the original image at the positions specified by the grid. Since the position of each grid point in the image is not necessarily integer valued, while the image is discretized in pixels, bilinear interpolation is used to compute the value of the image pixels at the specified positions. This is done for each channel separately, thus preserving the number of channels of the input. Since each grid point determines the value of one pixel in the top-view image, the width and height of the top-view is equal to the width and height of the grid.

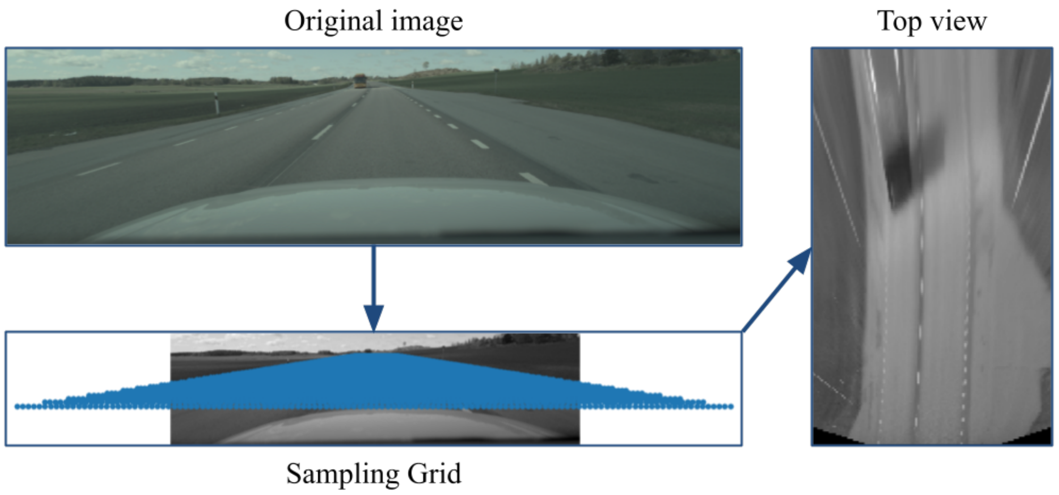


Figure 3.2: Illustration of how an image is transformed to top-view by sampling the image at the positions specified by the sampling grid. This transformation is then applied on each color channel.

3.3 Anchors

Since 3D-LaneNet essentially casts the lane detection task as an object detection one and works in a similar fashion as one-stage object detectors, it too uses the concept of anchors to make predictions. In this case, the set of anchors $\{A^i\}_{i=1}^N$ consists of equally spaced longitudinal lines with zero height and constant lateral offset. Each anchor is represented by the set of k points in the coordinate system $\boldsymbol{C}_{road}$ given by $\{(X_A^i, y_j, 0)\}_{j=1}^k$, where $\boldsymbol{y} = \{y_j\}_{j=1}^k$ is common for all anchors and specifies the predetermined values in the longitudinal direction and X_A^i is the constant lateral offset of anchor A^i. With respect to each of the N anchors, the output of the network consists of a confidence prediction p^i that describes whether anchor i is associated with a lane or not, as well as predictions $(\boldsymbol{x^i}, \boldsymbol{z^i}) = \{(x_j^i, z_j^i)\}_{j=1}^k$ that correspond to lateral and vertical offsets with respect to the k points of anchor i. In summary, the prediction (x_j^i, z_j^i) corresponds to the point in 3D space given by $(X_A^i + x_j^i, y_j, z_j^i)$ in the coordinate system $\boldsymbol{C}_{road}$. The anchor representation is illustrated in Figure 3.3.

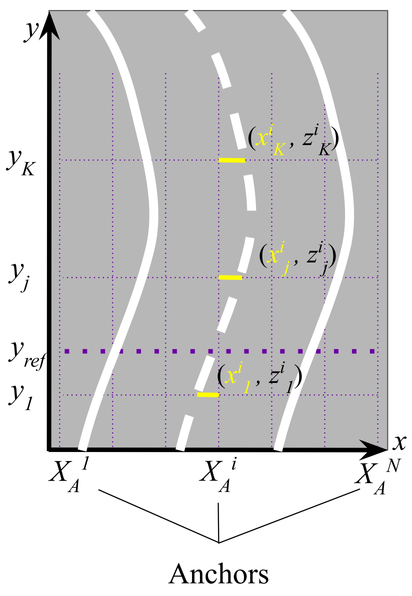


Figure 3.3: Illustration of the lane anchors and the predicted offsets at the predefined y-values to a given anchor.

For the purpose of training, the ground truth lanes are assigned to the closest anchor at $y_{ref} = 20m$. If more than one lane is closest to the same anchor the longest lane will be assigned to this anchor, while the other lane(s) are disregarded during training. Following the method of 3D-LaneNet, any lanes that do not cross y_{ref} inside the top view region (described in Section 3.2) are disregarded during both training and validation.

In our experiments, the number of anchors N was set to 16 and the anchors were chosen to be represented by ten points ($k = 10$). The predefined y-values of the anchors were set to $\{6.5, 10, 15, 20, 30, 40, 50, 60, 80, 100\}$ meters and the constant lateral offsets X_A^i were equally spaced between -10 and 10 meters, meaning that the distance between the anchors is ≈ 1.33 meters. This is the same setup that [10] used in their unofficial implementation of 3D-LaneNet, with the exception that we have changed the first y-position from 5 to 6.5. This was done because the hood of the vehicle usually covered the road at $y = 5$ meters and therefore we did not have many ground truth lanes starting this early.

3.4 Model

The model used in this study is a reimplementation of the 3D-LaneNet [7] in Keras, based on the code used in the article Gen-LaneNet [10].

3.4.1 Architecture

A schematic overview of the network is illustrated in Figure 3.4 and the specific details are listed in Table 3.1. The layers are clustered into what we call **L**−*layers*. Information passed to the network is split up and processed in two parallel pathways via the so called dual-pathway backbone [7]. The model can be divided into four quadrants, which in the 3D-LaneNet article are called: the *Image-view pathway*, the *Road plane prediction branch*, the *Top-view pathway* and the *Lane prediction head*.

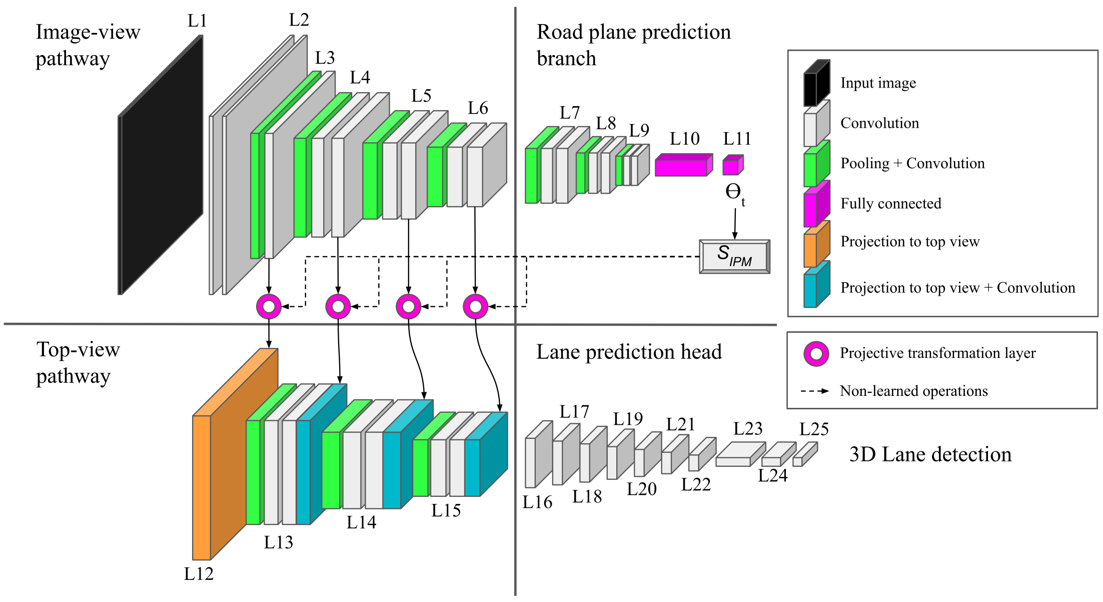


Figure 3.4: The 3D-LaneNet [7] architecture used in this thesis. The model consists of a dual pathway that includes convolutional, max pooling and dense layers together with a projective transformation layer that transforms the feature maps from image-view to top-view. The image is inspired by [7].

$\mathbf{L}-layer$	Width	Height	#Filters
1	w	h	N/A
2	w	h	64
3	$w/2$	$h/2$	128
4	$w/4$	$h/4$	256
5	$w/8$	$h/8$	512
6	$w/16$	$h/16$	512
7	$w/32$	$h/32$	256
8	$w/64$	$h/64$	128
9	$w/128$	$h/128$	64
10	$dim = 1 \times 1 \times 64$		
11	$dim = 1 \times 1 \times 1$		
12	$\hat{w} = 128$	$\hat{h} = 208$	N/A
13	$\hat{w}/2$	$\hat{h}/2$	128
14	$\hat{w}/4$	$\hat{h}/4$	256
15	$\hat{w}/8$	$\hat{h}/8 = 26$	256
16	$\hat{w}/8$	24	64
17	$\hat{w}/8$	22	64
18	$\hat{w}/8$	20	64
19	$\hat{w}/8$	16	64
20	$\hat{w}/8$	12	64
21	$\hat{w}/8$	8	64
22	$\hat{w}/8$	4	64
23	$dim = 1 \times \hat{w}/8 \times 256$		
24	$dim = 1 \times \hat{w}/8 \times 64$		
25	$dim = 1 \times \hat{w}/8 \times 3(2k+1)$		

Table 3.1: Table containing information about the dimensionality of the $\mathbf{L}-$*layers*. Here w and h are the width and height of the input image while $\hat{w}$ and $\hat{h}$ are the width and height of the first top-view feature map.

Image-view pathway

The first part of the network is the *Image-view pathway.* This part of the network takes in the RGB-channels of the input image and propagates the information through several convolutional and maxpooling layers, following the structure of the first part of the standard VGG16 [57] network. Here spatial features in the original image-plane are preserved. The final output of the *Image-view pathway* is sent to the *road plane prediction branch.* Furthermore, the output from **L**3, **L**4, **L**5 and **L**6 is sent to the *Top-view pathway* via a *Projective transformation layer* which transforms these image-view feature maps to top-view as described in Section 3.2.

Road plane prediction branch

From the *Image-view pathway* the output features of **L**6 is sent to the *Road plane prediction branch* which goes through a similar process as the *Image-view pathway* of convolving and maxpooling. The information is then sent to **L**10 and **L**11 which are two fully connected layers resulting in two values: the predicted height and pitch of the camera (Θ_t in Figure 3.4). The predicted pitch and height is then sent together with the outputs from **L**3, **L**4, **L**5 and **L**6 to a *Projective transformation layer*, see Section3.2.

Top-view pathway

The second part of the parallel pathway is the *Top-view pathway* where all features are represented in the top-view plane. Following a similar procedure as in the image-view pathway, the signals are convolved and maxpooled with the addition of concatenation of the projected outputs of the feature maps from the **L**3, **L**4, **L**5 and **L**6. This means that we are combining the spatial features gathered in both the image-view plane and the top-view plane. The signal is then sent to to Lane prediction head where the final predictions of the 3D-Lanes are made.

Lane prediction head

The final part of the network is the *Lane prediction branch*, which takes in the output features of **L**15 and further convolves the information. The final output of **L**25 are the predicted 3D lanes which are represented by the lane anchors defined in Section 3.3. The output contains the predicted lateral and height offsets to each of the anchors as well as confidence scores of the existence of a lane at each anchor.

The final output dimension of the network becomes $dim = N \times (2k+1) = 336$, where $N = 16$ is the number of anchors and $k = 10$ is the number of points that each anchor consists of.

3.4.2 Supervised Loss Function

The supervised training of the model is done using a loss function consisting of three parts:

$$\begin{aligned}\mathcal{L} = &-\left[\sum_{i=1}^{N}(\hat{p}^i \log(p^i) + (1-\hat{p}^i)\log(1-p^i))\right] \\ &+\sum_{i=1}^{N}\hat{p}^i \cdot \left(\|\boldsymbol{x}^i - \hat{\boldsymbol{x}}^i\|_1 + \|\boldsymbol{z}^i - \hat{\boldsymbol{z}}^i\|_1\right) \\ &+ |\theta - \hat{\theta}| + |h - \hat{h}|.\end{aligned} \tag{3.1}$$

In Equation 3.1, the variables with hat denotes the ground truth labels while the ones without are the predictions. Here p is the probability that there is a lane at the given anchor (the ground truth is simply an indicator taking the value 0 or 1), $\{\boldsymbol{x}^i\}_{i=1}^N$ and $\{\boldsymbol{z}^i\}_{i=1}^N$ are the offsets with respect to the lane anchor i for all N anchors, θ is the camera pitch and h is the camera height.

The first sum is a cross entropy loss over the confidence scores of the lane anchors.

The second sum is a l^1-norm penalty for the vertical and lateral offsets. Each term in the sum is multiplied with the ground truth lane anchor probability, which is either 1 or 0. Since we only have information about the lanes that exist and do not know anything about the other anchors, we cannot penalize the models outputs where lanes do not exist. Finally a penalty for the predicted height and pitch is added to the loss function.

3.4.3 Implementation Details

The model was implemented in Keras, Tensorflow, where the VGG16 architecture is easily accessible. We do not use any pre-trained weights for the VGG16 [57] backbone, or any other parts of the model, but instead use random initialization for all the weights in the network. We use the standard *Glorot normal initializer* available in Keras, which is also known as the *Xavier normal initializer*. Following the unofficial implementation of 3D-LaneNet by [10], we include batch normalization layers after every convolutional and dense layer (except for those layers that output the predictions of the network) and include a dropout layer before the camera height and pitch prediction (**L**11). However, since it is not clear if 3D-LaneNet [7] actually used batch normalization and dropout in their original implementation, we introduce the hyperparameters *batch norm* and *dropout* to allow for training models also without these optional layers. The batch normalization and dropout layers are included if the hyperparameters *batch norm* or *dropout* is True respectively, and excluded otherwise.

For supervised training, the learning rate is scheduled such that it is divided by a constant LR_d (learning rate decay) every LR_i (learning rate interval) epochs. Therefore, the learning rate schedule entails three hyperparameters given by the initial learning rate and the constants LR_d and LR_i. During all experiments, the Adam optimizer was used to train the network.

Regularization was also investigated as a way of facilitating the training process. The kernel och bias l^1-regularization available in Keras was applied on every convolutional and dense layer except for those responsible for the predictions of the network. To be able to adjust the amount of regularization we defined the input to the Keras regularizer, which determines the magnitude of the regularization, as a hyperparameter that we call *regularization*. No regularization is applied if *regularization* is zero and the magnitude of the applied regularization is scaled linearly with *regularization* if it is greater than zero.

Since we use a very small dataset compared with [7] (3194 labeled images in the training set instead of roughly 100,000 images) we also tried using a constant camera height and pitch instead of predicting these values. In this case, the mean height and pitch of the training set was used to transform image view feature maps to top-view in the projective transformation layers, rather than using the predicted height and pitch. The hypothesis was that using a constant height and pitch may be advantageous in the case of little available data since poor height and pitch predictions may result in skewed/erroneous top-view projections, which make it difficult for the model to accurately predict the 3D lane geometry. For this purpose, a hyperparameter denoted as *constant cam* was introduced, which takes the value True if a constant height and pitch is used and False otherwise.

3.5 Evaluation

To evaluate the predictions of the network on the validation and test datasets, the network's output was first transformed into 3D lanes. This is accomplished by first applying a probability threshold p_{th} on the predicted confidence scores of the network's sixteen anchors. Each anchor with predicted confidence less than p_{th} were discarded and the predictions with higher confidence than p_{th} are considered as positive predictions. For each positive prediction, the 3D lane geometry is then computed by using that the predictions $\boldsymbol{x}^i$ and $\boldsymbol{z}^i$ of anchor i correspond to the points in 3D space given by $\{(X_A^i + x_j^i, y_j, z_j^i)\}_{j=1}^k$, as explained in Section 3.3. This essentially transforms the predicted confidences and offsets of the sixteen anchors into a (small) set of predicted 3D lanes.

In order to evaluate how well the set of predicted 3D lanes approximates the set of ground truth lanes, the global optimal matching between the two sets is sought. Following the method of [10], this is accomplished by formulating the matching problem as bipartite matching problem and seeking the solution with the min-cost-flow solver from the *ORTOOLS* package in Python. For this purpose, the lanes are first resampled into a denser representation such that each lane is represented by a set of K points $\{x_j^i, y_j, z_j^i\}_{j=1}^K$, where $\{y_j\}_{j=1}^K = \{7, 8, 9, 10, ..., 100\}$ is common for all lanes and $\{x_j^i\}_{j=1}^K$ and $\{z_j^i\}_{j=1}^K$ depends on the corresponding prediction. The sampling is done via piece wise linear interpolation in x and z, meaning that these variables are viewed as functions of the position in the forward direction y. For example, to compute the x-value at a point $y = b$ that lies in between $y = a$ and $y = c$, for which the predictions are x_a and x_b respectively, the following equation is used:

$$x_b = (c - b)\frac{x_a}{c - a} + (b - a)\frac{x_c}{c - a}. \tag{3.2}$$

This holds for any point $a <= b <= c$ where a and c is in the set of original y-values $(\{6.5, 10, 15, 20, 30, 40, 50, 60, 80, 100\})$ and b is one of the new y-values. z-values at the interpolated points are computed analogously. After resampling the lanes into denser representations, the cost of matching lane m with lane n is defined as

$$cost_{mn} = \sum_{j=1}^{K} d_j^{mn},$$

where d_j^{mn} is given by the following equation:

$$d_j^{mn} = \begin{cases} \log(\sqrt{(x_j^m - x_j^n)^2 + (z_j^m - z_j^n)^2} + 1), & \text{if both lanes exist at j} \\ d_{th}, & \text{otherwise.} \end{cases} \tag{3.3}$$

This definition of the cost of matching two lanes is used by the min-cost-flow solver and the global optimal matching is defined as the one that matches the set of ground truth and predicted lanes with the lowest total cost. By adding a penalty of d_{th} for each point that is not covered by both lanes, it is possible to match partly overlapping lanes with this method. This is important since the ground truth lanes of our datasets don't always stretch over the full prediction range of 6.5-100 meters.

This formulation of the distance metric d_j^{mn} is slightly different from the one used by [10]. They use the Euclidean distance if both lanes exist for a point, instead of the natural logarithm of the Euclidean distance (plus one) as we do. The reason for adding the logarithm is that it makes the matching more stable.

Consider for example the fictional case shown in Figure 3.5, where the positions of a lane for simplicity is represented by a single point on a one dimensional line. In this case, the ground truth lanes A and B have the following distances to the predicted lanes C and D: $\|A - C\| = 1, \|A - D\| = 2, \|B - C\| = 0, \|B - D\| = 1$. The cost of the matching $A - C$ and $B - D$ is therefore equals 2, which is also the cost of matching $A - D$ and $B - C$ when d_j^{mn} is defined without the logarithm. However, when defining d_j^{mn} with the logarithm the cost of matching $A - C$ and $B - D$ becomes $\log(1 + 1) + \log(1 + 1) \approx 1.39$, while the cost of matching $A - D$ and $B - C$ is only $\log(2 + 1) + \log(0 + 1) \approx 1.10$. In this case, it seems likely that prediction C is trying to predict the ground truth lane B, while prediction D is a faulty detection. Therefore, the latter method for matching the lanes is preferred (the fact that A is matched with D in this case is not an issue and will be handled later as an invalid match).

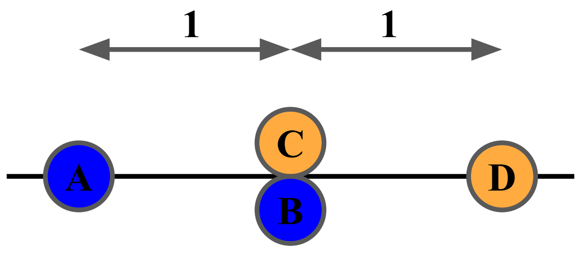


Figure 3.5: Simplified one dimensional representation of ground truth lanes (blue) and predicted lanes (orange).

This theoretical motivation also translates to practical scenarios such as the one shown in Figure 3.6, where ground truth and predicted lanes are shown in blue and orange respectively. In this example, the min-cost-flow solver not using the logarithm finds the matching between ground truths and predictions as 1→1, 2→2 and 3→3 (see the numbering of the lanes in Figure 3.6). This is obviously not desired and is a consequence of the linear scaling of the Euclidean distance. When incorporating the logarithm, the matching instead becomes 1→2, 2→3 and 3→1 between ground truths and predictions.

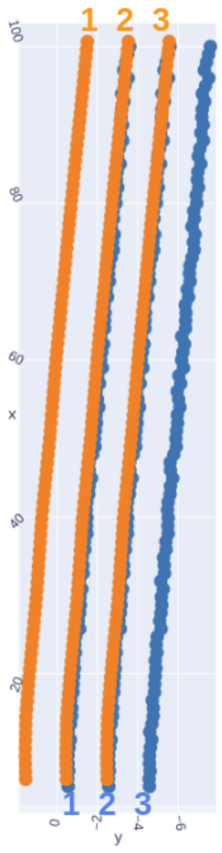


Figure 3.6: The figure shows three predicted lanes (orange) and three ground truth lanes (blue).

After the global optimal matching has been found, it is known what predictions correspond to what ground truths. It is then possible to define metrics such as precision and recall as well as measuring the average distance between the matched lane pairs. Ground truth lanes are considered detected successfully if at least 75 % of its points (may be fewer points than K since not all ground truth lanes stretch the full range from 7 to 100 meters) have a distance less than d_{th} to the matched lane. That is if d_j^{mn} is less than d_{th} for at least 75 % of its points. Similarly, a prediction is considered correct if at least 75 % of its (always K) points have a distance less than d_{th} to the matched ground truth. The precision and recall is computed as

$$\text{Precision} = \frac{Pred_{\text{correct}}}{Pred_{\text{total}}} \tag{3.4}$$

and

$$\text{Recall} = \frac{GT_{\text{detected}}}{GT_{\text{total}}}. \tag{3.5}$$

In the above equations, $Pred_{\text{correct}}$ and $Pred_{\text{total}}$ are the number of correct predictions and the total number of predictions respectively, while GT_{detected} and GT_{total} is the number of successfully detected ground truth lanes and the total number of ground truth lanes respectively. Note that the numerator is not necessarily the same in these definitions and therefore differs from the standard definitions of precision and recall in this sense. The numerators may differ if for instance a short ground truth lane is matched with a long predicted lane. In this case, the ground truth lane may be considered detected successfully since all of its points may be covered by the predicted lane, while the predicted lane is considered incorrect since less than 75 % of its points may be covered by the short ground truth lane.

The precision-recall curve is then computed by varying the probability threshold from 0.05 to 0.95 with step length 0.05 and computing precision and recall for each value. The F-score is also computed for each value of the probability threshold and the maximum F-score is reported as one of the key metrics. Precision and recall was set to 0 and 1 for $p_{th} = 0$ and set to 1 and 0 for $p_{th} = 1$ respectively for the purpose of computing the average precision (AP). The found values of recall and precision constitute a set of irregularly spaced points that are then interpolated at even intervals with piece wise linear interpolation. The mean of these interpolated points constitute the average precision of the model. Figure 3.7 illustrates the experimentally found points in blue (with the exception of the first and last points that are added manually), from which the red points are interpolated and used to compute AP. This method of computing AP is not exactly in line with what was found in the theory section and may give rise to some error since the end points of the graphs are not correctly accounted for. However, this is the method that Gen-LaneNet [10] used to compute AP and since this is one of the few sources for comparison available, we chose to do the same.

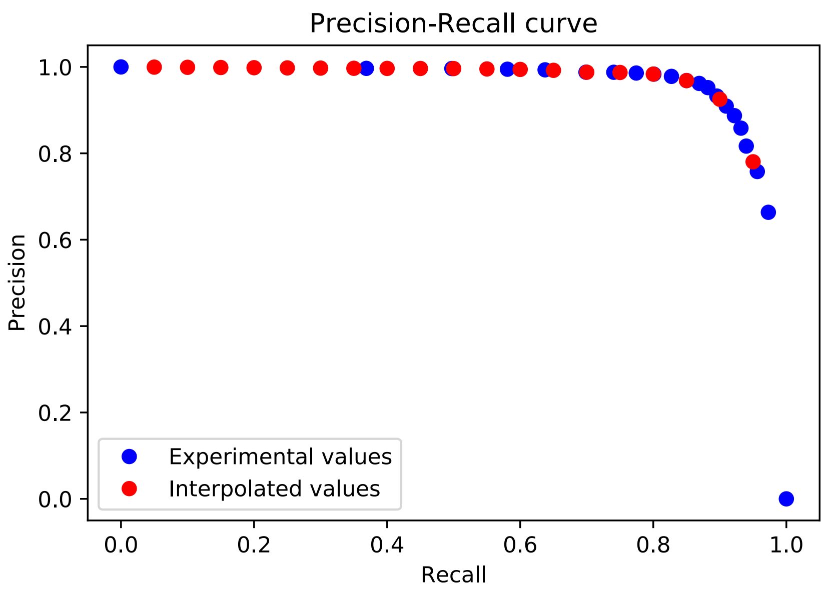


Figure 3.7: Illustration of an experimentally found precision-recall curve (blue) and the regularly spaced interpolated points of this curve (red).

Furthermore, for each matched lane pair with average point wise distance less than d_{th} (denoted as a valid match) the average absolute lateral and vertical error is computed. This is done in the close range (0-40 meters) and far range (40-100 meters) separately for each such lane pair. The close and far range errors are then averaged over all such lane pairs to get the close and far range error metrics for the whole dataset. The probability threshold p_{th} is set to the same value that gave rise to the maximum F-score described earlier, such that both the lane geometry errors and the reported (max) F-score are based on the same set of predictions (using the same probability threshold).

In our experiments, d_{th} is set to $\log(1.5+1)$ such that

$$\log(\sqrt{(x_j^m - x_j^n)^2 + (z_j^m - z_j^n)^2} + 1) < d_{th} \iff \sqrt{(x_j^m - x_j^n)^2 + (z_j^m - z_j^n)^2} < 1.5. \tag{3.6}$$

Meaning that $d_{th} = \log(1.5+1)$ does not allow for an Euclidean distance between points larger than 1.5 meters when for example computing precision and recall, which effectively is the same threshold as [10] used. In summary, the metrics used to evaluate the model's performance is (max) F-score, AP and x-error and z-error in the close and far range.

3.6 Semi-supervised Training

The developed method for unsupervised training is based on the assumption of consistency of 3D lanes in video sequences. In general, it is likely that some parts of the lanes observed in one frame of a video sequence are also observed in the next frame, given that the time interval between the two frames is not too large. Therefore, the predicted 3D lanes of two such frames should partly overlap/be close together in 3D space, if the predictions are close to correct. The proposed method leverages this fact and constitutes a framework for training 3D-LaneNet to make consistent predictions on unlabeled video data. The remainder of this section describes our method in detail as well as the assumptions and observations that lead up to the proposed method. Section 3.6.1 describes the consistency loss function that is used as an unsupervised training objective and Section 3.6.2 describes the semi-supervised training scheme used for training the model on labeled and unlabeled data simultaneously.

3.6.1 Consistency Loss

Given two images from a video sequence, one can impose some notion of consistency on the predicted 3D lanes as these should partly overlap for correct predictions. However, it is not certain that consistent predictions are correct. For example, if both images from the video sequences are taken from the same position (i.e. the vehicle is standing still) the predicted 3D lanes will be consistent as long as the predictions are the same in both frames. That is, the network can predict any kind of lanes in both frames and be consistent, but certainly not correct. On the other hand, if the vehicle has significant movement in between the two frames it is likely that the vehicle's relative position to the lanes as well as the lane curvature (as viewed from the vehicle) changes between the two frames. In this case, making correct predictions of lanes seems like one of the only possibilities for the network to be consistent. This observation poses a constraint on the unlabeled dataset used for consistency loss: the vehicle should have significant movement between the video frames in order to avoid any trivial consistent predictions.

Another concern about the consistency of lanes is that lanes visible in one frame are not always visible in the next. Particularly in urban environments, where lane topologies are complex and can change quickly over short distances, it is quite common that the lanes seen in one frame are not the same as those seen in the next frame, especially when it is desired that the vehicle has some movement between the frames as described previously. Due to the regularity of the lanes on highways, this road type seems particularly well suited for imposing consistency. The fact that the lane topologies are usually constant over far distances on highways makes it possible to impose consistency between lanes even though the vehicle has significant movement between the frames. Due to this reason, and to keep the unlabeled dataset and consistency between 3D lanes as simple as possible, the dataset used for unsupervised training only includes images from highways. Furthermore, the method that will be presented here only considers consistency between image pairs, although it could be possible to impose consistency over several frames.

It is now time to formalize the notion of consistency between two sets of lanes. Intuitively, if one knows that a predicted lane of frame t corresponds to the same ground truth lane as a predicted lane of frame $t+1$, it makes sense to enforce spatial consistency on these predictions such that they align well in 3D space. However, since the ground truth lanes are unknown for unlabeled data we are forced to guess which predictions correspond to the same ground truth. This guessing game becomes quite difficult since the vehicle's movement relative to the lanes between the two frames makes it so that the responsibilities of the 16 anchors of the network may change between the frames. For example, it may be the case that the seventh anchor in frame t predicts the same lane as the tenth anchor in frame $t+1$. Since there is no easy way of knowing which lane each anchor will predict in the two frames, we resort to investigating this explicitly by matching the predicted 3D lanes of the two frames. This is done by first transforming the predicted 3D lanes of frame $t+1$ into frame t using the 3D coordinate transformation between these two frames given by processed accurate GPS data. The global optimal matching between the two sets of lanes is then sought in analogy with the matching process used during evaluation as described in Section 3.5. When matches between the predicted lanes have been found, it is possible to enforce spatial consistency between the matched lanes as well as penalizing the network's confidence predictions. The method for computing the consistency loss is schematically illustrated in Figure 3.8.

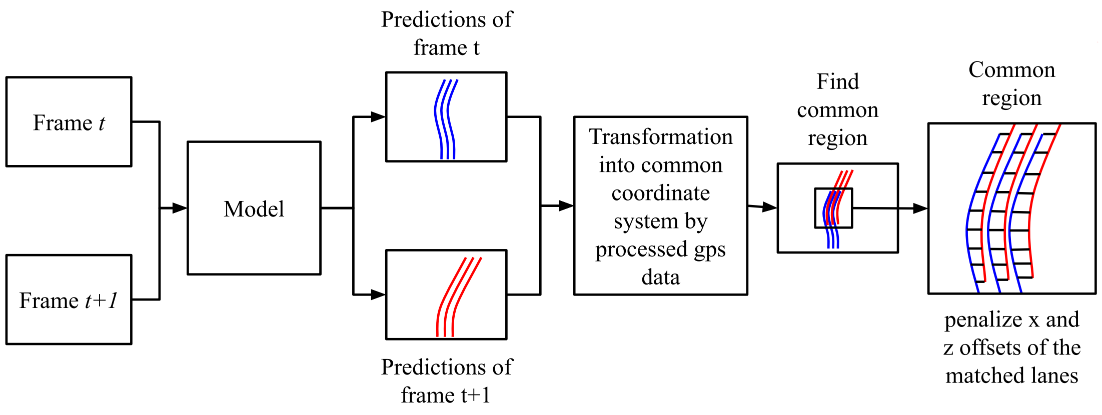

Predictions of frame t
Frame t
Frame t+1
Model
Transformation into common coordinate system by processed gps data
Find common region
Common region
Predictions of frame t+1
penalize x and z offsets of the matched lanes

Figure 3.8: Schematic illustration of how the consistency loss is computed between two sets of predicted lanes.

What follows is the mathematical definition of the consistency loss that is used to penalize both lane geometry and confidence predictions. Given the predictions $\{p^i\}_{i=1}^N$ and $\{(\boldsymbol{x^i}, \boldsymbol{z^i})\}_{i=1}^N$ of frame t and $\{\hat{p}^i\}_{i=1}^N$ and $\{(\boldsymbol{\hat{x}^i}, \boldsymbol{\hat{z}^i})\}_{i=1}^N$ of frame t+1, all the predictions with confidence less than p_{th} is disregarded. The remaining predictions are then transformed into 3D space by using the fact that the prediction (x_j^i, z_j^i) corresponds to the point in 3D space given by $(X_A^i + x_j^i, y_j, z_j^i)$ in the coordinate system $\boldsymbol{C}_{road}$, as described in Section 3.3. The predictions of frame $t+1$ are then transformed to $\boldsymbol{C}_{road}$ of frame t and the lanes are resampled at common longitudinal positions by piece wise linear interpolation. The resampled lane that originated from predictions p^i and $(\boldsymbol{x^i}, \boldsymbol{z^i})$ is now defined by a new set of points in 3D space given by $(\boldsymbol{x^i_{ext}}, \boldsymbol{y_{ext}}, \boldsymbol{z^i_{ext}})$, where each of these vectors are of some length q and $\boldsymbol{y_{ext}}$ is a vector of predetermined sampling positions. Similarly, the resampled lane that originated from the predictions $\hat{p}^i$ and $(\boldsymbol{\hat{x}^i}, \boldsymbol{\hat{z}^i})$ of frame $t+1$ is denoted by $(\boldsymbol{\hat{x}^i_{ext}}, \boldsymbol{y_{ext}}, \boldsymbol{\hat{z}^i_{ext}})$.

Note that the longitudinal positions given by $\boldsymbol{y_{ext}}$ is the same for the predictions of both frame t and $t+1$. Furthermore, $\boldsymbol{y_{ext}}$ only corresponds to points that lie within the common region of the lanes from frame t and $t+1$ as shown in Figure 3.8 (e.g. 50-100 meters in $\boldsymbol{C}_{road}$), which depends on the distance traveled by the vehicle between the two frames. In our work, $\boldsymbol{y_{ext}}$ is by default set to $10, 15, 20, ..., 100$ and then the points that don't lie within the common region are removed, such that both predictions of frame t and $t+1$ exist for all the remaining points in $\boldsymbol{y_{ext}}$.

After resampling the lanes at the positions defined by $\boldsymbol{y_{ext}}$, the lanes are matched using the min-cost-flow algorithm with the same cost function as described in Section 3.5. Again, the matched lane pairs that have a smaller average distance than d_{th} (again compared with the logarithm of the Euclidean distance as in Equation 3.6) are considered valid matches. The consistency loss then consists of two components, one for valid matches and one for invalid matches. If the predicted lane given by p^m and $(\boldsymbol{x^m}, \boldsymbol{z^m})$ of frame t forms a valid match with the prediction $\hat{p}^l$ and $(\boldsymbol{\hat{x}^l}, \boldsymbol{\hat{z}^l})$ of frame $t+1$, the consistency loss between this lane pair is computed as

$$\mathcal{L}_{valid}^{m,l} = \mathcal{L}_{p}^{m,l} + \mathcal{L}_{xz}^{m,l}, \tag{3.7}$$

where

$$\mathcal{L}_{xz}^{m,l} = \|\boldsymbol{x^m_{ext}} - \boldsymbol{\hat{x}^l_{ext}}\|_1 + \|\boldsymbol{z^m_{ext}} - \boldsymbol{\hat{z}^l_{ext}}\|_1 \tag{3.8}$$

$$\mathcal{L}_{p}^{m,l} = -\log(p^m) - \log(\hat{p}^l). \tag{3.9}$$

The total consistency loss of all valid matches is then given by

$$\mathcal{L}_{valid} = \sum_{(m,l)\in V} \mathcal{L}_{valid}^{m,l}, \tag{3.10}$$

where V is the set of all valid matches. Finally, a penalty for not finding a valid match is applied as

$$\mathcal{L}_{invalid} = -\left(\sum_{i\in U_1} \log(1-p^i) + \sum_{i\in U_2} \log(1-\hat{p}^i) \right), \tag{3.11}$$

where U_1 and U_2 are the set of all anchors of frame t and $t+1$ respectively that did not find a valid match.

The total consistency loss is the sum of the penalty applied on the valid and invalid matches, such that

$$\mathcal{L}_c = \mathcal{L}_{valid} + \mathcal{L}_{invalid}. \tag{3.12}$$

In summary, the anchors that correspond to valid lane matches are penalized as to predict 1.0 confidence and all other anchors are penalized to predict 0.0 confidence using the cross entropy loss. The underlying assumption of penalizing the confidence predictions in this way is that lanes that found a valid match over the two frames are likely to be correct, while the lanes that do not are likely to be incorrect predictions. Furthermore, the predictions of the lane geometry of valid matches are penalized with l^1-norm to be consistent.

Note that the geometric part of the consistency loss penalizes the predictions of the network implicitly by computing the loss based on, for example, $\boldsymbol{x^m_{ext}}$ and $\boldsymbol{\hat{x}^l_{ext}}$ rather than the predictions $\boldsymbol{x^m}$ and $\boldsymbol{\hat{x}^l}$. However, each of the points in $\boldsymbol{x^m_{ext}}$ depends linearly on some two points of $\boldsymbol{x^i}$, so it's straight forward to compute the gradients from this formulation.

The described consistency loss entails two hyperparameters, namely the distance threshold d_{th} that defines which matches are valid and the probability threshold p_{th} that defines which predictions are considered positive predictions. The distance threshold should be set low enough such that the predictions of two adjacent lanes can never be considered valid, meaning that d_{th} should be smaller than $log(x+1)$ (again using Equation 3.6) where x is the typical lane width (in Sweden the minimum width of a standard lane is 3.25m [58]). Of course, d_{th} should not be too close to zero either, because then essentially no lanes will constitute valid matches. However, there is a quite large span of values between 0 and the typical lane width that could work well, and it is difficult to argue which value of d_{th} is most suitable without doing any experiments. On the other hand, it is certainly easier to argue which value of the probability threshold should work best. Since the consistency loss is based on the assumption that the predicted (and matched) lanes describe the underlying ground truth lanes as well as possible, the obvious choice of p_{th} is to set it to the value that resulted in the maximum F-score on the validation set. This is straight-forward to compute since, in the semi-supervised approach (explained in the next section), predictions on unlabeled data are always preceded by supervised training. In a sense, this value of the probability threshold results in the most accurate predictions of the network (best trade-off between precision and recall). Using this value of p_{th} and some appropriate value of d_{th} together with the loss function explained above will later be referred to as the *standard consistency loss*.

Although this choice of the hyperparameter p_{th} should be suitable for the consistency loss, it could also be interesting to investigate the effects of setting $p_{th} = 0$. In this case, all the 16 predicted lanes of the network are considered positive predictions. This implies that the 16 lanes from frame t will be matched with the 16 lanes from frame $t+1$, which makes it likely that the number of found valid matches greatly exceeds the number of ground truth lanes. Therefore, the underlying assumption that valid matches over the two frames are likely correct will not hold in this case, and the proposed way of penalizing the confidence predictions is not appropriate. In order not to make any such assumption in the case of $p_{th} = 0$, Equation 3.9 can be replaced by $\mathcal{L}_p^{m,l} = |\log(p^m) - \log(\hat{p}^l)|$ such that valid matches are penalized to predict the same confidence, rather than predicting 1 in confidence. However, any invalid matches can still be penalized to predict zero confidence. Using $p_{th} = 0$ and this alternative confidence penalization was investigated as an option to the *standard consistency loss* and will later be referred to as the *alternative consistency loss*.

This concludes the definition of the consistency loss function that is used to supervise the network on unlabeled data. What remains is to formulate a semi-supervised training scheme that allows for utilizing both labeled and unlabeled data during training, which is done in the next section.

3.6.2 Semi-supervised Training Scheme

As described in the previous section, the formulated consistency loss requires a matching between the predicted lanes of frame t and those of frame $t+1$. The matching is essential in the consistency loss formulation as it defines how the network will be penalized during training (one type of penalization is applied to anchors that found a valid match and another to those that didn't). Since the matching is done between the predicted 3D lanes, it is evident that the computed matching is dependent on the predictions of the network. To ensure that the matching between the anchors in any given unlabeled image pair doesn't change sporadically during the course of training, we propose computing the matching only once and thus keeping the matching (and the consistency loss function) constant during the entirety of the training. If the matching is recomputed at every epoch, the way that the consistency loss penalizes the network may change from one epoch to another, which makes it difficult for the network to converge.

Therefore, we propose the following method for training the model semi-supervised, which we call the *Basic scheme*.

Basic scheme:

1. Train the network on the labeled dataset until convergence.
2. Make predictions on the unlabeled dataset and compute the global optimal matching for each unlabeled image pair given the current predictions.
3. Add the unlabeled data for which the consistency loss can be used (the network found at least one valid match) to the training set.
4. Train further on both labeled and unlabeled data using the regular loss and the consistency loss respectively (since the matching is kept constant for each unlabeled image pair it never has to be recomputed during training).

One drawback of computing the matching only once is that a poor matching will effect the network negatively during the whole training. For instance, if there exist three ground truth lanes in a given image pair but the network only found two valid matches, the network will be trained to only detect these two lanes and disregard the third one. Therefore, it is essential that the matches are of good quality for the consistency loss to work well. To ensure this, one could instead only add the unlabeled data for which all of the network's predicted lanes constitute valid matches to the training set. The assumption that motivates this approach is that it is less likely that the network missed any ground truth lanes, or predicted too many lanes, if every predicted lane of the two frames constitute a valid match.

However, using this constraint on the unlabeled data used for training makes it likely that only a small subset of the unlabeled dataset will be added when semi-supervised training is started. It is therefore motivated to attempt adding more unlabeled data at some later stage during semi-supervised training. This approach has similarities with commonly used self-training methods as mentioned in Section 2.3, where only the confident predictions are used as pseudo-labels and more and more unlabeled data is added iteratively as the network's performance increases. To investigate whether this attempt of ensuring good quality of the added unlabeled data (and the found matches) is beneficial for semi-supervised training, we also propose the iterative training scheme defined as follows.

Iterative scheme:

1. Train the network on the labeled dataset until convergence.
2. Make predictions on the (remaining) unlabeled data and compute the global optimal matching for each unlabeled image pair given the current predictions.
3. Add the unlabeled data for which **all** of the network's predictions constitute valid matches to the training set.
4. Train further on both labeled and unlabeled data for a given number of epochs.
5. Iterate over step 2-5 until the unlabeled dataset is exhausted or no further improvements are made.

The iterative training scheme is also illustrated in Figure 3.9.

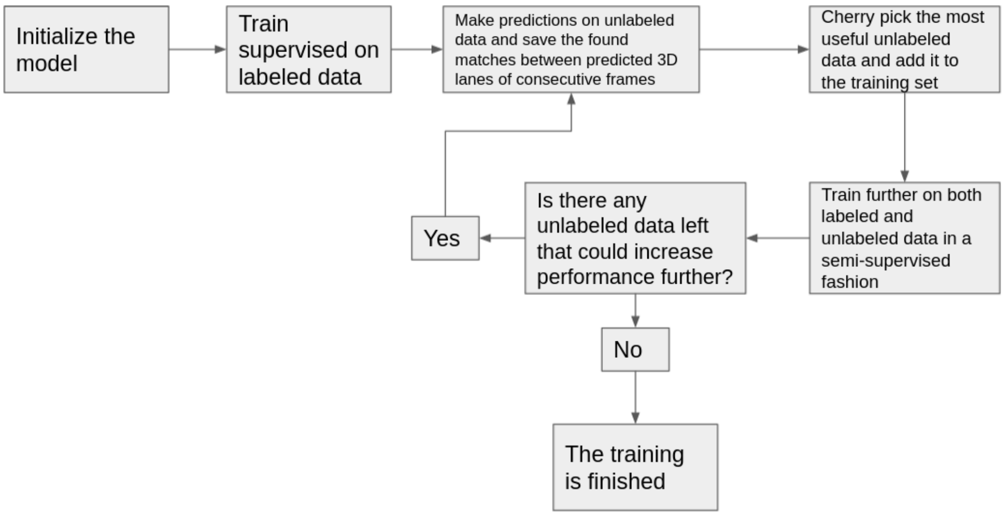

Initialize the model
Train supervised on labeled data
Make predictions on unlabeled data and save the found matches between predicted 3D lanes of consecutive frames
Cherry pick the most useful unlabeled data and add it to the training set
Train further on both labeled and unlabeled data in a semi-supervised fashion
Is there any unlabeled data left that could increase performance further?
Yes
No
The training is finished

Figure 3.9: Overview of the semi-supervised training scheme.

When training on the labeled data and unlabeled data simultaneously, it is necessary to decide how the supervised loss (from Section 3.4.2) and the consistency loss should be weighted compared to each other. For the possibility of varying the relative importance of the supervised and consistency loss during training, we introduced the hyperparameter α and computed the total loss during training according to

$$\mathcal{L}_{total} = (1 - \alpha)\mathcal{L}_{sup} + \alpha\mathcal{L}_c,$$

where $\mathcal{L}_{sup}$ is the supervised loss and $\mathcal{L}_c$ is the consistency loss.

3.7 Datasets

This section describes the collection/creation of the datasets used in this thesis. The training, validation and test sets consists of 3194, 510 and 510 labeled real-world images respectively. The unlabeled dataset used for semi-supervised training consists of 2525 sequences containing two images each and the synthetic dataset consists of 1609 labeled images.

3.7.1 Real-World Labeled Dataset

The real-world 3D lanes dataset was created by combining depth maps from aggregated LiDAR point clouds (the depth maps constitute ground truth depths of the pixels in the images) with 2D lane instance annotations and then applying some post-processing to refine the extracted lanes. A schematic illustration of the method is shown in Figure 3.10. Transforming the pixels that are annotated as lanes to 3D coordinates with the help of the camera intrinsic matrix and pixels depths results in 3D point clouds that correspond to the 3D ground truth lanes. Since the used 2D annotations were so called instance annotation, which constitute separate labels for each lane instance, the output is a separate point cloud for each annotated lane. Some post processing was then applied to these point clouds in order to remove noise from the data as well as creating a continuous representation of the lanes. A continuous representation is needed to be able to uniquely determine the lane's lateral and vertical offsets to the assigned anchor at every predefined position in the forward direction during training.

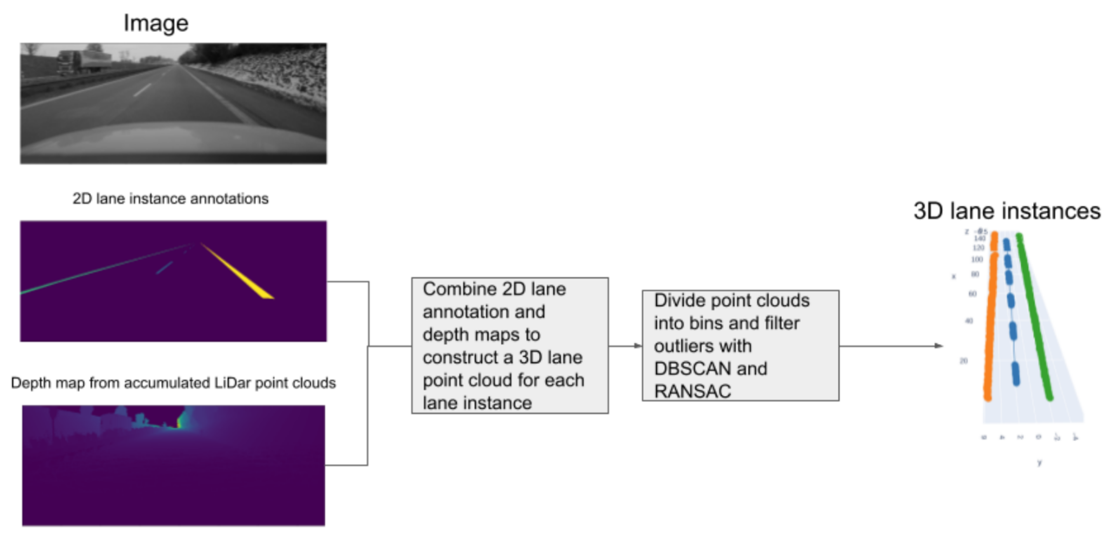


Image
2D lane instance annotations
Depth map from accumulated LiDar point clouds
Combine 2D lane annotation and depth maps to construct a 3D lane point cloud for each lane instance
Divide point clouds into bins and filter outliers with DBSCAN and RANSAC
3D lane instances

Figure 3.10: Overview of how 3D lanes are extracted from 2D lane annotations and depth maps from aggregated LiDAR point clouds.

In order to create continuous lanes each point cloud was divided into bins that each occupy 20 cm in the forward direction. The median of the points in each bin was computed and saved as the lane's position for this bin. Continuous lane lines was then created by simply connecting the computed median points of each bin with straight lines. The error introduced by approximating the lane's position with straight lines in between the median points of the bins is usually small since the point clouds are dense. However, it is not always the case that all the bins are occupied, especially when dealing with dashed lane markings, occlusions or lanes in the far range where information in the image is more scarce. In these cases, there may be gaps of several meters between the closest occupied bins. There is therefore a risk of introducing a non-negligible error when approximating the lane's position with a straight line between such distant bin points. However, since the lane's position needs to be defined at the prespecified y-values for training it is necessary to interpolate the lanes position at least a couple of meters (for example to overcome the gaps in dashed lane markings). To cover as many of the predefined y-values as possible, a quite generous threshold were used that allowed for connecting any bins that were less than 20 meters apart.

Since the depth maps from the accumulated LiDAR point clouds were not completely noise free there was also a need for filtering the point clouds for outliers. For this purpose, the density based clustering algorithm DBSCAN was applied. Due to the regularity of the typical lane geometry it is sensible to assume that the lane's lateral and vertical position changes slowly over distances in the forward direction. To leverage this assumption, the lateral and vertical axes were scaled up such that DBSCAN became more sensitive to changes in these dimensions. Therefore, if some points of the lane have different vertical or lateral position from most nearby points (in the forward direction) these will be classified as outliers. DBSCAN is effective in filtering outliers when the density of the outliers is much smaller than the density of the inliers (hence the name). However, the density of the inliers can be very low in the case of 3D lanes, especially in the far range, but this can also happen in the case of partial occlusions and dashed lane markings. To deal with sparse inliers in the far range, DBSCAN was applied in two different ranges separately, once in the close range and once in the far range, using different hyperparameters for the algorithm to match the general density of the lanes in both regions. Thereafter, RANSAC was also applied to filter any outliers that DBSCAN missed. This helps in cases where large parts of the data are faulty and therefore classified as inliers by DBSCAN due to high density. RANSAC instead leverages the fact that lanes locally are well estimated by lines and removes any points that deviate too much from the best line fit to the data. Therefore RANSAC can also remove dense clusters of faulty points. RANSAC was applied at three separate intervals (0-33, 33-67 and >67 meters in the forward direction) to make sure that not many points were removed simply due to lane curvature or other violations of the straight line assumption.

Finally, the ground truth pitch and height of the camera with respect to the local road plane was determined. In our case, a static pitch and height calibration was used for each separate drive of the vehicle. Any changes in pitch and height due to vehicle dynamics (e.g. large acceleration) was therefore not taken into consideration. Furthermore, the images were cropped and rescaled (we removed the top and bottom parts of the image that correspond to sky and vehicle hood) to reduce the number of computations during training. The final size of the images that are fed to the network is 962 (3848/4) times 274 (1094/4) pixels.

The real-world dataset was divided into a training set of 3194 images and a validation and test set of 510 images each. Since 3D-LaneNet had only been tested on highway roads before [7], it was decided to include only such images in the validation and test set (images taken on roads with speed limit at least 100 km/h in our case). However, the images in the training set come from both urban and highway scenes. Another reason for including only highway roads in the a validation and test set was that it makes sense to evaluate the model on long lanes that stretch over the full prediction range (≈6-100 meters) to estimate the close and far range predictions errors as well as possible.

3.7.2 Real-World Unlabeled Dataset

The collected unlabeled real-world dataset consists of 2525 sequences containing two images each. All image pairs were taken from highways (roads with speed limit of at least 100 km/h) and the time interval between any two pairs is between 0.5 and 2.5 seconds. The ground truth labels of this dataset consists of the camera pitch and height as well as the transformation between the vehicles position in the image pairs. The transformation was computed using processed GPS data that accurately measures the vehicle's movement between the two frames. The transformation allows for transforming the predicted 3D lanes of two consecutive images into a common coordinate system where they can then be compared.

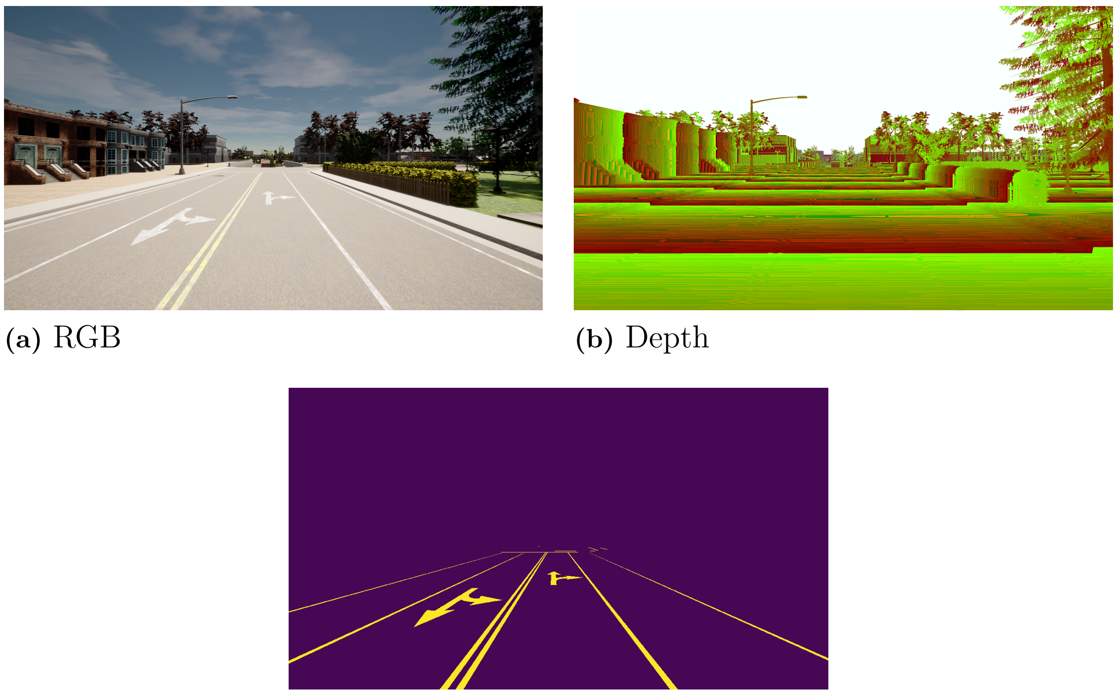
3.7.3 Synthetic Dataset

To extend our dataset further, we created synthetic data with the open-source simulator CARLA (version 9.11) [59].

The synthetic dataset is based on Towns 1-7, which are seven maps available for download from the CARLA [59] repository on GitHub. Each map can be modified by changing different environmental settings such as cloudiness, precipitation and sun altitude angle. CARLA [59] also provides a handful of options regarding vehicles and pedestrians/cyclists that are able to move around the environment and interact with each other, such as forming queues and following the rules of traffic. The data acquisition was made by the following steps:

1. Select a random Town and add randomly selected environmental settings.
2. Spawn 30 actors (either a type of vehicle or a pedestrian).
3. Attach a camera to one of the vehicles (randomly selected) that collects RGB, depth and segmentation images.
4. Collect 250 images at a frame rate of 2 images per second.

An example of the collected RGB, depth and segmentation images at a given timestamp is shown in Figure 3.11.



(a) RGB **(b)** Depth

(c) Lane segmentation

Figure 3.11: RGB (a), Depth (b) and Lane segmentation (c) images collected from CARLA [59].

The lane annotation from Carla is simply the pixels containing lane markings. By combining the lane annotations with the depth maps, we are able to create point clouds that correspond to the annotated lane markings in 3D-space. However, the labels gathered from CARLA [59] were simply all pixels containing road markings, including special road markings such as the arrows shown in Figure 3.11c. Hence some post-processing is needed to divide the markings into lane instances and remove the unwanted special road markings. For this purpose we adopted an iterative *RANSAC* algorithm. RANSAC is short for *Random sample consensus* and is an iterative method used to find linear dependence within data clusters. This is done by iteratively creating straight line fits based on a few points selected at random. From the straight line fit, the points within the vicinity are denoted as inliers and the rest outliers. This is done for a number of iterations and the line fit that gives the highest count of inliers is the selected best fit [60]. RANSAC is equipped with a residual threshold for the line fit and also a parameter for the number of iterations. These were set to 0.5m and 1 000 iterations respectively, and the number of points used for each fit was set to five.

Since the number of pixels for the lane markings increase vastly as we get closer to the camera, the point clouds were first binned into boxes with side length $bin_{size} = 0.2$ meters. All points in each bin were then replaced by a single representative point. This helps to even out the density of points over the whole point cloud. The binning was done by the following steps:

1. Find the maximum and minimum position of the point cloud in the x, y and z coordinates.
2. Create 3D-bins (cubes) with side length bin_{size} over the span:
$$([x_{min}, x_{max}], [y_{min}, y_{max}], [z_{min}, z_{max}]).$$
3. For each bin that contains at least one point, replace all points in the bin by a new one located at the mean in each dimension.

This helps for cases such as dashed lane markings, where the high density of points close to the camera can cause the RANSAC algorithm to falsely assign the closest dash of each lane instance to the same cluster, although they belong to different lane instances.

After binning the data, the RANSAC algorithm from *scikit-image* was applied, which separates the data into two parts; the most likely straight line fit and outliers. In order for the straight line to be considered a valid lane, it is required that it extends over $y_{ref} = 20$m so that it can be represented by a lane anchor. The straight line cluster is removed from the point cloud regardless if valid or not and saved as a lane instance if valid. The outliers are kept as the remaining point cloud. Lane instance points beyond $y = 100$m are discarded since these lie outside of the predictive range of our model. The RANSAC is applied on the remaining point cloud and repeated until one of the stop criteria is met.

Stop criterion

- There are too few points left in the lane markings data.
- The number of found lanes surpasses the number of lane anchors for the 3D-Lane detection model (in our case 16).

The extracted 3D lane instances from the example above are shown in Figure 3.12, where they are projected into the RBG image. As can be seen in this image, only fragments of the special road markings (arrows) remain after clustering the 3D point cloud into lane instances, and the method successfully distinguishes the apparent lane instances from each other. However, the lanes are rather straight in this example and the method is more prone to making errors in more complex scenarios, such as curved lanes, since the RANSAC algorithm is not well suited for these cases.

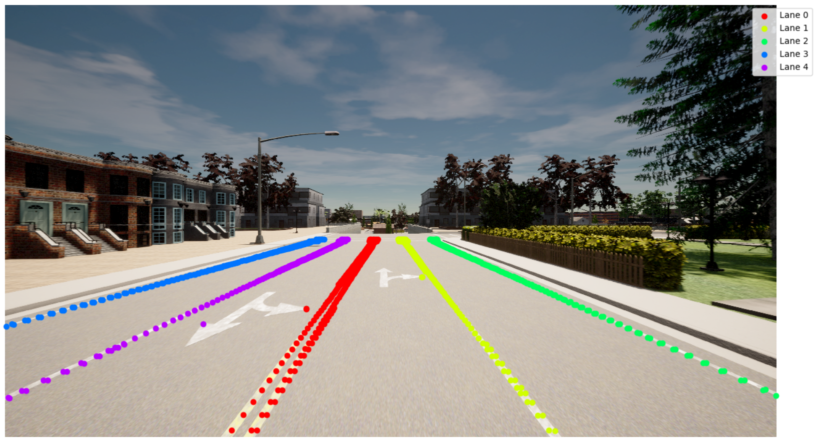


Figure 3.12: Resulting lane instance annotation from our iterative RANSAC algorithm, reprojected on the RGB image.

During training, the extracted synthetic 3D lanes are treated in the same way as the real-world 3D lanes. That is, using the same anchor assignment and representation of the lanes in the coordinate system C_{road}.

For a model to be able to learn efficiently, the quality of the ground truth is of great importance. After manual inspection 2/3 of the collected synthetic data was discarded, leaving 1609 images to train on.

3.8 Experiments

To investigate weather semi-supervised learning can be used for the task of 3D lane detection, 3D-LaneNet was first trained on the labeled real-world training set and evaluated on the validation set. Thereafter, the model was trained further using the proposed semi-supervised training method and evaluated again to see if this would increase the performance. Since adding unlabeled data to the training set can be expected to have greatest impact when the labeled dataset is relatively small, this experiment was repeated using only parts of the labeled dataset. Experiments were done with labeled datasets including 512, 1024, 1536, 2048 and 3194 images and this will later be referred to as training the model at different levels of supervision. In each case, the model was first trained on the labeled data for 100 epochs and then trained further using semi-supervised learning for five epochs with the chosen labeled data and all of the collected unlabeled data. To prevent overfitting during the first 100 epochs, the performance on the validation set was monitored during training and the best performing model over all epochs was saved/chosen for further semi-supervised training. On the other hand, it was assumed that overfitting would not occur during the five epochs of semi-supervised training and therefore the final model was simply chosen in this case.

Both supervised and semi-supervised training entails setting the hyperparameters to some sensible values during training. As explained in Section 3.4.3, the hyperparameters related to supervised training consists of (initial) *learning rate*, *learning rate decay* (LR_d), *learning rate interval* (LR_i), *regularization*, *batch norm*, *dropout* and *constant cam*. These hyperparameters determine the learning rate schedule and the regularization amount, as well as whether batch normalization, dropout and constant camera pitch and height should be used. For semi-supervised training, the hyperparameters consist of the distance threshold d_{th} related to the consistency loss, α that determines the relative importance of the supervised loss and the consistency loss, as well as the initial learning rate. The hyperparameters for both supervised and semi-supervised training were determined by doing randomized hyperparameter searches and selecting those that resulted in the best performance on the validation set.

After investigating the potential benefits of semi-supervised learning, two important aspects of the semi-supervised approach was investigated in detail. Firstly, the basic training scheme, which adds (almost) all unlabeled data at once, was compared with the iterative scheme, which entails cherry picking unlabeled data to be added successively during training. Secondly, the alternative consistency loss was compared with the standard consistency loss. As described in Section 3.6.1, the main difference between the alternative and standard consistency loss is that p_{th} is set to zero when using the alternative formulation while it is set to the value that gave rise to the highest F-score on the validation set when using the basic consistency loss. Due to limited computational resources and time, these investigations were done only at the lowest level of supervision.

Finally, the generated synthetic data was added during training in a separate experiment to study whether the synthetic data could increase the performance of the model. Since the main focus of this thesis is to develop an unsupervised training method, we don't investigate any of the domain adaption techniques presented by [13]. Instead, we used a straight forward approach of pre-training the model on the synthetic data and then training further on only the real-world data.

The validation set was used during the development of the methods in order to leave the test set untouched. Therefore, much of the analysis and conclusions made in this thesis are based on the performance on the validation set. In the end, the key results are however also validated on the test set. This is done to get an accurate estimate of the model's performance on unseen data and to give more credibility to the main conclusions.

4

Results

In this chapter, the results from the aforementioned experiments are presented. The test set was left untouched during the bulk of the experiments and therefore it is the performance on the validation set that is reported in this chapter if not stated otherwise.

4.1 Semi-supervised Training at Different Levels of Supervision

To study the effect of using semi-supervised learning at different levels of supervision, the model was trained on subsets of the labeled dataset and thereafter trained further on both labeled and unlabeled data in a semi-supervised fashion. The hyperparameters used for supervised training was decided based on a hyperparameter search that was done for the model trained on the full labeled dataset. During this search, which is described in detail in Section A.1.1 in Appendix, the hyperparameters were randomized and the parameters that resulted in the highest F-score on the validation set, without using any regularization techniques, are shown in Table 4.1. What is denoted as *Learning rate* is the initial learning rate during training.

Table 4.1: hyperparameters for supervised training that are denoted as *set 1*.

Learning rate	LR_d	LR_i	Regularization	Constant cam	Batch norm	Dropout
0.000123	6	40	0	0	0	0

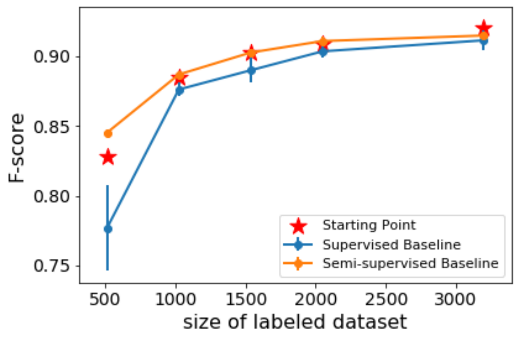
This set of parameters will be referred to as *set 1*. It was noted that a slight increase in performance could be achieved by using regularization and batch normalization, but in this case the semi-supervised learning did not work well, which is shown in Section A.1.2 in Appendix. Therefore, the parameters of *set 1* were used instead.

A hyperparameters search for semi-supervised training (described in Section A.1.2 in Appendix) was also completed on the highest level of supervision. In this case, the chosen hyperparameters were those that resulted in the largest decrease of geometrical error since no increase in F-score was observed. The chosen hyperparameters are shown in Table 4.2 and are later referred to as *set 2*.

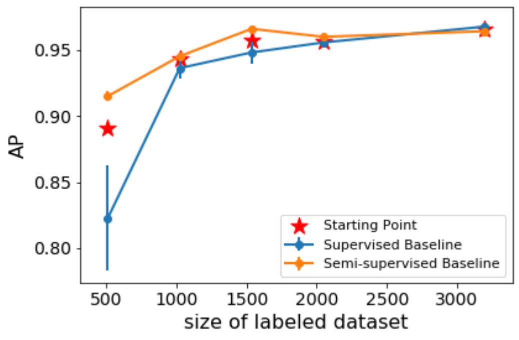
Table 4.2: hyperparameters for semi-supervised training that are denoted as *set 2*.

Learning rate	Distance Threshold (d_{th})	α
0.000003	0.779 ($\log(1.179 + 1)$)	0.552

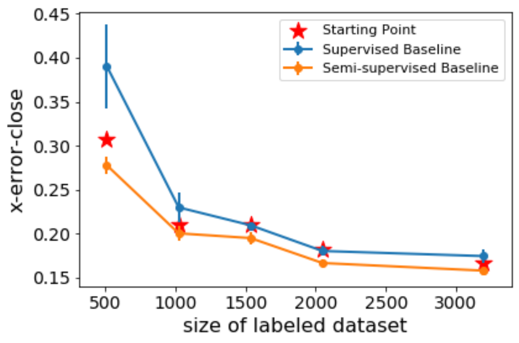
Continuing training in a semi-supervised fashion with hyperparameter *set 2* after training the models on labeled data with hyperparameters *set 1* gave the results shown in Figure 4.1. For each level of supervision, five models were trained supervised with hyperparameters of *set 1* until convergence (training for 100 epochs and preventing overfitting by monitoring the validation set performance as explained in Section 3.8). The model with the highest F-score at each level of supervision was then used for further semi-supervised training using hyperparameters of *set 2* for 5 epochs. Each chosen model was trained further with semi-supervised training five times since the training process is inherently stochastic. In Figure 4.1, the blue and orange graphs show the mean values and standard deviations of the five models trained supervised and the five models trained semi-supervised respectively. The red stars indicate the (common) starting point of all of the models trained semi-supervised.



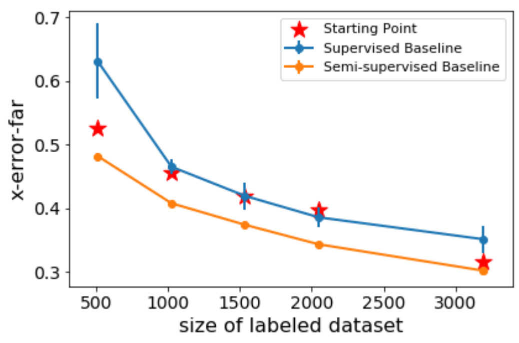
(a) F-score



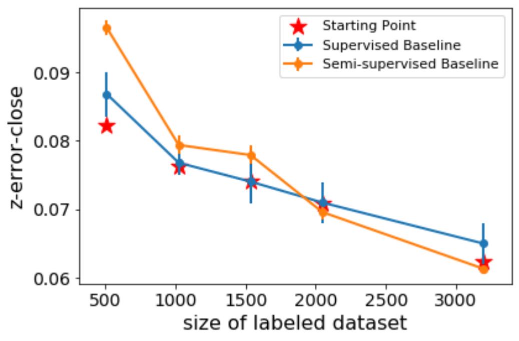
(b) AP



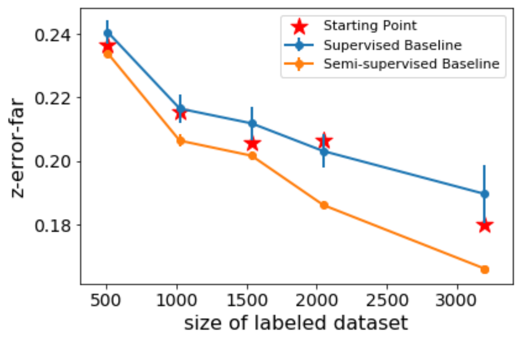
(c) x-error-close



(d) x-error-far



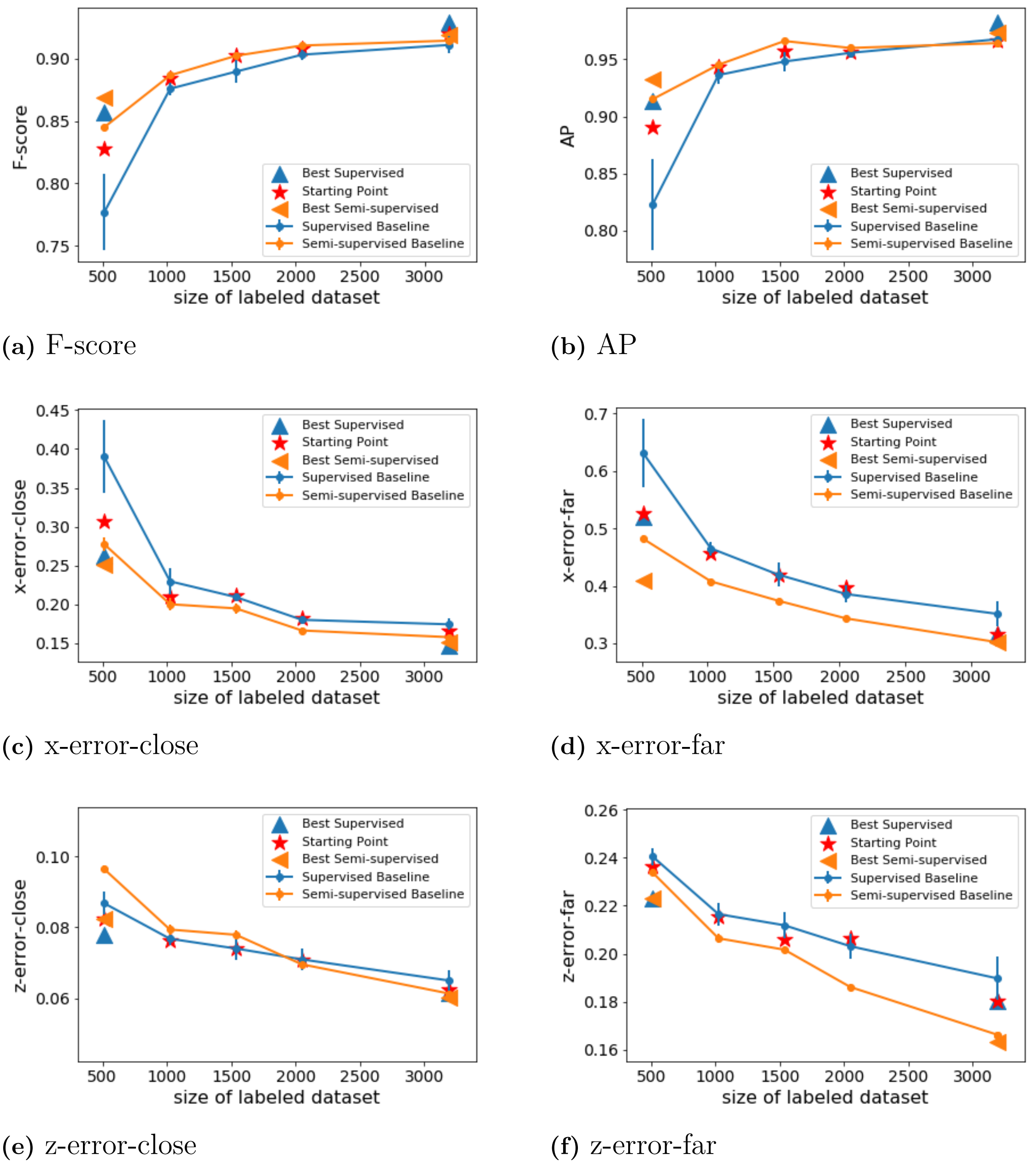
(e) z-error-close



(f) z-error-far

Figure 4.1: Performance comparison of models trained supervised with those trained semi-supervised.

In general, a slight performance increase can be observed across the board from utilizing semi-supervised training, except for z-error-close for low levels of supervision and F-score and AP for high levels of supervision where the performance actually dropped. One concern raised by these results is that the observed performance increase may be too subtle to be considered statistically significant. One reason for this may be that the chosen hyperparameters, both for supervised and semi-supervised training, were optimized at the highest level of supervision and may not be optimal also for lower levels of supervision. Therefore, a hyperparameter search for supervised and semi-supervised training was also done for the model trained on 512 labeled data to see if the performance could be increased further (see Section A.1.3 in Appendix for a detailed description of the parameter search). Figure 4.2 shows the same results as above but now also including the best performance achieved by supervised and semi-supervised models during the rigorous parameter search on the lowest level of supervision. For transparency, the best performance achieved during the supervised and semi-supervised hyperparameter searches on the full labeled dataset is also included.



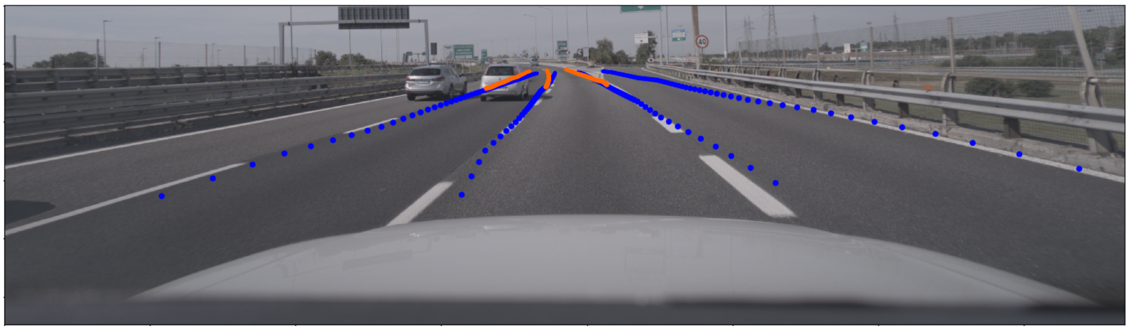
(a) F-score **(b)** AP **(c)** x-error-close **(d)** x-error-far **(e)** z-error-close **(f)** z-error-far

Figure 4.2: Performance of models trained only supervised and models trained semi-supervised. The best achieved performance from the hyperparameter searches on the lowest and highest levels of supervision are also included.

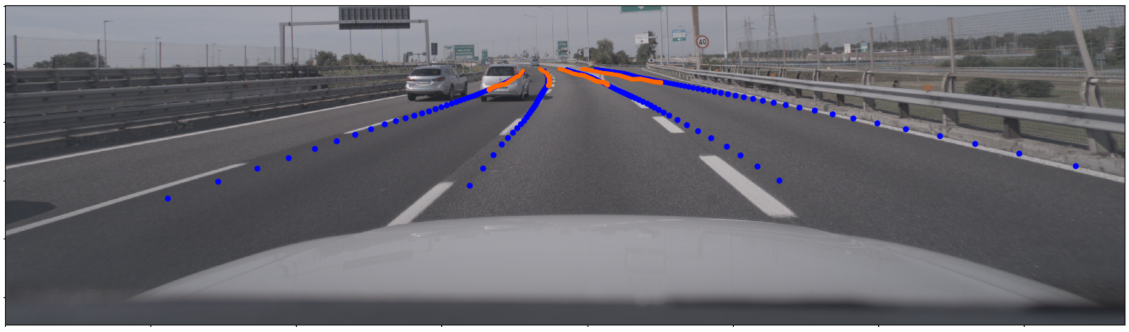
These results show that a general boost in performance was achieved when optimizing the hyperparameters of semi-supervised training for the model trained on 512 labeled images. In fact, an increased performance was observed in every metric compared to the results achieved when training semi-supervised with hyperparameter *set 2*. This concludes that hyperparameter *set 2* was not well suited for training the model on the lowest level of supervision. Also the best supervised model became a bit better after optimizing the supervised hyperparameters at the lowest level of supervision. Furthermore, the best performance achieved from semi-supervised training (orange triangle) exceeds the best performance achieved by supervised training (blue triangle) at the lowest level of supervision in the metrics x-error-far, F-score and AP while the performance between the models is very similar for x-error-close, z-error-close and z-error-far. The most noticeable performance increase is shown in x-error-far where semi-supervised training at the lowest supervision level accounts for an error decrease of more than 10 cm. At the highest level of supervision there was only a performance increase in z-error-far of about 2 cm while the performance did not increase in any other metric. As expected it seems like the semi-supervised approach works best when little labeled data is available.

Figures 4.3 and 4.4 show an example of the qualitative effect of training with the consistency loss on unlabeled data. The figures show the predictions of the network on a pair of unlabeled images (frame t and $t+1$) that was added to the training set during semi-supervised training. The predicted lanes for the two consecutive frames are transformed into the coordinate system of the first frame and shown projected into the image (frame t) in Figure 4.3 and in "3D" in Figure 4.4. In each figure, the predictions are shown both before and after semi-supervised training in order to illustrate how the network is trained on the unlabeled data. It is evident that the predicted lanes align better after semi-supervised training and it also seems like the lane curvature is predicted more accurately, although the ground truths are not known for this image.

What is surprising in this example is that the network predicts four lanes for both frame t and $t+1$ after semi-supervised training (four blue and four orange lanes), although only three matches were found for this example (there is only three orange lanes in the case of supervised training, so surely no more than three matches could be found). This does not reflect the goal of the consistency loss since the fourth, unmatched lane will be trained to predict zero in confidence (see the definition of $\mathcal{L}_{invalid}$ in Section 3.6.1 that describes how unmatched lanes are penalized). However, the network is trained on thousands of images simultaneously and it is the overall training loss that is expected to decrease during training, not every specific part of the training loss in every training example. Therefore, these kinds of deviations from the training objective can of course occur.

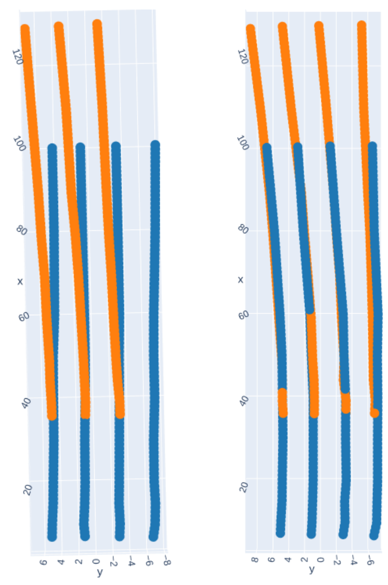


(a) Supervised



(b) Semi-supervised

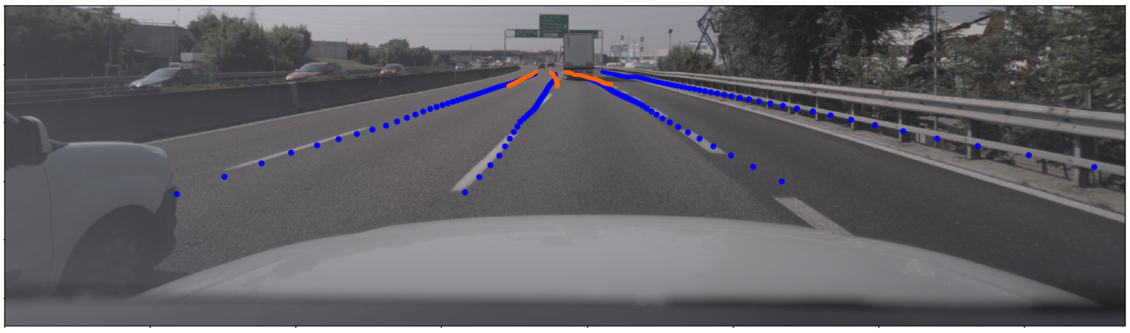
Figure 4.3: Projection of the predicted lanes from the supervised (a) and semi-supervised (b) model at the lowest level of supervision. The orange lanes are the predictions of frame $t+1$ while the blue lanes the predictions of frame t (the current frame).



(a) Supervised **(b)** Semi-supervised

Figure 4.4: The predicted lanes from the supervised (a) and semi-supervised (b) model at the lowest level of supervision. The orange lanes are the predictions of frame $t + 1$ while the blue lanes the predictions of frame t (the current frame).

Figures 4.5 and 4.6 show a similar example where instead the unmatched lane "dissappears" after training semi-supervised, which is the expected behavior of training with the consistency loss. Although the adjustments made in the lane geometry looks promising, this example highlights a drawback with the approach. That is the network will not be trained to predict any lanes that was missed during the matching procedure.


(a) Supervised

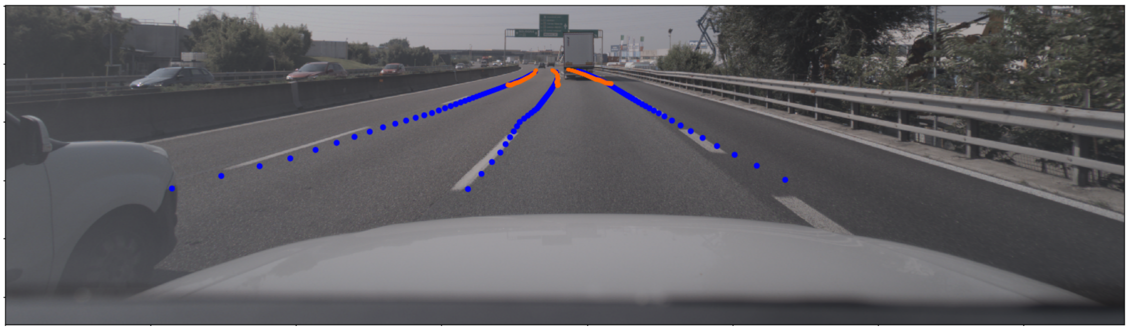
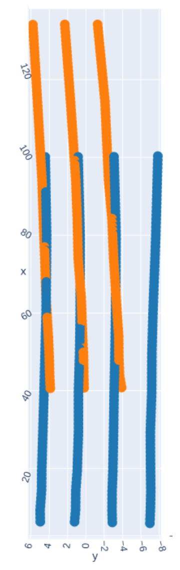
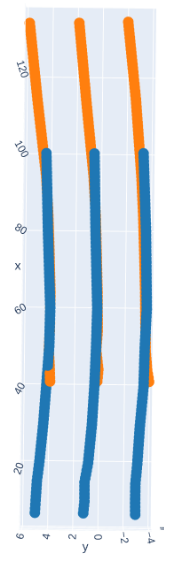

(b) Semi-supervised

Figure 4.5: Projection of the predicted lanes from the supervised (a) and semi-supervised (b) model at the lowest level of supervision. The orange lanes are the predictions of frame $t+1$ while the blue lanes the predictions of frame t (the current frame).



(a) Supervised

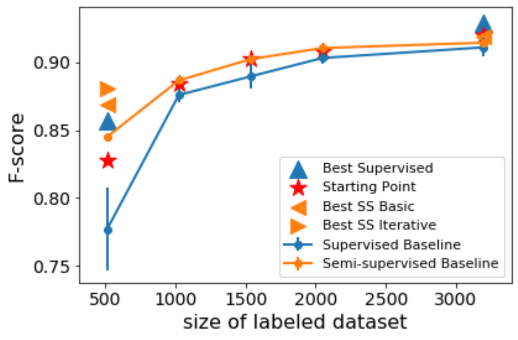


(b) Semi-supervised

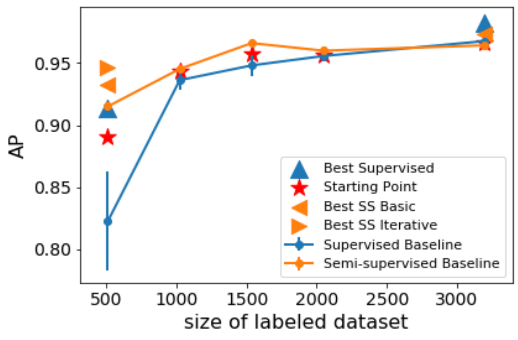
Figure 4.6: The predicted lanes from the supervised (a) and semi-supervised (b) model at the lowest level of supervision. The orange lanes are the predictions of frame $t + 1$ while the blue lanes the predictions of frame t (the current frame).

4.2 Iterative Training Scheme

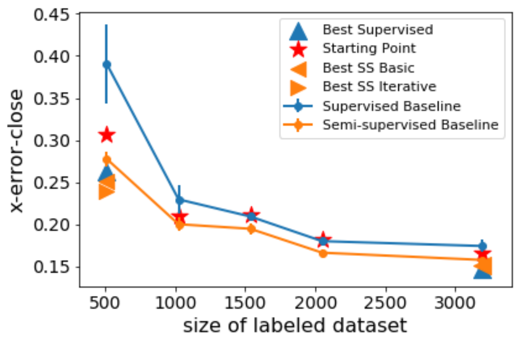
In order to investigate whether the iterative method for semi-supervised training is better than the basic method, another ten semi-supervised trainings were completed at the lowest level of supervision. The hyperparameters for semi-supervised training were randomized in the same way as before (see Section A.1.3 in Appendix), but this time the iterative training scheme was used instead of the basic scheme. The same supervised model as before was used as starting point for semi-supervised training and the unlabeled data was added at four different stages, letting the model train for five epochs after every addition to the training set. For one of the best performing models this results in adding around 2000 unlabeled images at the first iteration, 1000 images at the second and third iteration and 500 images in the last iteration. The results are shown in Figure 4.7. In the figure, *SS* is an abbreviation of *semi-supervised*, which will be used also later in the report. Furthermore, what is called *Best SS Basic* in this figure corresponds to the same results as *Best Semi-supervised* in Figure 4.2, now taking on a new name to distinguish it from the iterative approach.



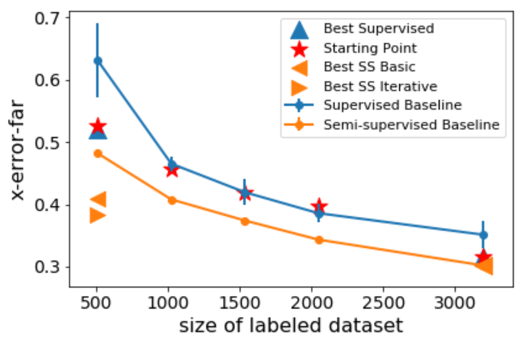
(a) F-score



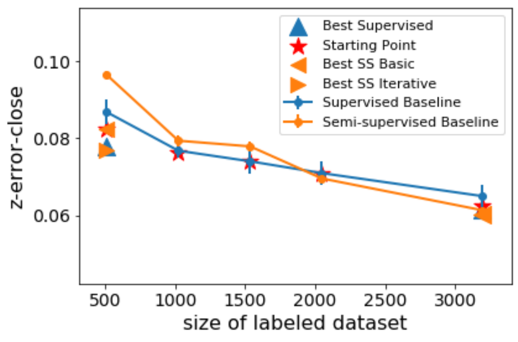
(b) AP



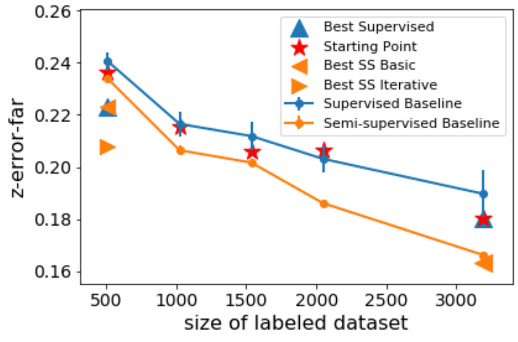
(c) x-error-close



(d) x-error-far



(e) z-error-close

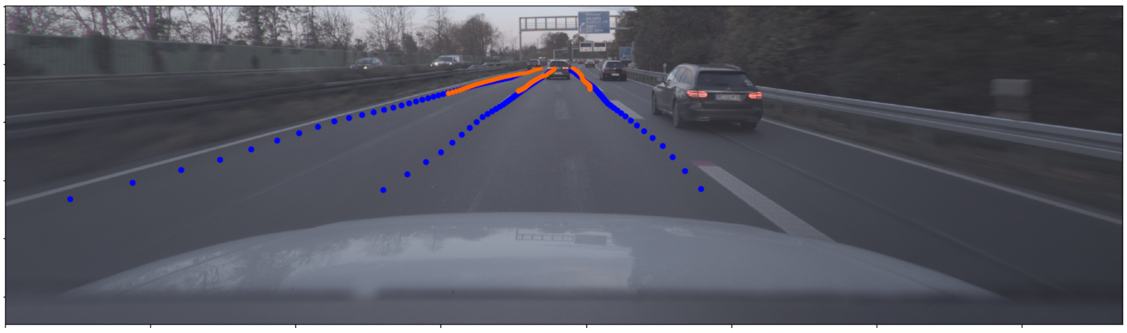


(f) z-error-far

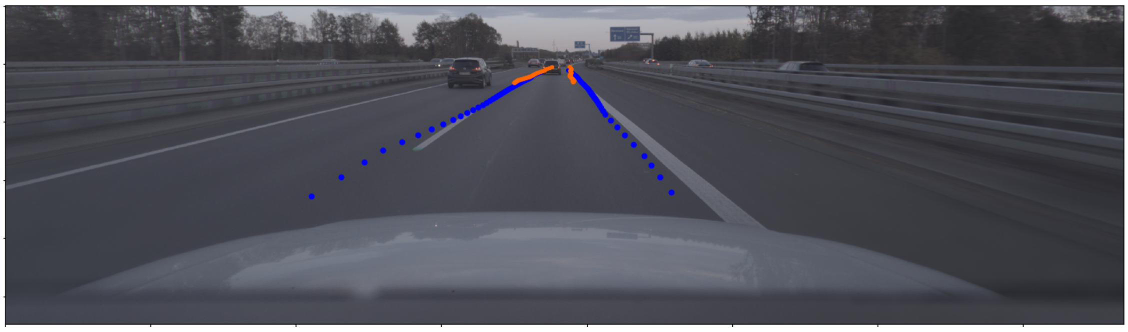
Figure 4.7: Comparing iterative training scheme with basic approach.

Slight improvements are observed across the board from utilizing the iterative training scheme instead of the basic one. F-score, AP and z-error-far are now competitive with the model trained supervised on 1024 labeled images. Furthermore, x-error-far decreased to 0.383 meters when using the iterative semi-supervised training scheme and it is competitive with the model trained on 2048 labeled images. The best supervised model at the lowest level of supervision only achieves 0.521 meters in x-error-far and the best supervised model at the highest level of supervision achieves 0.316 meters in x-error-far. The semi-supervised approach thus reduced this performance gap between the models at the lowest and highest level of supervision from over 20 cm to under 7 cm.

The purpose of the iterative training scheme was to ensure high quality of the added unlabeled data. It was hypothesized that only adding unlabeled data for which the model found valid matches for all its predictions would result in fewer missed and faulty predicted lanes, such as the right-most lane of Figure 4.5. Although this probably helped a bit, which can explain the improvements seen in Figure 4.7, some lanes are still missed as seen in the examples shown in Figure 4.8.



(a) An example from the unlabeled dataset where the right-most lane is missed.

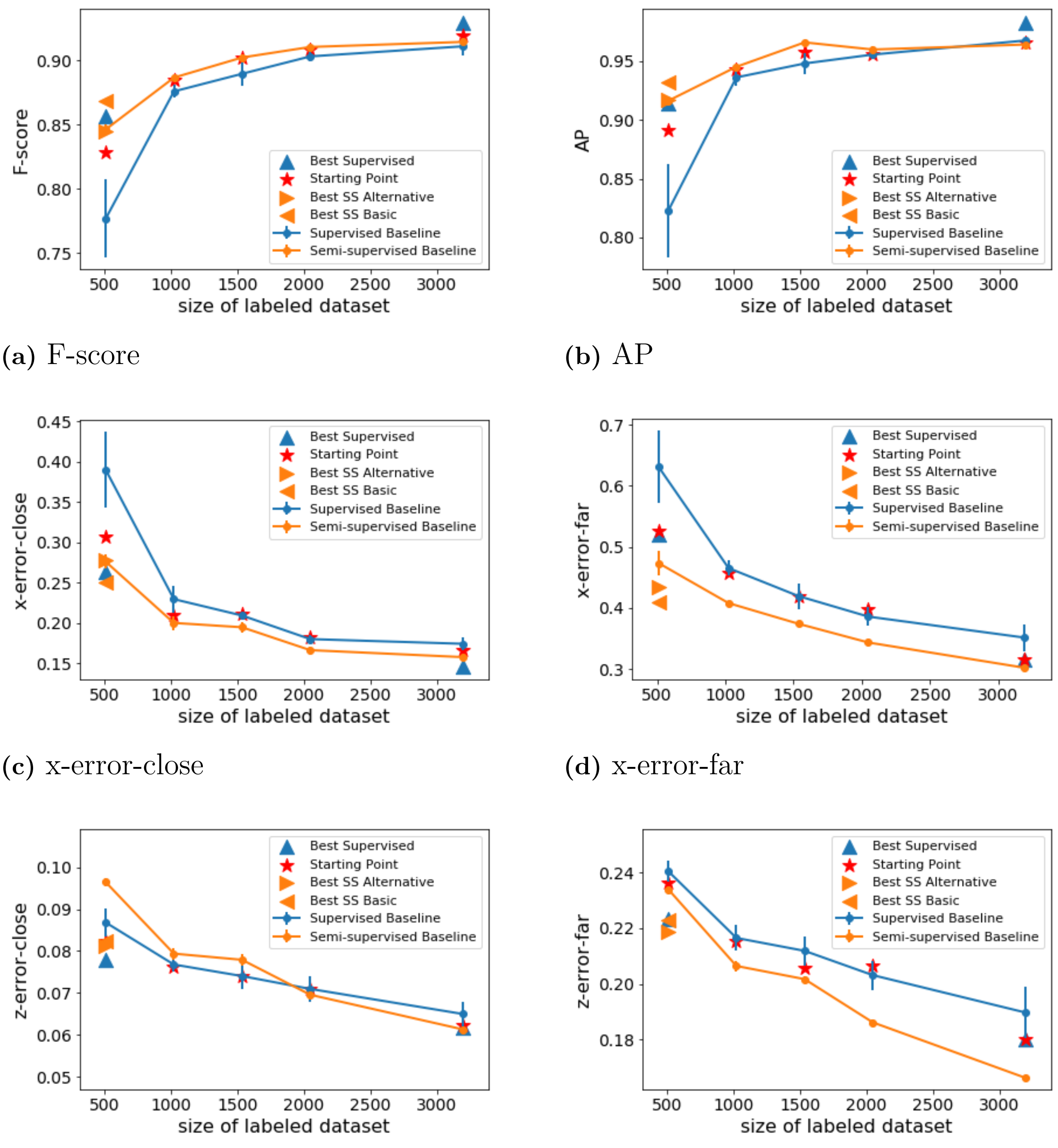


(b) An example from the unlabeled dataset where the left-most lane is missed.

Figure 4.8: Projection of the predicted lanes in two examples from the unlabeled dataset. The model found matches between all the predictions in both cases but still missed one lane in each image.

4.3 Alternative Consistency Loss

Twenty semi-supervised runs were made at the lowest level of supervision with the alternative consistency loss, which is defined in Section 3.6.1. The hyperparameters were randomized in the same way as before (see Section A.1.3) with the exception that p_{th} was set to zero. Furthermore, the iterative training scheme was not used for these experiments and therefore the results are compared with those of the previous experiments on the basic semi-supervised training scheme. It is evident that this method also achieves a decrease in x-error-far compared with the model trained on only labeled data, but it does not improve performance significantly in any of the other geometrical metrics. Furthermore, it is outperformed by the basic method for semi-supervised training in all metrics except for z-error-close and z-error-far, where the performance is similar. Semi-supervised training with the standard consistency loss also achieves increased F-score and AP compared to the supervised model while this is not seen when training with the alternative consistency loss. Based on these observations, it seems like the alternative consistency loss also penalizes the lane geometry in a reasonable manner, while the confidence scores are penalized more appropriately with the standard consistency loss, since no improvements in F-score and AP are seen when using the alternative consistency loss.



(a) F-score **(b)** AP

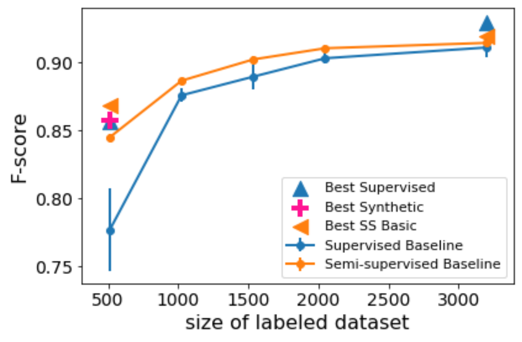
(c) x-error-close **(d)** x-error-far

(e) z-error-close **(f)** z-error-far

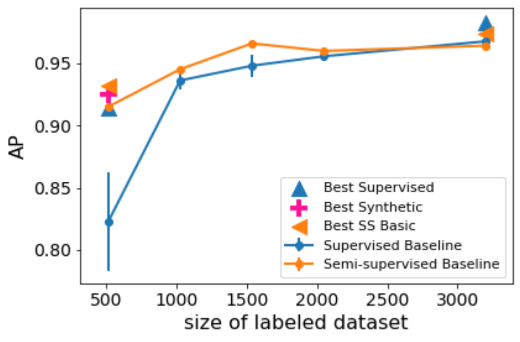
Figure 4.9: Results from using the alternative formulation of the consistency loss.

4.4 Leveraging Synthetic Data

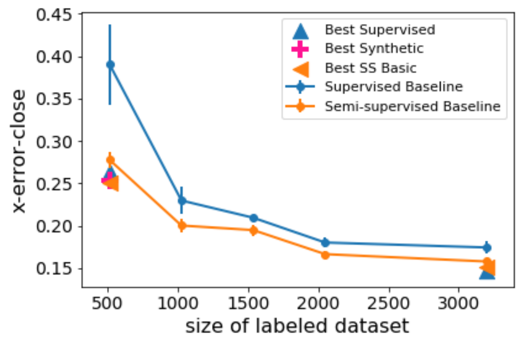
The generated synthetic data was also investigated as a tool for improving the model's performance at low supervision level. In this case, the model was pre-trained on the synthetic data and then trained further on only the real world dataset of 512 labeled images. In total 30 runs were made and the best achieved performance is shown in Figure 4.10



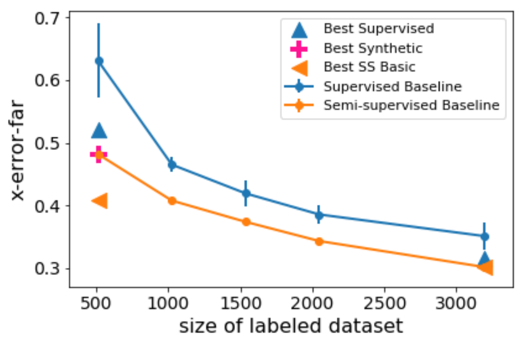
(a) F-score



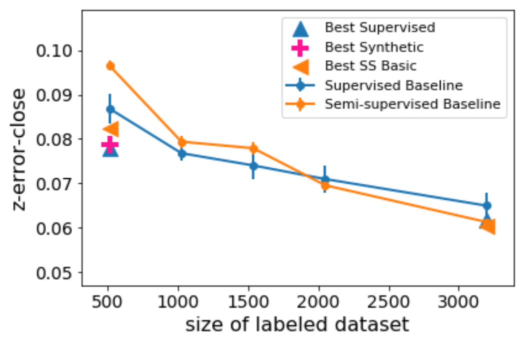
(b) AP



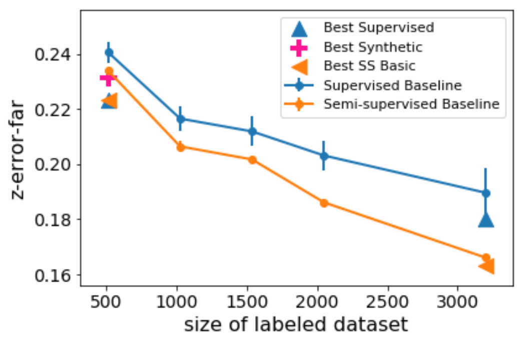
(c) x-error-close



(d) x-error-far



(e) z-error-close



(f) z-error-far

Figure 4.10: Results from training on both labeled real data and synthetic data compared with the models using only labeled real data and semi-supervised training.

In each of the 30 runs, 100 epochs were completed and overfitting was prevented by monitoring validation set performance as explained in Section 3.8. Hyperparameters were randomized in the same way as when using only the 512 labeled images (see Section A.1.3) and the same set of hyperparameters was used both when training on the synthetic and real world data. In Figure 4.10, small improvements are observed in x-error-far, F-score and AP while the performance in the other metrics decreased slightly. Overall, synthetic data does not seem to improve performance significantly.

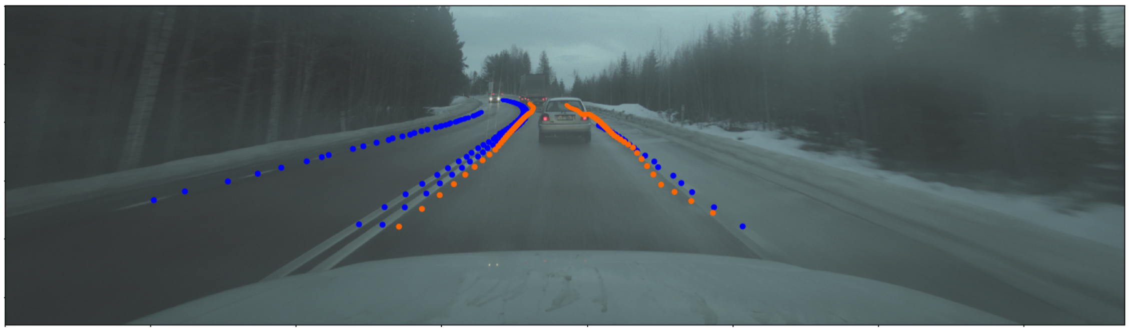
4.5 Validation on Test Set

The performance on the validation set was used when developing the method for semi-supervised training and thus all previously shown results are the performance on this set. Since a lot of experiments were made and the model with the best performance on the validation set was chosen in each case, the reported performance on the validation set is inherently inflated. Furthermore, since the aim of this project was to develop a semi-supervised training method, one may suspect that the performance of the semi-supervised models are more inflated than those of the supervised models, and thereby questioning the credibility of the results. In order to remove any such doubts and give a proper estimate of the models' performances on unseen data, the best supervised and semi-supervised models trained on 512 labeled images were also evaluated on the test set. The results are shown in Table 4.3 and show similar performance to that of the validation set presented in Figure 4.7. On the test set, x-error-far decreased from 0.509 to 0.397 and F-score and AP increased from 0.864 to 0.881 and 0.924 to 0.950 respectively when leveraging semi-supervised training. The changes observed in x-error-close, z-error-close and z-error-far are not very significant.

Table 4.3: The test set performance of the best (with respect to validation set performance) models trained supervised and semi-supervised with a labeled dataset of size 512.

Model	F-score	AP	x-error-close (m)	x-error-far (m)	z-error-close (m)	z-error-far (m)
Supervised	0.864	0.924	0.279	0.509	**0.076**	0.206
Semi-supervised	**0.881**	**0.950**	**0.254**	**0.397**	0.084	**0.190**

Figures 4.11 and 4.12 illustrate a qualitative comparison between the predictions of the supervised and semi-supervised models on a set of curved lanes from the test set that both models fail to predict accurately. However, improvements from using semi-supervised training are still observed in two aspects. Firstly, the semi-supervised model manages to predict the double lane marking while this is not done by the supervised model. Secondly, the predicted curvature in the far range is more pronounced after semi-supervised training and is closer to the ground truth lanes. It may not look like a large adjustment, but in fact the position of the lanes at the farthest away points changed with around **two meters** after semi-supervised training, which can be considered a huge improvement since the average lateral error in the far range is only in the order of half a meter as seen in Table 4.3.


(a) Supervised

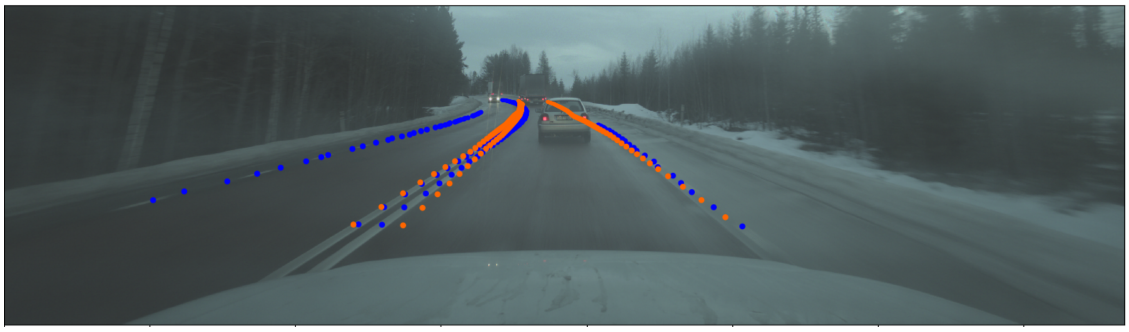
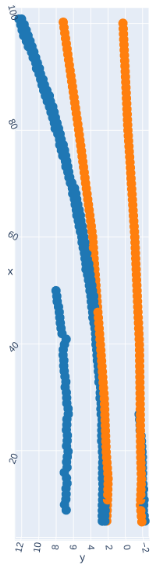
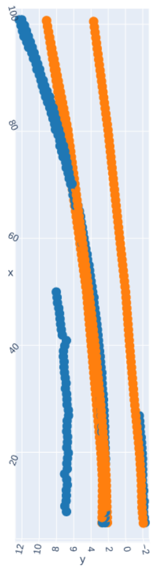

(b) Semi-supervised

Figure 4.11: Re-projected predictions on a test set example of the model trained supervised (a) and semi-supervised (b) at the lowest level of supervision.



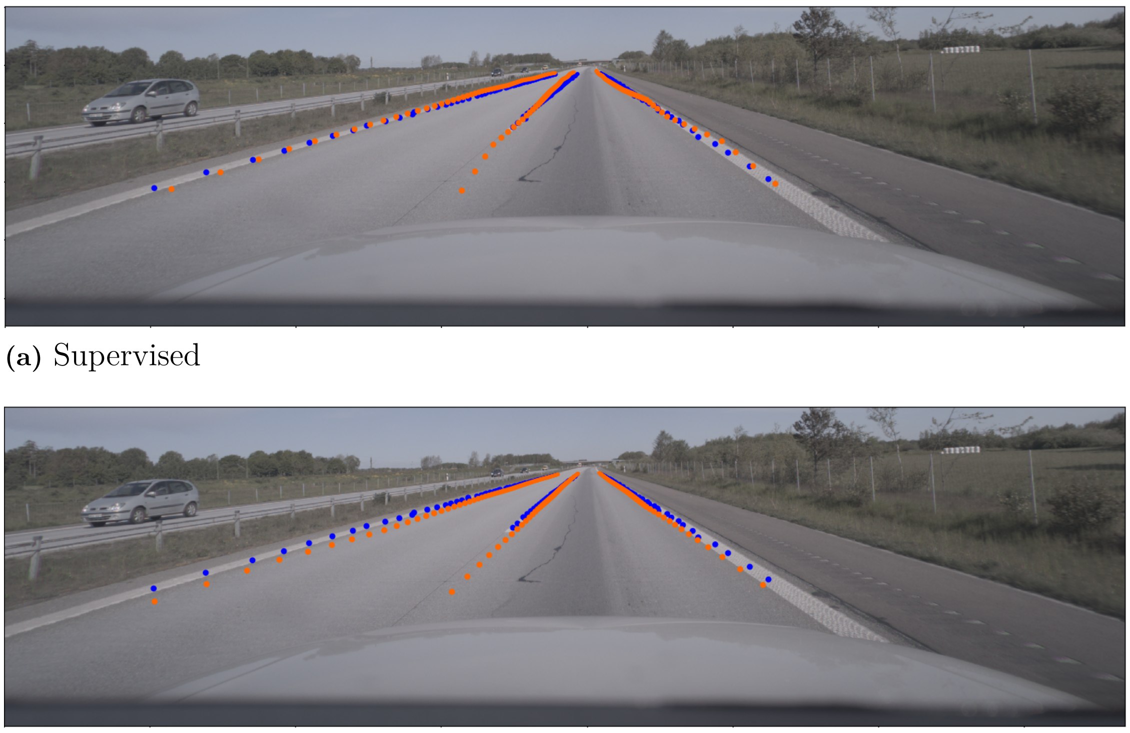
(a) Supervised



(b) Semi-supervised

Figure 4.12: 3D lanes predictions on a test set example (shown in Figure 4.11) of the model trained supervised (a) and semi-supervised (b) at the lowest level of supervision.

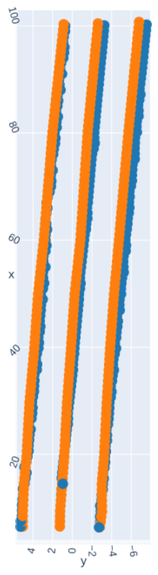
Figures 4.13 and 4.14 show another example from the test set where the predictions in the far range become more accurate after semi-supervised training. This example shows a typical case where both models predict the same set of lanes but the error in the far range is usually larger for the model trained only supervised, and there is little difference between the two models in the close range.



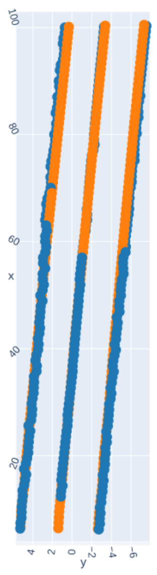
(a) Supervised

(b) Semi-supervised

Figure 4.13: Re-projected predictions on a test set example of the model trained supervised (a) and semi-supervised (b) at the lowest level of supervision.



(a) Supervised



(b) Semi-supervised

Figure 4.14: 3D lanes predictions on a test set example (shown in Figure 4.13) of the model trained supervised (a) and semi-supervised (b) at the lowest level of supervision.

5

Discussion

In this chapter, the analysis of the results as well as a discussion of future work are presented.

5.1 Analysis of Results

From Figure 4.7 it is clear that the semi-supervised approach leads to increased performance at the lowest level of supervision, which was also verified on the test set as shown in Table 4.3. Using the semi-supervised approach with 512 labeled and 2525 unlabeled image pairs results in competitive performance with the models trained supervised on 1024-2048 labeled images in several metrics. However, as noted in Figure 4.2 the performance of the supervised model using only 512 images increased a bit when optimizing the hyperparameters for this level of supervision. There is therefore a chance that the performance of the models trained supervised on the intermediate supervision levels (1024, 1536 and 2048) may also increase if the hyperparameters were optimized for them specifically, rather than using hyperparameter *set 1* that was found from the hyperparameter search at the highest level of supervision. However, since hyperparameter *set 1* was optimized at the highest level of supervision it is sensible to believe that it should work better at the supervision level 2048 than they do at 512, since the difference between the datasets is not that great in this case. Therefore, the performance at supervision level 2048 is not expected to increase a lot when optimizing the hyperparameters for this dataset, while the performance is likely to increase a bit more for the lower levels of supervision such as 1024.

Given the results shown in Figure 4.7, it is therefore reasonable to state that the semi-supervised model at the lowest level of supervision reaches competitive performance with the model trained supervised on 2048 labeled data in the metric x-error-far. However, as AP and F-score only barely reach the performance level of supervised training on 1024 labeled images we dare not state that much more than a few hundred labeled images can be replaced by the semi-supervised method for the purpose of achieving high F-score and AP with little labeled data. It would of course have been best if hyperparameter searches could have been done at each level of supervision, but due to limited time and computational resources we had to resort to this solution. In summary, what is certain is that the performance increased significantly with respect to F-score, AP and x-error-far when using semi-supervised learning at the lowest level of supervision, but it did not reach as good performance as the model trained on all of the labeled data.

One reason to the limited performance increase from semi-supervised training may be that the unlabeled dataset was not large enough. In particular, the relative size of the unlabeled dataset compared to the labeled dataset may be of importance. At the lowest level of supervision, the total size of the training set is increased to ≈ 10 times the size when adding the unlabeled data, and therefore it is very likely that the dataset in its entirety becomes more diversified when adding the unlabeled data. However, at the higher levels of supervision, the relative increase of the dataset size is not as significant, and it is more likely that the labeled data already covers most of the examples of the added unlabeled data. Since the goal of semi-supervised training essentially is to allow the network to train on many new examples (a more diversified dataset) it may be necessary to add a larger unlabeled dataset at the higher levels of supervision to see an increase in performance. Our unlabeled dataset of 2525 image pairs can be considered fairly small since unlabeled data can be collected with very little effort and there is a chance that improvements would be observed across the board if a larger unlabeled dataset had been used. Of course, this would entail longer training times, which is why we used a rather small unlabeled dataset during the development of our methods.

Furthermore, the results showed that the improvements in F-score and AP was not as pronounced as the improvements made in x-error-far. The semi-supervised approach therefore seems most suitable for adjusting the predictions of the lane geometry, rather than training the confidence predictions of the network. The example from the unlabeled training set shown in Figure 4.5 gives a hint of why this may be the case. Since the network's confidence predictions are penalized based on the found matches it happens that the network is faulty penalized when the matches don't describe the underlying ground truth lanes accurately. For instance, the network will be trained not to predict any lanes that are missed/not matched, which was the case of this particular example. On the other hand, the lane geometry is always penalized "correctly" even if some lanes are missed or faulty matched. In summary, the problem is that the number of ground truth lanes are unknown, and only if all of the lanes are predicted (and matched) will the consistency loss work in the intended way.

To alleviate this problem, the iterative semi-supervised training scheme was adopted. Although it resulted in improvements across the board, it did not solve the problem completely, which is seen in Figure 4.8 where one lane was missed in each example. Therefore, there is a good chance that developing an even better training scheme can increase the performance further. However, it should also be noted that the number of epochs was larger when using the iterative scheme rather than the basic one (twenty epochs instead of five). There is therefore a possibility that the enhanced performance of the iterative scheme simply comes from the fact that it spent more time training. This was not investigated explicitly since we only trained the models for just enough epochs to prove the feasibility of both methods, and therefore the performance of both methods could increase if longer trainings had been done.

We can therefore not establish that the iterative method certainly achieves higher performance than the basic method, although it seems likely given the identified problem with faulty lane matches and the iterative method's potential of dealing with this issue. To establish which of the two methods is most suitable for large-scale applications, it is also important to study the training speed. Although the iterative training scheme spends some time cherry picking the images, it saves time during training since less data is included in the training set. In our case of using only about 5000 unlabeled images, the overhead from cherry picking the data may be very large compared with the time saved during training, but in a practical case where hundreds of thousands of unlabeled images are available, it may be well worth while to exclude some unwanted images from the training set and thus speed up training.

Introducing the alternative consistency loss that uses $p_{th} = 0$ was another attempt of mitigating the effects of missed lanes in the unlabeled data. Although this method also resulted in a significant decrease in x-error-far compared with the supervised model as seen in Figure 4.9, it performed worse than the standard consistency loss in all metrics. The hope was that the alternative formulation of the consistency loss would be beneficial in scenarios where the standard consistency loss would penalize the confidence predictions inaccurately due to a missed or faulty match. For instance, in the case that the network predicts high confidence of a lane in one frame and low in the other, the regular consistency loss may not find a valid match between these lanes and penalize both to predict zero confidence. On the other hand, the alternative consistency loss would match the lanes (due to zero probability threshold) and penalize the predictions to be consistent, which would probably lead to the confidence predictions converging to some intermediate value (like 0.5) rather than zero. However, there is certainly a greater risk of finding inappropriate/faulty matches between the predictions when trying to match 16 lanes with another 16 lanes rather than matching a small set of (usually) 3-5 lanes which is the case of the standard formulation. This is probably the cause for the low general performance of the alternative consistency loss.

From the experiments on synthetic data, it is clear that training on the synthetic dataset of 1609 images did not increase the performance of the model significantly. The main reason for this is probably the difference in appearance between the synthetic and the real-world images. As mentioned in Section 2.4, Garnett *et al.* [13] adopted several domain adaption techniques to overcome the visual differences of the source domain (synthetic data) and target domain (real-world data) in their work on 2D lane detection. It is likely that some method of domain adaption also is required for our case of 3D lane detection. However, both [13] and [11] also concluded that training on labeled synthetic and real-world data resulted in better performance than only training on real-world data even without using any domain adaption techniques. Therefore, we believe that the difference in appearance between our synthetic and real-world dataset may not be the only reason for the failure in improving the performance, but rather also the quality of the synthetic dataset.

Since no instance annotations of lanes were available in the CARLA [59] simulator, we needed to apply post-processing in the form of binning and RANSAC to extract the lane instances. This may have resulted in reduced diversity of the dataset in the sense that the more difficult-to-extract lane instances, including curved lanes, merges and splits, may often have been excluded from the dataset, thereby leaving only the more simple cases of straight lanes over-represented. Since the model deals with the simple examples quite well already after training on the real-world data, there may not be much to gain from adding more examples of that type to the training set.

5.2 Future Work

The iterative training scheme was introduced to ensure that the consistency loss works as intended on the unlabeled data that is added during training, and our results indicate that it had a positive effect on the model's performance. One way to reduce the number of missed lanes/faulty matches further could be to use more than two consecutive frames and thereby give the network more chances to detect any inconsistencies. For example, if the network predicts the same lanes for ten consecutive images it should be very likely that these predictions indeed correspond to all ground truth lanes of the image. Although it should be possible to better leverage the assumed consistency of lanes through videos if more consecutive frames are used from each video sequence, this would also entail longer training times. Furthermore, since all frames from the same video are highly correlated (they are taken from close to the same geographical location and time) little is gained in the sense of diversifying the unlabeled dataset when adding additional frames from the same sequence. That said, there is a good chance that utilizing more than two frames could improve the performance and is worth investigating further. Furthermore, when applying semi-supervised training at larger scales, with access to potentially hundreds of thousands of unlabeled images, the training speed may become increasingly important. Therefore, the basic and iterative scheme should not only be evaluated based on performance, but also with respect to training speed, to see which of the methods is best suited for large-scale applications.

Another aspect of the consistency loss worth investigating more closely is the way that the predictions from the consecutive frames supervise each other. Currently, we assign equal importance to the predictions of both frames in the sense that they are both penalized equally if they do not align well in 3D space. Since the results show that the close range predictions are more accurate than the far range predictions (hardly surprising) one could argue that it would be better to assign greater importance to the close range predictions, and thus penalize the predictions of the first frame more than those of the second frame. One could in fact only penalize the lane geometry predictions of the first frame while not penalizing those predictions of the second frame at all, since it is believed that the far range predictions of the first frame cannot supervise the more accurate close range predictions of the second frame.

This idea of course makes most sense if the traveled distance between the frames is large. For instance, in the case that the traveled distance between the two frames is 90 meters, it does not make sense to adjust the ”very” close range predictions (0-10 meters) of the second frame based on the ”very” far range predictions (90-100 meters) of the first frame. In our case, the traveled distance between the frames was usually around 20-50 meters and therefore penalizing both predictions to some extent was probably a good idea. However, it would definitely be interesting to investigate a weighted loss function that takes into account the assumed importance of the two predictions (close and far range) in an attempt to improve the semi-supervised approach further. This could result in decreased lateral and vertical errors in both the close and far range. Furthermore, this idea of leveraging the close range predictions may be even more applicable if more than two images from each sequence are used. For instance, if ten images are used from each sequence and the vehicle travels around ten meters between each frame, the accurate close range predictions between 0 and 10 meters of each frame could effectively supervise the full extent of the predicted lanes (0-100 meters) in the first frame.

It would also be interesting to see what performance can be achieved if the unlabeled dataset is extended. In our work, we only used an unlabeled dataset of 2525 image pairs and in practice this could be extended significantly with very little effort. The downside of extending the dataset is of course increased training times, but if a larger dataset can increase the performance further, it could definitely be worth considering for real-world applications.

We would also like to see further research towards synthetic data generation with CARLA [59] and how it can be used for the purpose of lane detection. The main issue with our method for generating synthetic 3D lanes with CARLA [59] is that the proposed method for extracting lane instances often makes mistakes, which results in faulty labels. The main reason for this being that the topology of the lanes in the image is too complex and/or not linear enough for the RANSAC algorithm to be able to cluster the lane point clouds into lane instances correctly. Other clustering methods, such as DBSCAN, could be investigated as an alternative approach to the RANSAC algorithm, or one could also try using some heuristic methods for this purpose. However, the best solution would of course be if CARLA [59] would support lane instance annotations in their segmentation source code directly, thereby removing any needs for post-processing and clustering of the lanes. Some of the domain adaption techniques presented by [13] should also be investigated in the setting of 3D lane detection to overcome the differences in appearance between synthetic and real-world images.

6 Conclusion

The primary aim of this thesis was to investigate whether unlabeled data could be used to improve the performance of the 3D lane detection model 3D-LaneNet and in that way mitigating the need for large labeled 3D lanes datasets. The first research question was therefore formulated as follows:

Can training on unlabeled data improve the performance of the 3D lane detection model 3D-LaneNet?

To answer this question, we introduced a novel consistency loss that leverages the assumed spatio-temporal consistency of 3D lanes through videos and used this to train 3D-LaneNet on unlabeled data. Furthermore, an iterative semi-supervised training scheme was adopted, where the unlabeled data is added in turns. The results from our work show that training on unlabeled data in a semi-supervised fashion significantly improves the performance of the model when the size of the available labeled dataset is small. Improvements were mainly observed in the predictions of the lane geometry in the far range (40-100 meters) as well as in the detection metrics F-score and Average Precision. When using a labeled dataset of 512 images, the semi-supervised training accounted for a decrease of the lateral error in the far range from 50.9 to 39.7 cm. Improvements were also seen in F-score and AP that increased from 0.864 to 0.881 and from 0.924 to 0.950 respectively.

Since it is sensible to assume that the value of adding unlabeled data to the training set is dependent on the size of the available labeled dataset, a second research question was formulated as:

How does the (potential) performance gain from training on unlabeled data depend on the size of the labeled dataset?

To this end we trained 3D-LaneNet at different levels of supervision (using different amounts of labeled data) both supervised and semi-supervised to study the effect of adding unlabeled data to the training set when different amounts of labeled data is available. The results show that training on unlabeled data increased the performance significantly at the lowest level of supervision (512 labeled images) while little or no improvements were observed at the highest level of supervision (3194 labeled images). Semi-supervised training with 5050 unlabeled images at the lowest level of supervision resulted in competitive performance with supervised training on 2048 labeled images in terms of lateral error in the far range and with supervised training on 1024 images in terms of F-score and Average Precision.

In conclusion, it has been shown that the proposed consistency loss and semi-supervised training scheme for 3D lane detection has the potential of mitigating the need for expensive labeled 3D lanes datasets, but further research is needed to apply this on larger scale and successfully replace substantial amounts of labeled data.

The third and final research question of this thesis was stated as:

Can labeled synthetic data be used to improve the performance of 3D-LaneNet on real-world data?

To give an answer to this question, we generated synthetic data with the open-source simulator CARLA [59] and used it to pre-train 3D-LaneNet before training further on the labeled dataset. The result from this experiment was that the generated synthetic data did not increase the performance of the network significantly. The reason for this was probably the difference in appearance between synthetic and real-world images and the fact that the CARLA [59] simulator does not have support for lane instance annotations, which made it difficult to create a 3D lanes dataset of high quality.

Bibliography

[1] National Highway Traffic Safety Administration, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., Feb. 2015. [Online]. Available: `https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115` (visited on 07/10/2021).

[2] Association for Safe International Road Travel, "Road safety facts," Tech. Rep., 2020. [Online]. Available: `https://www.asirt.org/safe-travel/road-safety-facts/` (visited on 07/10/2021).

[3] A. B. Hillel, R. Lerner, D. Levi, and G. Raz, *Recent progress in road and lane detection: A survey*, 2012.

[4] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA grand challenge: the great robot race*. Springer, 2007, vol. 36.

[5] ——, *The DARPA urban challenge: autonomous vehicles in city traffic*. springer, 2009, vol. 56.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[7] N. Garnett, R. Cohen, T. Pe'er, R. Lahav, and D. Levi, *3d-lanenet: End-to-end 3d multiple lane detection*, 2019. arXiv: `1811.10203 [cs.CV]`.

[8] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool, *Towards end-to-end lane detection: An instance segmentation approach*, 2018.

[9] D. Chang, V. Chirakkal, S. Goswami, M. Hasan, T. Jung, J. Kang, S.-C. Kee, D. Lee, and A. P. Singh, *Multi-lane detection using instance segmentation and attentive voting*, 2020. arXiv: `2001.00236 [cs.CV]`.

[10] Y. Guo, G. Chen, P. Zhao, W. Zhang, J. Miao, J. Wang, and T. E. Choe, "Gen-lanenet: A generalized and scalable approach for 3d lane detection," *Lecture Notes in Computer Science*, pp. 666–681, 2020, ISSN: 1611-3349. DOI: `10.1007/978-3-030-58589-1_40`. [Online]. Available: `http://dx.doi.org/10.1007/978-3-030-58589-1_40`.

[11] N. Efrat, M. Bluvstein, N. Garnett, D. Levi, S. Oron, and B. E. Shlomo, *Semi-local 3d lane detection and uncertainty estimation*, 2020. arXiv: `2003.05257 [cs.CV]`.

[12] K. Behrendt and R. Soussan, *Unsupervised labeled lane markers using maps*, 2019.

[13] N. Garnett, R. Uziel, N. Efrat, and D. Levi, *Synthetic-to-real domain adaptation for lane detection*, 2020. arXiv: `2007.04023 [cs.CV]`.

[14] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 687–10 698.

[15] J. Jeong, S. Lee, J. Kim, and N. Kwak, "Consistency-based semi-supervised learning for object detection," 2019.

[16] C. Rosenberg, M. Hebert, and H. Schneiderman, *Semi-supervised self-training of object detection models*, Jun. 2018. DOI: `10.1184/R1/6560834.v1`. [Online]. Available: `https://kilthub.cmu.edu/articles/journal_contribution/Semi-Supervised_Self-Training_of_Object_Detection_Models/6560834/1`.

[17] A. Vezhnevets, V. Ferrari, and J. M. Buhmann, "Weakly supervised structured output learning for semantic segmentation," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 845–852. DOI: `10.1109/CVPR.2012.6247757`.

[18] N. Souly, C. Spampinato, and M. Shah, "Semi supervised semantic segmentation using generative adversarial network," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[19] G. J. B. Clément Godard Oisin Mac Aodha, *Unsupervised monocular depth estimation with left-right consistency*, 2020.

[20] Wikipedia contributors, *Activation function — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Activation_function&oldid=1026815756`, [Online; accessed 7-June-2021], 2021.

[21] C. D. Sa, *Lecture 7: Non-convex optimization*, 2014.

[22] Wikipedia contributors, *Gradient descent — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=1031739545`, [Online; accessed 9-July-2021], 2021.

[23] ——, *Backpropagation — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Backpropagation&oldid=1032593246`, [Online; accessed 9-July-2021], 2021.

[24] J. Brownlee. (Jan. 2021). "Gentle introduction to the adam optimization algorithm for deep learning," [Online]. Available: `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/` (visited on 07/09/2021).

[25] Wikipedia contributors, *Layer (deep learning) — Wikipedia, the free encyclopedia*, [Online; accessed 3-June-2021], 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Layer_(deep_learning)&oldid=1021579399`.

[26] ——, *Convolutional neural network — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=1026397751`, [Online; accessed 9-June-2021], 2021.

[27] B. Mehlig, "Artificial neural networks," *CoRR*, vol. abs/1901.05639, 2019. arXiv: `1901.05639`. [Online]. Available: `http://arxiv.org/abs/1901.05639`.

[28] Wikipedia contributors, *Batch normalization — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Batch_normalization&oldid=1017569708`, [Online; accessed 23-June-2021], 2021.

[29] ——, *Dilution (neural networks) — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Dilution_(neural_networks)&oldid=993904521`, [Online; accessed 23-June-2021], 2020.

[30] Z. Zou, Z. Shi, Y. Guo, and J. Ye, *Object detection in 20 years: A survey*, 2019. arXiv: `1905.05055 [cs.CV]`.

[31] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2014.

[32] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[33] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015. DOI: `10.1109/TPAMI.2015.2389824`.

[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. arXiv: `1512.02325`. [Online]. Available: `http://arxiv.org/abs/1512.02325`.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[37] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[38] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, *Microsoft coco: Common objects in context*, 2015. arXiv: `1405.0312 [cs.CV]`.

[39] Wikipedia contributors, *Information retrieval — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Information_retrieval&oldid=793358396`, [Online; accessed 10-July-2021], 2017.

[40] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," in *The elements of statistical learning*, Springer, 2009, pp. 485–585.

[41] K. Kluge, A. Arbor, and S. Lakshmanan, *A deformable-template approach to lane detection*, 1995.

[42] B. Ma, S. Lakshmanan, and A. Hero, *Road and lane edge detection with multisensor fusion methods*, 1999.

[43] R. Aufrère, R. Chapuis, and F. Chausse, *A model-driven approach for real-time road recognition*, 2001.

[44] Y. Wang, D. Shen, and E. K. Teoh, *Lane detection using spline model*, 2000.

[45] B. Yu and A. K. Jain, *Lane boundary detection using a multiresolution hough transform*, 1997.

[46] C. Kreucher and S. Lakshmanan, *Lana: A lane extraction algorithm that uses frequency domain features*, 1999.

[47] A. S. Huang, D. Moore, M. Antone, E. Olson, and S. Teller, *Finding multiple lanes in urban road networks with vision and lidar*, 2009.

[48] J. Douret, R. Labayrade, L. Jean, and R. Chapuis, "A reliable and robust lane detection system based on the parallel use of three algorithms for driving safety assistance," vol. 89, Jan. 2005, pp. 398–401. DOI: 10.1093/ietisy/e89-d.7.2092.

[49] S.-J. Wu, H.-H. Chiang, J.-W. Perng, C.-J. Chen, B.-F. Wu, and T.-T. Lee, "The heterogeneous systems integration design and implementation for lane keeping on a vehicle," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 2, pp. 246–263, 2008. DOI: 10.1109/TITS.2008.922874.

[50] M. Aly, *Real time detection of lane markers in urban streets*, 2014.

[51] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, *An empirical evaluation of deep learning on highway driving*, 2015. arXiv: 1504.01716 [cs.RO].

[52] Y. Wang, E. K. Teoh, and D. Shen, *Lane detection and tracking using b-snake*, 2003.

[53] S. Nedevschi, R. Schmidt, T. Graf, R. Danescu, D. Frentiu, T. Marita, F. Oniga, and C. Pocol, "3d lane detection system based on stereovision," in *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, 2004, pp. 161–166. DOI: 10.1109/ITSC.2004.1398890.

[54] S. Nedevschi, F. Oniga, R. Danescu, T. Graf, and R. Schmidt, "Increased accuracy stereo approach for 3d lane detection," in *2006 IEEE Intelligent Vehicles Symposium*, 2006, pp. 42–49. DOI: 10.1109/IVS.2006.1689603.

[55] M. Tian, F. Liu, and Z. Hu, "Single camera 3d lane detection and tracking based on ekf for urban intelligent vehicle," in *2006 IEEE International Conference on Vehicular Electronics and Safety*, 2006, pp. 413–418. DOI: 10.1109/ICVES.2006.371626.

[56] A. Pasad, A. Gordon, and T.-Y. Lin, *Improving semantic segmentation through spatio-temporal consistency learned from videos*, 2020.

[57] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[58] Association for Safe International Road Travel, "Råd för: Vägars och gators utformning," Tech. Rep., 2012. [Online]. Available: https://trafikverket.ineko.se/Files/sv-SE/12043/RelatedFiles/2012_180_rad_for_vagars_och_gators_utformning.pdf (visited on 07/14/2021).

[59] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[60] Wikipedia contributors, *Random sample consensus — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Random_sample_consensus&oldid=1031868520, [Online; accessed 16-July-2021], 2021.

A

Appendix 1

A.1 Hyper Parameter Search

A.1.1 Supervised Training

The first hyper parameter search constitutes 34 training runs on the full labeled dataset where the hyper parameters were chosen by random sampling in the following manner: *learning rate* was drawn form a uniform distribution on the interval $[5 \cdot 10^{-5}, 5.5 \cdot 10^{-4}]$, LR_d and LR_i were chosen randomly from the set $\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and $\{2, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$ respectively, *regularization* was chosen randomly to either 0 or drawn from a uniform distribution on the interval $[10^{-5}, 1.001 \cdot 10^{-2}]$, while *batchnorm*, *dropout* and *constant cam* was set to either true or false (1 or 0) with equal probability.

The five models that achieved highest F-score out of the 34 runs are presented in ascending order (best one last) in Table A.1.

Table A.1: The F-score and hyper parameters of the five models with the highest F-scores found in the initial hyper parameter search.

F-score	Learning rate	LR_d	LR_i	Regularization	Constant Cam	Batch Norm	Dropout
0.917	0.000449	2	10	0	0	1	1
0.918	0.000123	6	40	0	0	0	0
0.918	0.000534	9	45	0	0	1	1
0.919	0.000187	10	35	0	0	0	1
0.929	0.000185	7	50	0.002282	0	1	0

From these first set of experiments it was concluded that setting LR_i to some relatively high number (30-50), LR_d to 6-10 and learning rate to 0.0001-0.0003 seems to constitute a good learning rate schedule. It was yet unclear whether batch normalization enhanced the performance or not and although dropout was used in four of the top five performing models it was not used in the best one. However, it was clear that setting constant camera to 0 is best (that is using the predicted camera pitch and height to transform image view features to top view). Furthermore, zero regularization was used in all of the top 5 models except for the best one, which used a low regularization of ≈ 0.002. This resulted in trying smaller values of regularization in the next set of experiments (regularization was often 0.001-0.01 in the previous runs).

To search further for the best set of hyper parameters another 20 training runs were completed with the following randomized hyper parameters: LR_d and LR_i chosen randomly from the set $\{6, 7, 8, 9, 10\}$ and $\{30, 35, 40, 45, 50\}$ respectively, learning rate drawn from a uniform distribution on [0.0001,0.0003], dropout and batchnorm was chosen randomly to either True or False while constant cam was set to False in all experiments. Furthermore, a new randomization scheme was adopted for regularization to better cover values close to zero. Regularization was set to $r \cdot x$ where r was drawn randomly from the set $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ and x was a real valued random number drawn from the uniform distribution on $[0, 10]$.

After 20 completed runs with these settings, the top-5 F-score models were those of Table A.2. The new runs produced 4 new models in the top-5, but the best one was still the best performing model from the first set of runs. Therefore, it seems difficult to achieve any better performance than this and since computation resources and time is limited this concludes the hyper parameter search for supervised training.

Table A.2: The F-score and hyper parameters of the five models with the highest F-scores found in the second hyper parameter search.

F-score	**Learning rate**	LR_d	LR_i	**Regularization**	**Constant Cam**	**Batch Norm**	**Dropout**
0.926	0.000222	8	30	4.00E-06	0	1	0
0.926	0.000291	6	50	0	0	1	1
0.926	0.000188	9	40	2.00E-06	0	1	0
0.928	0.000161	9	40	0	0	1	1
0.929	0.000185	7	50	0.002282	0	1	0

The hyper parameters of the best model in Table A.2 will in coming sections be referred to as *set 3*, while the hyper parameters of the fourth best model in Table A.1 are those referred to as *set 1* in the report. The hyper parameter *set 3* and *set 1* gave rise to the model achieving the highest F-score with and without any regularization techniques respectively. These sets of hyperparameters are also summarized in Table A.3.

Table A.3: Hyperparameter sets 1 and 3.

Hyperparameter set	**Learning rate**	LR_d	LR_i	**Regularization**	**Constant Cam**	**Batch Norm**	**Dropout**
Set 1	0.000123	6	40	0	0	0	0
Set 3	0.000185	7	50	0.002282	0	1	0

A.1.2 Semi-supervised Training

The supervised model trained with hyper parameters of *set 3* (see Section A.1.1 for the definition) was trained further using semi-supervised training. The hyper parameters of the semi-supervised training were randomized and around ten runs were made on both the labeled dataset of size 512 and the full labeled dataset.

Learning rate was sampled as $x \cdot a + 10^{-6}$ where x was drawn from the uniform distribution on $[0, 1]$ and a was chosen randomly to 10^{-4} or 10^{-5}, α was sampled from a uniform distribution on $[0.1, 0.9]$ and d_{th} was set to $log(x + 1)$ where x was sampled uniformly on the interval $[0.3, 2.0]$. The probability threshold p_{th} was always set to the value that gave rise to the maximum F-score on the validation set from supervised training and the models were trained semi-supervised for 5 epochs. Results are shown in Tables A.4 and A.5 at the highest and lowest level of supervision respectively.

Table A.4: Results from the semi-supervised hyper parameter search on the highest level of supervision, using the model trained supervised with hyper parameters *set 3*.

Model	**x-error-close**	**x-error-far**	**z-error-close**	**z-error-far**	**F-score**	**AP**
Supervised	**0.157**	**0.339**	**0.070**	**0.210**	**0.929**	**0.981**
Semi-supervised 1	0.385	0.519	0.1282	0.327	0.752	0.791
Semi-supervised 2	1.745	1.152	0.1649	0.356	0.162	0.085
Semi-supervised 3	0.401	0.662	0.122	0.308	0.773	0.818
Semi-supervised 4	0.361	0.629	0.108	0.265	0.763	0.804
Semi-supervised 5	0.677	0.807	0.115	0.371	0.542	0.539
Semi-supervised 6	1.472	0.954	0.175	0.480	0.399	0.296
Semi-supervised 7	0.985	1.112	0.094	0.287	0.403	0.286
Semi-supervised 8	0.256	0.443	0.086	0.222	0.911	0.971
Semi-supervised 9	1.178	1.051	0.289	0.752	0.227	0.124
Semi-supervised 10	0.441	0.620	0.114	0.326	0.718	0.760
Semi-supervised 11	0.209	0.420	0.083	0.218	0.905	0.959

Table A.5: Results from the semi-supervised hyper parameter search on the lowest level of supervision, using the model trained supervised with hyper parameters *set 3*.

Model	**x-error-close**	**x-error-far**	**z-error-close**	**z-error-far**	**F-score**	**AP**
Supervised	**0.320**	**0.595**	0.103	0.278	**0.818**	**0.863**
Semi-supervised 1	0.735	0.948	0.116	0.272	0.552	0.526
Semi-supervised 2	0.657	0.915	0.123	0.282	0.563	0.505
Semi-supervised 3	0.682	0.919	0.121	0.304	0.560	0.548
Semi-supervised 4	0.519	0.861	0.111	**0.265**	0.655	0.624
Semi-supervised 5	0.863	0.998	0.305	0.337	0.435	0.291
Semi-supervised 6	0.663	0.956	**0.092**	0.270	0.537	0.485
Semi-supervised 7	0.781	1.037	0.143	0.293	0.458	0.331
Semi-supervised 8	0.754	0.899	0.205	0.372	0.437	0.270

As seen in Table A.4, the performance degraded in every metric when incorporating semi-supervised training at the highest level of supervision. At the lowest level of supervision the performance increased slightly in z-error-close and z-error-far as seen in Table A.5, but since F-score and AP dropped tremendously for all cases it can not be viewed as an improvement overall. The conclusion from these experiments is that semi-supervised training degrades the performance of the models trained with hyper parameter *set 3* on labeled data. We did not investigate very closely why this is the case, but we believe that the issue is related to batch normalization, dropout or regularization. One theory is that batch normalization doesn't work well for semi-supervised training due to the small batch size which equals two in this case.

The model trained with hyper parameter *set 1* on the full labeled dataset was also trained further using semi-supervised learning with hyper parameters randomized in the same way. In this case, 15 runs were completed and the results are shown in Table A.6. As seen in the table, no increase in F-score was achieved but there was a slight decrease in geometrical error.

Table A.6: Results from the semi-supervised hyper parameter search on the highest level of supervision, using the model trained supervised with hyper parameters *set 1*.

Model	**x-error-close**	**x-error-far**	**z-error-close**	**z-error-far**	**F-score**	**AP**
Supervised	0.166	0.316	0.0624	0.180	**0.920**	0.966
Semi-supervised 1	0.169	0.307	0.0615	0.167	0.917	**0.968**
Semi-supervised 2	**0.152**	0.315	0.062	0.170	0.918	0.966
Semi-supervised 3	0.158	0.342	0.062	0.171	0.919	0.965
Semi-supervised 4	0.164	0.313	**0.060**	0.166	0.915	0.959
Semi-supervised 5	0.156	0.309	0.061	0.169	0.911	0.961
Semi-supervised 6	0.179	0.376	0.074	0.186	0.901	0.954
Semi-supervised 7	0.161	0.359	0.063	0.171	0.912	0.964
Semi-supervised 8	0.160	0.313	0.062	**0.163**	0.909	0.958
Semi-supervised 9	0.178	0.346	0.068	0.200	0.898	0.954
Semi-supervised 10	0.156	0.329	0.065	0.176	0.916	0.966
Semi-supervised 11	0.168	0.348	0.068	0.172	0.912	0.961
Semi-supervised 12	0.160	0.308	0.062	0.168	0.917	0.966
Semi-supervised 13	0.158	**0.302**	0.061	0.166	0.915	0.964
Semi-supervised 14	0.159	0.359	0.070	0.191	0.919	0.973
Semi-supervised 15	0.170	0.326	0.067	0.186	0.910	0.962

Since it had been hypothesized that the semi-supervised training would be best suited for decreasing the geometric errors, we choose the hyper parameters for semi-supervised training based on the metric x-error-far. Table A.7 shows the five best performing models with respect to x-error-far and their corresponding hyper parameters. The best model in this aspect actually outperforms the corresponding supervised model slightly in all of x-error-close, x-error-far, z-error-close and z-error-far, although F-score and AP decreases slightly as seen in Table A.6.

Table A.7: The x-error-far and hyper parameters of the five models with the lowest x-error-far found in the hyper parameter search.

x-error-far	**Learning Rate**	**Distance Threshold (d_{th})**	$\boldsymbol{\alpha}$
0.302	3.00E-06	0.779	0.552
0.307	3.30E-05	0.494	0.394
0.308	2.00E-06	0.449	0.341
0.309	6.00E-06	0.922	0.569
0.313	8.00E-06	0.557	0.246

The hyper parameters used by the best performing model in Table A.7 are those referred to as *set 2* in the report. These hyperparameters are also shown in Table A.8.

Table A.8: Hyperparameter set 2.

Hyperparameter set	**Learning Rate**	**Distance Threshold (d_{th})**	α
Set 2	$3 \cdot 10^{-6}$	0.779	0.552

A.1.3 Extra Search at the Lowest Level of Supervision

To search for the best set of hyper parameters for supervised training at the lowest level of supervision, 20 training runs were completed with the same method for randomizing hyper parameters as in the latest search on the highest level of supervision. That is: LR_d and LR_i chosen randomly from the set $\{6, 7, 8, 9, 10\}$ and $\{30, 35, 40, 45, 50\}$ respectively, *learning rate* drawn from a uniform distribution on $[0.0001, 0.0003]$, *dropout* and *batchnorm* was chosen randomly to either True or False while *constant cam* was set to False in all experiments. Furthermore, *regularization* was set to $r \cdot x$ by first drawing a random number r from the set $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ and then drawing a random real valued number x uniformly from the interval $[0, 10]$.

Searching for appropriate hyper parameters for semi-supervised training at the lowest level of supervision was also done in the same way as at the highest level of supervision. That is: *learning rate* was sampled as $x \cdot a + 10^{-6}$ where x is sampled uniformly in the interval $(0, 1)$ and a is chosen randomly to 10^{-4} or 10^{-5}, α was sampled from a uniform distribution on $[0.1, 0.9]$ and d_{th} was set to $log(x + 1)$ where x is sampled uniformly at $[0.3, 2.0]$. The probability threshold p_{th} was always set to the value that resulted in the maximum F-score on the validation set. Twenty runs were completed and all models were trained semi-supervised for 5 epochs.