Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2019

End-to-end Road Lane Detection and Estimation using Deep Learning

Malcolm Vigren and Linus Eriksson

Master of Science Thesis in Electrical Engineering

End-to-end Road Lane Detection and Estimation using Deep Learning:

Malcolm Vigren and Linus Eriksson

LiTH-ISY-EX--19/5219--SE

Supervisor: **Mikael Persson, Ph.D. Student**
ISY, Linköpings universitet
Jon Bjärkefur, M.Sc.
Veoneer
Martin Nilsson, M.Sc.
Veoneer

Examiner: **Per-Erik Forssén, Docent**
ISY, Linköpings universitet

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Till morfar Börje

Abstract

The interest for autonomous driving assistance, and in the end, self-driving cars, has increased vastly over the last decade. Automotive safety continues to be a priority for manufacturers, politicians and people alike. Visual-based systems aiding the drivers have lately been boosted by advances in computer vision and machine learning. In this thesis, we evaluate the concept of an end-to-end machine learning solution for detecting and classifying road lane markings, and compare it to a more classical semantic segmentation solution. The analysis is based on the frame-by-frame scenario, and shows that our proposed end-to-end system has clear advantages when it comes detecting the existence of lanes and producing a consistent, lane-like output, especially in adverse conditions such as weak lane markings. Our proposed method allows the system to predict its own confidence, thereby allowing the system to suppress its own output when it is not deemed safe enough. The thesis finishes with proposed future work needed to achieve optimal performance and create a system ready for deployment in an active safety product.

Acknowledgments

This thesis concludes our time as undergraduate students at Linköping University, and to quote Jerry Garcia, what a long strange trip it's been. Since the end of this thesis also signifies the end of our time as students, some acknowledgements are in place.

First and foremost, our supervisors at Veoneer, Jon Bjärkefur and Martin Nilsson, deserve our deepest thanks. You have always looked out for us, and took time to help us both organize, prioritize and think. Your constant engagement and eager to see this work succeed has inspired us.

At Veoneer, Dennis Lundström deserves a special thank for his engaging interest in our work and competent help with the sometimes confusing world of neural networks.

Mikael Persson, our supervisor from Linköping University, is to be thanked for his enthusiasm and incredible knowledge - and how he shared them.

To our examiner, Docent Per-Erik Forssén, we direct our thanks for his deep interest in our thesis, sharp eye and sense for importance - you have made this thesis strive to be the best it can be.

We would like to take a moment to thank all of the wonderful people we have met during our time at Linköping University. The lectures, teachers, examiners and everyone else who have given us the knowledge to perform this thesis, the hard-working volunteers who keeps the student life amazing, and the fun comrades who makes everything easy.

Linus would like to direct his deepest thanks to Malcolm, for a fantastic time together. Your skills are amazing, and your dedication have kept this thesis floating. I regret our paths not crossing earlier. On that note, I would also like to thank Veoneer for pairing him with Malcolm - without their idea, this wonderful colaboration would not have happened. Finally, I would like to thank my family, friends and partner, without whose support and encouragement my work would in no way be what it has been - and this applies to everything.

Malcolm would also like do personally thank Linus for being an absolute pleasure to cooperate with in this project. He is a great listener, writes very readable code of high quality and has amazing mathematical skills. I would also like to thank my family, my girlfriend and my friends for showing support and interest in this project.

Linköping, Maj 2019
Malcolm Vigren och Linus Eriksson

Contents

Notation

Glossary

Word	Meaning
Ego-lane	The lane in which we are currently driving in.
Ego-left/right	The left/right edge of the lane in which we are currently driving in.
Neighbor-left/right	The edge of the left/right to the current neighboring lane.
True Positive (TP)	A detection corresponding to a detection in ground truth.
False Positive (FP)	A detection not corresponding to a detection in ground truth.
True Negative (TN)	Lack of a detection corresponding to a lack of detection in ground truth.
False Negative (FN)	Lack of a detection corresponding to a detection in ground truth.
Intersection-over-union (IoU)	The fraction of the intersection of two sets over their union. Used to determine similarity.

1

Introduction

For the past century, cars have been one of the main forms of transportation. It is, however, also one of the most dangerous. In 2013, 1.25 million people died in car crashes [28]. For this reason, automotive safety has for a long time been a serious concern for car manufacturers. Traditional safety equipment has usually consisted of *passive* devices, that is equipment such as airbags, crumple zones and seat belts, designed to reduce the dangers of crashes. Recently, with the advance of computers, cameras and sensors, *active safety* systems are becoming increasingly common. These are systems designed to prevent crashes from happening in the first place, and includes automatic emergency braking, lane departure warning and driver monitoring systems. [10]

Active safety features like lane departure warning require the vehicle to have some understanding of where the road lane boundaries are located and how they curve ahead. A conceptual example of such a system is shown in Figure 1.1, where the overlaid colors represent a systems output, showing pixel-wise detection of lane markings. This is called *semantic segmentation*, and is a common way of solving this problem [24].

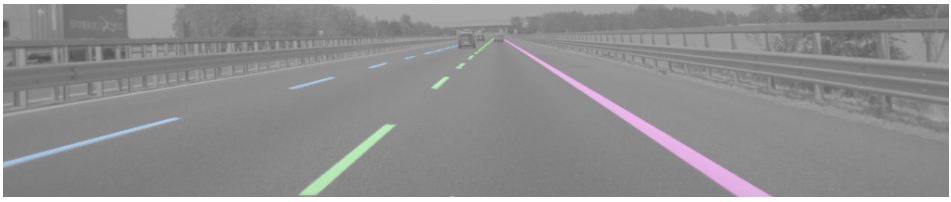


Figure 1.1: *Conceptualized output of a system finding road lane markings. Note how this system only detects the actual markings, and not the lane boundaries they define.*

Comparing to how a human driver would perceive the road, the output shown in Figure 1.1 has some fundamental differences. The main one is that the system only detects the markings, while a human driver can subconsciously connect them and imagine the lanes themselves as objects of importance. This concept is illustrated in Figure 1.2, where the output from a system more closely mimicking the human perception of the road is shown.

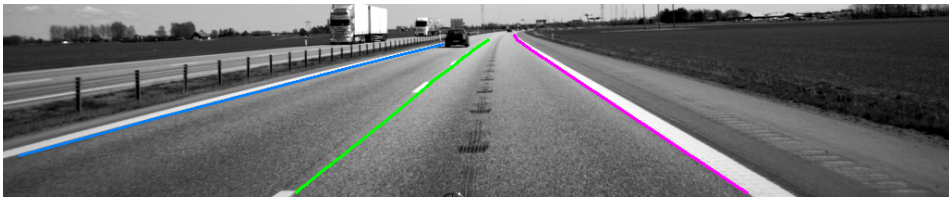


Figure 1.2: *Conceptualized output of a system finding road lanes. Note how this system detects and outputs the lane boundaries as defined by the markings, and not the markings themselves.*

Note how Figure 1.1 and Figure 1.2 differs, in that the system in Figure 1.2 does not output the position of the markings, but the lanes described by them, while the system shown in Figure 1.1 outputs the exact position of each pixel corresponding to a lane marking, but does not output anything for the pixels between two subsequent dashed markings.

It is possible to connect the segmented lane markings outputted in Figure 1.1 to form a similar representation to the one shown in Figure 1.2. Such post-processing would however require hand-written algorithms, the development of which is slow and tedious. If the system could directly output the final representation of the lanes, there would be no need to create and fine-tune these algorithms. Therefore, this thesis aims to create a system which could do this, and compare it to a classical semantic segmentation solution. This is to be implemented using *deep learning*, described below.

1.1 Introduction to Deep Learning

A common method of solving problems which are easily conceptualized by humans, but difficult to break down into an explicit procedure, is to use a *neural network*. This is a kind of *machine learning* system, which is inspired by the biological brain, and uses *layers* of *neurons* to calculate an output [21]. A *neuron* in this context is a summation of different weighted inputs, which is fed to an *activation function* in order to create an output. The activation function is some non-linear function used to give the network the ability to output non-linear solutions, which would otherwise be impossible given the linear sums. A layer is a group of neurons, fed either with input data, called the *input layer*, fed with data from a previous layer, called a *hidden layer*, or feeding data out of the system, called the *output layer* [11].

These networks are *trained* to output the wanted data given the input. This means that rather than explicitly describing how to arrive at a computational result, as with classical algorithms, the network *learns* how to arrive at a result by feeding it data from a *training dataset*, containing input data and corresponding desired output. This process involves designing a *loss function*, describing how similar the predicted output is to the wanted one. Based on this loss, the weights on the neurons are optimized using some optimization algorithm, where mainly *Stochastic Gradient Descent (SGD)* or its successor *ADAM* is used. [11] The optimization is done iteratively by feeding the network with samples from the training data, and performing an optimization step. This is repeated until wanted performance is achieved.

Especially useful for computer vision-related task are the type of networks called *Convolutional Neural Network* (CNNs). These networks do not connect all neurons in one layer with all neurons in the next, allowing for spatial locality. This gives the behavior of a *feature extractor*, where the network can extract both fine features, such as an edge, and coarse ones, such as a human, from a given image. These features may then be combined to create the desired output.[17, 22] Using multiple convolutional layers in a CNN allows for efficient feature extraction, resulting in the concept of *deep neural networks*, or *deep learning*.

Given the data available and the concepts described above, this thesis seeks to implement the systems using CNNs.

1.2 Motivation

This thesis project is done at Veoneer in Linköping, which is one of the leading companies in active automotive safety. This problem is of great interest for Veoneer, specifically for their active systems that make use of lane detection. The current implementation relies on machine learning algorithms coupled with a number of hand-made algorithms. This could be streamlined by developing a holistic approach, where the CNN directly outputs some representation of the lanes, such that less post-processing is needed in order to interpret the output of the system. This would reduce the complexity of the system and require less expert knowledge to maintain it.

1.3 Aim

The aim of this project is to design a deep neural network which inputs camera images and directly outputs some representation of road lanes. This representation allows one to view the system as an end-to-end system, in the sense that the representation needs minimal post-processing to determine the lines corresponding to the positions of the lanes in the image.

The study focuses mainly on achieving the best performance of the end-to-end model, as shown in Figure 1.2 within the time frame, and comparing it to the traditional semantic segmentation solution, as shown in Figure 1.1. This ap-

proach requires post-processing to produce the actual lines which represent the lanes.

1.4 Research Questions

The following questions are to be answered in this thesis:

1. How well does the end-to-end system predict the existence and positions of lanes in the image, compared to a semantic segmentation based system?

2. Which advantages does the end-to-end representation have over the semantic segmentation representation?

3. Is it feasible to let the end-to-end system estimate a confidence measure in the predicted lane positions?

4. How should the semantic segmentation ground-truth data be pre-processed for the end-to-end representation?

1.5 Delimitations

This section will list the delimitations of this study, describing what will not be considered.

- The study is limited to exploring different deep convolutional neural networks for road lane descriptions. Other machine learning methods will not be considered.

- The study will not try to determine the lane markings continuity type, that is, if the marking is dashed or solid. Neither will the study concern itself with whether the marking is a painted marking or a Bott's dots marking, or the color of the markings. Only the positions and existence of the lanes are relevant.

- The study will only focus on detecting at maximum the four closest lane markings, corresponding to the lane currently driven in and the neighboring lane on each side.

- The study will limit itself to only consider neural networks using ENet[27] as a backbone, as described in Section 4.2.1.

- The study will only consider frame-by-frame methods, thus limiting itself to using the current image frame as the sole input data for the system.

1.6 Author Contributions

This thesis has two authors, and each author has had main areas of responsibility. Though most of the work and ideas have been provided by both authors, Linus has been mainly responsible for solving the lane change discontinuity problem and outputting of regression confidences. Malcolm has been mostly responsible for the data pre-processing, network architectures and the semantic segmentation representation. The relevant sections for each contribution is displayed in Table 1.2.

Area	Sections	Author
Introduction	1	Both
Theory	2	Both
Data Pre-processing	4.3, 6.5	Malcolm Vigren
Lane Representation	4.1	Malcolm Vigren
Neural Network Architectures (not confidences)	4.2	Malcolm Vigren
Performance Evaluation	4.5,	Malcolm Vigren
Supporting Points with Confidence	4.1.2, 6.6	Linus Eriksson
Loss Gradient	4.2.4	Linus Eriksson
Confidence Dependent Regression	4.2.5	Linus Eriksson
Lane-change Discontinuity	4.4, 6.4	Linus Eriksson
Semantic Segmentation	4.1.3, 6.7	Malcolm Vigren
Training Results	5	Both
Validation Loss Minimum	6.2	Linus Eriksson
Horizon Loss Discussion	6.3	Linus Eriksson
Discussion of method	6.8	Both
Conclusions	7	Both
Training Predictions	Appendix A	Both
Lane Change Comparison	Appendix B	Linus Eriksson
Gradient Calculation	Appendix C	Linus Eriksson
Gradient Extreme Values	Appendix D	Linus Eriksson
Full Network Architectures	Appendix E	Both

Table 1.2: *The main contributions by each author*

2 Theory

This chapter describes the relevant theory for the study. It should be noted that the descriptions in this chapter will be brief and general, and that the authors warmly recommend the book Deep Learning by Goodfellow et al.[11], which covers the concepts of neural networks in good detail and gives the necessary fundamental backgrounds in mathematics and information theory.

2.1 Overfitting

Just as a polynomial of degree $N - 1$ can fit N data points perfectly, without describing the trend between and outside the points, it is possible for a sufficiently large network to achieve perfect performance on all the data it is fed with without performing well on unseen data. This behavior is called *Overfitting*, where a network fits the data it has trained on well but fails to generalize to unseen data. Therefore, during training, the available data with known output is divided into a *Training Set*, a *Validation Set* and a *Test Set*. The *Training Set* is used to perform the optimization, while the *Validation Set* is used to repeatedly evaluate how the network performs on data on which it has not been optimized. The parameters giving the lowest loss on the *Validation Set* (the *Validation Loss*) are then used to test the network on the *Test Set*. This final test on the test set lets different networks be benchmarked to each other, and gives a final indication of performance on unseen data.

2.2 Loss Function

The *Loss Function* used when training a neural network needs to have certain properties. This follows from the use of gradient-based optimization. Such a

feature is to not be constant everywhere, since such a function has a gradient equal to 0. Based on this, a loss function should not have a *Vanishing Gradient* anywhere. This corresponds to the function not having asymptotes, close to which its gradient approaches zero. Should a loss function exhibit a vanishing gradient, optimization risks being slow, and sometimes impossible given computational constraints in numerical accuracy. An optimal loss function has only a single minimum, which allows Stochastic Gradient Descent (SGD) to converge with high probability. On real world problems, such loss functions are next to impossible to create, given the complexity of the optimization problems.

2.3 Multi-stream Neural Networks

The purpose of a *Multi-stream Neural Network* is to let the "stream" of data flow separately through some parts of the network. This is conceptualized by copying the output from the network at some layer, and feeding it to two or more new layers, instead of the classical one-layer-to-the-next approach. This aims to let the network extract different features for different purposes, instead of forcing it to use the same data for all outputs. The idea of a multi-stream neural network is best thought of as a network splitting into several other networks, which may or may not merge together again. This river-like analogy is connected to the concept of seeing the data as "flowing" through the network, and a multi-stream approach lets it branch out into different paths.

2.4 Smooth Maximum

The purpose of a *Smooth Maximum* is to create a smooth approximation for the function $\max(\mathbf{x})$. One such function is the $S_\alpha(\mathbf{x})$ function, defined as

$$S_\alpha(\mathbf{x}) = \frac{\sum_{i=1}^{n} x_i e^{\alpha x_i}}{\sum_{i=1}^{n} e^{\alpha x_i}} \tag{2.1}$$

where the parameter $\alpha > 0$ is used to control smoothness.[20] Letting $\alpha < 0$ in the smooth maximum function results in the corresponding *Smooth Minimum* function.

3

Related work

This section describes relevant previous efforts to solve the problem, as well as other literature relevant for the project.

3.1 Lane Detection using Deep Learning

This section will describe work related to the concept of lane detection using deep learning, presenting several approaches.

3.1.1 LaneNet

One method involving deep learning was proposed by Neven et al. [24], where the image is input to two neural networks. The first network, called "LaneNet", is a network which performs segmentation of the lanes and pixel embeddings, to produce a labeled set of lanes. Before lane curves are fitted through the results from LaneNet, the lanes are projected via a homography onto a "bird's eye view", that is, a view in which the road is viewed from above. This is to make sure that the road can be represented with polynomials of low degree. To perform this homography, the homography matrix H is learned by the second neural network, called "H-Net". H is used to project the labeled lanes from LaneNet to a bird's eye view, from which lane curves are estimated using second degree polynomials. These polynomials are finally projected back using H^{-1}. The authors reported an accuracy of 96.4% on the tuSimple benchmark, discussed in Section 3.7.1, and 52 frames per second performance on an Nvidia 1080 Ti with an image size of 512×256.

In this thesis, focus laid on an end-to-end approach with simplicity. Therefore, a solution including a homography transformation, as the one described by Neven et al., was not investigated.

3.1.2 Raw Pixel to Steering Command

Another attempt to build a completely end-to-end system based on a Convolutional Neural Network (CNN) in the context of self driving cars is presented by Bojarski et al. [4]. In this article, the authors have succeeded in developing a neural network that takes as input images from cameras on the car and directly outputs a steering parameter that controls the car. In the article they show their process of data collection through driving and recording, data selection of sequences with more curves and data augmentation by creating images with artificial shift and rotations. In order to train the neural network they used a simulator for offline training before testing the results on a real car.

Since this thesis focuses on the detection and position estimation of road lanes, the steering based output proposed by Bojarski et al. was not deemed relevant to implement.

3.1.3 Simple CNN

A similar attempt to train a CNN to output steering angles directly from images was done by Zhilu Chen and Xinming Huang [6]. Here, they use a simple CNN, consisting of only three convolutional layers and two fully connected ones. They also train the system on a relatively small dataset, consisting of only 2.5 hours of video. To deal with this, they used several techniques to make the best use of the available data. One was realizing that their training data was unbalanced in the sense that most of the highway driving was done on straight roads, which could lead the model to always output small steering angles, so they oversampled the images with curved roads. They also added dropout layers to prevent overfitting.

As discussed in Section 3.1.2, the steering angle output was not of interest in this thesis, and therefore the method proposed by Zhilu Chen and Xinming Huang was not tested.

3.1.4 Spatial CNN

Pan et al. proposes a network focusing further on the spatial relationship expected in images, presenting a novel network type called *Spatial CNN* which proposes replacing the standard convolution with slice-by-slice convolutions. [26] Their implementation achieved a 1st place on the tuSimple benchmark 3.7.1, with an accuracy of 96.53%.

The Spatial CNN approach, while showing promising results and an interesting concept, was not deemed simple enough to validate the necessary implementation time. It was also concluded to be outside of the scope of this thesis, based on its complexity.

3.2 Uncertainty Prediction using Deep Learning

This section will present some work related to the concept of letting a neural network predict its own uncertainty.

3.2.1 Uncertainty Types

Kendall and Gal presents two major types of uncertainty possible to model, *Aleatoric* uncertainty connected to the noise of the observation, and *Epistemic* uncertainty connected to the ignorance surrounding which model generated the data collected. [18] They propose performance increasing solutions to semantic segmentation systems using Bayesian modeling.

The work of Kendall and Gal centers around semantic segmentation and pixel-wise depth estimation, and uses Bayesian Neural Networks to increase performance. Since this thesis does use uncertainties in its semantic segmentation approach, does not use Bayesian Neural Networks and does not work with depth data or estimation, the findings from Kendall and Gal are not applicable here.

3.2.2 Multitask Learning

Kendall et al. proposes a method of using uncertainty types discussed in [18] (see Section 3.2.1) to select the weights used to combine the losses from several tasks in a multi-task learning scenario. [19] They create a multi-stream model (see Section 2.3) to simultaneously perform semantic segmentation, object detection and depth estimation. Illustrating the importance of correctly weighing the losses when combining tasks, their proposal is a method of using Bayesian modeling to learn weights for the losses, thereby increasing performance.

Since the uncertainty handled by Kendall et al. here is based on Bayesian modeling, which is not used in this thesis, their methods were deemed inapplicable. The learning weighing scheme proposed was of interest, but deemed out of scope for this thesis to implement.

3.3 Neural Network Architectures

This section highlights some of the public neural networks relevant to this study.

3.3.1 ENet

There exists several public CNN architectures for the task of semantic segmentation of images, which is a closely related problem to the problem of lane detection. One such architecture is the ENet architecture by Abhishek et al. [27], which has the advantage of being highly efficient compared to other architectures. It uses, for instance, 75 times less floating point operations and runs 18 times faster than SegNet [3], which is another widely used CNN for semantic segmentation, while still achieving comparable performance.

ENet is an *encoder-decoder* network, meaning it first performs several down-samplings together with convolutions and other layers, to create some abstract representation of the features in the input image. This part of the network is called the *encoder*. The feature maps are then upsampled together with other layers in several steps until the output is of the same height and width as the input image. This is the *decoder* part of the network. The output contains the same

number of images as the number of classes, and for each position, the pixels sum up to 1, where the class of the highest value is selected as the class to output. In order for the pixels to sum up to 1, a *softmax* layer is used.

Given the high performance of ENet, it can be concluded that it is highly suitable as a basis for the network architectures to be used in this project. Specifically, the feature extracting part of the network up until the first upsampling is suitable for the networks outputting the end-to-end representation, and the entire network is suitable for the semantic segmentation representation.

3.3.2 ResNet

One of the most important neural network architectures is the architecture described in the *Deep Residual Learning for Image Recognition* paper by He et al.[15]. In the paper, the authors describe the degradation problem, which is a situation where accuracy of the network gets saturated, and gets worse the more layers that are added. The authors note that this is not caused by overfitting, but by the fact that these networks are simply harder to optimize. As a solution to this problem, the authors propose a network designed to not explicitly learn an underlying mapping $H(\mathbf{x})$. Instead, the network is designed to learn the residual mapping $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$, making the original mapping $F(\mathbf{x}) + \mathbf{x}$. The motivation behind this is that it is probably easier for solvers to learn mappings close to the identity mapping, as the weights would be driven towards 0. The mappings are constructed by the use of shortcut connections, where the output of a few chained layers are added to the input of the layers, producing the mapping $F(\mathbf{x}) + \mathbf{x}$. F could be two or more convolutional layers, or other types of layers.

Since the ResNet style skips are presented as a solid solution to the degradation problem, such connections were deemed useful to include in the networks to be designed in this thesis. This is furthered by the fact that ENet 3.3.1 utilizes such skips, adding to their credibility.

3.4 Multi-stream Neural Networks

Several papers have been published showing promising performance on computer-vision tasks using multi-stream neural networks. One such shows that fusing standard RGB images with data extracted from the images, such as Optical Flow, Pose Estimation et cetera, may increase accuracy in activity recognition [2].

Interesting work connected to automotive safety from Du et al. also shows that a multi-stream approach which combines features from early and late convolutional layers to detect pedestrians with state-of-the art performance. [9].

Connected to this thesis is the work by Neven et al., who proposes a multi-stream network combining binary segmentation masks with embeddings for the lane-classes. This shows that separating the classification of which lane each marking is on from the regression of the exact position of the lane may increase performance [25].

Another automotive safety use of multi-stream neural networks is shown by Zhang et al.[29]. They propose using two images from two different cameras as

data to two neural networks, which are then fused and shares features between them, in order to recognize driver behavior.

The above mentioned applications and their results makes a multi-stream approach to the task of road-lane marking detection and classification interesting, and worth including in this thesis.

3.5 PReLU

Proposed by He et al. in [14] is the activation function *Parametric Rectified Linear Unit*, PReLU. This is defined as

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i \geq 0 \\ a_i y_i, & \text{if } y_i < 0 \end{cases} \tag{3.1}$$

for the output of the i:th channel, with a_i as a learned parameter. The reasoning for this activation function is to allow the network to learn a suitable slope for the negative part of the input. Compared to the standard *Rectified Linear Unit* ReLU [12], defined as

$$f(y_i) = \max(0, y_i), \tag{3.2}$$

it is seen to have the same effect as letting $a_i = 0$. The difference between the PReLU and the ReLU is that the PReLU allows for a non-zero derivative for negative values, which is not possible with the ReLU implementation. This has been shown to greatly increase performance of the network, based on the importance of non-zero derivatives for Stochastic Gradient Descent. [14]

Based on this, and the fact that PReLU is used as activation function in ENet, it was regarded as a suitable function to use in the network architectures in this thesis.

3.6 Global Average Pooling

Pooling is an important stage of a convolutional neural network, where the feature maps are downsampled in some way, before being fed to the next layer. A pooling scheme introduced by Lin et al. in [23] is Global Average Pooling (GAP). This concept replaces each feature map in a layer with its average, effectively reducing the size by a factor equal to the feature map size for that layer. The concept behind the scheme is that every feature map can be approximated by its average without too large of a loss in information, and that the resulting scalar valued feature maps inherently keep structural data which makes it more native to the convolutional nature of the network, since CNNs by construction work with structural data.

Given the high performance presented using GAP, combined with the decrease in parameters, GAP was selected to be used in the end stages of the networks constructed in this thesis.

3.7 Datasets

This section will present and discuss some datasets relevant to this thesis study, as well as other work related to datasets.

3.7.1 tuSimple

The tuSimple dataset contains 3 226 annotated frames for lane detection, and is used for the tuSimple benchmark [1]. The data is marked with ego-left, ego-right, neighbor-left and neighbor-right lanes. The dataset's chosen representation is to have a list of N sample heights for each image, corresponding to height positions (y) in the image, and 4 lists of N width positions, corresponding to the width positions (x) in the image containing each of the lanes at said position. A lane which does not exist in on a sampled height is represented as having an x value of -2.

This dataset appears to contain mostly daytime highway driving, and is relatively small compared to the other datasets discussed in this section. The representation presented is an interesting one, but is not compatible with the semantic segmentation approach. Therefore, the tuSimple dataset was not used further in this study.

3.7.2 CULane

The CULane dataset contains 133 235 frames labeled with continuous lane markings [26]. The frames in the CULane dataset are labeled in the same conceptual style as shown in Figure 1.2, meaning that the labels connect dashed markings. The labels are also continued through occlusions such as cars. The CULane dataset is divided into a training set, a validation set and a test set.

Investigating the CULane dataset, it was discovered that the validation set only contained frames gathered from a single day of driving. This led to the frames presenting only highway conditions with little traffic and good lighting, while the rest of the dataset was more diverse. This was remedied by mixing the sequences from the validation set with the ones from the training set and thus creating a new training set and a new validation set. It was then noted that the frames in the dataset often contained glare from light sources, scratches in the filmed-through windshield, reflections from the dashboard and adhesive stickers stuck to the windshield. These unnatural additions, combined with the fact that it was unusable for the semantic segmentation based approach, led to the CULane dataset being deemed sub-optimal for this thesis.

3.7.3 Veoneer Data

A subset of Veoneer's ground-truth frames was provided for the purpose of this thesis. It contained 41 562 frames to be used for training, 11 538 frames to be used for validation, and 10 007 frames to be used for testing. The frames in this dataset contained a mixture of many situations and locations, as only a few

frames were selected out of every 30 second sequence. It was labeled in the same conceptual way as displayed in Figure 1.1, as it's meant to be used as ground truth for semantic segmentation.

The labeling of the Veoneer dataset did make it suitable for the semantic segmentation approach, while requiring some processing to be suitable for the end-to-end representation from Figure 1.2. Given the other datasets' (Sections 3.7.1 and 3.7.2) disadvantages, and that the thesis was done for Veoneer, this dataset was chosen as the main one to be used in this thesis.

3.7.4 Dataset Imbalance

Sometimes, the dataset used is unbalanced, meaning it has a bias towards certain situations or classes. This can lead to poor performance on the underrepresented situations in the dataset.

A simple and common way of solving this problem is to use a technique *oversampling*. Here, the samples of an underrepresented class is randomly repeated until it matches the number of other classes. Buda et al.[5] showed that this technique works well with convolutional neural networks, better than many other techniques for remedying an unbalanced dataset. This makes oversampling a suitable technique for weighting underrepresented situations in the Veoneer dataset.

3.8 Evaluating Lane Detecting Systems

This section will present some proposed methods of evaluating the performance of lane detecting systems.

3.8.1 Intersection over Union

In their paper on Spatial CNN (Section 3.1.4) Pan et al. proposes an evaluation method consisting of viewing the pixel-wise lane positions as lines with width 30, and calculate the IoU between the lines produced by a system and the lines created from ground-truth data [26]. Prediction IoU larger than a threshold was seen as TP. They consider thresholds 0.3 and 0.5 for the classification of correctly detected points, corresponding to loose and strict evaluations, respectively. Their final metric is then to calculate the F1 measure, corresponding to $\frac{\text{PrecisionRecall}}{\text{Precision+Recall}}$ with Precision = $\frac{\text{TP}}{\text{TP+FP}}$ and Recall = $\frac{\text{TP}}{\text{TP+FN}}$.

This evaluation method is closely related to those often used for object detection purposes. The method chooses thresholds and line widths arbitrarily, creating a large parameter space for the evaluation process. It also do not distinguish between systems barely passing the threshold, and those being pixel-wise exact. Therefore, this evaluation method was not chosen to be used in this study.

3.8.2 Accuracy

The tuSimple dataset, as used in the tuSimple benchmark and described in Section 3.7.1, uses an Accuracy measure to evaluate performance. This is calculated by comparing each outputted point with the ground-truth point. If the difference between the outputted x-position and the ground-truth x-position is within some threshold, it is classified as a correct point and added to the set $\mathbf{C}_i$, the set of correct points for image i. This is then compared to the set $\mathbf{S}_i$, the set of ground-truth points for image i. The accuracy is then given as $\frac{\sum_i \mathbf{C}_i}{\sum_i \mathbf{S}_i}$. Should the number of outputted lanes be over 2 more than the number of lanes in the ground truth, the accuracy of that image is set to 0.

This evaluation metric focuses on correct detection within a threshold. Therefore, it is only accurate up to said threshold, and neither does it allow to differentiate between systems outputting positions outside the threshold. This led to this metric being disregarded in this study.

4

Method

This chapter describes the methodology followed in the study. The study in practice was conducted iteratively, basing each experiment upon the results of the previous one. Therefore, each section in this chapter contains a motivation to the investigation, as well as intermediate results and conclusions drawn from it.

4.1 Lane Representation

This section describes the representation used for the end-to-end system, as well as the baseline segmentation representation. Both of these representations are in 2D, in the sense that they represent points in the image. These representations have inherently fewer degrees of freedom than representations based on 3D world coordinates, which should in theory make training easier. They are also easier to create ground-truth data for, since one need not know the mapping from image to world coordinates.

4.1.1 Supporting Points

The end-to-end representation needs to be consistent, meaning regardless of which lanes are represented, the number of numbers in the representation should be constant. This is because Convolutional Neural Networks (CNN) typically have a fixed output size. Furthermore, the representation should directly represent the positions of the lane in the image, such that little or no post-processing of the output is necessary. The representation should also have limited degrees of freedom, such that it can only represent shapes which are realistic.

Based on these requirements, we propose a representation called *Supporting Points*, which is similar to the representation used in the tuSimple dataset [1] from Section 3.7.1. This representation involves representing lanes in the image

by points through which the lanes run. The lanes are sampled row-wise in the image, where each row is separated by a number of pixels. For each row, a point where the row intersects with the lane is created, which means each lane boundary is represented by a list of x-coordinates. We denote the coordinate at pixel row i on lane j as $x_i^{(j)}$. A list of x-coordinates is however not enough, as depending on the road profile and curvature, the lanes might not intersect all rows in the image. To deal with this problem, each $x_i^{(j)}$ is coupled with a binary value $m_i^{(j)}$. If $m_i^{(j)} = 1$, the corresponding $m_i^{(j)}$ represents a point on the road, while if $m_i^{(j)} = 0$, there is no intersection of $x_i^{(j)}$ on the road. See Figure 4.1 for an illustration of supporting points.

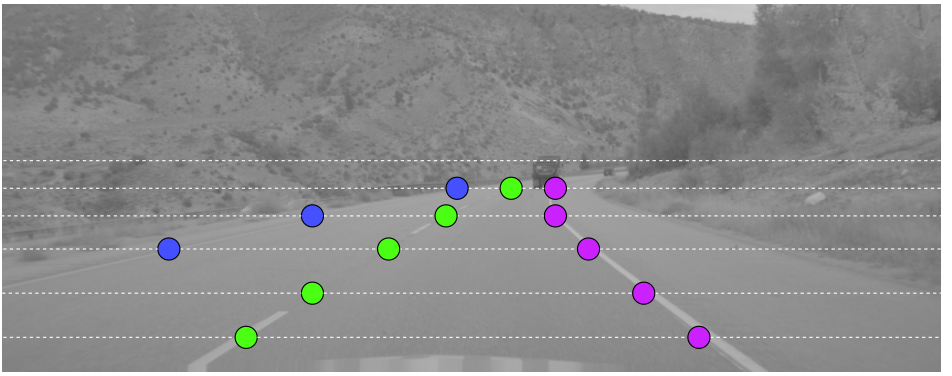


Figure 4.1: *Simplified illustration of the supporting points representation. The dashed lines depict the rows of the image that are sampled, and the color of the points depict different lane-ID:s.*

Due to the fact that the road is not observed from above when viewing it from the perspective of the car, rows higher up in the image correspond to locations further away from the vehicle than the lower rows in the image. This motivates placing the rows for sampling closer together higher up in the image, to approximately extract the same amount of information for every real-world length unit of the lane. A simple way to do this is to have the distance d_i between row $i-1$ and i to be decreasing exponentially with i,

$$d_i = \alpha d_{i-1}. \tag{4.1}$$

The parameter $\alpha \in [0, 1]$ is the rate at which the distance decreases, where $\alpha = 1$ corresponds to there being a constant distance between each row, and $\alpha = 0$ means all rows share the same y coordinate.

In order for the network to learn this representation, it needs to be trained with a cost function which takes the binary mask variables into account. These need to strictly output 0 or 1, and if they are 0, the corresponding x should have no impact on the cost.

This representation has the following hyper-parameters:

- The rate at which the distance between the rows decreases, α.
- The initial distance between the rows, d_0.
- The number of sample rows, N.
- The y coordinate of the bottom row, y_0.

4.1.2 Supporting Points with Confidences

After working with the supporting points representation, described in Section 4.1.1, it was found lacking in some aspects. Mainly, false positives were common. These appeared when the network correctly predicted the existence of a lane, while wrongly providing its position. This leads to a case where the network outputs a confidence of existence close to 1 for a point, while its regression is not close to its desired coordinate. This is illustrated in Figure 4.2, where it is shown how the edge between the newly repaired road and the old road is taken as the edge of the lane, instead of the actual lane.

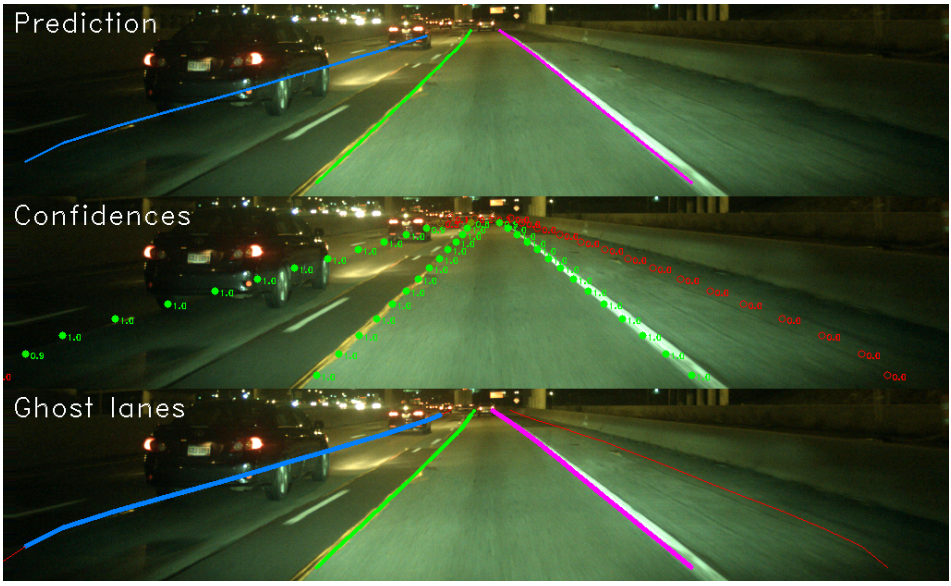


Figure 4.2: *Example of high confidence but wrong position regression.*

Based on the observation described above and shown in Figure 4.2, it was decided to investigate if a network could learn to predict its own performance and thereby output a confidence for each prediction. It was hypothesized that the network could learn to recognize situations where the regression performs badly, and thereby output a confidence value for it. This value was theorized to be useful to avoid false positives of the types shown in Figure 4.2, by recognizing a difficult situation and then outputting low confidence, which can then be deemed

as the network outputting high confidence in the existence of a lane, but at the same time informing that its positional output should not be trusted.

This confidence was chosen to be seen as a third output for the neural networks. It was deemed adequate to view the confidence as a number $c_i^{(j)} \in (0, 1]$. This value is to be a representation of the error in regression, with no error/a perfect prediction corresponding to $c_i^{(j)}(d) = 1$ for $e = 0$, and a prediction infinitely far away corresponding to $\lim_{e\to\infty} c_i^{(j)}(e) = 0$, for absolute error between ground truth and regression prediction e. This design choice led to viewing the confidence as an exponential decaying with the error in position (*regression error*). Since the regression error can not be know to the network beforehand, work needed to be done to allow the network to train to predict something for which there exists no prior ground truth. This was solved by letting the ground truth from the ground-truth file contain a confidence value of 0, which was then promptly replaced with the correct, regression-error based confidence value during loss calculation. This lead the network to calculate the sought confidence for each point at runtime, and then use it as ground truth for its prediction of the confidence. This ground truth for the confidence loss was implemented by calculating a confidence value $c_i^{(j)}$ for each row i on lane j, chosen as

$$c_i^{(j)} = \exp\left(-L\left(\frac{\left|x_i^{(j)} - \bar{x}_i^{(j)}\right|}{T}\right)\ln 2\right) \tag{4.2}$$

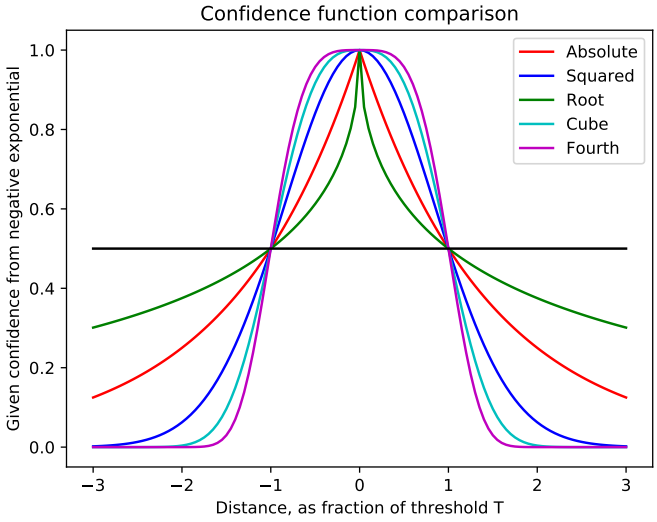
for a continuous, strictly increasing distance function L, predicted x-position $\bar{x}_i^{(j)}$, ground-truth x-position $x_i^{(j)}$, threshold $T > 0$. The threshold T is set as a tolerance distance in pixels, which together with the $\ln 2$ factor ensures that the confidence is $\frac{1}{2}$ for a positional prediction T pixels away from the ground truth, regardless of distance function. This design allows for an easy interpretation of the confidence, since a confidence value $> \frac{1}{2}$ corresponds to a regression error $< T$, while a confidence value $< \frac{1}{2}$ corresponds to a regression error $> T$.

The distance function L can be any continuous, strictly increasing function. Intuitively, it was thought that it should be an exponent of the distance $\left|x_i^{(j)} - \bar{x}_i^{(j)}\right|$, corresponding to L1, L2 norms. This is described in Equation 4.3, where the exponent k determines the type of distance function.

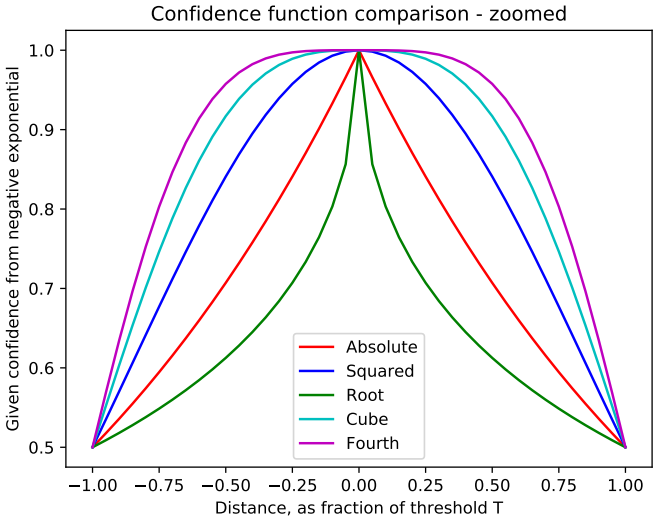
$$L_k = \left(\frac{\left|x_i^{(j)} - \bar{x}_i^{(j)}\right|}{T}\right)^k \tag{4.3}$$

A comparison between different values for k is given in Figure 4.3, for $k = \frac{1}{2}, 1, 2, 3, 4$, corresponding to a root, linear, square and cube function, as well as a fourth power, respectively.

As is shown in Figure 4.3a, the higher the exponent, the greater the slope at $|x_i - \bar{x}_i| = T$, and the lower the confidence at $|x_i - \bar{x}_i| > T$. Figure 4.3b shows that a higher exponent lets the confidence be closer to 1 for larger distances than



(a) *Confidence as a function of distance. Horizontal black line shows* $c = \frac{1}{2}$



(b) *Confidence as a function of distance, detail for distance between* $-T$ *and* T*.*

Figure 4.3

lower exponents. This is interpreted as a higher exponent allowing larger errors up to T, but then gravely loosing confidence, while a lower exponent more clearly

decays with increasing T, while still giving some confidence for predictions with error only slight larger than T.

Based on this interpretation, the root distance function was not considered worth investigating further, since the drop in confidence for distances close to 0 was unwanted. Furthermore, distance functions based on exponents higher than 2 was also discarded, since they were shown in Figure 4.3b to be close to constantly equal to 1 for distances smaller than T, and shown in Figure 4.3a to be almost constantly equal to 0 for distances larger than T. This almost rectangular-function like behavior was unwanted, as gradually changing confidence were deemed more useful than binary-like behavior. This led to only the absolute and squared distance functions being examined.

4.1.3 Semantically Segmented Images

Semantic Segmentation is a common technique in computer vision, where each pixel is classified as one of several classes [27]. This problem can be solved using deep convolutional neural networks, and as described in Section 3.3.1, ENet [27] is an example of such a network. *Semantically Segmented Images* is a representation based on semantic segmentation, where each pixel in the image is classified one of five classes; lane marking of neighbor-left, ego-left, ego-right or neighbor-right, as well as "other". In the ground truth, this is represented as five binary images, one for each class, with ones in the active class and zeros elsewhere. In practice, the network outputs for each pixel a number for each class, indicating how likely the pixel is of that class. These numbers sum up to 1 in each pixel, meaning each pixel has something similar to a probability distribution. For example, the network might output 0.7 for neighbor-left, 0.1 for ego-left, 0.1 for ego-right, 0.05 for neighbor-right and 0.05 for other, in which case, the pixel is probably part of a neighbor-left marking.

See Figure 4.4 for an illustration of a ground-truth segmentation image.

Figure 4.4: *Visualization of the segmentation ground truth. The colors indicate the different classes, and uncolored parts of the image depict the "other" class.*

Post-processing Semantic Segmentation

This representation does not directly specify where each lane is in the image, it merely specifies where the lane markings are. This means the semantic segmen-

tation output needs to be post-processed to be able to specify where each lane is.

The following method was used to post-process the output: Each class was assigned a label, for instance 100 for neighbor-left and 0 for "other". The network output was then converted to a grayscale image, where each pixel in the image was assigned the label of the class with the highest probability in the network output. This produced an image of the same format as the unprocessed ground truth of the supporting points representation, meaning the same algorithm used for generating supporting points from segmentation ground truth was used. This algorithm is described in Section 4.3.1, but in short, the algorithm samples the rows of the segmentation image and finds the left or right edge of each lane label. These edges are then interpolated and resampled to create the supporting points representation. As with the supporting points ground-truth generation, the lanes are extrapolated to the bottom of the image and resampled to the same number of points and row positions as was used for the supporting points, to make them directly comparable. The extrapolation and resampling procedure is described in Section 4.3.2.

See Figure 4.5 for an example of a semantic segmentation prediction and its post-processed output.

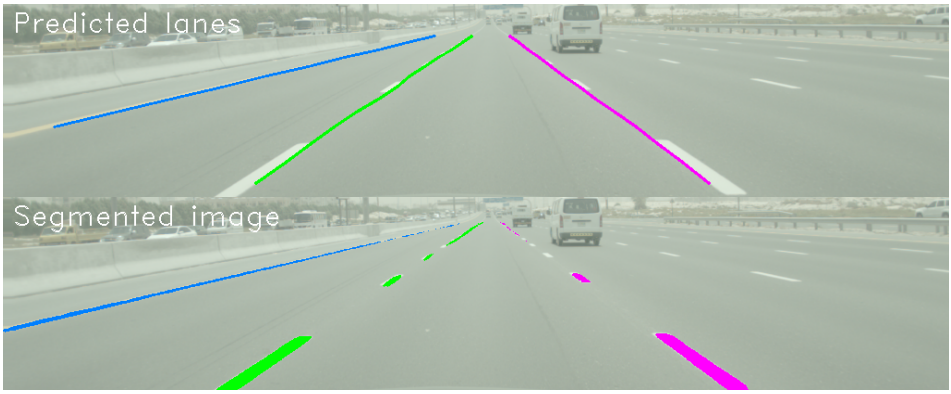


Figure 4.5: *Example of a post-processed semantic segmentation prediction. Below is a visualization of the output of the network, where the maximum probability class has been selected, and above is the output of the supporting points conversion.*

4.2 Neural Network Architectures

The focus of this study is not to explore many different neural network architectures or design an architecture from scratch. This means existing architectures are used with minor modifications. However, the networks still need to be designed and tuned for the purpose. This section describes this design procedure and which architectures are used.

4.2.1 ENet Backbone

The network needs to be fairly deep in order to account for the diversity of situations the system could face, while being efficient enough to be used in a real-time application. This makes ENet [27] a suitable network to base the architecture on, as described in Section 3.3.1. The encoder-part of this network is used for the supporting points representation, as the "backbone" of the network, onto which other architectures are appended, called *heads*, as illustrated in Figure 4.6.

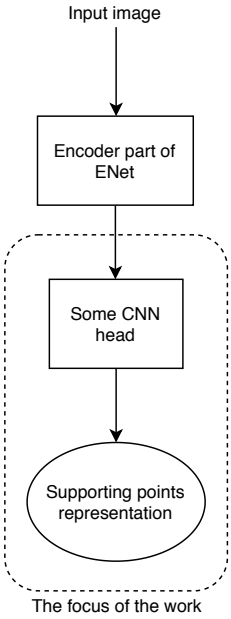


Figure 4.6: *An illustration of the focus of the work done regarding the network architecture design for the supporting points.*

4.2.2 Architecture of the Network Head

This section describes the neural network architecture used for the part after the ENet-backbone, and the design choices that went into it. The full network architectures are disclosed in Appendix E.

ResNet-style Shortcut-connections

An architecture inspired by ResNet [15] was developed, by using the shortcut connections from this architecture, described in Section 3.3.2. As shown in Figure 4.7, two convolutional layers, along with a few activation and regularization[1]

[1] Regularization refers to actions taken to reduce validation error, possibly at the expense of training error.

layers, were bypassed. This kind of shortcut only works when the number of channels in the input equal the ones in the output, which is a problem whenever a convolutional layer increases the number of channels. The authors of the ResNet paper [15] propose two methods of dealing with this, one is by zero-padding the input to match the output of the skipped layers, and one is to use convolutions with a kernel size of 1×1 on the input to increase the dimensionality, see Figure 4.8. In this thesis, a simpler method of having the convolutional layer which increases the number of channels not be bypassed at all was used, shown in Figure 4.9. The reasoning behind this was that the depth of the network head was not high, meaning the number of non-bypassed layers would not be high. This is supported by the fact that the final networks that were used only had one or two such layers. This network type was also easier to implement.

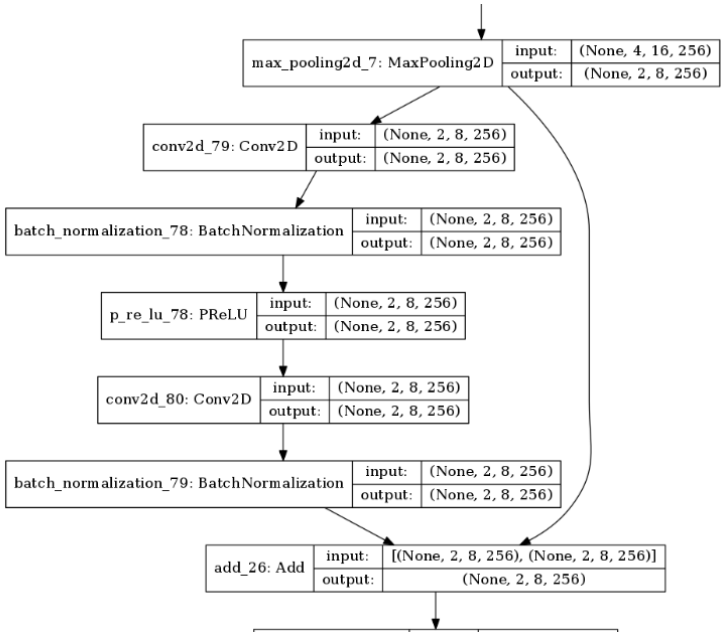


Figure 4.7: *A ResNet-style shortcut. A set of layers are bypassed by adding the input to the layers to the output.*

Batch-Normalization Layers

Each convolutional layer was followed with a batch-normalization layer, for regularization. This is in part because the ENet-backbone uses it, and because it normalizes the activations from the convolutional layers over a batch, which increases the speed of the training. It also has regularization effects, which improves generalized performance of the network [16].

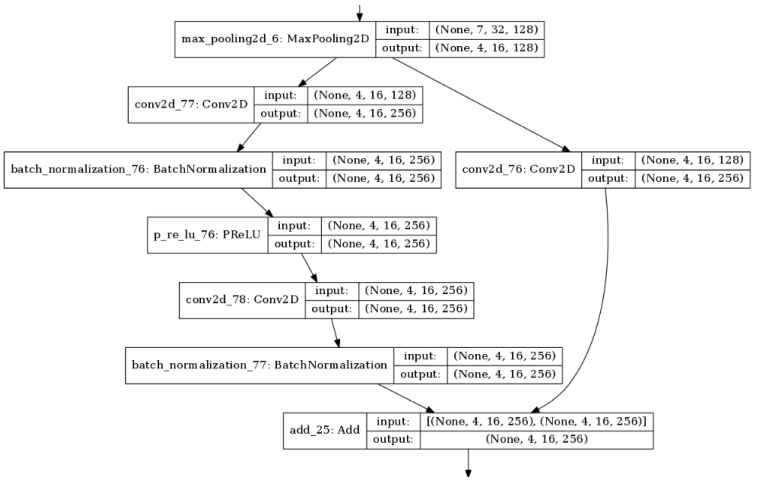


Figure 4.8: *Shortcut with 1x1 convolution to increase the number of channels*

Activation Functions

Parametric Rectified Linear Unit (PReLU) [14] were used as activation functions after the convolution and batch-normalization layers, for the reasons described in Section 3.5.

Global Average Pooling

After the last PReLU activation and before the fully-connected layers, a global average pooling [23] layer was used. This was done to drastically reduce the number neurons being fed to the fully-connected layers, which reduces the number of parameters in the network considerably, as described in Section 3.6.

Fully Connected Layers

Two fully-connected layers conclude the network. This was done as a final step, as a function from the higher-level input extracted by the convolutional part of the network to the output classification and regression. A ReLU activation layer was used after the first fully-connected layer.

4.2.3 Representation-specific Architectures

Each lane representation needed its own neural network, due to their differences in output, which puts different requirements on the network. This section outlines the differences in network design between the representations.

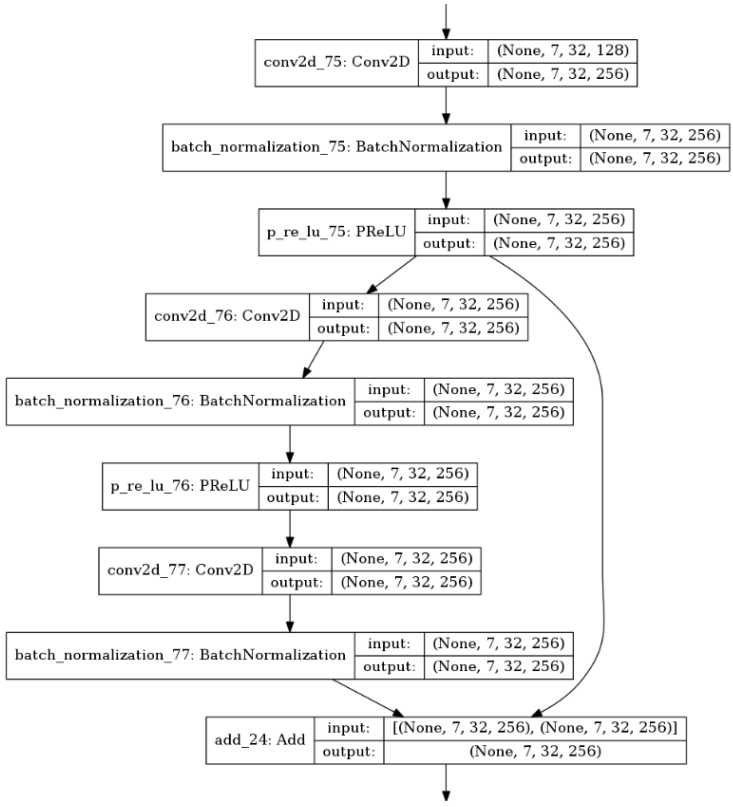


Figure 4.9: *Convolution before shortcut to increase the number of channels*

Supporting Points

There are two different types of outputs for the supporting points, the x-coordinates $x_i^{(j)}$ and the mask variables $m_i^{(j)}$. Estimating the coordinates is a regression problem, while estimating the correct mask values can be regarded as a classification problem. This means different output units are to be used. For the x-coordinates, a linear output unit is used,

$$\mathbf{x} = \mathbf{W}^T\mathbf{h} + \mathbf{b}, \tag{4.4}$$

where $\mathbf{W}$ is the weight matrix, $\mathbf{h}$ are the activations from the previous layer and $\mathbf{b}$ are the biases. This means the output unit is a simple linear neural network layer without activation functions. This was done to avoid vanishing gradients during training, as linear units do not saturate. For the mask variables, a sigmoid output unit is used,

$$\mathbf{m} = \sigma(\mathbf{W}^T\mathbf{h} + \mathbf{b}). \tag{4.5}$$

This was done to ensure that the mask variables are constrained in the interval

[0, 1], letting them be interpreted as the likelihood of the corresponding lane boundary point existing. A reason for preferring this over a hard threshold is that the gradients are always non-zero, whereas with a threshold they become 0 as soon as the unit outputs a 0 or 1, making correction of a wrong decision impossible. The gradients do however saturate when the magnitude of the input to the sigmoid is high, meaning care has to be taken when designing the loss function.

For the supporting points, the loss function needs to account for the error in both the mask variables $m_i^{(j)}$ and the error in the x-coordinates $x_i^{(j)}$. It should also have the property that if the mask variable $m_i^{(j)}$ is 0, then the corresponding $x_i^{(j)}$ should have no impact on the loss.

To achieve this, the loss function $J(\theta)$ for network parameters θ is constructed as the sum of the error in the mask variables ($J_m(\theta)$) and the error in the x-coordinates (regression loss, $J_x(\theta)$),

$$J(\theta) = \gamma J_x(\theta) + (1 - \gamma) J_m(\theta), \tag{4.6}$$

where $\gamma \in [0, 1]$ is a hyper-parameter controlling which term of the loss to weigh more.

Denote $n_i^{(j)}$ the ground-truth mask variables. The $J_m(\theta)$ loss is defined as the binary cross entropy between the training data and the predicted values:

$$J_m(\theta) = -\sum_{i,j} (n_i^{(j)} \log(m_i^{(j)}) + (1 - n_i^{(j)}) \log(1 - m_i^{(j)})). \tag{4.7}$$

This loss function is motivated by the fact that the output unit for the mask variables is a sigmoid. The logarithm in the loss function compensates for the exponentiation in the sigmoid function, which eliminates the gradient saturation problem. This function is also the one typically used for classification problems [8].

The regression error was experimented with more than the error of the mask variables. Denote $x_i^{(j)}$ as the ground-truth x-coordinates. The first variant of the regression loss, $J_{x,1}(\theta)$, is defined as the mean square error between the ground-truth positions and the predicted positions $\bar{x}_i^{(j)}$,

$$J_{x,1}(\theta) = \frac{1}{N} \sum_{i,j} n_i^{(j)} (x_i^{(j)} - \bar{x}_i^{(j)})^2, \tag{4.8}$$

where N is the number of mask variables set to 1. With this loss function, the network gets penalized for the wrong $x_i^{(j)}$ if $n_i^{(j)} = 1$, otherwise no value is placed in what position network outputs. The errors are normalized by dividing by the number of active points, but if there are no active points for this lane, they are divided by the machine epsilon instead of 0. Another variant of this loss function is one with the mean absolute error instead:

$$J_{x,2}(\theta) = \frac{1}{N} \sum_{i,j} n_i^{(j)} |x_i^{(j)} - \bar{x}_i^{(j)}|. \tag{4.9}$$

Here, the rate of penalization increases linearly with the error, rather than quadratically. We hypothesized that the quadratic error would perform better than the absolute error. However, early experiments showed that the networks trained on the absolute error converged faster and produced better results than the ones trained with a quadratic error. See Figures 4.10 and 4.11 for examples of training curves with quadratic and absolute loss. One reason for this might be that when the network outputs a wrong prediction with the quadratic error, it dominates the cost in such extent that it the network does not try optimize the mask variables.

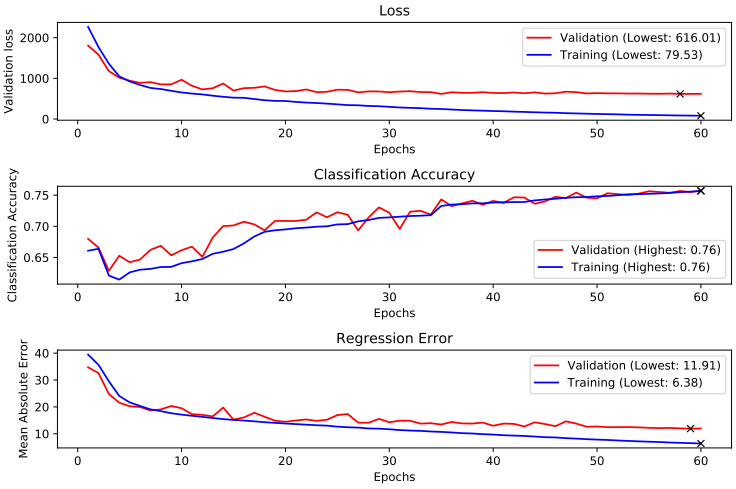


Figure 4.10: *Training curves for the quadratic $J_{x,1}(\theta)$ loss*

The *Huber loss* [13, p. 349] was another variant of the loss function was tested. We hypothesized that this loss function combines the desirable property of smoothness around 0 from the quadratic loss with the slower growing absolute value loss. The Huber loss $H(x, y)$ is defined as (scaled with $\frac{1}{2}$ compared to the equation given in [13, p. 349]):

$$H(x, y) = \begin{cases} \frac{1}{2}(x - \bar{x})^2, & \text{if} |x - \bar{x}| \leq \delta \\ \delta |x - \bar{x}| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases} \tag{4.10}$$

where δ is a parameter which controls a distance from 0, outside which the loss has a slope equal to the absolute error, and inside a slope equal to the quadratic error. Total Huber-based $J_{x,3}(\theta)$ loss then becomes:

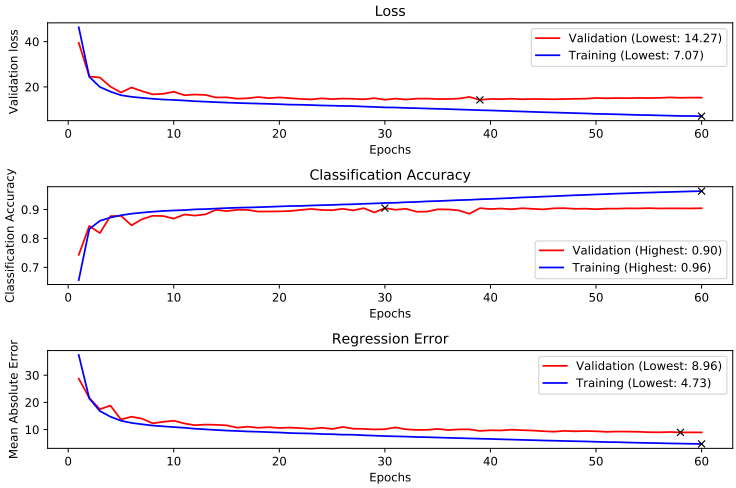


Figure 4.11: *Training curves for the absolute $J_{x,2}(\theta)$ loss*

$$J_{x,3}(\theta) = \frac{1}{N} \sum_{i,j} n_i^{(j)} H(x_i^{(j)}, \bar{x}_i^{(j)}). \tag{4.11}$$

Contrary to the hypothesis, however, this still did not perform as well as the absolute $J_{x,2}(\theta)$ loss, at least in regards to the regression error, as can be seen in Figure 4.12. While it did certainly better than the quadratic $J_{x,1}(\theta)$ loss, it still did not perform well enough to validate using the Huber loss over the absolute error. Therefore, the mean absolute error was used as the error function in the losses.

The last basic regression loss function which was tested was one based on the observation that it might be more important to weigh points higher up in the image more, as these correspond to larger distances in the real world. For this reason, a variant of the absolute error loss function, which we call the *Horizon Loss* ($J_{x,4}(\theta)$), was developed. With this regression loss, the mean absolute error is used, but it weighs the regression error more the higher up the points are in the image. Formally, a weight vector $\mathbf{w}$ is created, which is a linear range where the first element $w_1 = 0.5$ and the last $w_M = 1.5$ (M is the number of rows). The elements of this vector is multiplied with the regression error in each row:

$$J_{x,4}(\theta) = \frac{1}{N} \sum_{i,j} n_i^{(j)} w_i |x_i^{(j)} - \bar{x}_i^{(j)}|. \tag{4.12}$$

This loss was hypothesized to improve the performance of points high up in the image. It should also improve the performance on samples of curvy roads, since the road visually curves the most high up in the image.

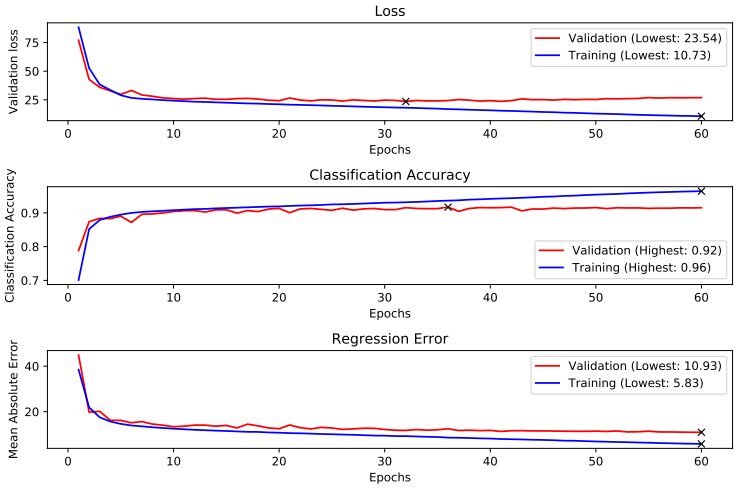


Figure 4.12: *Training curves for the Huber-based $J_{x,3}(\theta)$ loss, where $\delta = 20$ pixels*

Semantic Segmentation

With semantic segmentation, the output is an entire image, with the same number of channels as the number of classes. In this implementation, the image size of the output is the same as the size of the input, with five output channels. This means the network needs a radically different architecture than the one for supporting points. For this reason, the entire ENet [27] network is used for semantic segmentation, both the backbone and head.

The ENet network was trained using a weighted binary cross entropy loss. Here, a set of class weights $w^{(c)}$ is used, indicating how important that each class c is. Also, positive to negative ratio weights $w_{pn}^{(c)}$ are used, where a high value weighs positive predictions more, and a lower value weighs negative more, for each class c. In other words, a high $w_{pn}^{(c)}$ decreases the number of false negatives, and while a low one decreases the number of false positives. Denote $x_{i,j}^{(c)}$ a ground-truth pixel at row i and column j for class c, and $\bar{x}_{i,j}^{(c)}$ the corresponding predicted pixel. The loss function $J(\theta)$ becomes

$$J(\theta) = -\sum_{i,j}\sum_{c}(x_{i,j}^{(c)}\log(\bar{x}_{i,j}^{(c)})w_p^{(c)} + (1 - x_{i,j}^{(c)})\log(1 - \bar{x}_{i,j}^{(c)})w_n^{(c)}) \tag{4.13}$$

where

$$w_p^{(c)} = \frac{w^{(c)}}{1 + \frac{1}{w_{pn}^{(c)}}} \tag{4.14}$$

and

$$w_n^{(c)} = \frac{w^{(c)}}{1 + w_{pn}^{(c)}}. \tag{4.15}$$

Supporting Points With Confidence

The calculated confidence value $c_i^{(j)}$ for each point i was used as ground truth for predicted confidences $\bar{c}_i^{(j)}$. These were compared using a loss function $J_c(\theta)$, where the following two variants were investigated for each of the two ways to calculate the confidence c,

$$J_{c,1}(\theta) = \frac{1}{N} \sum_{i,j} n_i^{(j)} |\bar{c}_i^{(j)} - c_i^{(j)}| \tag{4.16}$$

$$J_{c,2}(\theta) = \frac{1}{N} \sum_{i,j} n_i^{(j)} (\bar{c}_i^{(j)} - c_i^{(j)})^2 \tag{4.17}$$

where $J_{c,1}$ corresponds to a linear absolute difference, and $J_{c,2}$ corresponds to a squared error, letting $n_i^{(j)}$ denote ground-truth mask variable for lane existence, in order to not penalize confidences when no lane markings exists. N once again corresponds to the number of points in the sample where the ground-truth existence is 1, as in Section 4.2.3. These were chosen for investigation since a continuous confidence was desired, and not a binary classifier. Therefore, regular regression was selected as a goal. Applying the same reasoning as with the confidence calculation described in Section 4.1.2, distance-based loss functions of lower order than 1 or higher order than 2 was not deemed worth investigating. To ensure that the predicted confidence were bounded to the acceptable interval $(0, 1)$, a sigmoid was applied to the network output for confidence predictions as an activation function.

This led to a total loss function given by

$$J(\theta) = \lambda \left(\gamma J_x(\theta) + (1 - \gamma) J_m(\theta)\right) + (1 - \lambda) J_c(\theta) \tag{4.18}$$

with $J_x(\theta)$ as the regression loss, $J_m(\theta)$ as the classification loss $J_c(\theta)$ and the confidence loss. The parameter $\gamma \in [0, 1]$ is, as earlier, used to weigh the regression and classification loss, while the parameter $\lambda \in [0, 1]$ is used to weigh the original loss, that is, the regression and classification loss, with the confidence loss. This weighting scheme was chosen to allow the prior weight knowledge from the original representation without confidence prediction to be used, as well as to reduce dimensionality from three freely chosen weights to two.

In order to simplify the implementation of the confidence prediction and to let its network complexity be easily comparable to the representation without it, it was decided to implement it by changing the final layers of the network to output confidences as well. This was hypothesized to allow it to use the same spatial information as the regression and confidence prediction, which was believed to

give the confidence prediction the needed information to correctly predict the confidence of the regression.

This led to the networks being identical in design until the final dense layers, which were used to calculate the actual output. Figure 4.13 illustrates the final layers of a network used without confidence predictions, while Figure 4.14 illustrates the final layers of a network with confidence prediction. Note that both the incoming arrows on the top of both figures represent the data coming from the last skip connection, into the global average pooling layer. Also note that, as mentioned, the networks only differ in the dense layers and in the final output layer. The different names of the final layer, SelectiveSigmoid versus SelectiveSigmoidConfidence, distinguish between the two variants of the output layer - one that takes a regression and a classification output and applies a Sigmoid to the classification output, and one that takes a regression, a classification and a confidence output and applies a Sigmoid to both the classification output and the confidence output.

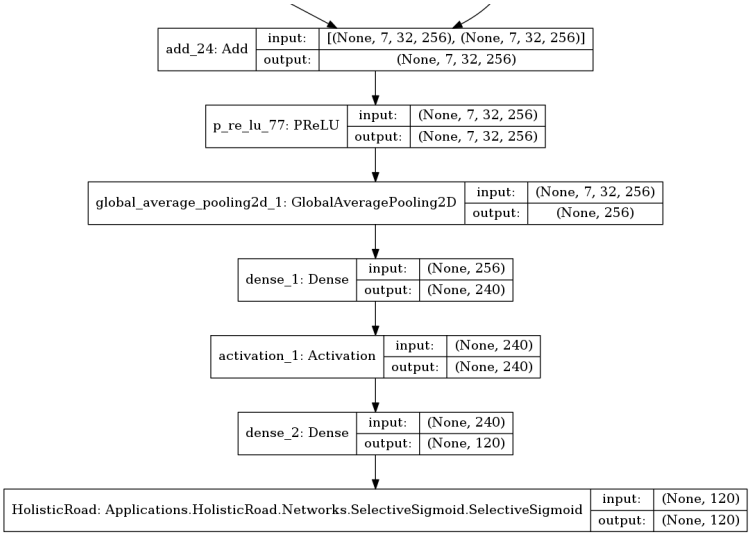


Figure 4.13: *Final layers of network without confidence output*

Note that both networks share the same structure of their ENet backbone, which allows a network with confidence predictions to be initialized with weights from a pre-trained network without confidence predictions. This is believed to decrease training time and increase performance, since the feature extraction learned from the first network is believed to be a good starting solution for the second network's feature extracting ENet backbone.

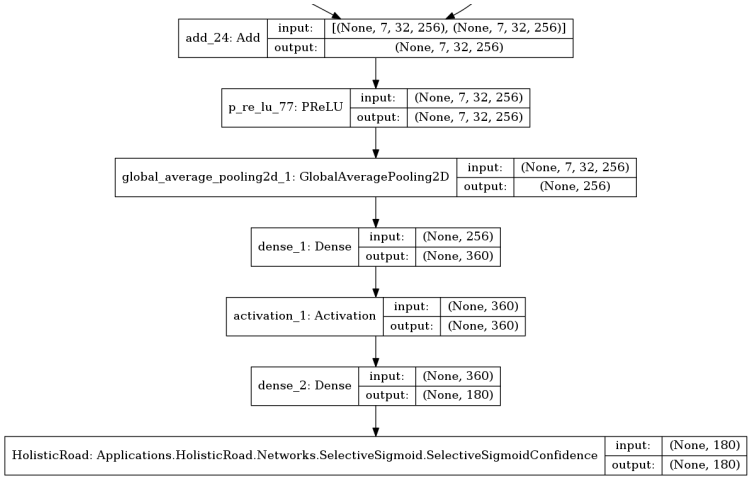


Figure 4.14: *Final layers of network with confidence output*

4.2.4 Loss Gradient

The loss functions described in Section 4.2.3 results in the following gradients, calculated in Appendix C:

$$\nabla J_{x,1} = 2\,(x-\bar{x})\,\hat{x} \tag{4.19a}$$

$$\nabla J_{x,2} = \frac{x-\bar{x}}{|x-\bar{x}|}\hat{x} \tag{4.19b}$$

$$\nabla J_m = \left(\frac{1-n}{1-m} - \frac{n}{m}\right)\hat{m} \tag{4.19c}$$

$$\nabla J_{c,1} = \frac{\bar{c} - e^{-\frac{|x-\bar{x}|}{T}\ln 2}}{\left|\bar{c} - e^{-\frac{|x-\bar{x}|}{T}\ln 2}\right|}\hat{c} - \frac{\bar{c} - e^{-\frac{|x-\bar{x}|}{T}\ln 2}}{|\bar{c} - e^{-\frac{|x-\bar{x}|}{T}\ln 2}|}\frac{x-\bar{x}}{|x-\bar{x}|}\frac{\ln 2}{T}e^{-\frac{|x-\bar{x}|}{T}\ln 2}\hat{x} \tag{4.19d}$$

$$\nabla J_{c,2} = \frac{\bar{c} - e^{-\frac{(x-\bar{x})^2}{T}\ln 2}}{\left|\bar{c} - e^{-\frac{(x-\bar{x})^2}{T}\ln 2}\right|}\hat{c} - \frac{\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}}{|\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}|}2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}\hat{x} \tag{4.19e}$$

$$\nabla J_{c,3} = 2\left(\bar{c} - e^{-\frac{|x-\bar{x}|}{T}\ln 2}\right)\hat{c} - 2(\bar{c} - e^{-\frac{|x-\bar{x}|}{T}\ln 2})\frac{x-\bar{x}}{|x-\bar{x}|}\frac{\ln 2}{T}e^{-\frac{|x-\bar{x}|}{T}\ln 2}\hat{x} \tag{4.19f}$$

$$\nabla J_{c,4} = 2\left(\bar{c} - e^{-\frac{(x-\bar{x})^2}{T}\ln 2}\right)\hat{c} - 2(\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 ln2})2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}\hat{x} \tag{4.19g}$$

with

- $J_{x,1}$ corresponding to the squared loss for regression
- $J_{x,2}$ corresponding to the absolute regression loss

- J_m corresponding to the classification loss
- $J_{c,1}$ corresponding to the absolute confidence loss for linear distance
- $J_{c,2}$ corresponding to the absolute confidence loss for squared distance
- $J_{c,3}$ corresponding to the squared confidence loss for linear distance
- $J_{c,4}$ corresponding to the squared confidence loss for squared distance
- x corresponding to the ground truth for regression
- $\bar{x}$ corresponding to the predicted regression
- n corresponding to the ground truth for the classification
- m corresponding to the predicted classification
- $\bar{c}$ corresponding to the predicted confidence
- $\hat{x}$ representing a unit-vector in the direction of increasing regression error
- $\hat{m}$ representing a unit-vector in the direction of increasing classification error
- $\hat{c}$ representing a unit-vector in the direction of increasing confidence error

Note that the gradients presented in 4.19 are viewed as gradients of a single sample, and ignores the ground-truth existence parameter $n_i^{(j)}$ as well as the multiplicative weights for the different losses. This was chosen to increase readability. Listed below is a description of the interpretations of the different gradient parts.

- Equation 4.19a shows that the only effects the $J_{x,1}$ loss has on the gradient lies along to a growing difference between x and $\bar{x}$, scaled with twice the distance. Given to a gradient descent algorithm, the next iteration of the search is therefore thought be positioned towards a smaller error, with a larger step taken proportional to the error.
- Equation 4.19b shows that the effects the $J_{x,2}$ loss has on the gradient lies along a growing difference between x and $\bar{x}$, constant with regards to the error. Given to a gradient descent algorithm, the next iteration of the search is therefore thought be positioned towards a smaller error, but with constant step size regardless of error magnitude.
- Equation 4.19c shows that the effect J_m loss has on the gradient is that it gives it in a $(0, 1)$ interval in the direction of the classification, $\hat{m}$. This corresponds to a gradient pointing towards increasing classification error, but with its magnitude < 1. Fed to a gradient descent algorithm, this is believed to have the effect of letting the next iteration position itself towards a smaller error with a variable but bounded step size.

- Equation 4.19d shows that the effect of the $J_{c,1}$ loss on the gradient is that it, in a manner similar to that of Equation 4.19a, adds a constant part towards increasing confidence error. It also adds a part in the direction of the regression error. If $\bar{c} < e^{-\frac{|x-\bar{x}|}{T}\ln 2}$, it will add $\frac{\ln 2}{T}e^{-\frac{|x-\bar{x}|}{T}\ln 2}$ in the direction of increasing regression error, while it will add $\frac{\ln 2}{T}e^{-\frac{|x-\bar{x}|}{T}\ln 2}$ in the direction of decreasing regression error if $\bar{c} > e^{-\frac{|x-\bar{x}|}{T}\ln 2}$. Used in a gradient descent algorithm, the part in $\hat{c}$ direction is believed to have the effect of letting the next iteration be that of a smaller confidence error, with a constant step size. The part in the $\hat{x}$ direction is thought to let the algorithm take a larger step towards lower regression error if the confidence prediction is too low ($\bar{c} < e^{-\frac{|x-\bar{x}|}{T}\ln 2}$), while making the algorithm take a smaller step towards lower regression error if the confidence prediction is too high ($\bar{c} > e^{-\frac{|x-\bar{x}|}{T}\ln 2}$). The regression affecting part is shown in Appendix D to be bounded between $(-\frac{\ln 2}{T}, \frac{\ln 2}{T})$.

- Equation 4.19e shows that the effect of the $J_{c,2}$ loss on the gradient is that it, in a manner similar to that of Equation 4.19b, adds part towards increasing confidence error, scaled with twice the confidence error. It adds a part in the direction of the regression error, similar to Equation 4.19d. If $\bar{c} < e^{-\frac{(x-\bar{x})^2}{T}\ln 2}$, it will add $2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}$ in the direction of increasing regression error, while it will add $2(x-\bar{x})\frac{\ln 2}{T^2}^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}$ in the direction of decreasing regression error if $\bar{c} > e^{-\frac{(x-\bar{x})^2}{T}\ln 2}$. Used in a gradient descent algorithm, the part in $\hat{c}$ direction is believed to have the effect of letting the next iteration be that of a smaller confidence error, with a step size proportional to the confidence error. The part in the $\hat{x}$ direction is thought to let the algorithm take a larger step towards lower regression error if the confidence prediction is too low ($\bar{c} < e^{-\frac{(x-\bar{x})^2}{T}\ln 2}$), while making the algorithm take a smaller step towards lower regression error if the confidence prediction is too high ($\bar{c} > e^{-\frac{(x-\bar{x})^2}{T}\ln 2}$). This is similar to Equation 4.19d. It is shown in Appendix D that the regression affecting part is bounded to $(-\frac{1}{T}e, \frac{1}{T}e)$.

- Equation 4.19f shows that the effect of the $J_{c,3}$ loss on the gradient is that it, with respect to the confidence error, behaves similarly to Equation 4.19a, in that that it adds a part towards increasing confidence error, scaled with twice the error. As Equations 4.19d and 4.19e it also adds a part in the direction of the regression error. This part behaves like the one described in Equation 4.19d, but is scaled with twice the confidence error. Used in a gradient descent algorithm, the part in $\hat{c}$ direction is believed to let the algorithm take a step towards smaller confidence error, with a step size proportional to the confidence error. The part in $\hat{x}$ is believed to let the algorithm take a larger step in the direction of smaller regression error if the confidence prediction was too high ($\bar{c} > e^{-\frac{|x-\bar{x}|}{T}\ln 2}$), and a larger step

towards smaller regression error if the confidence prediction was too low ($\bar{c} < e^{-\frac{|x-\hat{x}|}{T}\ln 2}$), both with step sizes scaled with twice the confidence error. It is shown in Appendix D that the regression affecting part is bounded to $(-\frac{\ln 4}{T}e, \frac{\ln 4}{T}e)$.

- Equation 4.19g shows that the effect of the $J_{c,4}$ loss on the gradient is that it as Equation 4.19a adds a part towards increasing confidence error, scaled with twice the error. As Equations 4.19d and 4.19e it also adds a part in the direction of the regression error. This part behaves like in Equation 4.19e, but is scaled with twice the confidence error. Used in a gradient descent algorithm, the part in $\hat{c}$ direction is believed to let the algorithm take a step towards smaller confidence error, with a step size proportional to the confidence error. The part in $\hat{x}$ is believed to let the algorithm take a larger step in the direction of smaller regression error if the confidence prediction was too high ($\bar{c} > e^{-\frac{|x-\hat{x}|}{T}\ln 2}$), and a larger step towards smaller regression error if the confidence prediction was too low ($\bar{c} < e^{-\frac{|x-\hat{x}|}{T}\ln 2}$), both with step sizes scaled with twice the confidence error. It is shown in Appendix D that the regression affecting part is bounded to $(-\frac{2}{T}e, \frac{2}{T}e)$.

Based upon the reasoning above, the L2-norm based confidence error was disregarded, with the same reasoning as then the L2-norm for regression error was abandoned in favor of the L1-norm. This led to only $J_{c,1}$ and $J_{c,2}$ being implemented. Early experiments, however, showed better performance with $J_{c,1}$, meaning it is the only one used in the final results.

4.2.5 Confidence Dependent Regression

As shown in Section 4.2.4, the addition of the confidence prediction to the network will affect the networks learning of regression prediction. This is shown in Equations 4.19d, 4.19e, 4.19f, and 4.19g. A behavior that was attributed to this was that regression prediction on networks sometimes was consistently bad for specific points in the sampling. These points i, j did often have a regression prediction $\bar{x}_i^{(j)}$ which seemed to be consistently off against the sought prediction $x_i^{(j)}$, by some constant value $\bar{\delta}_i^{(j)}$. The points confidence prediction $\bar{c}_i^{(j)}$ was seen to be consistently 0. This led to a situation where the regression for some point i, j was consistently off by a seemingly constant amount for all predictions from the network, while the network at the same time was confident in the fact that its regression in that point was sub-par. This is illustrated in Figure 4.15.

Note how the ninth and eleventh point of the ego-left lane (points 9, 2 and 11, 2) as well as the eight point of the ego-right lane (point 8, 3) has positional predictions considerably worse than the other detected lanes in the images, while their confidence prediction is 0.[2] The described behavior visible in both networks

[2]Note that the same points $(i, j) = (9, 2), (11, 2), (8, 3)$ showed this behavior for all predicted images from the given network.

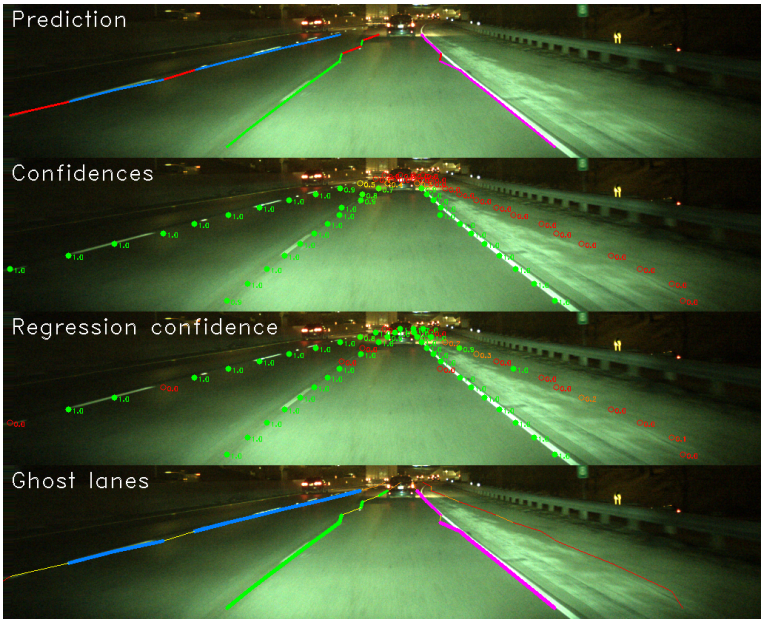


Figure 4.15: *Illustration of how confidence prediction affects regression. The top image illustrates the final network output (red segments indicate that the regression confidence is below 0.5), the second image from the top illustrates the existence variables, the third illustrates the regression confidences and the bottom image draws the lanes even when the corresponding existence variables are below the threshold.*

initialized with weights from networks trained without confidence predictions, as well as networks initialized with random weights.

It was theorized that this behavior was an effect of the network learning to consistently predict bad regression for some points, and give it low confidence. This behavior might be explained by the constructed loss function $J(\theta)$ allowing for a trade-off between regression quality $J_x(\theta)$ and confidence quality $J_c(\theta)$, where the network can achieve a lower loss by knowingly outputting bad regression and giving it low confidence, thus achieving a higher regression loss $J_x(\theta)$ and lower confidence loss $J_c(\theta)$ for some points. This was deemed plausible considering the effect of the confidence loss on the regression part of the gradient, as discussed in Section 4.2.4.

In order to avoid this behavior, two solutions were tested. One was to simply allow the network to train during more epochs and with carefully chosen weights λ and γ, hypothesizing that the network will eventually learn the correct predictions. This solution was based on the idea that the added complexity of predicting the confidence needed more training to converge. The other solution was to attempt to de-couple the regression prediction from the confidence prediction. This was hypothesized to allow the neurons responsible for prediction

regression and classification to focus on that task, while allowing other neurons to learn how well the regression performs in the given context. This led to the implementation of a Multi-stream Neural Network, described below.

Multi-stream Implementation for Confidence De-coupling

A Multi-stream implementation was designed to separate the regression prediction from the confidence prediction, with the goal of eliminating the behavior described in Section 4.2.5 and illustrated in Figure 4.15. The multi-stream implementation aims to let the network learn the regression prediction separately from the confidence prediction, by taking the output from the ENet Backbone and feeding it to two separate decoding heads. One of these heads is then used to predict regression and classification, while the other is used to predict confidence. The heads were constructed to be identical in layer design, and to then merge sigmoid layer, as illustrated in Figure 4.16. This was chosen to let the two heads independently extract the final features needed, and interpret them.

Given the network design shown in Figure 4.16 and the weight λ, it was deemed possible to first set $\lambda = 1$ to have the network only learn to predict the regression and classification. A network trained like this could then be further trained, but with $\lambda = 0$, to have it learn to predict confidences. This approach also allows the second training, with $\lambda = 0$, to freeze the weights of every layer except the ones in the confidence head, leading to the confidence prediction not being able to affect the learned classification and regression prediction in any way. Other possibilities was to set λ close to 1 during the first training, to allow the network to learn a bit about the confidence prediction. Similarly, during the second training, λ could be set close to 0, allowing the confidence prediction to affect the regression prediction and vice versa. Several combinations of this were tested.

As Figure 4.16 shows, the two heads have layers of the same shape during the two skip connections. Given this design choice, it was possible to directly add the output from a layer in one head to the corresponding layer in the other. Based on this, it was hypothesized that the confidence head could benefit from being given output from the regression head, once again applying the intuition that predicting a confidence x is intrinsically connected to predicting x. Therefore, it was deemed worthy to investigate the performance on the confidence prediction if the output of each of its skip-connections was fed the output from the corresponding skip-connection in the regression head. This is illustrated in Figure 4.17, where the added addition layers in the confidence head are visible, as well as its input from the regression head. Note how Figure 4.16 does not show the connections, nor the addition layers. This feature-sharing design did not show competitive results in early experiments, which when combined with the reasoning that the two tasks should be independent in the multi-stream designs, led to this design being abandoned.

Comparing the networks illustrated in Figures 4.14 and 4.16, and given that the two networks have identical ENet backbones, it is clear that the multi-stream network has a higher number of layers than the single-stream network. In order

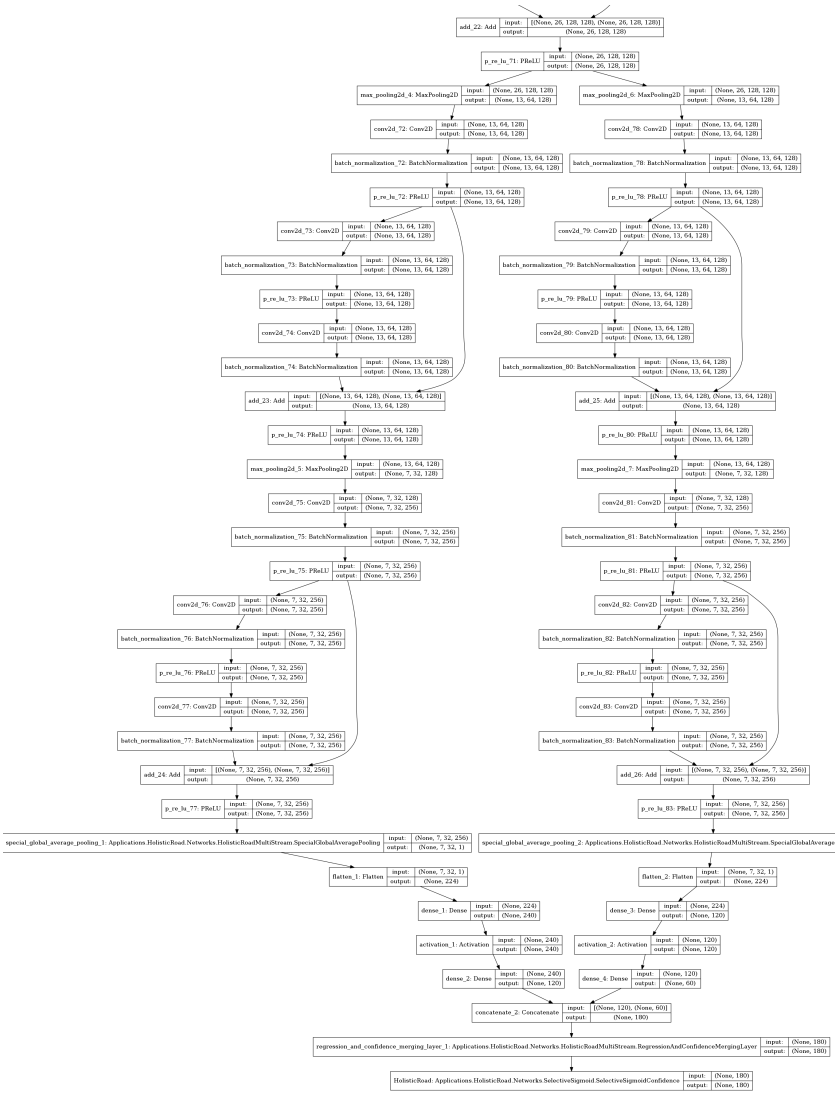


Figure 4.16: *Final layers of multi-stream network*

to evaluate the performance between the single- and multi-streamed networks given the same amount of layers, a variant of the multi-stream network was created where only one skip-connection was used in each head. To allow for this, the max-pooling before the first skip-connection was doubled in kernel size, effectively letting the output from the first skip-connection in the smaller head have the same output shape as the second skip-connection in the larger head. This is illustrated in Figure 4.18. Note the single skip in each head, and the larger max-pooling kernel before them, compared to Figure 4.16.

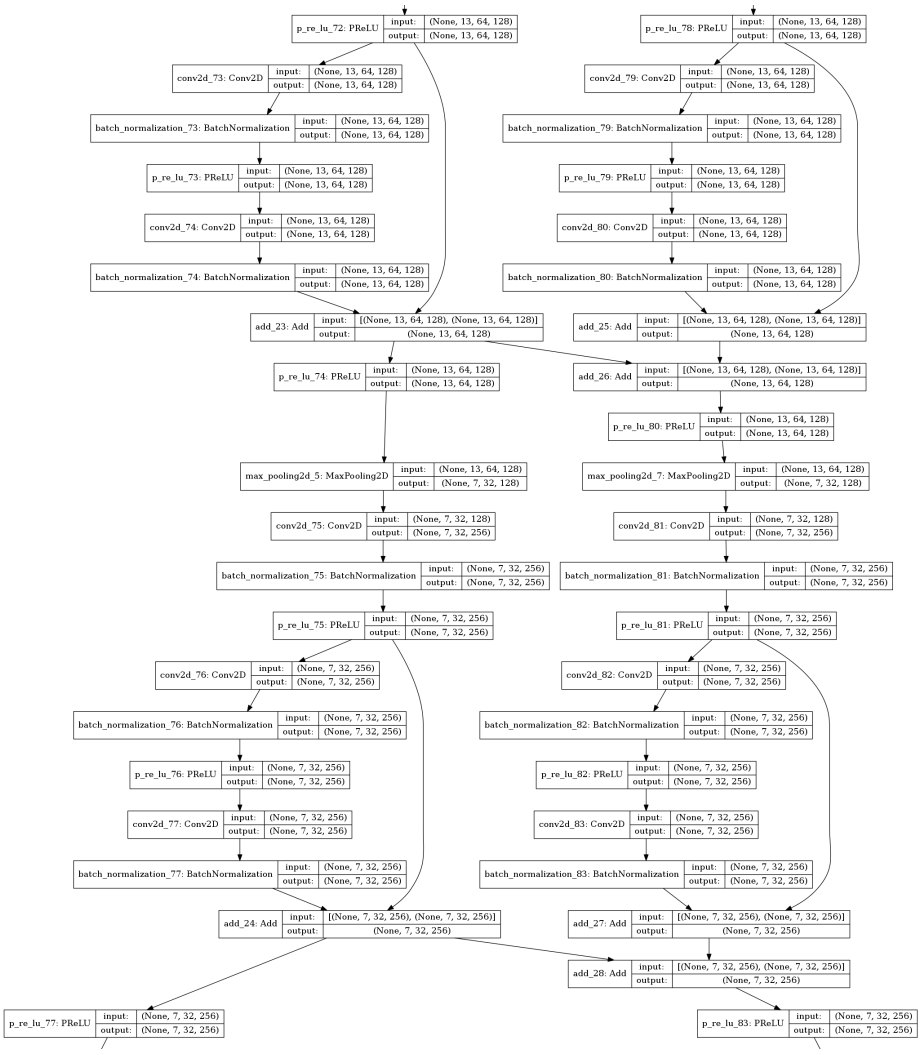


Figure 4.17: *Multi-stream network with connections between the heads*

4.2.6 List of Networks

All architectures tested in this study are listed below. The full networks can be found in Appendix E.

- **Full ENet**: This network was, as described in Section 4.2.3, used for the segmentation trainings. This means using the ENet backbone together with the ENet decoder as the head.
- **Network with 2 skip-connections**: This is the network used for the trainings without regression confidences. It features two blocks of ResNet-style

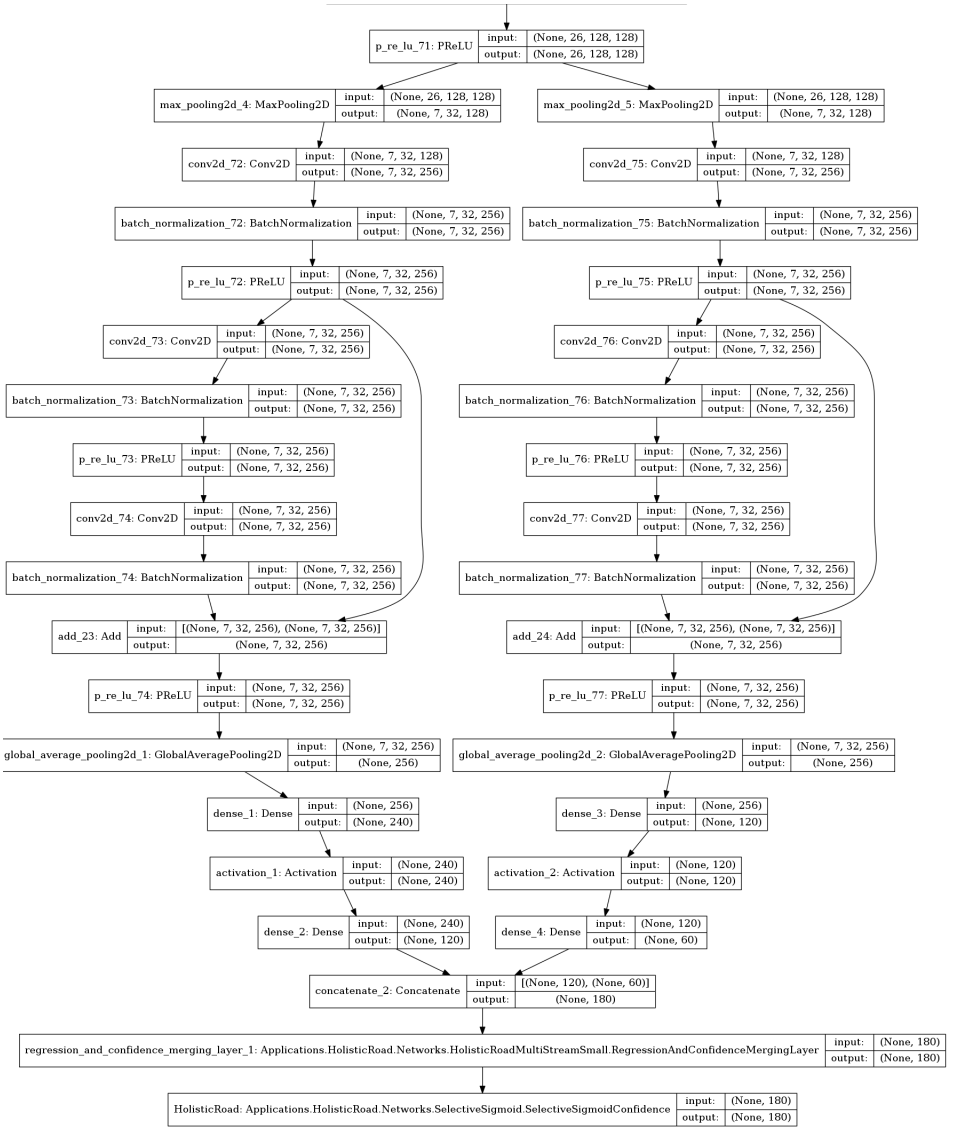


Figure 4.18: *Multi-stream network with one skip-connection removed and larger max-pooling kernel*

skip-connections. This network can be found in Appendix E.2.

- **Confidence network with 2 skip-connections**: This network is identical to the previous network, except that it has a larger output layer, since it outputs regression confidences. This network can be found in Appendix E.3.

- **Confidence network with 1 skip-connection**: This network also outputs

confidences, but uses only one ResNet-style skip-connection, and is thus shallower. This network can be found in Appendix E.3.

- **Multi-stream confidence network with 2 skip-connections**: This is one of the multi-stream networks described in Section 4.2.5. It features two almost identical branches with 2 ResNet-style skip-connections each, one for prediction of the x-positions and mask variables, and one for prediction of the regression confidences. The x-position and mask variable branch is identical to the *Network with 2 skip-connections*. This network can be found in Appendix E.5.

- **Multi-stream confidence network with 1 skip-connection**: This is identical to the other multi-stream network, but uses only one ResNet-style skip-connection. This network can be found in Appendix E.6.

4.3 Data Pre-processing

Ground-truth data of high quality is essential for training and evaluating neural networks. The data also needs be compatible with the network, meaning the data must be expressed in the representation which is to be output by the network. Unfortunately, the dataset provided by Veoneer is ground-truth data for semantic segmentation. This works for the semantic segmentation representation, but not the supporting points. For this reason, the data needs to be pre-processed to convert it to the supporting points representation. This section details how this was done, as well as problems faced and how they were overcome.

4.3.1 Converting Segmentation to Supporting Points

A proof-of-concept solution for a representation similar to supporting points was developed by Veoneer before the start of the thesis. This solution included an algorithm for converting the segmentation representation into the supporting points representation. For each lane-ID in the ground truth, the algorithm finds the points of the left or right edges of the segmentation polygons of the lane marking. These points are interpolated cubically for each lane, connecting points between lane markings when they are dashed. These interpolated lanes are then sampled at the desired rows, to create the supporting points representation. This algorithm was not modified in the thesis, only the rows at which the last sampling of the splines were changed. These rows were generated with $\alpha = 0.95$, $d_0 = 10$, $N = 35$ and $y_0 = 20$. An example ground truth generated from a segmentation image is shown in Figure 4.19.

4.3.2 Lane Ground truth Extrapolation and Resampling

It was noted that the ground-truth data produced by the ground-truth generator described in Section 4.3.1 produced lanes which did not continue fully down to

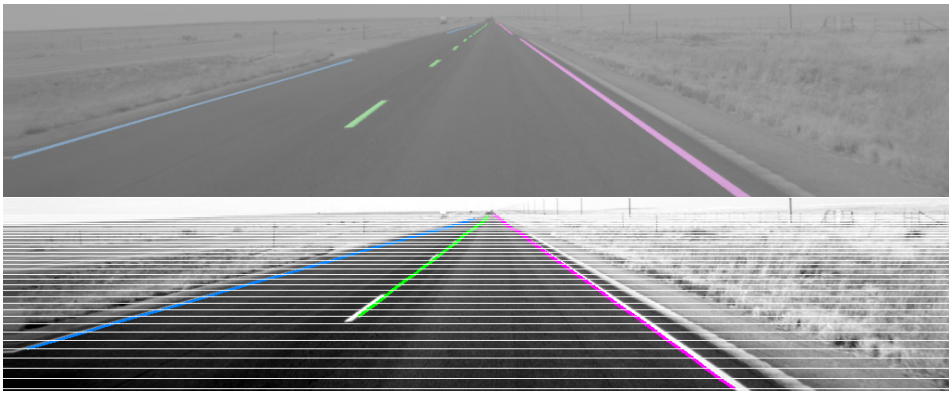


Figure 4.19: *A segmentation ground-truth image (top) and the generated supporting points from it (below)*

the bottom of the image. This was especially clear on roads with dashed markings, where the labels only stretched from the bottom-most visible marking to the topmost, and not entirely down to the end of the image, see Figure 4.20. This was a problem, as the networks should ideally understand that a dashed lane is a solid construct and continues along the road, even if the markings themselves by design were not continuous. Based on this observation, a method was developed to extrapolate lane markings to the bottom of the image when necessary.

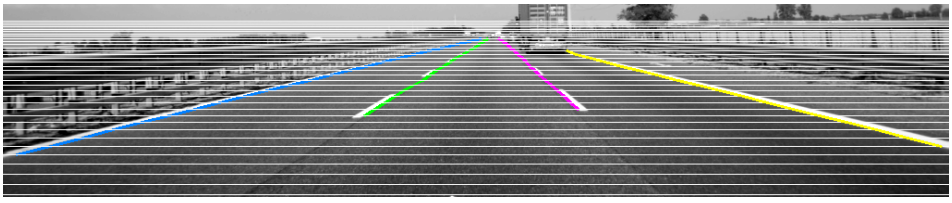


Figure 4.20: *An example of a ground-truth image where the lanes starts at the first visible markings*

This method consists of fitting univariate splines through the supporting points generated from the procedure described in Section 4.3.1. These splines were fitted with linear smoothing and extrapolation, allowing for the ground truth to be extrapolated towards the bottom of the image. The extrapolation was done by resampling the splines at new rows, to produce new supporting points. Care was taken to make sure the lanes are not extrapolated too far in the image, by limiting the amount of new points to add. For instance, if only two points are active in the original ground truth, extrapolating these will probably result in a lane which strays far away from the road boundary. This was especially a problem in road exits, as in Figure 4.21. Here, since the new lane begins at an incline to the rest of the road, extrapolating this would cause the lane boundary to cross the road in the ground truth, which is incorrect. For this reason, the neighbor-right lane was

subject to extra caution of not extrapolating too far. The lane extrapolation and resampling procedure is described in detail in Algorithm 1.

In the extrapolation procedure, the points are resampled such that they are more sparsely sampled than the original ground truth. These new rows were generated with $\alpha = 0.90$, $d_0 = 50$, $N = 15$ and $y_0 = 20$. The reason for doing this is to initially capture as much information as possible from the segmentation ground truth, by originally sampling them densely. It is however unnecessary for the neural networks to output such a dense representation, as a more sparse sampling can still represent virtually any road while limiting the degrees of freedom. Therefore, the lanes are sampled more sparsely to produce the final ground truth. See Figure 4.22 for an example of a ground-truth image where the lanes have been extrapolated and resampled.

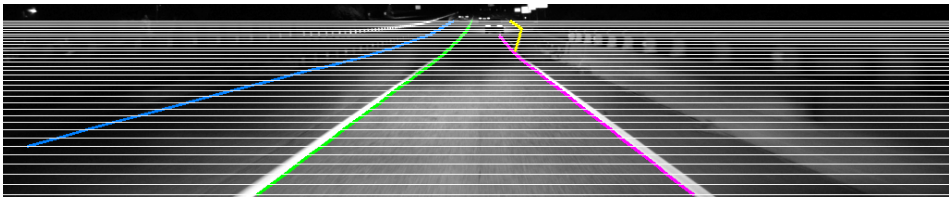


Figure 4.21: *Ground-truth image with road exit at the horizon. If neighbor-right (yellow) was to be extrapolated here, it would incorrectly cross the other lanes.*

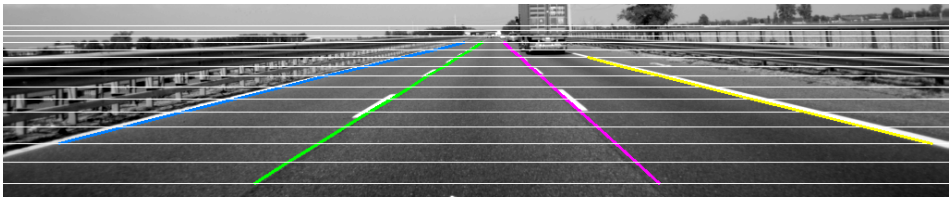


Figure 4.22: *Ground truth with lane continuation. A resampling to a less dense representation was also done with this image.*

4.3.3 Data Enhancement

It was discovered that several of the generated ground-truth images contained erroneous markings. These were caused by a number of issues, such as incorrect labeling in the original segmentation images, artifacts from the ground-truth generation process and badly extrapolated lanes. Some images had incorrect lane-ID:s, others had lane boundaries which did not correctly follow the lane and others had some or all markings missing. Examples of these situations are shown in Figure 4.23.

Since the ground truth in these images are incorrect, they should not be included in the dataset. However, it is highly impractical to manually review all

Algorithm 1 Extrapolation and resampling of lanes

Input: List of x-positions $\mathbf{x}^{(i)}$, y-positions $\mathbf{y}^{(i)}$ and existence variables $\mathbf{c}^{(i)}$ for each lane $i \in \{1, 2, 3, 4\}$, corresponding to neighbor-left, ego-left, ego-right and neighbor-right.
Input: New row positions $\mathbf{r}$ to sample the lanes at.
Output: Resampled and extrapolated supporting points $\hat{\mathbf{x}}^{(i)}$, $\hat{\mathbf{y}}^{(i)}$ and $\hat{\mathbf{c}}^{(i)}$.

Initialize set of splines $S = \emptyset$
for $i \in \{1, 2, 3, 4\}$ **do**
  Set $\bar{\mathbf{x}}^{(i)}$ and $\bar{\mathbf{y}}^{(i)}$ to the $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ for which $\mathbf{c}^{(i)} = 1$
  if the number of points in $\bar{\mathbf{x}}^{(i)}$ or $\bar{\mathbf{y}}^{(i)}$ is greater than 1 **then**
    Fit a univariate spline $s^{(i)}$ to $\bar{\mathbf{y}}^{(i)}$ and $\bar{\mathbf{x}}^{(i)}$ ($\bar{\mathbf{y}}^{(i)}$ as x-values and $\bar{\mathbf{x}}^{(i)}$ as y-values) with linear smoothing.
    Add $s^{(i)}$ to S.
  else
    Add null value to S.
  end if
end for

for all $s^{(i)}$ in S **do**
  Set $\hat{\mathbf{y}}^{(i)} = \mathbf{r}$
  if $s^{(i)}$ is null value **then**
    Set $\hat{\mathbf{x}}^{(i)} = \hat{\mathbf{c}}^{(i)} = \mathbf{0}$
  else
    ▷ Avoid extending road exits, thus have lower t on neighbor-right.
    Set extension threshold t to 4 if $i \neq 4$, otherwise 1.5.
    if extrapolation would increase the number of points more than t-fold **then**
      Set $y_{max} = \max(\bar{\mathbf{y}}^{(i)})$
    else
      Set y_{max} = image height
    end if
    Set $\mathbf{y}_{new}$ to the rows in $\mathbf{r} \geq \min(\bar{\mathbf{y}}^{(i)})$
    Interpolate $s^{(i)}$ using $\mathbf{y}_{new}$, producing $\mathbf{x}_{new}$
    Set $\hat{\mathbf{x}}^{(i)}$ to $\mathbf{x}_{new}$ padded with zeros, ensuring correct length of $\hat{\mathbf{x}}^{(i)}$
    Set $\hat{\mathbf{c}}^{(i)} = 0$ where the corresponding $\hat{\mathbf{x}}^{(i)}$ was zero-padded, set to 1 elsewhere.
    Set $\hat{\mathbf{c}}^{(i)} = \hat{\mathbf{x}}^{(i)} = 0$ where the corresponding y-position is greater than y_{max} or the $\hat{\mathbf{x}}^{(i)}$ is outside the image bounds.
  end if
end for

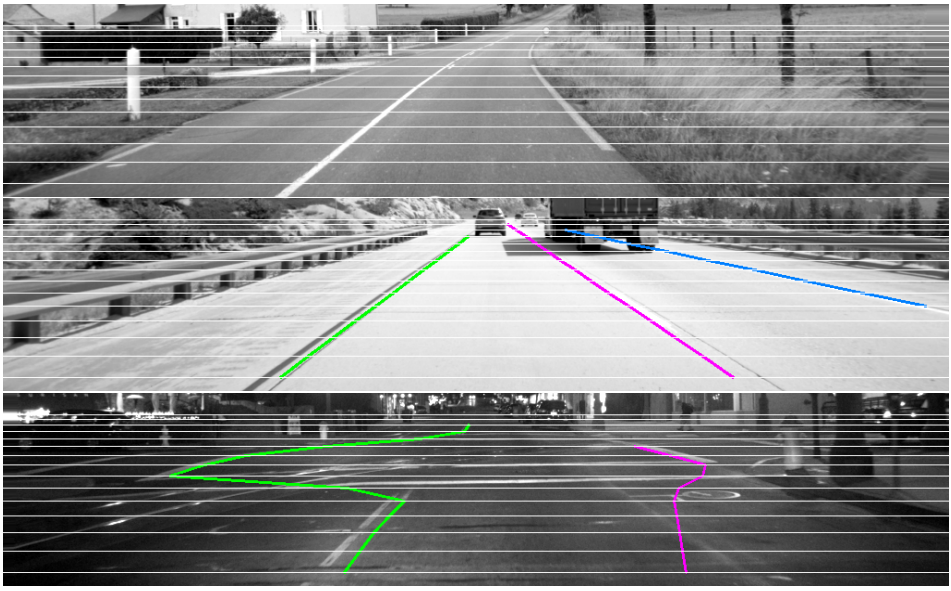


Figure 4.23: *Examples of bad ground truth. The bottom is caused by artifacts in the ground-truth generation, the middle has neighbor-right incorrectly labeled as neighbor-left (should be yellow), and the top image lacks lane markings entirely.*

images in the dataset, as the training set alone contains more than 40 000 images. For this reason, a method based on detecting suspicious ground-truth samples was developed. This way, only a small number of images need to be manually reviewed, instead of the entire dataset. The images confirmed to be bad ground truth in the manual review were removed from the dataset. Two different complementary detectors for detecting suspicious ground truth was developed, and are described in the following subsections.

Second Derivative-based Detector

The first detector for suspicious ground truth tried to identify situations like the bottom image in Figure 4.23. The characteristic property of these samples is that excessive sharp curvature, which actual road lanes typically do not exhibit. The main idea of the detector is based on the fact that if the x-positions of these lanes are expressed as functions of their y-positions, f, sharp curvature can be detected as a high absolute value of the function's second derivative f''.

The detector worked by first viewing the x-positions of each lane boundary as functions of their y-positions, as described in the previous paragraph, and removing linear trends from them. The second derivative of the functions were estimated, and if any of these had absolute values anywhere above a certain threshold t, it was marked as suspicious for manual review.

This turned out to be an efficient detector of this kind of erratic ground truth.

However, many false positives were produced by it, mainly highly curvy roads. Fortunately, a method for detecting curvy roads was developed for the purpose of weighting samples, which is described in Section 4.3.4. This detector was used to filter out the samples which had curvy roads which would otherwise have been labeled as suspicious, making the set of suspicious samples more relevant. This allowed for a lower threshold for detecting bad samples to be used, without making the set of suspicious samples excessively littered with false positives.

Deep Learning-based Detector

The detector based on the second derivative only detects ground truth with excessive jumps in the ground truth, such as the one in the bottom image of Figure 4.23. The other situations are hard to detect. For instance, it is hard to tell whether an image with no ground-truth lanes is missing the markings, or really should not have any lanes in it. Furthermore, there might be problems with the ground truth that has not even been discovered, and thus is impossible to write manual detectors for.

To solve this problem, it was realized that the neural networks themselves could be used as a detector. The best trained network at the time was used to output predictions on the dataset. These predictions were evaluated with regards to the mean absolute error for the regression, as well as the classification accuracy of the binary mask variables. The worst 100 predictions with respect to each performance metric was then labeled as suspicious, for manual review. This was repeated until the set of suspicious images were all false positives.

4.3.4 Dataset Imbalance

Since the available data mostly consists of images depicting a somewhat straight road with several lanes marked and clearly visible, some bias towards those situations are to be expected. The first versions of the system struggled with lane changes and curvy roads, in part because the dataset did not contain many of these samples. This unbalance in the database needed to be fixed in order to expect better performance on these situations.

As described in Section 3.7.4, oversampling is a suitable solution to this problem. However, to use this technique, one must be able to identify which samples are those with lane changes and those with curves, and the samples have no associated metadata with this information. Just as with the problem of identifying faulty ground truth, described in Section 4.3.3, the data volume makes manually stepping through the dataset and identifying all lane changes and curvy roads impractical. An automated procedure is needed here to be able to detect the lane change and curve samples, and repeat them in the dataset. The following sections describe how this was done.

Lane Change Detector

This detector takes the bottom-most point of ego-left and ego-right in the ground truth, and checks if they are approximately in the center of the image. It specifi-

cally checks whether they are within a certain radius r of the image center, since it was hypothesized that during lane changes, the vehicle drives over one of the ego-lane markings, which causes it to be perceived in the center of the image. Only the points at the bottom of the image are checked, since the points higher up in the image might be at the center even when no lane-change is occurring, due to foreshortening and curvature of the road.

Curvy Road Detector

This detector works in a similar way as the erroneous-ground-truth detector from Section 4.3.3, in the sense that it views the x-positions as functions of their y-positions. This, however, uses both the first and second derivative to determine whether a sample is a curve.

It first removes linear trends from the lane functions, to avoid straight but slanted lanes being identified as curves. Then, it estimates the first derivative, and checks its maximum absolute value. If it is above a certain threshold, t, it might be a curve, since only curves have a non-zero first derivative after being de-trended. This is not sufficient to determine whether it is a curve, but if any of the lane functions do not satisfy this property, it can not be a curve.

To further make sure the sample is a curve, the observation is made that if a curve is really in the sample, then all lane boundaries should curve in the same direction. The direction of a curve can be found out by the sign of its second derivative. Therefore, the second derivative of the curve is estimated, and the sign of its mean is calculated. If the sign is the same for all lane boundaries, then it is determined to be a curve sample. This means that only a sample having two or more active lane boundaries can be considered containing a curve.

Oversampling Procedure

The oversampling was done by first finding all curve samples and lane-change samples, using the detectors above. These samples are then repeated by some factor, and inserted into the list of samples. The samples are not repeated to such extent that they match the number of samples of non-lane-change samples and straight roads, as this was thought to cause overfitting.

4.3.5 Results of the Data Pre-processing

The dataset generator managed to convert segmentation ground truth to 41 562 training samples, 11 538 validation samples and 10 007 test samples. Using the second derivative based detector with a threshold $t = 60$, 469 bad training samples, 130 bad validation samples and 136 bad test samples were removed from these datasets. Using the deep learning based detector, another 302 bad training samples, 102 bad validation samples and 54 test samples were removed.

The training dataset was detected to have 510 lane change samples, using a radius $r = 50$. These were oversampled such that the amount of lane changes increased by a factor of 6. Similarly, 321 curvy samples was detected using a threshold of 25. These were oversampled by a factor of 7. The non-oversampled

training set therefore contained 40 791 samples, and the oversampled one contained 45 191 samples.

4.4 Lane-change Discontinuity

Reviewing the performance of different networks using both the Supporting Points representation described in Section 4.1.1 and the Supporting Points with Confidences described in Section 4.1.2, it was a common denominator for all networks to drop in performance for a lane-change sequence, that is, a sequence when the vehicle change from one lane to another. This is illustrated in Figure 4.24, where one may observe how the lane boundaries are predicted in the middle of the lanes when one of the lane markings is positioned in the image center.

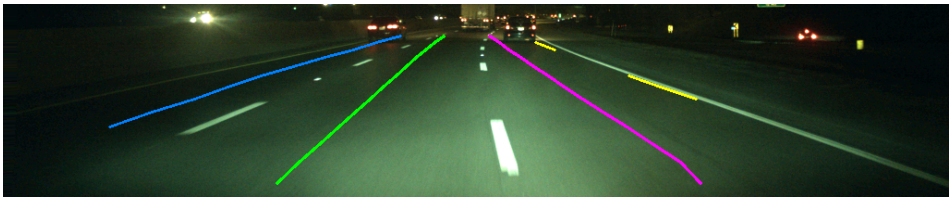


Figure 4.24: *Example of lane-change situation with bad prediction*

The sudden drop in regression performance during a lane-change, as clearly visible in Figure 4.24 was theorized to be an intrinsic effect of the supporting points representation, combined with the intrinsic question of "at what time do the lane crossing occur", leading to inconsistent labeling of the ground truth by humans as well. The human error can be explained by the fact that, judging from an image on a computer screen, it is not evident when the crossing of the lane occurs - that is, when do ego-left become ego-right for a lane-change to the left, et cetera.

The intrinsic effect of the supporting points representation is that the position of each point is connected to the ID of the lane it lies on. This is best explained with the following example: Consider a lane $\mathbf{x^i}$, with ID i. If the network predicts a lane $\bar{\mathbf{x}}^j$, with the same exact positions as $\mathbf{x^i}$, but with $i \neq j$, the loss will be non-zero. This follows from the fact that the loss function $J(\theta)$ compares every estimation $\bar{x}_i^{(j)}$, $\bar{m}_i^{(j)}$ and $\bar{c}_i^{(j)}$ with the corresponding ground-truth value $x_i^{(j)}$, $m_i^{(j)}$ and $c_i^{(j)}$. If the ground truth has "performed the lane-change" and thus have $x_i^{(j)} = \bar{x}_{i\pm1}^{(j\pm1)}$ the estimated position, classification and confidence will be identical to ground truth, but will still generate a high loss given the incorrect ID.

In the perfect lane-change scenario, the outputted lane ids match the ids of the ground truth. This is illustrated in Figure 4.25, where the predicted, dashed lanes match the ground truth solid lanes, both in position and ID. Here the position is represented by the lines being close together, and the ID is represented by the color of the lines.

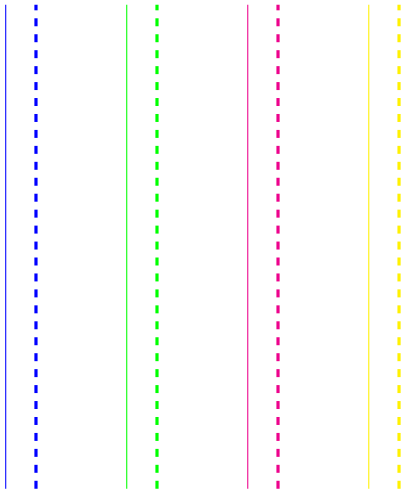


Figure 4.25: *Lane change situation with predictions (dashed) correct with regards to ground truth (solid)*

As Figure 4.25 shows, this situation require no further handling and may be fed to the loss function directly, expecting a low loss.

Illustrated in Figure 4.26 is a situation where the predicted lanes from the network are one lane off to the right, corresponding to a left-wise lane change where the ground truth has "performed the change" earlier than the network, or a right-wise lane change where the network has "performed the change" before the ground truth. As one can see, three of the predicted lanes, neighbor-left (blue), ego-left (green), ego-right (magenta), are still positioned correctly but labeled incorrectly. Here neighbor-left corresponds to the ground truths ego-left, ego-left corresponds to the ground truths ego-right and ego-right corresponds to the ground truths neighbor-right.

Note in Figure 4.26 that the network does not detect the lane that the ground truth labels as neighbor-left (solid blue), while detecting a lane that the ground truth has not labeled (dashed yellow). This is an effect of the chosen representation, where only four lanes are available at the same time.

Figure 4.27 illustrate the other possible erroneous situation, where the predicted lanes from the network are one lane off to the left, corresponding to a right-wise lane change where the ground truth has "performed the change" earlier than the network, or a left-wise lane change where the network has "performed the change" before the ground truth. Similarly to Figure 4.26, Figure 4.27 shows how three of the predicted lanes, neighbor-right (yellow), ego-right (magenta) and ego-left (green), are positioned correctly but labeled incorrectly. Here neighbor-right corresponds to the ground truths ego-left, ego-right corresponds to the ground truths ego-left and ego-left corresponds to the ground truths neighbor-left.

As Figure 4.26, Figure 4.27 shows the network detecting a lane to the left of the ground truth (neighbor-left, dashed blue) without corresponding ground-

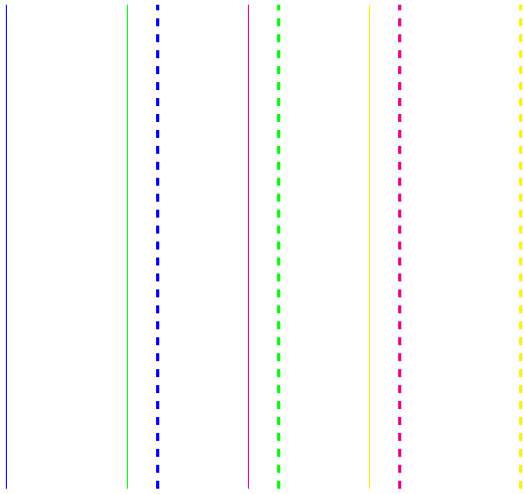


Figure 4.26: *Lane change situation with predictions (dashed) one lane off to the right with regards to ground truth (solid)*

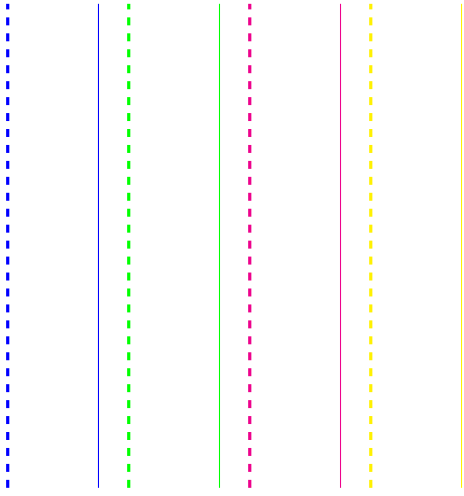


Figure 4.27: *Lane change situation with predictions (dashed) one lane off to the left with regards to ground truth (solid)*

truth lane, while the ground-truth labels a lane to the right of the prediction (neighbor-right, solid yellow) without corresponding prediction. This is, as discussed above, an effect of the chosen four-lane representation.

As Figures 4.26 and 4.27 shows, there may occur lane-change situations during which the network actually outputs lanes close to the ones in the ground truth, but the inherent difficulty of telling exactly when a lane change occurs and ego-right becomes ego-left or vice versa leads to a discrepancy between the

ground truth and the predictions. This discrepancy results in a high loss, and gradients pointing in the opposite direction. Take the situation in Figure 4.27 as an example: the predicted ego-left (dashed green) is almost perfect compared to the ground-truth neighbor-left (solid blue), but the negative gradient direction will be towards the ground-truth ego-left (solid green), which is away from the closer lane. This pushes the network to predict lanes further away from the closest one in lane-change situations, which is hypothesized to cause the behavior illustrated in Figure 4.24.

Returning to Figure 4.26, it is clear that if the ground-truth ids were to be shifted one step to the left, letting ego-left become neighbor-left, ego-right become ego-left and neighbor-right become ego-right, the ground-truth ids would correspond with the prediction ids, and the loss would therefore be smaller. More importantly, the negative gradient direction would point towards the closest lane, and not away from it. This is illustrated in Figure 4.28. Note how the old ground-truth neighbor-left, here red, no longer has a corresponding predicted lane. The same happens to the predicted neighbor-right, here dashed red.

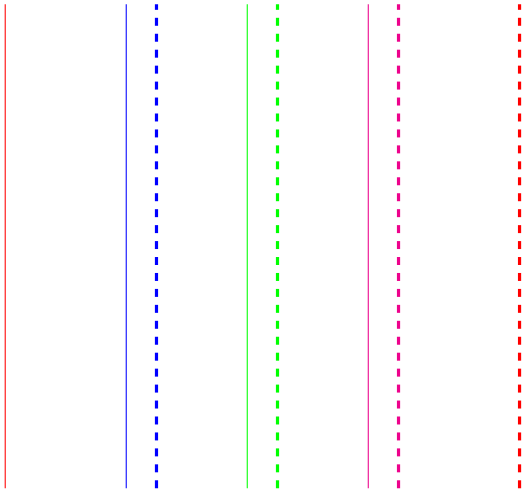


Figure 4.28: *Lane change where the ground truth has been shifted one step to the left to match the prediction ids*

From Figure 4.28 it is clear that every prediction closely corresponds to the ground truth, with an exception for neighbor-right since there is no ground truth available for that lane. If neighbor-right were to be ignored, together with the no longer necessary old ground-truth neighbor-left, the remaining three lanes would give a low loss and correct gradient when fed to the loss function. This is illustrated in Figure 4.29.

The corresponding steps and their effects on the situation illustrated in Figure 4.27 are visible in Figures 4.30 and 4.31.

This behavior led to the development of an algorithm which shifts the lanes when needed, with the goal of letting the network learn to give better predictions

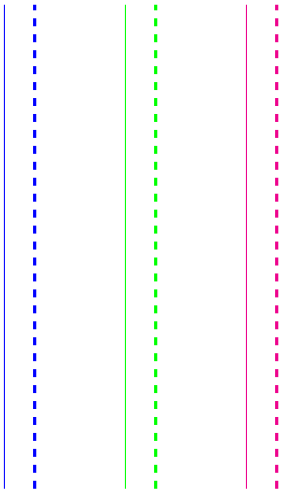


Figure 4.29: *Lane change situation with ground truth shifted to the left and leftover lanes removed*

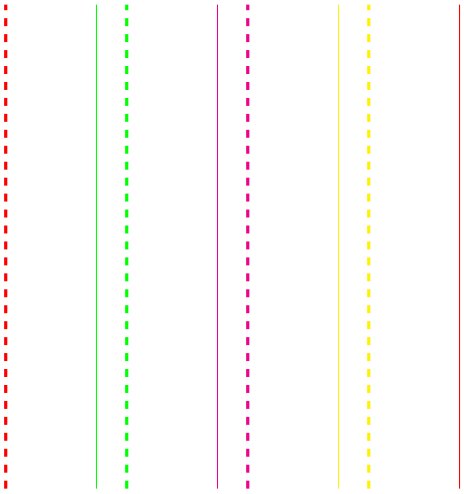


Figure 4.30: *Lane change where the ground truth has been shifted one step to the right to match the prediction ids*

during lane-change situations and not be affected by gradients pointing in the wrong direction.

4.4.1 Lane-change Shift Algorithm

The *Lane-change Shift* Algorithm is based upon the intuition described in Section 4.4, which states that there exists situations where the ids of the prediction

Figure 4.31: *Lane change where the ground truth has been shifted one step to the right and leftover lanes removed.*

does not match the ids of the ground truth, even if the predicted values closely match the ground-truth values. Therefore, in order to allow the networks to train correctly, it is sometimes necessary to shift the ground-truth ids in order to match the ones of the predictions.[3] Since there is no prior knowledge when the shifts are to be done, except that there might be need for them in a lane-change situation, but not all lane-change situations, the idea was to calculate the loss for shifted ground truth, and compare it to the loss for non-shifted ground truth. It was noted, as illustrated in Figures 4.28 and 4.30, that every shift results in one prediction lacking ground truth. Removing one lane in that way in the given representation corresponds to setting both the prediction and the ground-truth classifications to $m_i^{(j)} = n_i^{(j)} = 0$ for constant lane j and $i \in [1, N]$, which in accordance to the loss function $J(\theta)$ results in loss 0 for said lane j. Since $J(\theta)$ is constructed to give strictly non-negative losses, having one lane giving 0 loss results in a lower loss than having that lane giving any other loss. Since a loss equal to 0 is deemed to occur naturally with probability 0, this was deemed to risk leading to an unfair comparison between the shifted and non-shifted loss, with advantage to the shifted. Therefore, the choice was made to also zero-out the corresponding lane in the non-shifted prediction, only comparing three lanes for every lane-change situations. The Lane-change Shift Algorithm is described in Algorithm 2

As shown in Algorithm 2, the classification ground truth is not changed. This is a design choice made to separate the learning of position from the learning of classification in the lane-change case. This is motivated by the fact that the problems and ambiguity in the positional predictions should be avoided in the

[3]Note that the effect is supposed to be the same if the prediction is shifted instead, and that the choice was to shift the ground truth was arbitrarily made by the authors.

Algorithm 2 Lane-change shift algorithm

Input: List of ground-truth x-positions $\mathbf{x}^{(i)}$ and confidences $\mathbf{c}^{(i)}$ for each lane $i \in \{1, 2, 3, 4\}$, corresponding to neighbor-left, ego-left, ego-right and neighbor-right for one sample.
Input: List of predicted x-positions $\bar{\mathbf{x}}^{(i)}$ and confidences $\bar{\mathbf{c}}^{(i)}$ for each lane $i \in \{1, 2, 3, 4\}$, corresponding to neighbor-left, ego-left, ego-right and neighbor-right for one sample.
Input: Image width d and tolerance distance r in pixels.
Output: Loss $E(\mathbf{x})$ for the sample.

Decide whether the sample is a lane-change sample, and what type
if $|\frac{d}{2} - x_N^{(2)}| < r$ **then** ▷ Ground truth has ego-left close to the image center.
Create mask predictions without neighbor-left: Set $\dot{\mathbf{m}}^{(j)} = \begin{cases} 0, & \text{if} j = 1 \\ \mathbf{m}^{(j)}, & \text{otherwise} \end{cases}$
Create mask ground truth without neighbor-left: Set $\dot{\mathbf{n}}^{(j)} i = \begin{cases} 0, & \text{if } j = 1 \\ \mathbf{n}^{(j)}, & \text{otherwise} \end{cases}$
Create position ground truth shifted left: Set $\mathbf{y}^{(j)} = \begin{cases} 0, & \text{if } j = 1 \\ \mathbf{x}^{(j-1)}, & \text{otherwise} \end{cases}$
Calculate loss for non-shifted case $E_n = J(\dot{\mathbf{n}}, \mathbf{x}, \dot{\mathbf{m}}, \bar{\mathbf{x}})$
Calculate loss for shifted case $E_s = J(\dot{\mathbf{n}}, \mathbf{y}, \dot{\mathbf{m}}, \bar{\mathbf{x}})$
Set loss to be minimum of the shifted and non-shifted loss: $E = \min(E_s, E_n)$
else if $|\frac{d}{2} - x_N^{(3)}| < r$ **then** ▷ Ground truth has ego-right close to the image center.
Create predictions without neighbor-right: Set $\dot{\mathbf{m}}^{(j)} = \begin{cases} 0, & \text{if } j = 4 \\ \mathbf{m}^{(j)}, & \text{otherwise} \end{cases}$
Create ground truth without neighbor-right: Set $\dot{\mathbf{n}}^{(j)} = \begin{cases} 0, & \text{if } j = 4 \\ \mathbf{n}^{(j)}, & \text{otherwise} \end{cases}$
Create ground truth shifted right: Set $\mathbf{y}^{(j)} = \begin{cases} 0, & \text{if } j = 4 \\ \mathbf{x}^{(j+1)}, & \text{otherwise} \end{cases}$
Calculate loss for non-shifted case $E_n = J(\dot{\mathbf{n}}, \mathbf{x}, \dot{\mathbf{m}}, \bar{\mathbf{x}})$
Calculate loss for shifted case $E_s = J(\dot{\mathbf{n}}, \mathbf{y}, \dot{\mathbf{m}}, \bar{\mathbf{x}})$
Set loss to be minimum of the shifted and non-shifted loss: $E = \min(E_s, E_n)$
else ▷ Ground truth has neither ego-left or ego-right in the image center, or both - all corresponding to non-lane-change scenarios.
Calculate loss normally $E = J(\theta, \mathbf{x})$
end if

classification prediction.

4.4.2 Smooth Minimum

It should be noted that the solution presented in Algorithm 2 makes use of the minimum function $\min(x, y) = \begin{cases} x, & \text{if } x \leq y \\ y, & \text{otherwise} \end{cases}$ which is not differentiable along the line $x = y$. This was theorized to potentially affect the networks ability to learn, since there may be cases, especially early on in the training, where both the shifted and the non-shifted losses are close together. Thus the discontinuous derivative of the min-function may result in suboptimal gradients for the algorithm in the cases when $E_n \approx E_s$.

As mentioned in Section 2.4, a *Smooth Minimum* function $\mu_\alpha(x, y)$ can be defined as

$$\mu_\alpha(x, y) = \frac{xe^{-\alpha x} + ye^{-\alpha y}}{e^{-\alpha x} + e^{-\alpha y}} \tag{4.20}$$

Thus, modifying Algorithm 2 to use the *smooth minimum* from Equation 4.20 function results in Algorithm 3.

The parameter α, which controls the smoothness of the function, adds yet another parameter to the investigation. Since the distance between the ground-truth position x and the predicted one $\bar{x}$ may be in the hundreds, given the image size, it was quickly discovered that $\alpha > 2$ was not feasible due to hard-ware restrictions. With an error of 100, $e^{-200} \approx 1.4 * 10^{-87}$, which is clearly outside the $\sim 10^{-30}$ precision which was available. Therefore $\alpha = 2$ was chosen, and the inputted losses were scaled down by a factor of 20, after which the outputted minimum loss was scaled up by a factor of 20, resulting in the following implementation of $\mu(x, y)$:

$$\mu(x, y) = \frac{xe^{-\alpha \frac{x}{20}} + ye^{-\alpha \frac{y}{20}}}{e^{-\alpha \frac{x}{20}} + e^{-\alpha \frac{y}{20}}} \tag{4.21}$$

This allowed the numbers to be within precision, while maintaining the wanted smoothness. It should be noted that $20\mu(\frac{x}{20}, \frac{y}{20}) \neq \mu(x, y)$ for $x \neq y$. This was not deemed crucial, since the wanted minimum behavior of $20\mu(\frac{x}{20}, \frac{y}{20}) \in [x, y]$ for $(x, y) \in \mathbb{R}, x \leq y$ still holds.

4.5 Performance Evaluation

The networks were evaluated on the test dataset, which contained 9817 samples (after the ground-truth cleaning described in Section 4.3.3). The performance was evaluated using the following metrics which were all averaged over the dataset:

- **Mean Absolute Error (MAE)**: This measures the general error in the regression. For predicted regression positions $\bar{x}_j^{(i)}$, ground-truth positions $x_j^{(i)}$ and

Algorithm 3 Lane-change shift algorithm with smooth minimum

Input: List of ground-truth x-positions $\mathbf{x}^{(i)}$ and confidences $\mathbf{c}^{(i)}$ for each lane $i \in \{1, 2, 3, 4\}$, corresponding to neighbor-left, ego-left, ego-right and neighbor-right for one sample.
Input: List of predicted x-positions $\bar{\mathbf{x}}^{(i)}$ and confidences $\bar{\mathbf{c}}^{(i)}$ for each lane $i \in \{1, 2, 3, 4\}$, corresponding to neighbor-left, ego-left, ego-right and neighbor-right for one sample.
Input: Image width d and tolerance distance r in pixels.
Output: Loss $E(\mathbf{x})$ for the sample.

Decide whether the sample is a lane-change sample, and what type
if $|\frac{d}{2} - x_N^{(2)}| < r$ **then** ▷ Ground truth has ego-left close to the image center.
Create mask predictions without neighbor-left: Set $\dot{\mathbf{m}}^{(j)} = \begin{cases} 0, & \text{if} j = 1 \\ \mathbf{m}^{(j)}, & \text{otherwise} \end{cases}$
Create mask ground truth without neighbor-left: Set $\dot{\mathbf{n}}^{(j)} i = \begin{cases} 0, & \text{if } j = 1 \\ \mathbf{n}^{(j)}, & \text{otherwise} \end{cases}$
Create position ground truth shifted left: Set $\mathbf{y}^{(j)} = \begin{cases} 0, & \text{if } j = 1 \\ \mathbf{x}^{(j-1)}, & \text{otherwise} \end{cases}$
Calculate loss for non-shifted case $E_n = J(\dot{\mathbf{n}}, \mathbf{x}, \dot{\mathbf{m}}, \bar{\mathbf{x}})$
Calculate loss for shifted case $E_s = J(\dot{\mathbf{n}}, \mathbf{y}, \dot{\mathbf{m}}, \bar{\mathbf{x}})$
Set loss to be minimum of the shifted and non-shifted loss: $E = \mu_\alpha(E_s, E_n)$
else if $|\frac{d}{2} - x_N^{(3)}| < r$ **then** ▷ Sample has ego-right close to the image center.
Create predictions without neighbor-right: Set $\dot{\mathbf{m}}^{(j)} = \begin{cases} 0, & \text{if } j = 4 \\ \mathbf{m}^{(j)}, & \text{otherwise} \end{cases}$
Create ground truth without neighbor-right: Set $\dot{\mathbf{n}}^{(j)} = \begin{cases} 0, & \text{if } j = 4 \\ \mathbf{n}^{(j)}, & \text{otherwise} \end{cases}$
Create ground truth shifted right: Set $\mathbf{y}^{(j)} = \begin{cases} 0, & \text{if } j = 4 \\ \mathbf{x}^{(j+1)}, & \text{otherwise} \end{cases}$
Calculate loss for non-shifted case $E_n = J(\dot{\mathbf{n}}, \mathbf{x}, \dot{\mathbf{m}}, \bar{\mathbf{x}})$
Calculate loss for shifted case $E_s = J(\dot{\mathbf{n}}, \mathbf{y}, \dot{\mathbf{m}}, \bar{\mathbf{x}})$
Set loss to be minimum of the shifted and non-shifted loss: $E = \mu_\alpha(E_s, E_n)$
else ▷ Sample has neither ego-left or ego-right in the image center, or both - all corresponding to non-lane-change scenarios.
Calculate loss normally $E = J(\theta, \mathbf{x})$
end if

ground-truth existence variables $n_j^{(i)}$ for lane i and row j it is defined as

$$\text{MAE}(\mathbf{x}, \bar{\mathbf{x}}) = \frac{1}{N} \sum_{i,j} n_j^{(i)} |\bar{x}_j^{(i)} - x_j^{(i)}|, \tag{4.22}$$

where N is the number of points where $n_j^{(i)} = 1$. For semantic segmentation, the x-positions are set to 0 when the mask variables are set to 0, since the network does not directly predict regression. This means the regression error will be incorrect when the ground truth mask $n_j^{(i)} = 1$ but the predicted $m_j^{(i)} = 0$. Therefore, for the semantic segmentation, the regression error is only calculated when both $n_j^{(i)} = 1$ and $m_j^{(i)} = 1$:

$$\text{MAE}(\mathbf{x}, \bar{\mathbf{x}}) = \frac{1}{N} \sum_{i,j} n_j^{(i)} m_j^{(i)} |\bar{x}_j^{(i)} - x_j^{(i)}|, \tag{4.23}$$

where N is the number of points where $n_j^{(i)} m_j^{(i)} = 1$.

- **Classification Accuracy (CA)**: This measures how well the network detects the existence of lanes. The predicted existence variables $m_j^{(i)}$ are threshold such that $\hat{m}_j^{(i)} = \begin{cases} 1, & m_j^{(i)} \geq 0.5 \\ 0, & m_j^{(i)} < 0.5 \end{cases}$. The classification accuracy measures the fraction of $\hat{m}_j^{(i)}$ which equal the ground-truth existence variables $n_j^{(i)}$.

- **Curve MAE/CA**: This measures the MAE and CA on samples with curves. The curve detector from Section 4.3.4 is used to find the samples from the test set with curves, on which the MAE and CA is calculated.

- **Lane Change MAE (LC-MAE)**: This measures the MAE on samples with lane changes, with modification. Since it is ambiguous which lane should have which ID during a lane change, the MAE is calculated both normally and when the lanes have shifted ID:s to the left or right (depending on if it is ego-left or ego-right close to the center of the image) and the smallest loss chosen, in a similar way as the Lane Change-shift algorithm from Section 4.4.1. This way, it becomes a measure of how well the regression performs during lane changes with regards to positions, and not lane-ID:s. The samples with lane changes are extracting using the lane change detector from Section 4.3.4.

- **Confidence Mean Absolute Error (cMAE)**: This metric measures how well the network estimates the regression confidences, when applicable. This is simply the MAE between the predicted confidence and what it should be given the predicted regression. What the predicted position should be is given by Equation 4.2 from Section 4.1.2.

- **Existence Confusion Matrix**: Here, the classification performance of the existence variables is evaluated in terms of amount of TP, TN, FP and FN. The existence predictions are thresholded with five thresholds between 0.5 and 0.9, and whenever the value is above the threshold, it is considered a positive detection, otherwise negative. If it equals the ground-truth value, it is true, otherwise false. For each threshold, a confusion matrix containing the TP:s, TN:s, FP:s and FN:s, as percentage of the total amount of points, is crated.
- **Confidence Confusion Matrix**: This is the same as the Existence Confusion Matrix, but for the regression confidence predictions.

For optimal performance, all **MAEs** are 0, while all **CAs** are 1, and the **Confusion Matrices** have 0% FP and FN. Based on that, low **MAEs**, high **CAs** and low FP% and FN% are sought.

4.5.1 Semantic Segmentation

The performance of the semantic segmentation was evaluated by converting the segmentation output to the supporting points representation, and using the same ground truth and evaluation metrics as the other trainings.

Since ground-truth labeling semantic segmentation images is significantly slower than drawing simple lines along the lanes, it was deemed interesting to compare the segmentation to the supporting points when the segmentation network has been trained on only 25% the amount of data as the supporting points. Therefore, segmentation trainings were done with both the full dataset, and with the dataset being reduced by a factor of 4.

4.6 Network Training

The networks were implemented using a deep learning framework developed by Veoneer, which in turn uses the framework Keras[7]. All networks were trained using cosine learning rate annealing, often with different starting learning rates. For all trainings except the semantic segmentation, data augmentation was used, where the image was randomly horizontally flipped for some samples. Regression and classification weight $\gamma = 0.5$ was used in all trainings.

In Chapter 5, the results from training on several different configurations are presented, along with said configurations. The reasoning for presenting these results are as follows:

- **Regular Absolute Error Loss (RAE)**: Baseline training to compare modifications to.
- **Horizon Loss (HL)**: Investigate the effect of the *Horizon loss*.
- **Lane Change Loss with Hard Minimum (LChM)**: Investigate the effect of the *Lane Change Algorithm*.

- **Lane Change Loss with Smooth Minimum (LCsM)**: Compare the effect of the two minimum functions.
- **Lane Change Loss with No Oversampling (LCnO)**: Investigate the effect of Oversampling lane change samples.
- **Confidences on Larger Network with Pre-training (cLP)**: Investigate the effect of adding confidences to the output.
- **Confidences on Smaller Network with Pre-training (cSP)**: Compare different network sizes for confidence prediction.
- **Confidences on Larger Network without Pre-training (cLnP)**: Compare whether pre-training is beneficial for confidence prediction or not.
- **Confidences on Smaller Network without Pre-training (cSnP)**: Compare whether pre-training is beneficial for confidence prediction or not, with network size in consideration.
- **Confidences on Larger Multi-stream Network (cLM)**: Investigate if a multi-stream implementation improves performance for confidence outputting networks.
- **Confidences on Smaller Multi-stream Network (cSM)**: Compare the effect of network size on the multi-stream confidence outputting network.

5
Results

In this chapter, the training results most relevant for answering the research questions are presented. The reasoning behind the relevance of the trainings are described in Section 4.6.

The results are shown in the following sections with their settings and training curves. For each training, the weights from the epoch at which the network produced the lowest validation regression mean absolute error were chosen, except with the semantic segmentation and the multi-stream networks, in which case the model with the lowest validation loss was chosen. The metrics for all presented trainings are seen in Table 5.1 in Section 5.6. This was done to increase readability and ease of comparison.

5.1 Baseline Trainings

This section shows the results from the trainings done without the lane change loss and regression confidences.

5.1.1 Regular Absolute Error Loss (RAE)

This training had the following settings:

- **Network:** Network with 2 skip-connections (see Appendix E.2)
- **Learning rate:** 0.005
- **Number of epochs**: 60
- **Pretraining**: Initialized ENet-backbone with pretrained weights, trained with absolute error loss and regression confidences.

Example prediction images produced by this training are shown in Figures A.1 and A.2 in Appendix A, with a specific lane change sequence shown in Figure B.1 in Appendix B. The training curves are shown in Figure 5.1.

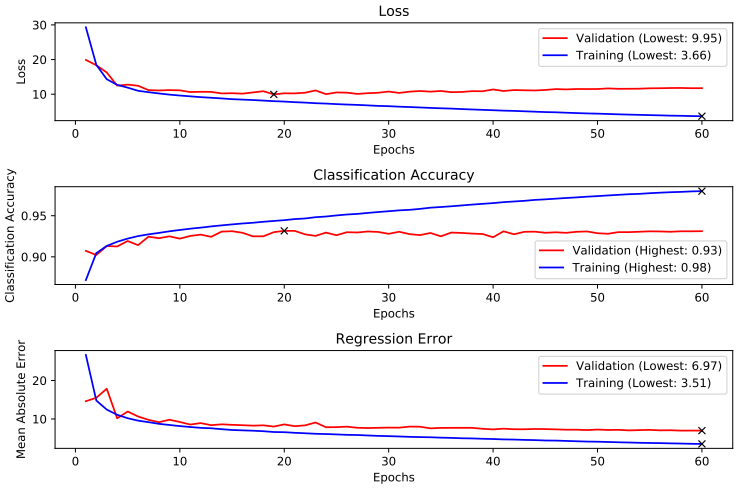


Figure 5.1: *Training curves for the training*

Below are confusion matrices for the existence variables:

Threshold 0.5	
TP: 39.7%	**FP**: 3.3%
FN: 3.5%	**TN**: 53.5%

Threshold 0.6	
TP: 39.2%	**FP**: 2.9%
FN: 3.9%	**TN**: 53.9%

Threshold 0.7	
TP: 38.7%	**FP**: 2.5%
FN: 4.4%	**TN**: 54.3%

Threshold 0.8	
TP: 38.1%	**FP**: 2.1%
FN: 5.1%	**TN**: 54.7%

Threshold 0.9	
TP: 36.9%	**FP**: 1.6%
FN: 6.2%	**TN**: 55.2%

5.1.2 Horizon Loss (HL)

This training had the following settings:

- **Network:** Network with 2 skip-connections (see Appendix E.2)
- **Learning rate:** 0.005
- **Number of epochs**: 60

- **Pretraining**: Initialized ENet-backbone with pretrained weights, trained with absolute error loss and regression confidences.

Example prediction images produced by this training are shown in Figures A.3 and A.4 in Appendix A. The training curves are shown in Figure 5.2.

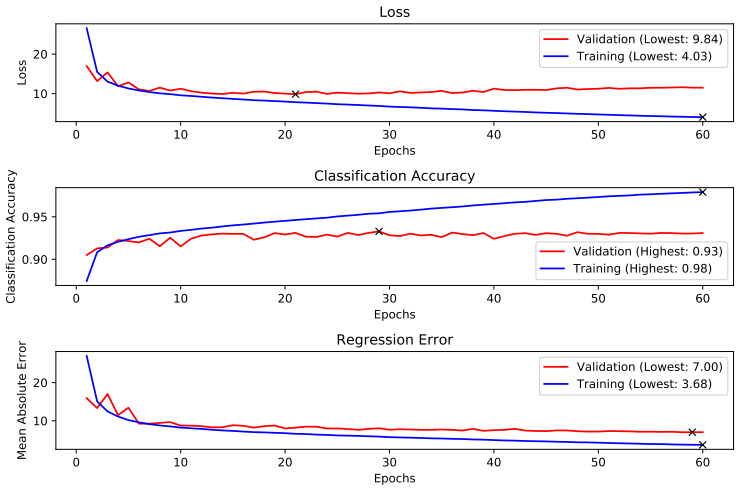


Figure 5.2: *Training curves for the training*

Below are confusion matrices for the existence variables:

Threshold 0.5	
TP: 39.6%	**FP**: 3.4%
FN: 3.6%	**TN**: 53.4%

Threshold 0.6	
TP: 39.2%	**FP**: 3.0%
FN: 4.0%	**TN**: 53.8%

Threshold 0.7	
TP: 38.7%	**FP**: 2.6%
FN: 4.5%	**TN**: 54.3%

Threshold 0.8	
TP: 38.0%	**FP**: 2.1%
FN: 5.1%	**TN**: 54.7%

Threshold 0.9	
TP: 36.9%	**FP**: 1.6%
FN: 6.3%	**TN**: 55.2%

5.2 Lane Change Loss Trainings

This section shows the results from the trainings done with the lane change loss, without regression confidences.

5.2.1 Lane Change Loss with Hard Minimum (LChM)

This training had the following settings:

- **Network:** Network with 2 skip-connections (see Appendix E.2)
- **Learning rate:** 0.005
- **Number of epochs**: 60
- **Pretraining**: Initialized ENet-backbone with pretrained weights, trained with absolute error loss and regression confidences.
- **Type of Lane Change Loss**: Hard minimum

Example prediction images produced by this training are shown in Figures A.5 and A.6 in Appendix A, with a specific lane change sequence shown in Figure B.2 in Appendix B. The training curves are shown in Figure 5.3.

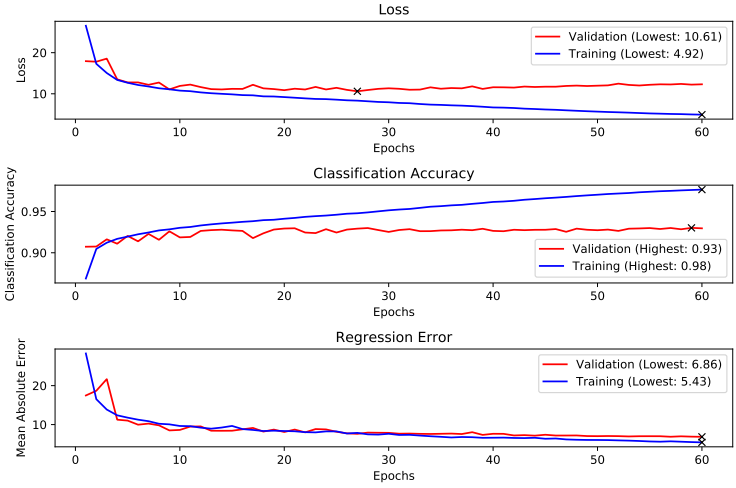


Figure 5.3: *Training curves for the training*

Below are confusion matrices for the existence variables:

Threshold 0.5		Threshold 0.6	
TP: 39.4%	**FP**: 3.3%	**TP**: 39.0%	**FP**: 2.9%
FN: 3.8%	**TN**: 53.5%	**FN**: 4.2%	**TN**: 53.9%

Threshold 0.7		Threshold 0.8	
TP: 38.5%	**FP**: 2.5%	**TP**: 37.8%	**FP**: 2.1%
FN: 4.7%	**TN**: 54.3%	**FN**: 5.4%	**TN**: 54.8%

Threshold 0.9	
TP: 36.6%	**FP**: 1.5%
FN: 6.5%	**TN**: 55.3%

5.2.2 Lane Change Loss with Smooth Minimum (LCsM)

This training had the following settings:

- **Network:** Network with 2 skip-connections (see Appendix E.2)
- **Learning rate:** 0.005
- **Number of epochs**: 60
- **Pretraining**: Initialized ENet-backbone with pretrained weights, trained with absolute error loss and regression confidences.
- **Type of Lane Change Loss**: Smooth minimum

Example prediction images produced by this training are shown in Figures A.7 and A.8 in Appendix A, with a specific lane change sequence shown in Figure B.3 in Appendix B. The training curves are shown in Figure 5.4.

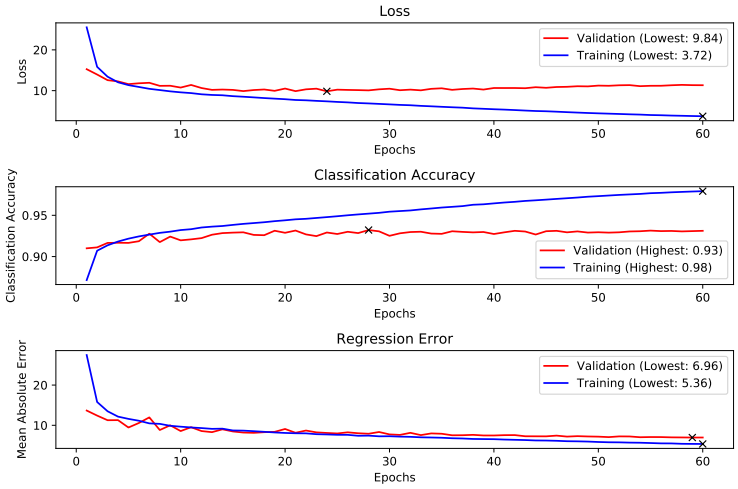


Figure 5.4: *Training curves for the training*

Below are confusion matrices for the existence variables:

Threshold 0.5		Threshold 0.6	
TP: 39.6%	**FP**: 3.3%	**TP**: 39.2%	**FP**: 2.9%
FN: 3.6%	**TN**: 53.5%	**FN**: 4.0%	**TN**: 53.9%

Threshold 0.7		Threshold 0.8	
TP: 38.7%	**FP**: 2.5%	**TP**: 38.0%	**FP**: 2.1%
FN: 4.5%	**TN**: 54.4%	**FN**: 5.2%	**TN**: 54.8%

Threshold 0.9	
TP: 36.8%	**FP**: 1.6%
FN: 6.3%	**TN**: 55.3%

5.2.3 Lane Change Loss with No Oversampling (LCnO)

This training had the following settings:

- **Network:** Network with 2 skip-connections (see Appendix E.2)
- **Learning rate:** 0.005
- **Number of epochs**: 60
- **Pretraining**: Initialized ENet-backbone with pretrained weights, trained with absolute error loss and regression confidences.
- **Type of Lane Change Loss**: Hard minimum

Example prediction images produced by this training are shown in Figures A.9 and A.10 in Appendix A. The training curves are shown in Figure 5.5.

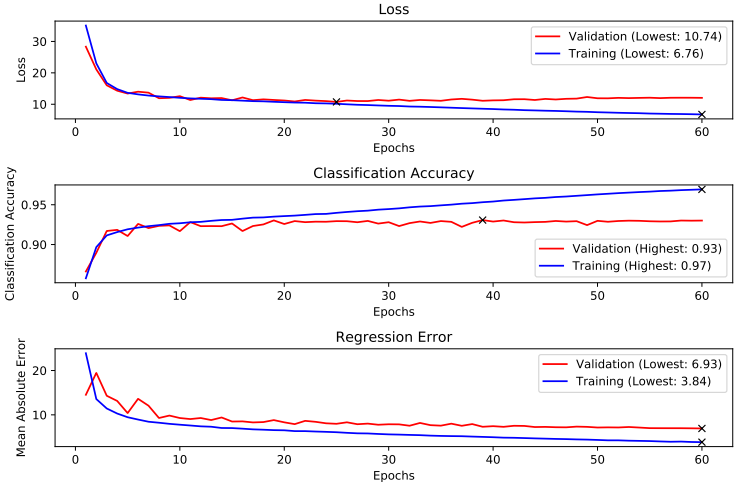


Figure 5.5: *Training curves for the training*

Below are confusion matrices for the existence variables:

Threshold 0.5		Threshold 0.6	
TP: 39.5%	FP: 3.4%	TP: 39.0%	FP: 3.0%
FN: 3.7%	TN: 53.4%	FN: 4.1%	TN: 53.9%

Threshold 0.7		Threshold 0.8	
TP: 38.5%	FP: 2.5%	TP: 37.8%	FP: 2.1%
FN: 4.6%	TN: 54.3%	FN: 5.4%	TN: 54.8%

Threshold 0.9	
TP: 36.6%	FP: 1.5%
FN: 6.6%	TN: 55.3%

5.3 Lane Change Loss Combined with Confidences

This section shows the trainings with the lane change loss combined with the regression confidences. These were all trained with the confidences together with the regression and classification.

5.3.1 Confidences on Larger Network with Pre-training (cLP)

This training had the following settings:

- **Network:** Confidence network with 2 skip-connections (see Appendix E.3)
- **Learning rate:** 0.001
- **Number of epochs**: 80
- **Pretraining**: Initialized entire network with pretrained weights, trained with absolute error loss and regression confidences.
- **Type of Lane Change Loss**: Smooth minimum
- **Regression Confidence Weight** λ: 0.1

Example prediction images produced by this training are shown in Figures A.11 and A.12 in Appendix A. The training curves are shown in Figure 5.6.

Below are confusion matrices for the existence variables:

Threshold 0.5		Threshold 0.6	
TP: 39.5%	FP: 3.8%	TP: 39.1%	FP: 3.4%
FN: 3.7%	TN: 53.1%	FN: 4.1%	TN: 53.5%

Threshold 0.7		Threshold 0.8	
TP: 38.6%	FP: 3.0%	TP: 38.1%	FP: 2.6%
FN: 4.5%	TN: 53.9%	FN: 5.1%	TN: 54.3%

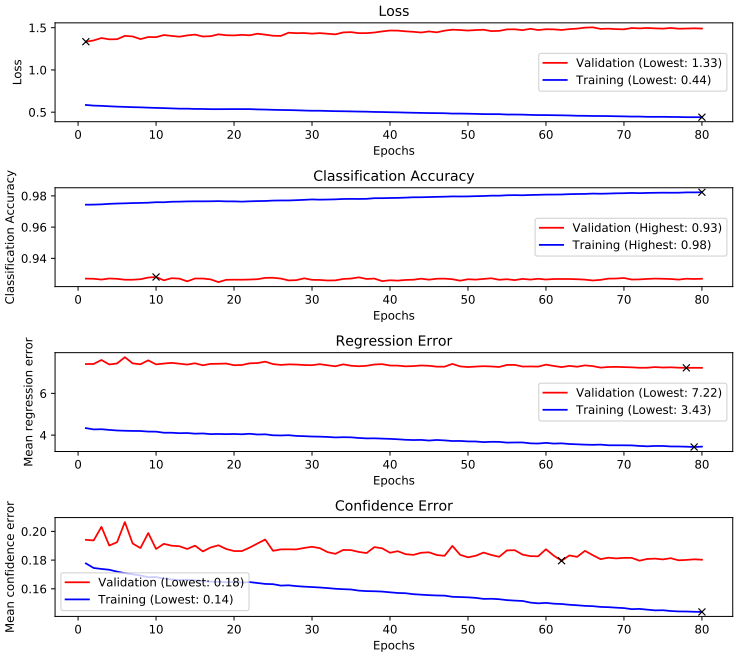


Figure 5.6: *Training curves for the training*

Threshold 0.9	
TP: 37.0%	**FP**: 2.0%
FN: 6.1%	**TN**: 54.8%

Below are confusion matrices for the regression confidence variables:

Threshold 0.5		Threshold 0.6	
TP: 87.0%	**FP**: 8.8%	**TP**: 86.6%	**FP**: 8.4%
FN: 3.0%	**TN**: 1.2%	**FN**: 3.4%	**TN**: 1.6%

Threshold 0.7		Threshold 0.8	
TP: 85.7%	**FP**: 7.8%	**TP**: 82.9%	**FP**: 6.9%
FN: 4.3%	**TN**: 2.2%	**FN**: 7.1%	**TN**: 3.1%

Threshold 0.9	
TP: 72.1%	**FP**: 6.1%
FN: 17.9%	**TN**: 3.9%

5.3.2 Confidences on Smaller Network with Pre-training (cSP)

This training had the following settings:

- **Network:** Confidence network with 1 skip-connection (see Appendix E.4)
- **Learning rate:** 0.001
- **Number of epochs**: 80
- **Pretraining**: Initialized entire network with pretrained weights, trained with absolute error loss and regression confidences.
- **Type of Lane Change Loss**: Smooth minimum
- **Regression Confidence Weight** λ: 0.1

Example prediction images produced by this training are shown in Figures A.13 and A.14 in Appendix A. The training curves are shown in Figure 5.7.

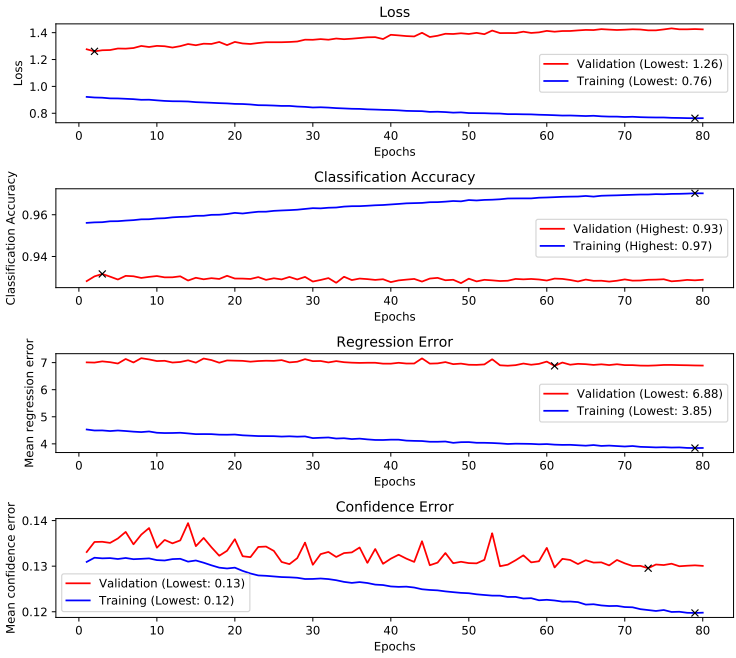


Figure 5.7: *Training curves for the training*

Below are confusion matrices for the existence variables:

Threshold 0.5		Threshold 0.6	
TP: 39.2%	**FP**: 3.2%	**TP**: 38.7%	**FP**: 2.8%
FN: 4.0%	**TN**: 53.6%	**FN**: 4.4%	**TN**: 54.0%

Threshold 0.7		Threshold 0.8	
TP: 38.2%	**FP**: 2.3%	**TP**: 37.4%	**FP**: 1.9%
FN: 5.0%	**TN**: 54.5%	**FN**: 5.8%	**TN**: 54.9%

Threshold 0.9	
TP: 36.0%	**FP**: 1.4%
FN: 7.2%	**TN**: 55.5%

Below are confusion matrices for the regression confidence variables:

Threshold 0.5		Threshold 0.6	
TP: 85.7%	**FP**: 6.7%	**TP**: 85.0%	**FP**: 5.7%
FN: 4.5%	**TN**: 3.0%	**FN**: 5.3%	**TN**: 4.0%

Threshold 0.7		Threshold 0.8	
TP: 82.9%	**FP**: 4.3%	**TP**: 75.8%	**FP**: 2.4%
FN: 7.4%	**TN**: 5.4%	**FN**: 14.5%	**TN**: 7.4%

Threshold 0.9	
TP: 48.1%	**FP**: 0.9%
FN: 42.2%	**TN**: 8.8%

5.3.3 Confidences on Larger Network without Pre-training (cLnP)

This training had the following settings:

- **Network:** Confidence network with 2 skip-connections (see Appendix E.3)
- **Learning rate:** 0.01
- **Number of epochs**: 150
- **Pretraining**: None
- **Type of Lane Change Loss**: Smooth minimum
- **Regression Confidence Weight** λ: 0.1

Example prediction images produced by this training are shown in Figures A.15 and A.16 in Appendix A. The training curves are shown in Figure 5.8.

Below are confusion matrices for the existence variables:

Threshold 0.5		Threshold 0.6	
TP: 38.9%	**FP**: 3.6%	**TP**: 38.5%	**FP**: 3.3%
FN: 4.3%	**TN**: 53.2%	**FN**: 4.7%	**TN**: 53.6%

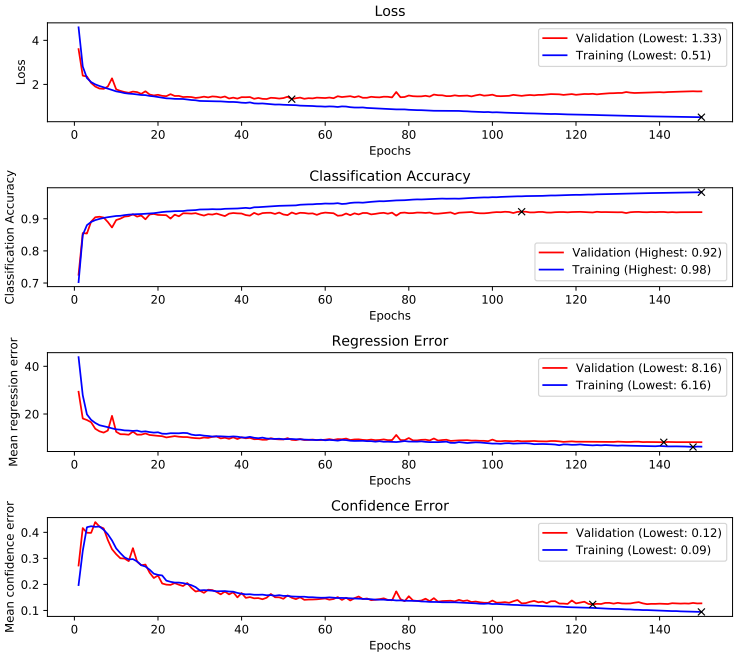


Figure 5.8: *Training curves for the training*

Threshold 0.7		Threshold 0.8	
TP: 38.1%	**FP**: 2.9%	**TP**: 37.5%	**FP**: 2.5%
FN: 5.1%	**TN**: 53.9%	**FN**: 5.6%	**TN**: 54.3%

Threshold 0.9	
TP: 36.5%	**FP**: 2.0%
FN: 6.6%	**TN**: 54.8%

Below are confusion matrices for the regression confidence variables:

Threshold 0.5		Threshold 0.6	
TP: 86.6%	**FP**: 8.6%	**TP**: 85.8%	**FP**: 7.4%
FN: 2.2%	**TN**: 2.6%	**FN**: 3.0%	**TN**: 3.7%

Threshold 0.7		Threshold 0.8	
TP: 83.4%	**FP**: 5.5%	**TP**: 75.4%	**FP**: 2.8%
FN: 5.4%	**TN**: 5.7%	**FN**: 13.5%	**TN**: 8.4%

Threshold 0.9	
TP: 44.3%	**FP**: 0.4%
FN: 44.5%	**TN**: 10.8%

5.3.4 Confidences on Smaller Network without Pre-training (cSnP)

This training had the following settings:

- **Network:** Confidence network with 1 skip-connection (see Appendix E.4)
- **Learning rate:** 0.01
- **Number of epochs**: 150
- **Pretraining**: None
- **Type of Lane Change Loss**: Smooth minimum
- **Regression Confidence Weight** λ: 0.1

Example prediction images produced by this training are shown in Figures A.17 and A.18 in Appendix A. The training curves are shown in Figure 5.9.

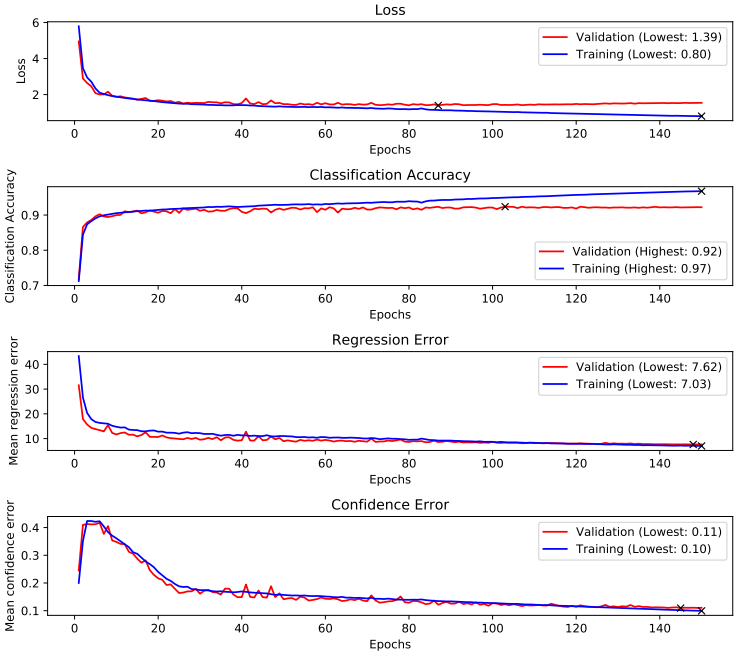


Figure 5.9: *Training curves for the training*

Below are confusion matrices for the existence variables:

Threshold 0.5		Threshold 0.6	
TP: 38.9%	**FP**: 3.5%	**TP**: 38.5%	**FP**: 3.0%
FN: 4.2%	**TN**: 53.3%	**FN**: 4.7%	**TN**: 53.8%

Threshold 0.7		Threshold 0.8	
TP: 37.9%	**FP**: 2.6%	**TP**: 37.0%	**FP**: 2.1%
FN: 5.3%	**TN**: 54.3%	**FN**: 6.1%	**TN**: 54.7%

Threshold 0.9	
TP: 35.6%	**FP**: 1.5%
FN: 7.6%	**TN**: 55.3%

Below are confusion matrices for the regression confidence variables:

Threshold 0.5		Threshold 0.6	
TP: 88.1%	**FP**: 8.2%	**TP**: 87.1%	**FP**: 6.9%
FN: 0.9%	**TN**: 2.8%	**FN**: 1.9%	**TN**: 4.1%

Threshold 0.7		Threshold 0.8	
TP: 84.6%	**FP**: 5.1%	**TP**: 76.0%	**FP**: 2.4%
FN: 4.5%	**TN**: 5.9%	**FN**: 13.0%	**TN**: 8.6%

Threshold 0.9	
TP: 42.9%	**FP**: 0.3%
FN: 46.1%	**TN**: 10.7%

5.4 Multi-stream Trainings

This section show the trainings done with the multi-stream networks. These networks have all pre-trained weights in the regression and classification branch of the networks which were frozen during training, meaning only the regression confidence branch was trained.

5.4.1 Confidences on Larger Multi-stream Network (cLM)

This training had the following settings:

- **Network:** Multi-stream confidence network with 2 skip-connections (see Appendix E.5)
- **Learning rate:** 0.005
- **Number of epochs**: 80

- **Pretraining**: Initialized and froze entire network except regression confidence branch with pretrained weights, which were trained with horizon loss.

- **Regression Confidence Weight** λ: 0

Example prediction images produced by this training are shown in Figures A.19 and A.20 in Appendix A. The training curves are shown in Figure 5.10.

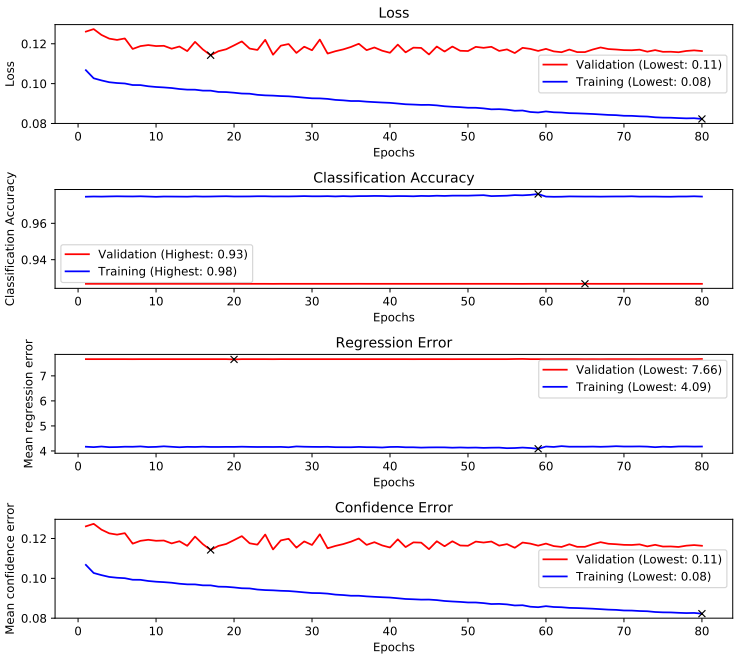


Figure 5.10: *Training curves for the training*

Below are confusion matrices for the existence variables:

Threshold 0.5	
TP: 39.0%	**FP**: 3.4%
FN: 4.1%	**TN**: 53.4%

Threshold 0.6	
TP: 38.6%	**FP**: 3.0%
FN: 4.6%	**TN**: 53.9%

Threshold 0.7	
TP: 38.1%	**FP**: 2.6%
FN: 5.1%	**TN**: 54.3%

Threshold 0.8	
TP: 37.4%	**FP**: 2.2%
FN: 5.8%	**TN**: 54.7%

Threshold 0.9	
TP: 36.1%	**FP**: 1.6%
FN: 7.0%	**TN**: 55.2%

Below are confusion matrices for the regression confidence variables:

Threshold 0.5	
TP: 87.9%	FP: 8.7%
FN: 0.9%	TN: 2.5%

Threshold 0.6	
TP: 86.8%	FP: 7.3%
FN: 2.0%	TN: 3.9%

Threshold 0.7	
TP: 83.7%	FP: 5.1%
FN: 5.1%	TN: 6.1%

Threshold 0.8	
TP: 72.7%	FP: 2.0%
FN: 16.1%	TN: 9.2%

Threshold 0.9	
TP: 32.4%	FP: 0.1%
FN: 56.3%	TN: 11.1%

5.4.2 Confidences on Smaller Multi-stream Network (cSM)

This training had the following settings:

- **Network:** Multi-stream confidence network with 1 skip-connection (see Appendix E.6)
- **Learning rate:** 0.005
- **Number of epochs**: 80
- **Pretraining**: Initialized and froze entire network except regression confidence branch with pretrained weights, which were trained with absolute error loss.
- **Regression Confidence Weight** λ: 0

Example prediction images produced by this training are shown in Figures A.21 and A.22 in Appendix A. The training curves are shown in Figure 5.11.

Below are confusion matrices for the existence variables:

Threshold 0.5	
TP: 39.6%	FP: 3.3%
FN: 3.6%	TN: 53.5%

Threshold 0.6	
TP: 39.0%	FP: 2.8%
FN: 4.1%	TN: 54.1%

Threshold 0.7	
TP: 38.4%	FP: 2.3%
FN: 4.8%	TN: 54.6%

Threshold 0.8	
TP: 37.4%	FP: 1.7%
FN: 5.7%	TN: 55.1%

Threshold 0.9	
TP: 35.8%	FP: 1.1%
FN: 7.4%	TN: 55.7%

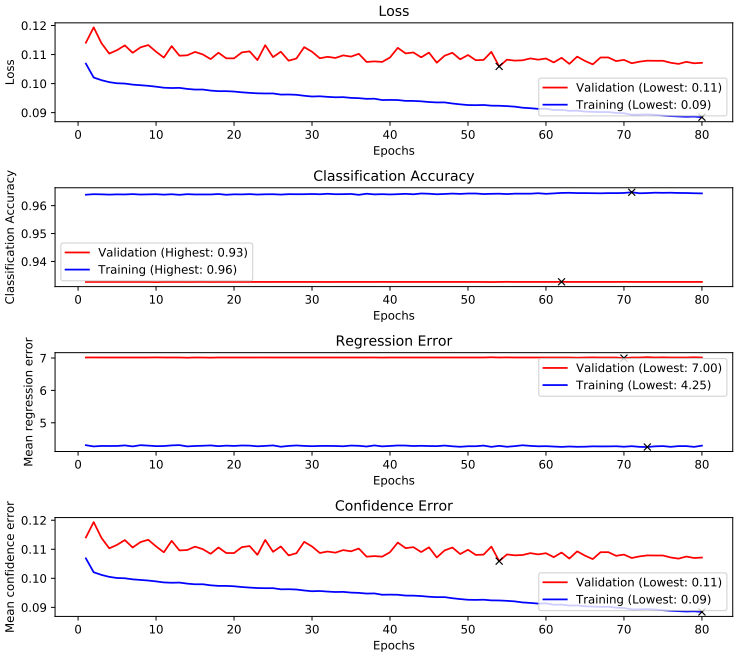


Figure 5.11: *Training curves for the training*

Below are confusion matrices for the regression confidence variables:

Threshold 0.5	
TP: 88.2%	**FP**: 6.2%
FN: 1.8%	**TN**: 3.7%

Threshold 0.6	
TP: 86.8%	**FP**: 5.0%
FN: 3.2%	**TN**: 5.0%

Threshold 0.7	
TP: 83.5%	**FP**: 3.4%
FN: 6.5%	**TN**: 6.6%

Threshold 0.8	
TP: 73.5%	**FP**: 1.4%
FN: 16.5%	**TN**: 8.5%

Threshold 0.9	
TP: 37.3%	**FP**: 0.1%
FN: 52.8%	**TN**: 9.9%

5.5 Semantic Segmentation

This section shows the results from the most successful semantic segmentation trainings.

5.5.1 Segmentation on Full Dataset (SF)

This training had the following settings:

- **Network:** Full ENet
- **Learning rate:** 0.001
- **Number of epochs**: 60
- **Pretraining**: None
- **Class weights** $w^{(c)}$: 8 for all lane classes, 1 for "other".
- **Positive to negative ratio weights** $w_{pn}^{(c)}$: 4 for all lane classes, 1 for "other".

Example prediction images produced by this training are shown in Figures A.23 and A.24 in Appendix A. The training curves are shown in Figure 5.12.

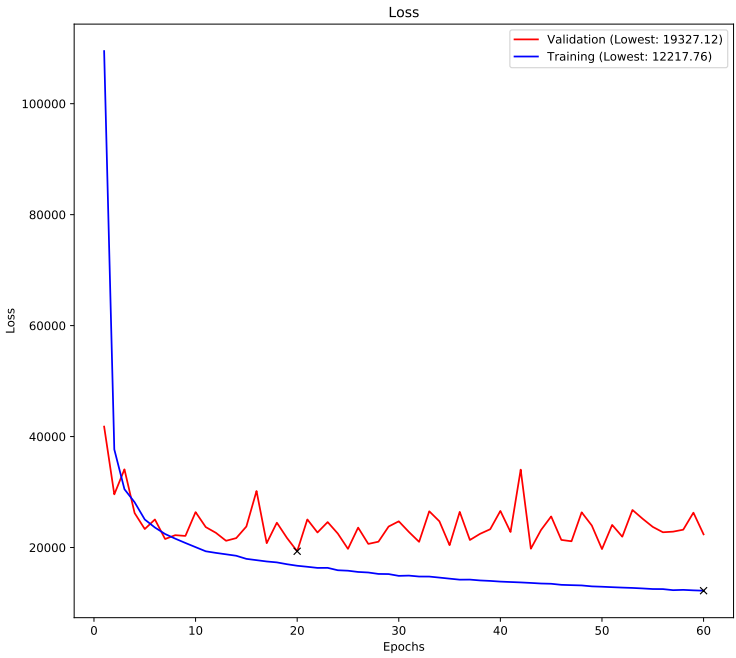


Figure 5.12: *Training curves for the training*

Below is the confusion matrix for the existence variables:

Threshold 0.5	
TP: 31.7%	**FP**: 0.9%
FN: 11.4%	**TN**: 55.9%

5.5.2 Segmentation on Quarter Dataset (SQ)

This training had the following settings:

- **Network:** Full ENet
- **Learning rate:** 0.001
- **Number of epochs**: 60
- **Pretraining**: None
- **Class weights** $w^{(c)}$: 8 for all lane classes, 1 for "other".
- **Positive to negative ratio weights** $w_{pn}^{(c)}$: 4 for all lane classes, 1 for "other".

Example prediction images produced by this training are shown in Figures A.25 and A.26 in Appendix A. The training curves are shown in Figure 5.13.

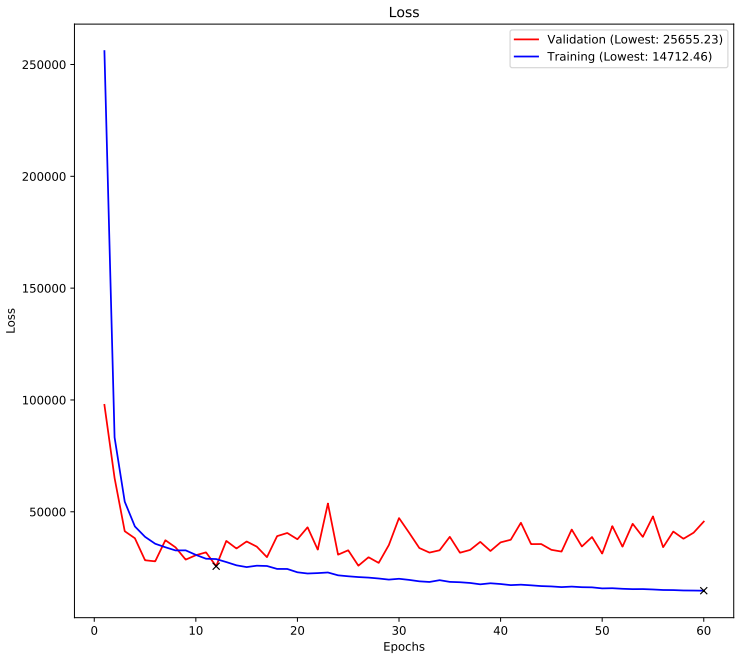


Figure 5.13: *Training curves for the training*

Below is the confusion matrix for the existence variables:

Threshold 0.5	
TP: 27.9%	**FP**: 2.7%
FN: 15.2%	**TN**: 54.2%

5.6 Result Comparison

This section provides Table 5.1, where the metrics from all presented results are presented for comparison.

Network	MAE	CA	LC-MAE	Curve MAE	Curve CA	cMAE
RAE	7.28	**93.19**	27.47	20.55	**93.12**	-
HL	7.30	93.03	25.46	21.64	92.32	-
LChM	7.34	92.91	24.18	20.99	91.67	-
LCsM	**7.27**	93.11	**21.90**	23.05	91.41	-
LCnO	7.37	92.87	31.65	22.06	91.27	-
cLP	7.67	92.53	28.33	23.46	92.87	0.18
cSP	7.29	92.76	25.42	21.59	91.27	0.13
cLnP	8.47	92.08	33.34	20.91	91.38	0.12
cSnP	8.02	92.23	24.39	21.76	91.16	0.11
cLM	7.82	92.46	27.39	21.17	91.16	0.11
cSM	7.44	93.10	25.33	**19.45**	92.57	**0.10**
SM	**3.89**	**87.61**	30.52	**4.7**	**85.19**	-
SQ	5.84	82.08	**19.90**	13.67	78.05	-

Table 5.1: *Metric comparison between all presented results. Best performance in each category is bolded, with one selected from the end-to-end approach and one for the semantic segmentation approach. Results from the two approaches are separated with a line. MAEs are measured in pixels and CA in %. cMAE is dimensionless.*

6
Discussion

In this chapter, the findings presented in Chapter 5 will be discussed and interpreted, in preparation for conclusions to be drawn in Chapter 7. It also contains comments on the used method, and the sources upon which this work is built.

6.1 Evaluation Metrics

We judge the results from the evaluation metrics described in Section 4.5 to be reliable, as they are run on a test set of 9 817 images. However, the *Curve MAE/CA* and *Lane Change MAE* metrics are deemed to be the least reliable. This is because only 59 curve samples and 129 lane change samples were found using the detectors in the test set. Either the true number of these samples in the dataset is very low, or the detectors produce many false negatives. Therefore, a difference in one or two pixels or percentage points in these metrics should not be seen as significant.

6.2 Validation Loss Minimum

In this section, a general behavior of the training graphs will be discussed, namely that the validation loss has a minimum at some point, after which it increases again, while both the regression error decreases and the classification accuracy increases after this point. Note for example the *Horizon Loss*, where Figure 5.2 shows how the optimum of the validation loss occurs before the optimum for both the classification accuracy and regression loss. It is also noteworthy how the regression error never obtains its optimum before the classification accuracy, for any training which optimizes said parameters. These observations might indicate that the networks begin to overfit with respect to the classification accuracy

before they have reached their optimum with respect to the regression error.

6.3 Horizon Loss

In this section, the effects of the *Horizon Loss*, described in Equation 4.12 will be discussed. Comparing it to the *Regular Absolute Loss*, one may note that the *Horizon Loss* produces a marginally higher mean absolute error and a marginally lower classification accuracy. The confusion matrices of resulting from both implementations are similar almost within rounding error. Comparing their prediction images, it is slightly noticeable that the *Horizon Loss* positions the lanes closer to the markings near the horizon than the *Regular Absolute Loss*. This is as expected, since the idea with the *Horizon Loss* was to increase performance for the predictions further away. These results are therefore somewhat mixed, since the metrics for the *Regular Absolute Loss* slightly suggests it being better. It is therefore believed that the concept of the *Horizon Loss* is solid, but that further work is needed to implement it satisfactory.

6.4 Lane Change Algorithm

In this section the effects of the algorithms presented in Section 4.4 will be discussed. It should directly be noted that the prediction images for the lane change situation from both the *Regular Absolute Loss* and the *Horizon Loss* from Section 5.1, Figures A.2 and A.4 as well as from *Lane Change Loss with Hard Minimum* and *Lane Change Loss with Soft Minimum* from Section 5.2, Figures A.6 and A.8, all predict similarly on the lane change situation. Instead, note the Figures in Appendix B. Figure B.1 shows how the prediction from *Regular Absolute Loss* varies much between the samples, and sometimes predicts lane markings in the middle of the lane. Compare that to Figures B.2 and B.3, where one may note that both variants of the lane change algorithm predicts stable lane markings during the sequence, performing notably better than the *Regular Absolute Loss*. Combining this with the lower lane change mean absolute error for both variants of the Lane Change Algorithm, it is believed that the use of the Lane Change Algorithm increases performance for the system in lane change situations.

Comparing the classification accuracy, mean absolute error and the confusion matrices for the *Regular Absolute Loss* and *Horizon Loss* with the *Lane Change Loss with Hard Minimum* and *Lane Change Loss with Soft Minimum*, it is noticed that they are all very similar. This furthers the belief that the use of the Lane Change Algorithm is beneficial, since it adds performance during lane change situations without hampering it otherwise.

6.4.1 Minimum Function Comparison

Comparison between the two different minimum functions available for the Lane Change Algorithm can be done by investigating the results from Section 5.2, noting the differences between *Lane Change Loss with Hard Minimum* and *Lane*

Change Loss with Soft Minimum. It can be seen that their performance is similar with regards to all presented metrics. Investigating their prediction images in Figures A.6 and A.8, the only clear difference is that the *Lane Change Loss with Soft Minimum* has a somewhat better prediction at the end of the curve sample. Focusing on the comparison for the lane change situation presented in Appendix B, their performance is once again similar. A slight edge has to be given to the *Lane Change Loss with Soft Minimum*, since it appears to predict both neighbor-left and neighbor-right slightly better than the *Lane Change Loss with Hard Minimum*. The *Lane Change Loss with Soft Minimum* somewhat also outperforms *Lane Change Loss with Hard Minimum* when it comes to Mean Absolute Error, both overall and during lane changes. The *Lane Change Loss with Soft Minimum* does however have a higher mean absolute error for curve situations. It is therefore deemed possible to conclude that the soft minimum is preferable to the normal minimum, given these results and the more continuous gradient.

6.5 Data-preprocessing

In this section, the results regarding the data pre-processing are discussed.

6.5.1 Ground-truth Generation

Judging by the results of the network trainings and the fact that the vast majority of the ground-truth data was not marked as corrupt by the data enhancement methods, it is safe to assume that it is completely feasible to generate ground-truth data from ground truth meant for semantic segmentation. The relatively simple method of sampling rows in the image and interpolating the edges of the lane markings will for most situations produce accurate ground truth.

One of the most obvious flaws in this method is that it only works where there are visible lane markings. For roads without road markings, or where it is covered by snow or heavy traffic, there are no segmentation markings from which to generate supporting points. This will lead the network to not output any lanes when there are no lane markings, which may or may not be desirable.

Another flaw is the problem of lanes starting at the first visible marking, which is mostly remedied with the lane extrapolation procedure. This proved to efficiently make the networks understand that the lane does not start at the first visible dashed markings. One flaw with the method that was implemented, however, relates to the handling of road exits. As described in Section 4.3.2, in order to not extract road exits, a harsher threshold was put on extracting neighbor-right. There are two problems with this, one being that this assumes there is only right hand driving in the dataset, which did not turn out to be true, and it led the network to be reluctant to expanding neighbor-right to the bottom of the image. This is can be seen in some of the prediction images, especially the rain sample in for example Figures A.1, A.5, A.9 and A.11, and the lane change sequences in Figures B.1 and B.1.

The road exit extrapolation problem suggests that there is room for improvement for this method of extrapolation. One possible solution would be to check

if neighbor-left or neighbor-right have similar curvature to ego-left and ego-right, in which case it is not a road exit and thus should be safe to extrapolate. However, this problem combined with the problem of ground-truth markings only existing when road labels exist suggests that it would be better to label the ground-truth data as supporting points, or some similar representation, from the start. This would eliminate the need for ground-truth pre-processing, and would probably result in less erroneous ground-truth data, as many bad ground-truth samples are caused by artifacts in the generation or extrapolation. It is also estimated to be faster label ground-truth data this way, as only lines would need to be drawn along the lanes rather than individual polygons on each lane marking.

6.5.2 Data Enhancement

Removing bad ground truth by creating detectors and manually viewing suspicious images, as described in Section 4.3.3, turned out to be highly time efficient. Only a few hundred images needed to be manually reviewed out of tens of thousands of images, which means it could be done within an hour. The deep-learning based detector was especially effective at finding samples with ground-truth problems that were until then unknown to exist. Examples of this are the samples with missing markings and those with incorrect lane ID:s. The second-derivative based detector was, however, also important, as some ground-truth errors might not have caused a large prediction error but were detected for being abnormally curvy.

Since the entire dataset was not manually examined, it is unclear how many incorrect samples were not detected by either detector. There is the possibility of an incorrect ground-truth situation existing which has not been thought of, without being excessively curvy or producing a high prediction error. To fully assess the accuracy of this method, it should be tested in a controlled environment where it is known how many samples should be removed.

6.5.3 Data Oversampling

It is unclear whether the data oversampling, described in Section 4.3.4, improved the performance in lane changes and curves. Looking at the two otherwise equivalent trainings *Lane Change Loss with No Oversampling* and *Lane Change Loss with Hard Minimum*, it seems that the curve performance is not very different at all. It is about one pixel worse when trained on the non-oversampled dataset, but this is within the margin of error considering the low number of curve samples it was tested on. The lane change performance, however, is substantially worse when no oversampling was done, differing with more than 7 pixels. It is, however, an open question whether this is due to the low number of evaluated lane change samples, as oversampling seemed to make little difference for the curve performance.

The reason for why oversampling did not show a clear improvement might have been that it was not done to a high enough degree. As is can be seen from the training curves, it does not seem that oversampling makes the models overfit, which suggests that it can be done more. However, as only 321 curve samples

and 510 lane change samples out of 40 191 were detected, it is unlikely that oversampling these to the point where they match the number of other situations would not cause overfitting, even with augmentation.

6.6 Confidence Prediction

This section will discuss the results containing the confidence predictions, and how they affect the overall results. A primary observation based on the outputted prediction images is that the addition of confidence prediction allows the system to suppress bad regression output. An example of this is given in Figure A.16, where the regression output in the curve image is suppressed (turned red) when the regression is notably bad close to the horizon. This is the wanted behavior, and points to the merit of the confidence prediction concept.

It should be noted that the pre-trained networks with confidence prediction produces similar metrics to the regular losses with the same lane change algorithm applied.

6.6.1 Confidence Constantly Zero

As shown in Figures A.11 and A.12, the *Confidences on Larger Network with Pre-training* implementation shows a behavior similar to the one described in Section 4.2.5 where the confidence for some points are constantly zero. This may be a sign that the network needs more training in order to fully converge, since Figure 5.6 shows that the mean confidence error might not have converged yet, albeit it is still very flat. Noting that *Confidences on Smaller Network with Pre-training* and *Confidences on Larger Network without Pre-training* shows the same behavior, as visible in Figures A.13, A.14 and A.15, A.16 respectively, it is possible to argue that the large confidence networks are problematic both with and without pre-training, and that pre-training did not give sought results for the small network either. This presents a trade off, since the metrics and the prediction images shows that the networks perform well, with the zeroed points being the only notable exception.

It should be noted that the *Confidences on Smaller Network without Pre-training* implementation did not show this behavior, as visible in Figures A.17 and A.18. However, the total regression error is still higher than for the trainings without confidences, which might indicate that the network has more potential. Something which contradicts this is that the validation loss seems to plateau, or even increase, as seen in Figure 5.9. This is strengthened by the fact that the lowest validation loss occurs before the last epoch, something discussed in Section 6.2.

6.6.2 Pre-training Confidence Outputting Networks

This section will discuss the effect of pre-training the confidence outputting networks. This process is described in Section 4.2.3 and initialized weights for the confidence outputting network with weights from previously high-performing networks of identical design but without confidence output.

Larger Network

Beginning the comparison with the large network, the metrics shows an improvement in mean absolute error for all situations except curves. Comparing the confusion matrices for regression confidence shows that the pre-trained network has notably lower FP and TN for the higher threshold. The network initialized with random weights do have a lower confidence mean absolute error. Comparing their prediction images, it is noticeable that the pre-trained network (Figures A.11 and A.12) has a somewhat preferable output to the network initialized with random weights (Figures A.15 and A.16). This is visible in the curve sample, where the pre-trained network keeps its prediction closer to the markings, and especially in the lane change sample where the pre-trained gives correct predictions, while the randomly initiated network both positions the lanes worse and, most notably, gives low regression confidence and therefore does not correctly output the lanes.

Smaller Network

The smaller networks show, just as for the large network, that pre-training lowers the mean absolute error, this time for all situations except lane change. The classification accuracy is almost identical, while the network initialized with random weights do present a lower confidence mean absolute error. Comparing their confusion matrices no substantial differences are detected regarding classification. The confusion matrices for regression confidence does show the same difference as for the larger networks, with the pre-trained network obtaining lower FPs and FNs. Comparing the prediction images for the two cases, one may note that the network with randomly initialized weights (Figures A.17 and A.18) do seem to perform a bit worse regression-wise than the pre-trained one (Figures A.13 and A.14), albeit with a smaller difference than the larger networks. Note however how the randomly initiated network has worse regression in its curve sample than the pre-trained, and how the randomly initiated one outputs very low regression confidences for the same situation. The low confidences might be wanted, since it may indicate that the network have correctly identified lane change situations as difficult, but the current output is worse than it should be, since the regression is quite close to the sought one. A major difference between the smaller networks is that the pre-trained one shows the behavior discussed in Section 6.6.1, while the randomly initiated one does not. The cause for this has not been established during this study, but it is believed that further training of both networks would not yield increased performance since both show increasing validation loss, indicating overfitting.

Learning Rate and Training Time

The networks treated above in this section required a long time to train - 150 epochs took almost a week of GPU time, compared to the normal 60 epochs taking approximately two days. This affected the study in the way that fewer parameters were tested for these networks, with one of them being the learning

rate. There is a possibility that a different learning rate, possibly with a different annealing scheme, could improve the results for the networks with randomly initiated weights. This is argued since the confidence error affects the regression gradient.

6.6.3 Multi-stream Networks

This section will discuss the effect of implementing the confidence predictions using a multi-stream network, as described in Section 4.2.5 and compare the larger and smaller variants of it. It should be noted that the regression-wise performance of these networks are not perfectly comparable with the performance from other network, since they were initialized with slightly different pre-trained weights as a result of their depths. Therefore, comments based on regression performance for these networks are not as reliable as for others.

Begin by noting that the smaller multi-stream network achieves a lower mean absolute error than the larger network, as well as a higher classification accuracy. The smaller network also outperforms the larger one for both lane change mean absolute error and curve mean absolute error, as well has as a marginally lower confidence mean absolute error. Comparing their prediction images, one may note by comparing Figure A.19 with Figure A.21 that the larger network has a somewhat better regression on the lane change sample than the smaller one. Comparing Figure A.20 with Figure A.22 instead shows that the smaller network appears to give better output on both the curve sample and the weak markings sample. This is noticeable sin the larger network has low regression confidence for ego-right for the curve sample, while the smaller one does not, while keeping their regression almost equally good. For the weak markings sample, it is highly noticeable that the smaller network outputs predictions for ego-right, while the larger network does not. Noting that the larger network still has high regression confidence for ego-right, it can be concluded that it fails to correctly classify the lane and thus does not output it. In favor of the larger network is the rain sample, where the larger network correctly finds and predicts neighbor-right, while the smaller network outputs a low confidence value there and thus does not detect said lane. The confidence value is however not 0.0 as it often is when the confidence output behaves like described in Section 6.6.1, but 0.2, 0.3 instead. This might indicate that the network is on its way to correctly learn the true confidence predictions, something that is further indicated by the fact that the validation loss does not seem to plateau as much for the smaller network as for the larger one, as seen when comparing Figure 5.10 with Figure 5.11.

Multi-stream and Single-stream Comparison

It should also be noted that the multi-stream networks were not trained with either of the lane-change losses, as a result of time constraints. Had this been done, better performance for the lane-change situations were to be expected.

Based on the given results, the multi-stream networks shows potential compared to the single-stream ones, especially considering that the multi-stream net-

works does not show the behavior of zeroed points, described in Section 6.6.1. Another conclusion is that the larger networks, both single-stream and multi-stream, were too large, since the smaller networks often outperformed the larger ones. Noteworthy is that the smaller single-stream network performed better than the larger multi-stream network, further indicating that the smaller networks shows the most promise. Especially noteworthy is that the small single-stream network initialized with random weights showed none of the zeroed-point behavior described in Section 6.6.1.

6.7 Semantic Segmentation

Before discussion is commenced, it should be noted that the semantic segmentation system does not output continuous classification, but rather by design only outputs a binary class value. This follows from the underlying system construction where each pixel is assigned exactly one class. Therefore, the confusion matrices for the semantic segmentation trainings will be the same for all threshold $T \in (0, 1)$, since the existence only takes a value of 0 or 1, which is why only the confusion matrix with threshold 0.5 is shown.

As is clear from the metrics on the semantic segmentation trainings, they produce a very low regression error, especially with the curve samples. Even with reduced dataset size, the average regression error is quite low. The regression error could in principle be even lower, since a simple post-processing algorithm was used. The biggest problem with the post-processing algorithm is that it assumes there is very little noise in the output, since this is true for the ground-truth, but not for actual segmentation output. As can be seen in the lane change image in Figure A.23, and the curve sample in Figure A.26, there are small areas where an incorrect lane was detected in multiple places, which causes the lines to be drawn incorrectly. With better post-processing, like thresholding probabilities rather than always choosing the maximum-probability class, applying morphological operations or clustering, one should expect better performance. Unfortunately, there was not enough time in this thesis to explore such methods, as most of the focus lay on achieving high performance with the supporting points.

6.7.1 Comparing Semantic Segmentation with Supporting Points

Looking at the prediction images and metrics from the supporting points and the segmentation trainings, it is clear that the semantic segmentation has some strong advantages over the supporting points. One being that it very rarely predicts that pixels which are not lane markings to be part of a lane. This means that it performs very well when markings are clearly visible, and will not produce false positives when lane markings are not visible. This explains why the total regression error is much lower for the segmentation images, even for curve samples. This is especially clear in the curve sample in Figure 6.1, the segmentation prediction is far better than any of the supporting points' predictions. Another advantage

is that this representation inherently outputs confidences, as it outputs a probability distribution for each pixel. An equal probability on all classes means the network is uncertain, and a probability of 1 on one class means the network is highly certain. This means special solutions like the regression confidences for the supporting points are not needed.

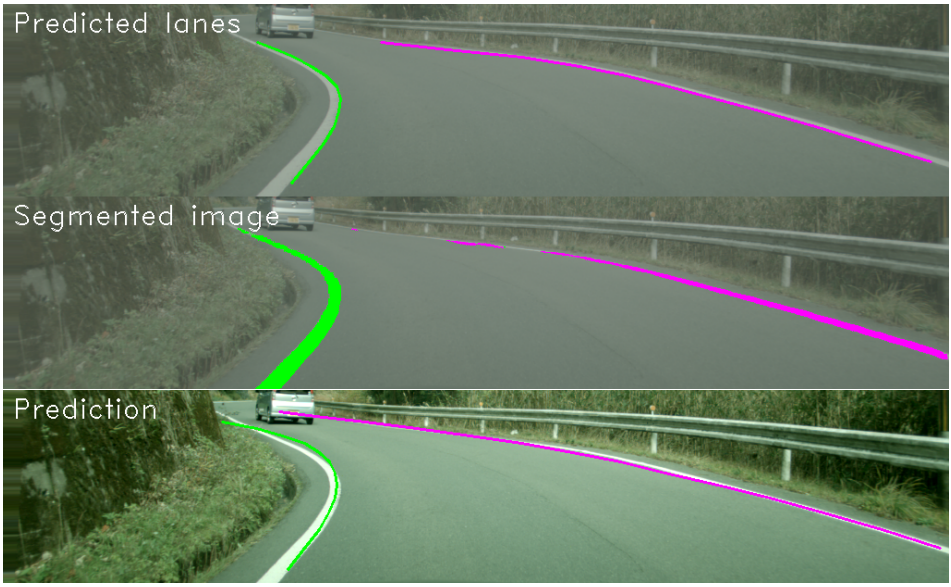


Figure 6.1: *Post processed semantic segmentation on the full dataset (above), raw semantic segmentation output (middle) and the supporting points trained with soft minimum lane change loss (below), evaluated on a curve sample.*

The supporting points have, however, clear advantages over the semantic segmentation. Apart from not needing to process the output using a complicated post-processing algorithm, it performs well when the lane markings are weak or occluded by other vehicles. Looking at the segmentation outputs from weak markings sample, shown in Figure 6.2, the network does not detect the existence of the lanes at all, in contrast to virtually all the supporting points trainings. In the rain sample, shown in Figure 6.3, it is also clear the supporting points do a better job at predicting neighbor-right even though it is occluded by a bus. This indicates there is great potential in the supporting points or another end-to-end representation to predict the existence of lanes even without actual lane markings, provided the lanes are marked in the ground truth in those cases.

Judging by the classification accuracy and the confusion matrices, the supporting points have an upper edge here as well. The overall classification accuracy and TP rate is higher for the supporting points and the FN rate is lower. This is probably in part due to supporting points finding lanes better on weak markings, which was discussed in the previous paragraph, but also that the support-

Figure 6.2: *Post processed semantic segmentation on the full dataset (above), raw semantic segmentation output (middle) and the supporting points trained with soft minimum lane change loss (below), evaluated on a sample with weak lane markings.*

ing points look further ahead in the image. Interestingly, the FP rate is slightly higher for the supporting points, which is to be expected when the segmentation network is specifically trained to find lane markings, while the supporting points sometimes predicts lanes when there are not really any lane markings on the road, but there are contours which marks the edge of the road.

What is also clear with the semantic segmentation is that it performs much worse when reducing the size of the dataset. The regression error is still fairly good, but the problem of FN is much worse. The network does not predict lanes as far as it does when there is more data, which can be seen in Figures A.25 and A.26. Considering this, and the fact that labeling this amount of data specifically as lines in the image takes much less time than drawing segmentation polygons, it is reasonable to assume that you get more performance per hour spent labeling data with the supporting points than with the segmentation approach.

6.8 Comments on Method

This section comments the method used in the thesis, pointing out potential flaws.

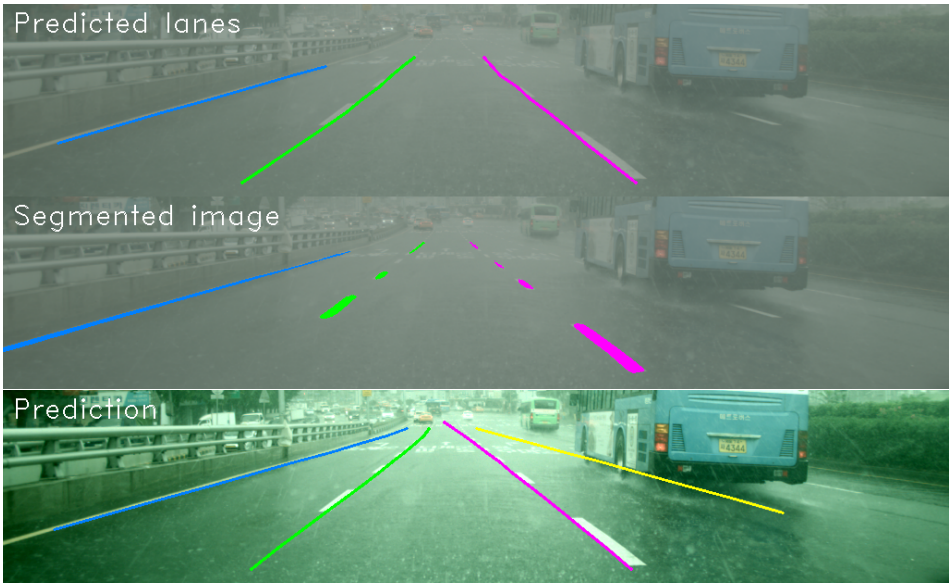


Figure 6.3: *Post processed semantic segmentation on the full dataset (above), raw semantic segmentation output (middle) and the supporting points trained with soft minimum lane change loss (below), evaluated on a rainy sample.*

6.8.1 Source Discussion

It is noted that while computer vision and machine learning is a fairly novel scientific field, it is based on solid mathematics and statistics. Furthermore, since the sources cited in this thesis are mainly academic research papers, their integrity and accuracy are deemed to be high. The only times non-academic papers where used they were used to show miscellaneous statistics or other facts which did not significantly affect the work, such as [28]. It should also be noted that many of the sources used also build upon one another, and that no contradictions have been found.

6.8.2 Criticism of the Method

The method of this study was mostly conducted via data-driven research, building upon intermediate results between iterations. We have thus continuously learned what needs to be tested, but it has also led to less planned trainings and tests. This could mean that combinations of different methods might work, while their individual parts presents no improvement. This is an effect of the *curse of dimensionality*. Since training neural networks is a slow process, requiring days of GPU time, it was deemed impractical to test every combination of parameters, networks and other implemented ideas. This is, for example, one of the reasons for the *Horizon Loss* not being tested together with other solutions, as well as for

the lane change loss being initialized with pre-trained weights.

An alternative process would have been to more thoroughly plan the work and theorize about every possible situation and how to implement solutions to them. This is not deemed efficient, since a lot of problems which need to be tackled are hard to think of without seeing the results of tests.

With infinite resources, the thesis could have tested every combination and parameter searched for optimum with high accuracy, though this is not to be seen as anything but an unreachable utopia.

7

Conclusions

This chapter will conclude the thesis, and present some of the authors' thoughts about future work on the subject. We begin by reasoning that, from this thesis work, based on the results in Chapters 5 and the findings in Chapter 6, we can conclude that an End-to-end Road Lane Detection and Estimation using Deep Learning is feasible. Our presented system shows several advantages over a semantic segmentation based system on many fronts, and while its performance is far from ready for deployment in real vehicles, it definitely shows a large potential.

7.1 Concluding Remarks

This section will attempt to answer the research questions stated in Section 1.4.

1. *How well does the end-to-end system predict the existence and positions of lanes in the image, compared to a semantic segmentation based system?*
   The semantic segmentation representation predicts the position of lanes better when clear lane markings are visible, since it predicts exactly where the lane markings are. The semantic segmentation for this reason performs better in curves. The end-to-end system does, however, better predict the positions and especially the existence of lanes under less ideal conditions, such as in the case of weak markings and occlusion by vehicles. It therefore also is better at predicting the existence of lanes than the segmentation representation. We therefore recommend investing further development of the end-to-end system rather than a semantic segmentation based one, as it shows greater potential to handle more general driving situations. It also seems feasible to reproduce the low regression error from the semantic segmentation solution on the end-to-end system during ideal driving conditions, given more work.

2. *Which advantages does the end-to-end representation have over the semantic segmentation representation?*
   The biggest advantages of the end-to-end representation has to do with practicality. The end-to-end approach does not need a very complicated post-processing algorithm for determining the positions of the lanes, saving time in development and maintenance of such an algorithm. It also uses simpler lane markings, which means a larger and more diverse ground-truth database can be produced in a shorter amount of time, compared to a semantic segmentation database. The end-to-end system is also more robust against situations where lane markings are not clearly visible, and can often see further ahead in the image. You can expect higher performance per hour spent marking and designing post-processing algorithms with the end-to-end system.

3. *Is it feasible to let the end-to-end system estimate a confidence measure in the predicted lane positions?*
   Both the single-stream and the multi-stream approach shows that letting the system output a confidence value of its own prediction is not only doable, but may increase performance. Especially the multi-stream approach shows that the system might avoid predicting lane positions in the wrong place with the implementation of a confidence prediction.

4. *How should the semantic segmentation ground-truth data be pre-processed for the end-to-end representation?*
   If semantic segmentation ground truth is all ground truth which is available, the described procedure in Section 4.3 of sampling the rows, extrapolating the lanes and removing bad samples caused by conversion artifacts clearly works to produce data for training the networks with an end-to-end approach. We, however, believe it is more ideal to label the data directly for the purpose instead, as it would probably be a faster and more reliable way of creating ground-truth data, with the added bonus of marking lanes even when there are no visible lane markings.

7.2 Future Work

This section aims to leave finishing remarks on how future work in this field can be done. The authors firstly wish to point towards the classical thought that more data should always be beneficial for machine learning tasks. In this case, special weight lies on correctly labeled data for all situations, in a balanced way. As noted in Section 4.3.5, only a handful of all samples contained the challenging lane-change situation. More training data on this and other situations where the current system performs sub-par is recommended.

Noted in Section 6.2 is how the optimum for the classification is obtained before the optimum for the regression. This might be an indication that it is worth investigating a solution decoupling the position of the lanes from their

classification. Such solution may be better specialized than the ones presented in this thesis, but might add a need for post processing to combine the two outputs.

As mentioned in Section 6.6.2, the authors did not prioritize testing different learning rates and annealing schemes for the confidence outputting networks. This is an area which may be further explored, perhaps yielding performance increases in the form of both better regression and no false suppression of points based on bad regression confidence.

Another point in the data category is that an interesting and somewhat novel solution may be to abstract the lane not as markings, but as conceptual "where-to-drive" fields, albeit often designated by markings. This would allow for data to be labeled with consistent lanes marking where vehicles would drive. This is theorized to create a more robust system, since human drivers are believed to seldom react to the markings themselves, but more to the greater concept of "this is where to drive". Mimicking this human behavior might increase performance vastly.

Appendix

A

Training Predictions

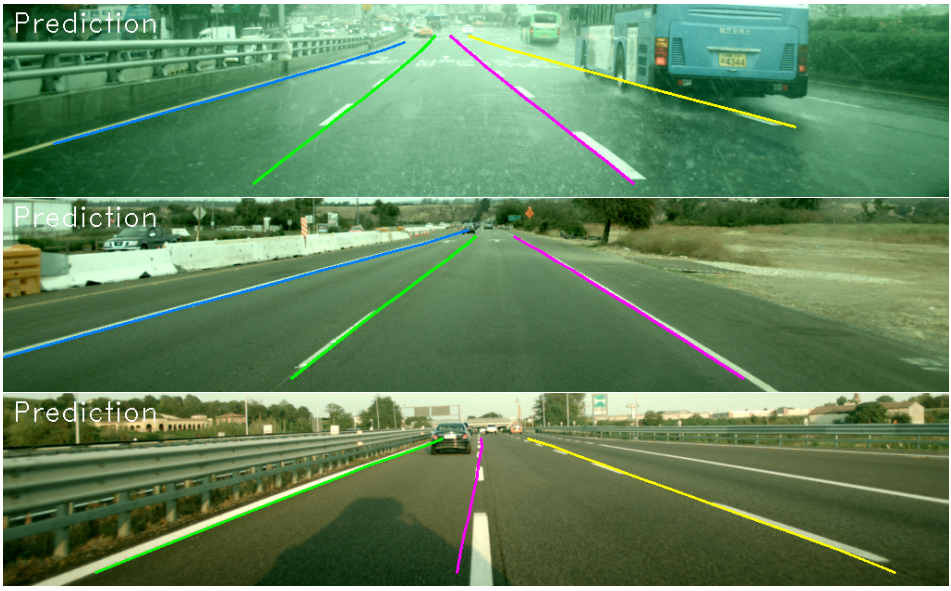


Figure A.1: *Regular Absolute Error Loss: predictions for rain, straight road and lane change*

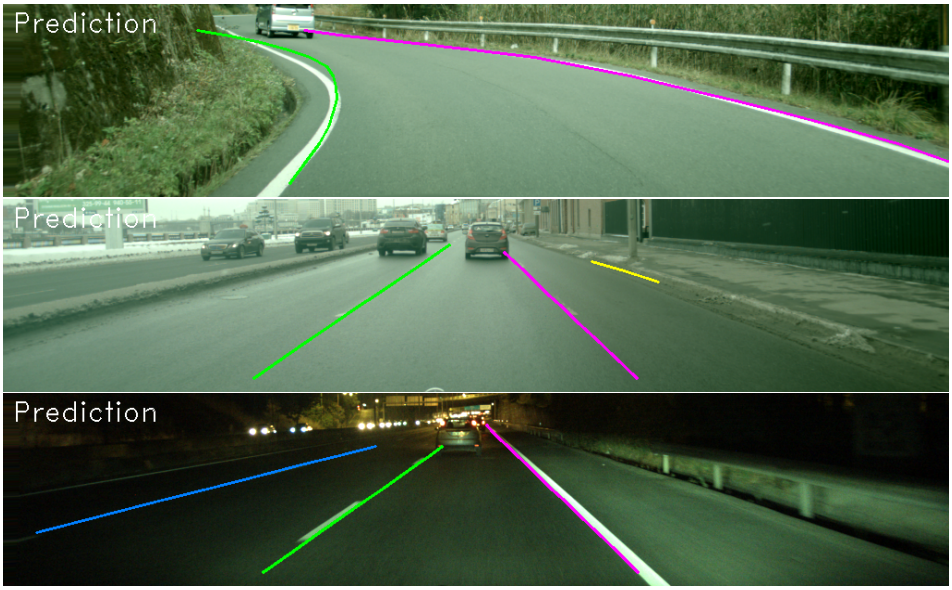


Figure A.2: *Regular Absolute Error Loss: predictions for curve, weak markings and night*

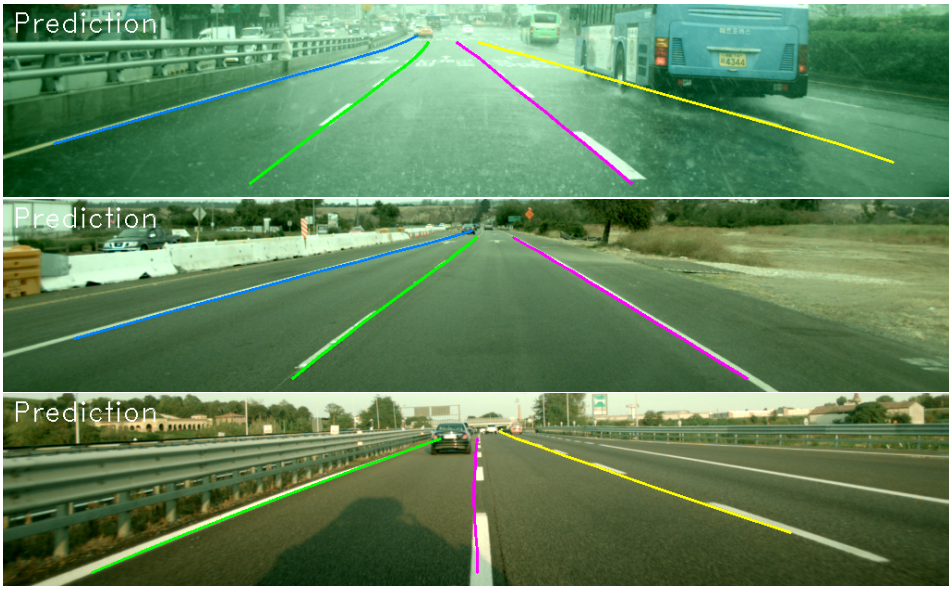


Figure A.3: *Horizon Loss: predictions for rain, straight road and lane change*

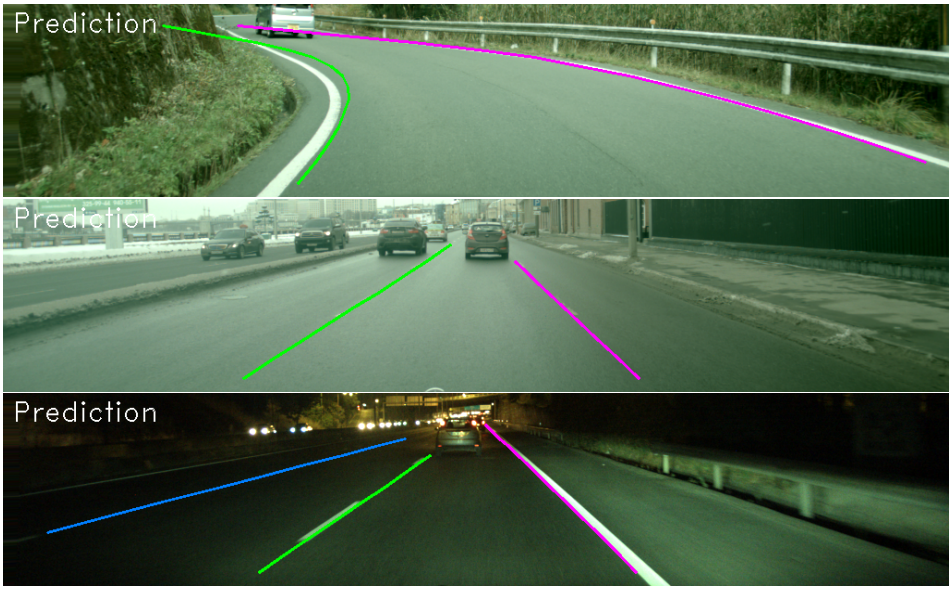


Figure A.4: *Horizon Loss: predictions for curve, weak markings and night*

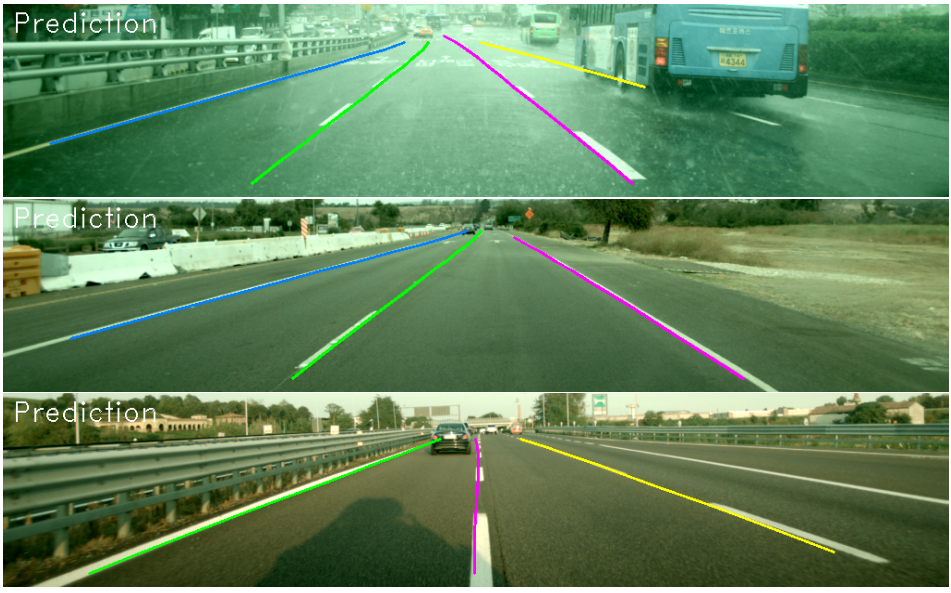


Figure A.5: *Lane Change Loss with Hard Minimum: predictions for rain, straight road and lane change*

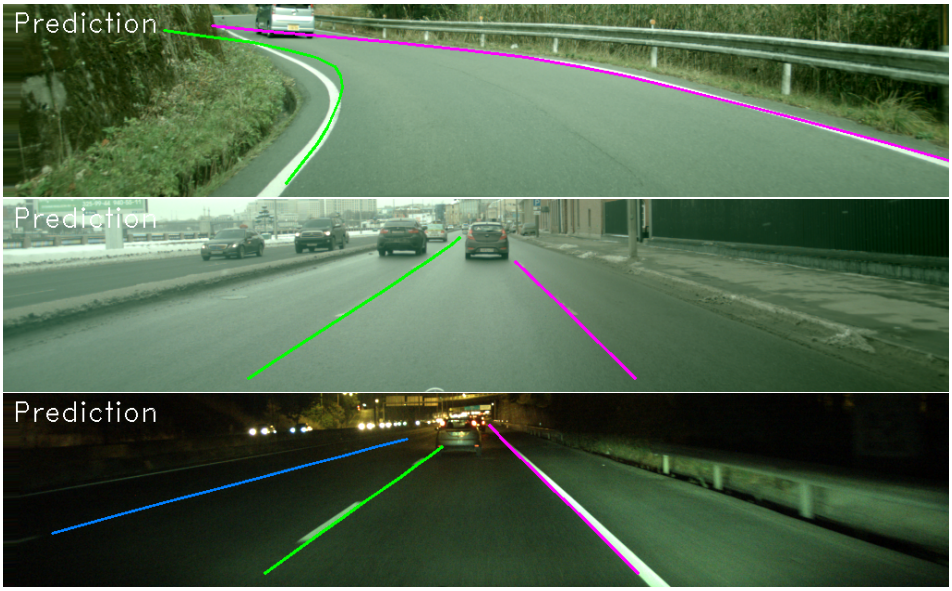


Figure A.6: *Lane Change Loss with Hard Minimum: predictions for curve, weak markings and night*

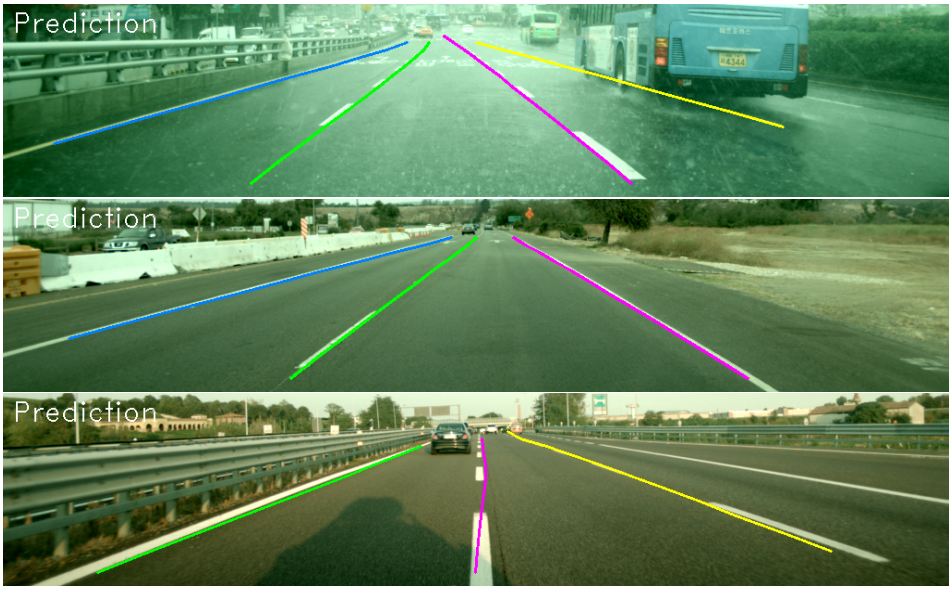


Figure A.7: *Lane Change Loss with Soft Minimum: predictions for rain, straight road and lane change*

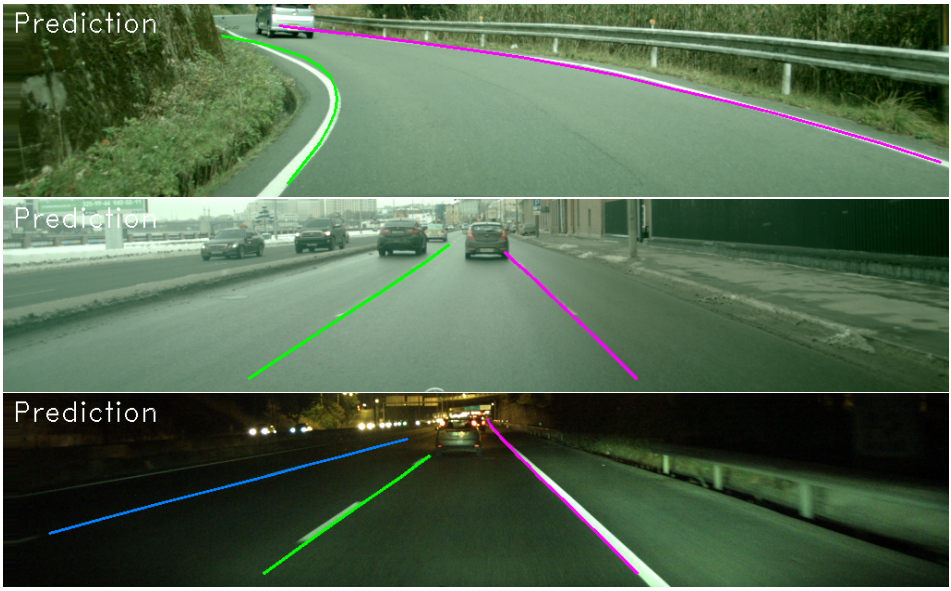


Figure A.8: *Lane Change Loss with Soft Minimum: predictions for curve, weak markings and night*

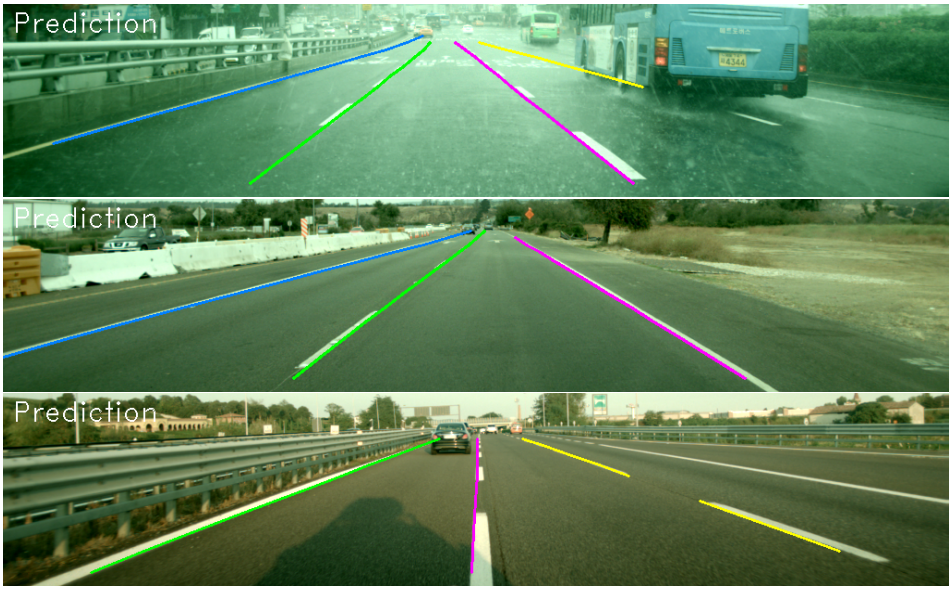


Figure A.9: *Lane Change Loss with No Oversampling: predictions for rain, straight road and lane change*

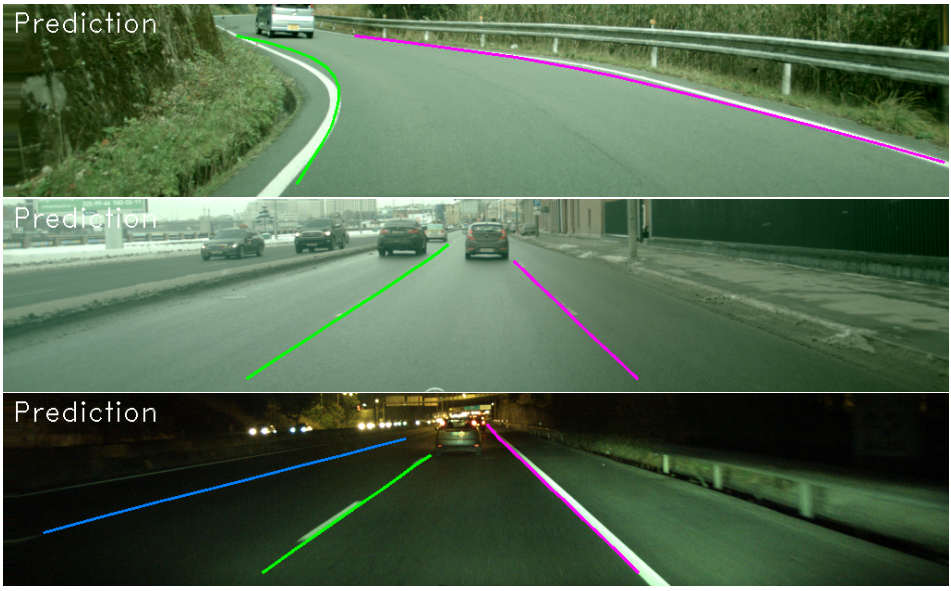


Figure A.10: *Lane Change Loss with No Oversampling: predictions for curve, weak markings and night*

Figure A.11: *Confidences on Larger Network with Pre-training: predictions for rain, straight road and lane change*

Figure A.12: *Confidences on Larger Network with Pre-training: predictions for curve, weak markings and night*

Figure A.13: *Confidences on Smaller Network with Pre-training: predictions for rain, straight road and lane change*

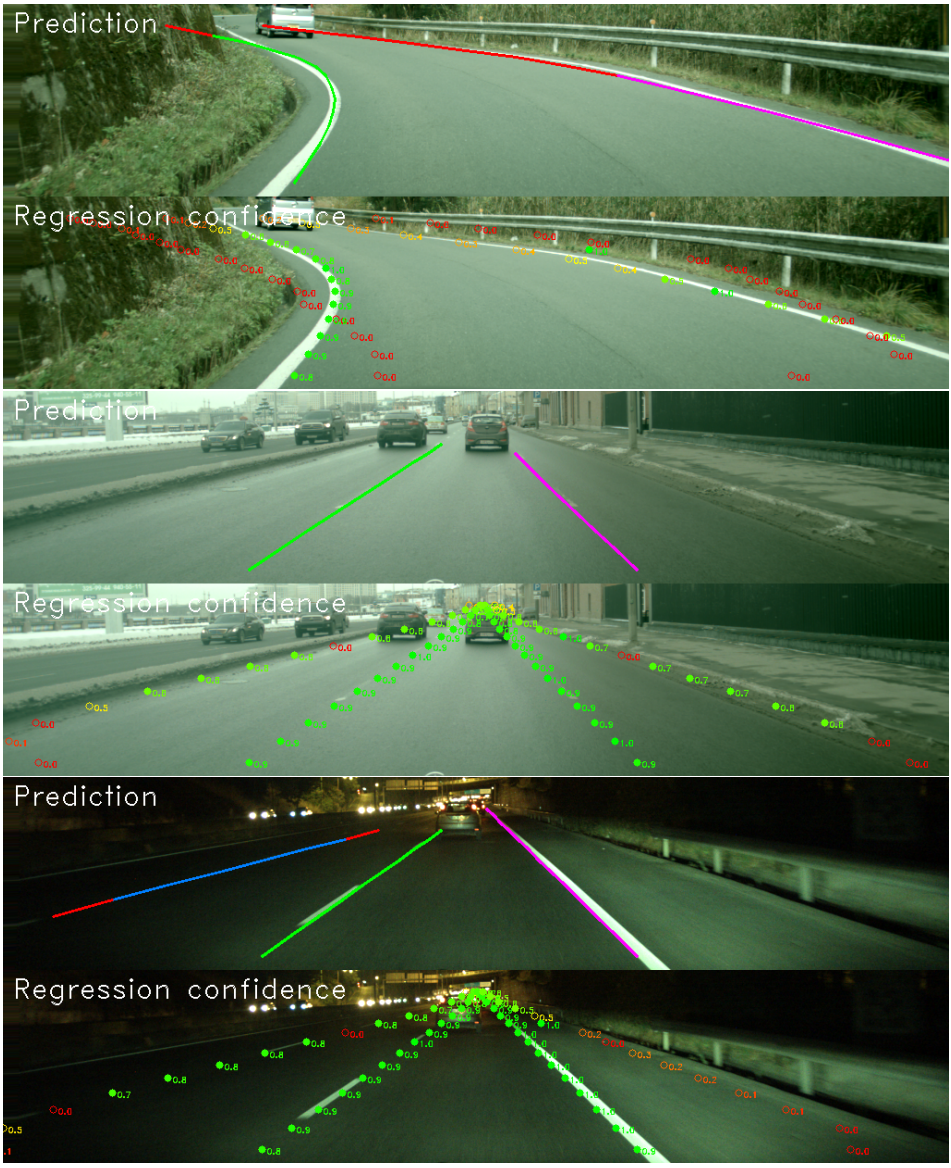


Figure A.14: *Confidences on Smaller Network with Pre-training: predictions for curve, weak markings and night*

Figure A.15: *Confidences on Larger Network without Pre-training: predictions for rain, straight road and lane change*

Figure A.16: *Confidences on Larger Network without Pre-training: predictions for curve, weak markings and night*

Figure A.17: *Confidences on Smaller Network without Pre-training: predictions for rain, straight road and lane change*

Figure A.18: *Confidences on Smaller Network without Pre-training: predictions for curve, weak markings and night*

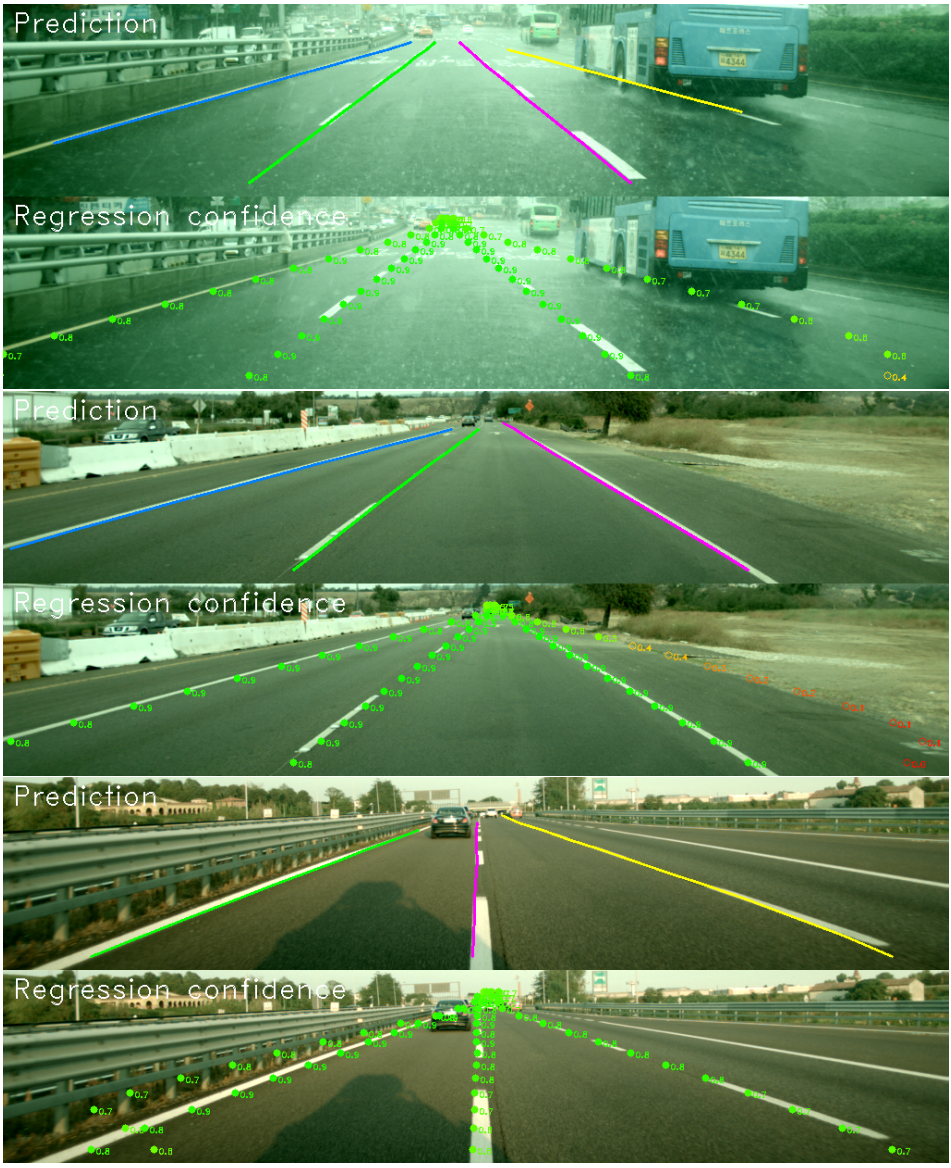


Figure A.19: *Confidences on Larger Multi-stream Network: predictions for rain, straight road and lane change*

Figure A.20: *Confidences on Larger Multi-stream Network: predictions for curve, weak markings and night*

Figure A.21: *Confidences on Smaller Multi-stream Network: predictions for rain, straight road and lane change*

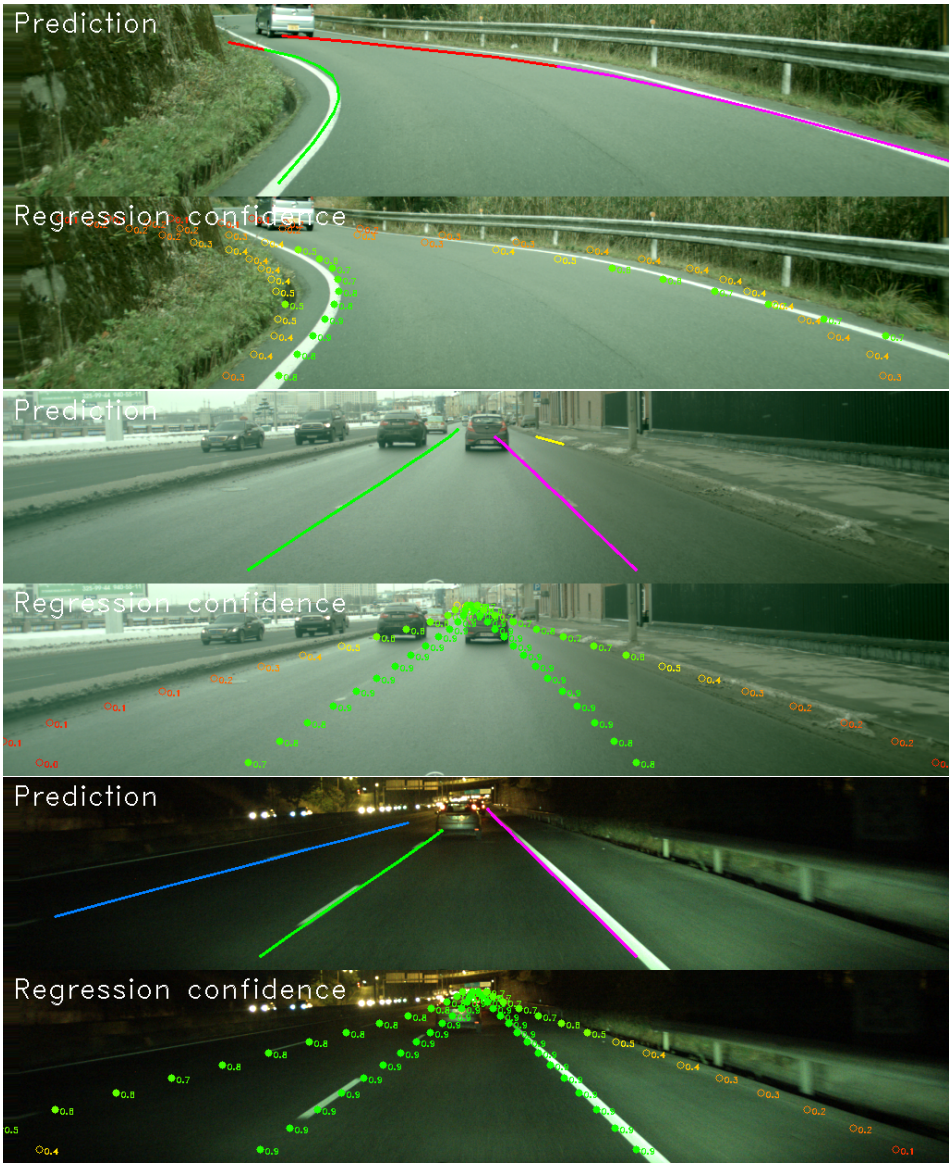


Figure A.22: *Confidences on Smaller Multi-stream Network: predictions for curve, weak markings and night*

Figure A.23: *Segmentation on Full Dataset: predictions for rain, straight road and lane change*

Figure A.24: *Segmentation on Full Dataset: predictions for curve, weak markings and night*

Figure A.25: *Segmentation on Quarter Dataset: predictions for rain, straight road and lane change*

Figure A.26: *Segmentation on Quarter Dataset: predictions for curve, weak markings and night*

B

Lane Change Comparison

This appendix will show comparison images from a lane change sequence for *Regular Absolute Loss*, *Lane Change Loss with Hard Minimum* and *Lane Change Loss with Soft Minimum*.

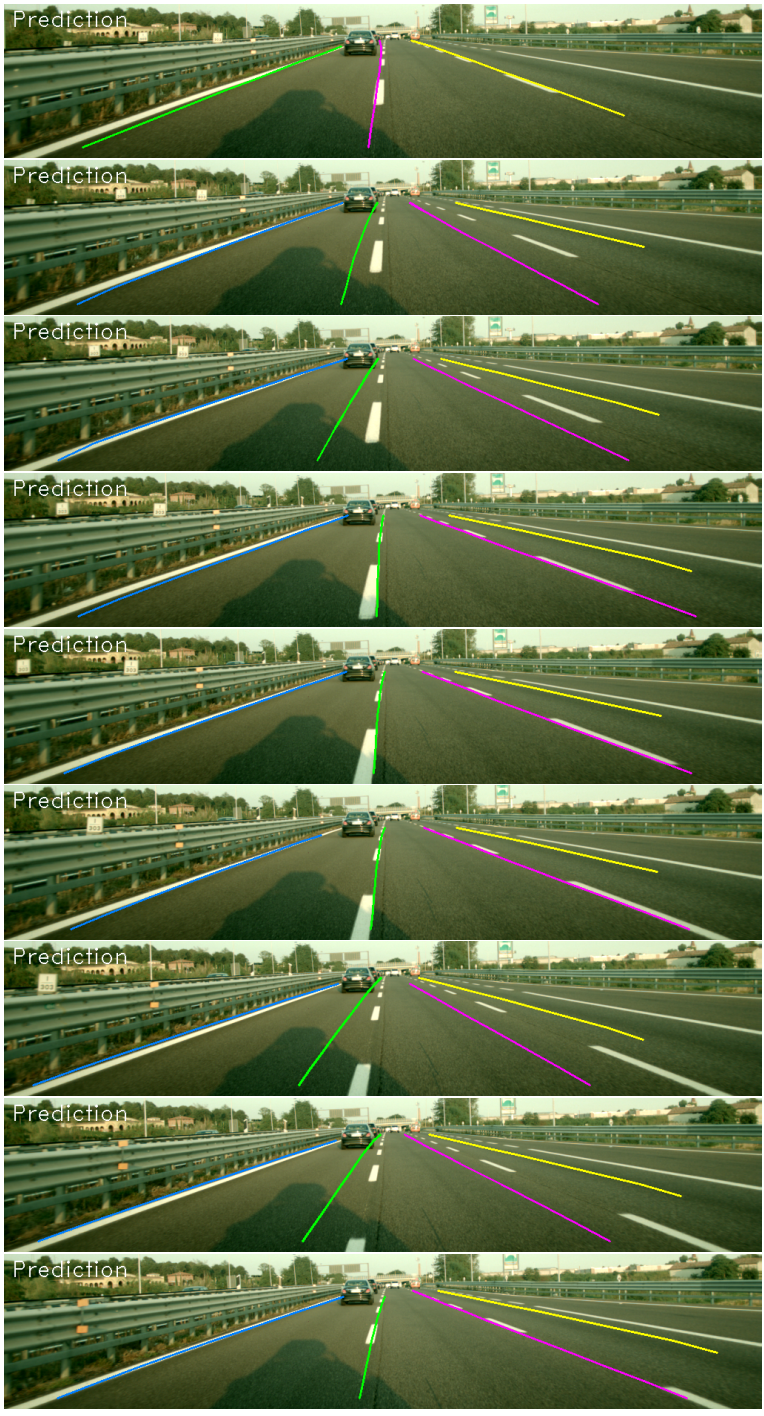


Figure B.1: *Lane change sequence for Regular Absolute Loss*

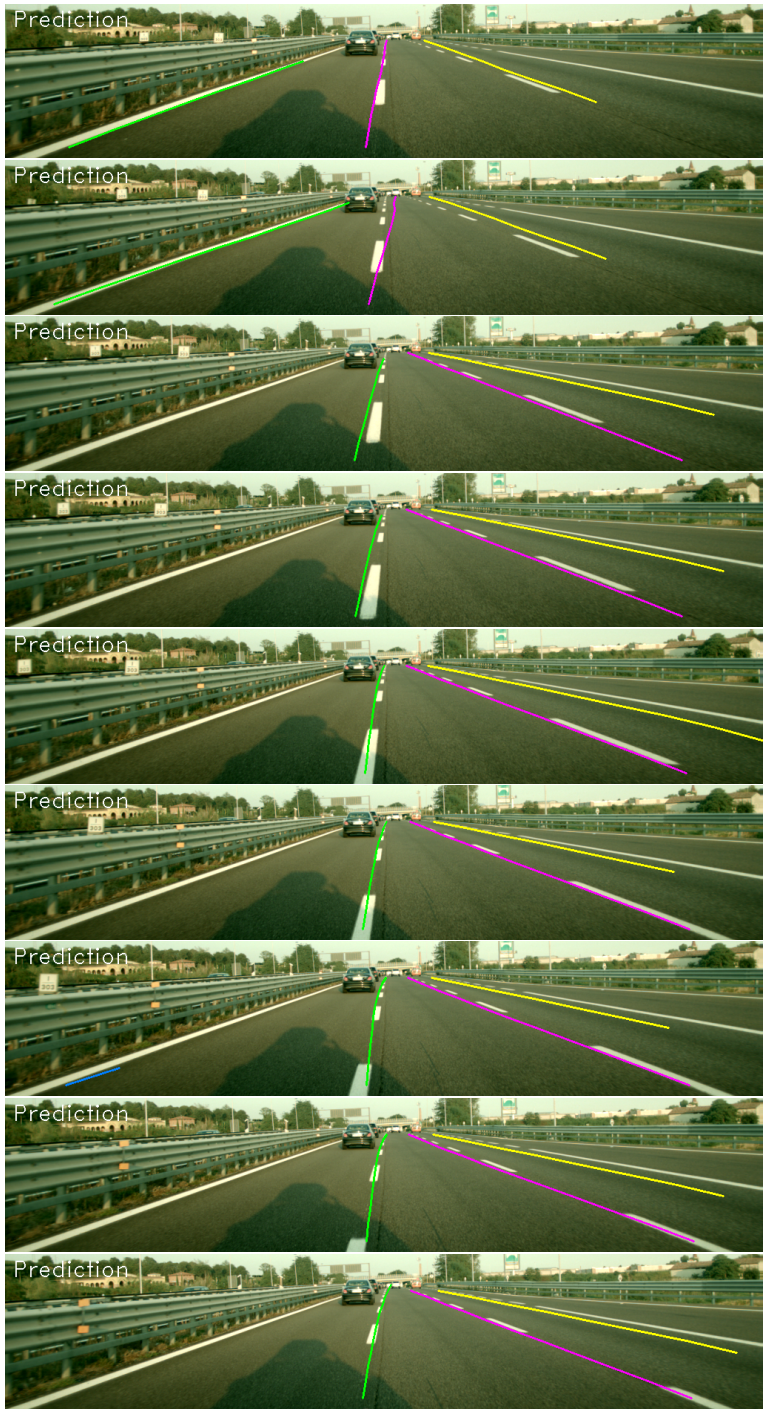


Figure B.2: *Lane change sequence for Lane Change Loss with Hard Minimum*

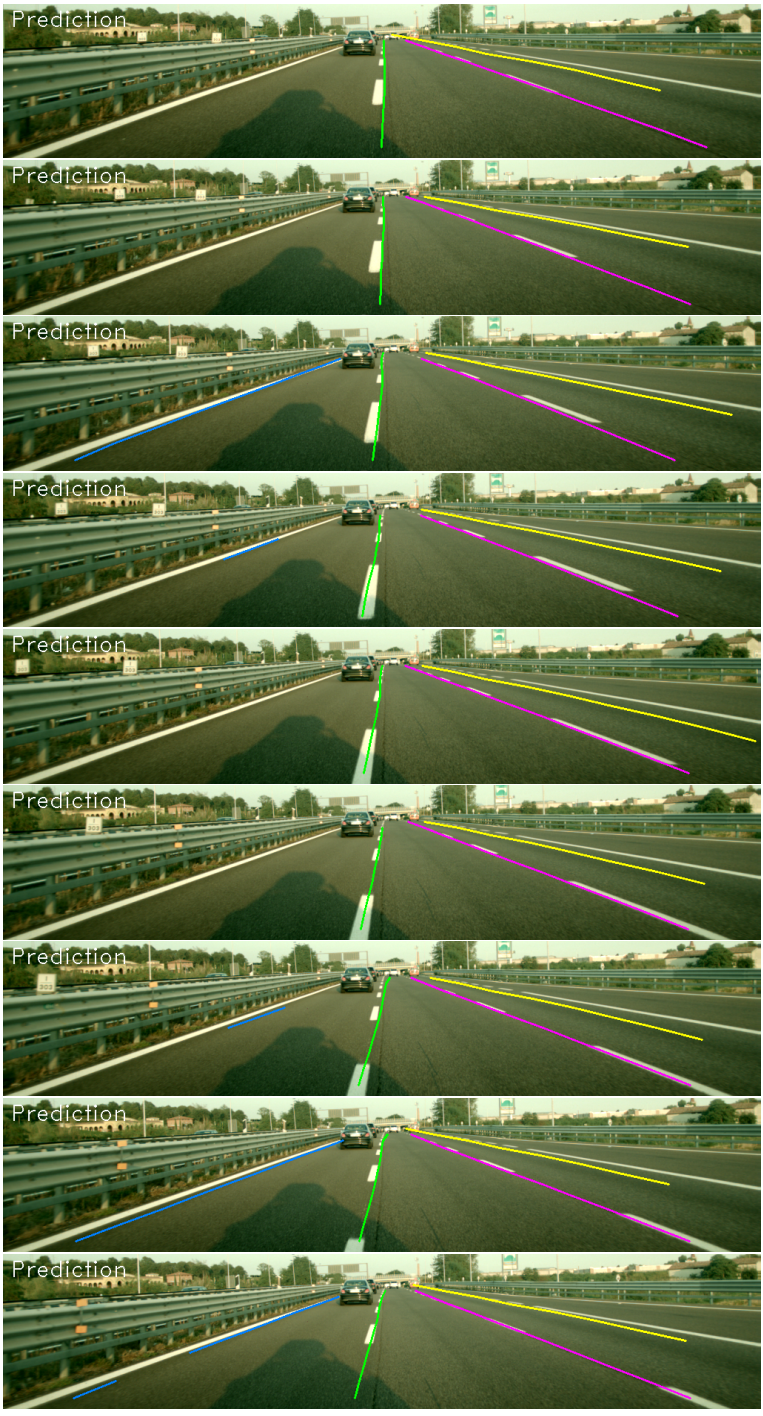


Figure B.3: *Lane change sequence for Lane Change Loss with Soft Minimum*

C

Gradient Calculation

Here the partial derivatives of the loss functions are calculated, beginning with the trivial case of the regression and classification losses.

$$\frac{\partial}{\partial \bar{x}}(x-\bar{x})^2 = 2(x-\bar{x}) \tag{C.1a}$$

$$\frac{\partial}{\partial m}(x-\bar{x})^2 = 0 \tag{C.1b}$$

$$\frac{\partial}{\partial \bar{c}}(x-\bar{x})^2 = 0 \tag{C.1c}$$

$$\frac{\partial}{\partial \bar{x}}|x-\bar{x}| = \frac{x-\bar{x}}{|x-\bar{x}|} \tag{C.1d}$$

$$\frac{\partial}{\partial m}|x-\bar{x}| = 0 \tag{C.1e}$$

$$\frac{\partial}{\partial \bar{c}}|x-\bar{x}| = 0 \tag{C.1f}$$

Where Equation C.1d holds for $x \neq \bar{x}$.

$$\frac{\partial}{\partial \bar{x}} - (n\log(m) + (1-n)\log(1-m)) = 0 \tag{C.2a}$$

$$\frac{\partial}{\partial m} - (n\log(m) + (1-n)\log(1-m)) = \left(\frac{1-n}{1-m} - \frac{n}{m}\right) \tag{C.2b}$$

$$\frac{\partial}{\partial \bar{c}} - (n\log(m) + (1-n)\log(1-m)) = 0 \tag{C.2c}$$

For the classification losses, it should first be noted that

$$\frac{\partial}{\partial \bar{x}} e^{-\frac{|x-\bar{x}|}{T}\ln 2} = \frac{x-\bar{x}}{|x-\bar{x}|}\frac{\ln 2}{T} e^{e^{-\frac{|x-\bar{x}|}{T}\ln 2}} \tag{C.3}$$

and that

$$\frac{\partial}{\partial \bar{x}} e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} = 2(x-\bar{x}) \frac{\ln 2}{T^2}^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \tag{C.4}$$

thus

$$\frac{\partial}{\partial \bar{x}} |\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}| = -\frac{\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}}{|\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}|} \frac{x-\bar{x}}{|x-\bar{x}|} \frac{\ln 2}{T} e^{-\frac{|x-\bar{x}|}{T} \ln 2} \tag{C.5}$$

and

$$\frac{\partial}{\partial \bar{x}} |\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}| = -\frac{\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}}{|\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}|} 2(x-\bar{x}) \frac{\ln 2}{T^2}^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \tag{C.6}$$

while for the L2-norm,

$$\frac{\partial}{\partial \bar{x}} (\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2})^2 = -2(\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}) \frac{x-\bar{x}}{|x-\bar{x}|} \frac{\ln 2}{T} e^{-\frac{|x-\bar{x}|}{T} \ln 2} \tag{C.7}$$

and

$$\frac{\partial}{\partial \bar{x}} (\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2})^2 = -2(\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}) 2(x-\bar{x}) \frac{\ln 2}{T^2}^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \tag{C.8}$$

so for the L1-norm for regression distance,

$$\frac{\partial}{\partial \bar{x}} |\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}| = -\frac{\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}}{|\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}|} \frac{x-\bar{x}}{|x-\bar{x}|} \frac{\ln 2}{T} e^{-\frac{|x-\bar{x}|}{T} \ln 2} \tag{C.9a}$$

$$\frac{\partial}{\partial m} |\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}| = 0 \tag{C.9b}$$

$$\frac{\partial}{\partial \bar{c}} |\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}| = \frac{\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}}{|\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}|} \tag{C.9c}$$

$$\frac{\partial}{\partial \bar{x}} (\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2})^2 = -2(\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}) \frac{x-\bar{x}}{|x-\bar{x}|} \frac{\ln 2}{T} e^{-\frac{|x-\bar{x}|}{T} \ln 2} \tag{C.10a}$$

$$\frac{\partial}{\partial m} (\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2})^2 = 0 \tag{C.10b}$$

$$\frac{\partial}{\partial \bar{c}} (\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2})^2 = 2(\bar{c} - e^{-\frac{|x-\bar{x}|}{T} \ln 2}) \tag{C.10c}$$

and for the L2-norm for regression distance,

$$\frac{\partial}{\partial \bar{x}}|\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}| = -\frac{\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}}{|\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}|} 2(x - \bar{x}) \frac{\ln 2}{T^2} e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \tag{C.11a}$$

$$\frac{\partial}{\partial m}|\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}| = 0 \tag{C.11b}$$

$$\frac{\partial}{\partial \bar{c}}|\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}| = \frac{\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}}{|\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}|} \tag{C.11c}$$

$$\frac{\partial}{\partial \bar{x}}(\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2})^2 = -2(\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}) 2(x - \bar{x}) \frac{\ln 2}{T^2} e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \tag{C.12a}$$

$$\frac{\partial}{\partial m}(\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2})^2 = 0 \tag{C.12b}$$

$$\frac{\partial}{\partial \bar{c}}(\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2})^2 = 2(\bar{c} - e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}) \tag{C.12c}$$

Where it should be noted that given the discontinuity of $\frac{d}{dx}|x|$ at $x = 0$, the derivatives including absolute values only holds when the difference is non-zero. Thus, these equations only hold when the predictions are not perfect, which occurs which is true with probability 1 in a neural network application such as this one.

D

Gradient Extreme Values

Here the extreme values for the gradients of the confidence errors are calculated, with respect to the regression prediction.

Begin by noting that the fraction $-\frac{\bar{c}-e^{-\frac{|x-\bar{x}|}{T}\ln 2}}{|\bar{c}-e^{-\frac{|x-\bar{x}|}{T}\ln 2}|}$ only takes on values of 1 or -1, and that it thus does not affect for which values of $\bar{x}$ the function have its extreme values. The same argument applies to the fraction $\frac{x-\bar{x}}{x-\bar{x}}$, which will be ignored in the same way - albeit noting that the original equations only holds for $\bar{x} \neq x$ and $\bar{c} \neq e^{-\frac{|x-\bar{x}|}{T}\ln 2}$.

For $\frac{\ln 2}{T}e^{-\frac{|x-\bar{x}|}{T}\ln 2}$ from Equation C.9a, it is easily seen that the maximum value is dependent on the decaying exponent, and that it is thus taken at $\bar{x} = x$, for which that part of the equation takes the value $\frac{\ln 2}{T}$. Since $\bar{x} \neq x$ was set as a constraint, it can be concluded that

$$\lim_{\bar{x}\to x} -\frac{\bar{c}-e^{-\frac{|x-\bar{x}|}{T}\ln 2}}{|\bar{c}-e^{-\frac{|x-\bar{x}|}{T}\ln 2}|}\frac{x-\bar{x}}{|x-\bar{x}|}\frac{\ln 2}{T}e^{-\frac{|x-\bar{x}|}{T}\ln 2} = \pm\frac{\ln 2}{T} \tag{D.1}$$

from trivial limits of the decaying exponent. The sign is decided by the fractions mentioned above.

For $-2(\bar{c}-e^{-\frac{|x-\bar{x}|}{T}\ln 2})\frac{x-\bar{x}}{|x-\bar{x}|}\frac{\ln 2}{T}e^{-\frac{|x-\bar{x}|}{T}\ln 2}$ from Equation C.10a the same reasoning applies, letting

$$\lim_{\bar{x}\to x} -2(\bar{c}-e^{-\frac{|x-\bar{x}|}{T}\ln 2})\frac{x-\bar{x}}{|x-\bar{x}|}\frac{\ln 2}{T}e^{-\frac{|x-\bar{x}|}{T}\ln 2} = \pm 2(\bar{c}-1)\frac{\ln 2}{T} \tag{D.2}$$

with the sign once again decided by the absolute value. From Equation D.2 it is noted that the maximum value is achieved when $\bar{c} = 0$, at $\pm 2\frac{\ln 2}{T}$

For $-\frac{\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}}{|\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}|}2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}$ from Equation C.11a, it is one again

noted that the fraction $\frac{\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}}{|\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}|}$ can be ignored. This allows us to focus on $2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}$. Differentiating gives

$$\begin{aligned}\frac{\partial}{\partial \bar{x}} 2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} &= \frac{\ln 4}{T^2}\frac{\partial}{\partial \bar{x}}(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \\ &= \frac{\ln 4}{T^2}\frac{\partial}{\partial \bar{x}}\left(-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} + 2\left(\frac{x-\bar{x}}{T}\right)\ln 2 e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}\right) \\ &= \frac{\ln 4}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}\left(\ln 4\frac{x-\bar{x}}{T} - 1\right)\end{aligned} \tag{D.3}$$

It is easily seen that Equation D.3 $= 0 \iff \ln 4\frac{x-\bar{x}}{T} - 1 = 0$

$$\begin{aligned}\ln 4\frac{x-\bar{x}}{T} - 1 = 0 &\iff \frac{x-\bar{x}}{T} = \frac{1}{\ln 4} \\ &\iff x-\bar{x} = \frac{T}{\ln 4} \\ &\iff \bar{x} = x - \frac{T}{\ln 4}\end{aligned} \tag{D.4}$$

Using the result from Equation D.4 in Equation C.11a gives the following extreme value

$$\begin{aligned}2(x-x+\frac{T}{\ln 4})\frac{\ln 2}{T^2}e^{-\left(\frac{x-x+\frac{T}{\ln 4}}{T}\right)^2} &= \frac{2T}{\ln 4}\frac{\ln 2}{T^2}e^{-\left(\frac{1}{\ln 4}\right)^2 \ln 2} \\ &= \frac{1}{T}e\end{aligned} \tag{D.5}$$

with sign determined from the confidence prediction.

For $2(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2})2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}$ from Equation C.12a, the following differential is constructed

$$\begin{aligned}&\frac{\partial}{\partial \bar{x}} 2(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2})2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \\ &= \frac{4\ln 2}{T^2}\frac{\partial}{\partial \bar{x}}(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2})(x-\bar{x})e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \\ &= \frac{4\ln 2}{T^2}[\left(\frac{\partial}{\partial \bar{x}}\left(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}\right)\right)(x-\bar{x})e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \\ &\quad + \left(\frac{\partial}{\partial \bar{x}}(x-\bar{x})\right)\left(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}\right)e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2} \\ &\quad + \left(\frac{\partial}{\partial \bar{x}}e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}\right)\left(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}\right)(x-\bar{x})]\end{aligned} \tag{D.6}$$

where the four differentials may now be calculated:

$$\frac{\partial}{\partial \bar{x}}\left(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}\right)=-2(x-\bar{x})\frac{\ln 2}{T}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2} \tag{D.7a}$$

$$\frac{\partial}{\partial \bar{x}}(x-\bar{x})=-1 \tag{D.7b}$$

$$\frac{\partial}{\partial \bar{x}}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}=2(x-\bar{x})\frac{\ln 2}{T}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2} \tag{D.7c}$$

which may be combined with their respective factors:

$$-2(x-\bar{x})\frac{\ln 2}{T}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}(x-\bar{x})\,e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}=-(x-\bar{x})^2\,\frac{\ln 4}{T}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 4} \tag{D.8}$$

$$-\left(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}\right)e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2} \tag{D.9}$$

$$2(x-\bar{x})\frac{\ln 2}{T}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2})(x-\bar{x})=\frac{\ln 4}{T}\,(x-\bar{x})^2\,(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2})e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2} \tag{D.10}$$

Now using Equations D.8,D.9 and D.10 in Equation D.6, we get the following

$$\begin{aligned}
&\frac{\partial}{\partial \bar{x}}2(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2})2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}\\
&\qquad=-(x-\bar{x})^2\,\frac{\ln 4}{T}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 4}\\
&\qquad-\left(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}\right)e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}\\
&\qquad+\frac{\ln 4}{T}\,(x-\bar{x})^2\,(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2})e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}\\
&\qquad=\frac{\ln 4}{T}\,(x-\bar{x})^2\,\bar{c}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}+e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 4}\\
&-\bar{c}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}-\frac{\ln 4}{T}\,(x-\bar{x})^2\,e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 4}-\frac{\ln 4}{T}\,(x-\bar{x})^2\,e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 4}\\
&\qquad=\frac{\ln 4}{T}\,(x-\bar{x})^2\,\bar{c}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}+e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 4}\\
&\qquad-\bar{c}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}-\frac{\ln 16}{T}\,(x-\bar{x})^2\,e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 4}
\end{aligned} \tag{D.11}$$

It is seen from Equation D.11 that the extreme values of Equation C.12a is not trivial to calculate analytically. Therefore, it is reasoned based on its similarity with Equation D.5 that it will at most have the same maximum value scaled by a factor of ± 2. This follows from the same factor $2(x-\bar{x})\frac{\ln 2}{T^2}e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2}$ with maximum value $\frac{1}{T}e$ determined from Equation D.5. The factor $2(\bar{c}-e^{-\left(\frac{x-\bar{x}}{T}\right)^2\ln 2})$ is

trivially determined to be limited to $[-2, 2]$ since both the confidence prediction $\bar{c}$ and the calculated confidence $^{-\left(\frac{x-\bar{x}}{T}\right)^2 \ln 2}$ by design are limited to $(0, 1]$. Thus, their difference is limited to $(-1, 1)$, which is then scaled by the factor 2.

E

Full Network Architectures

This Appendix will disclose the full architecture of the different networks used in this thesis.

E.1 ENet Backbone

The ENet Backbone used was directly implemented from the paper presented by Paszke et al. [27]. This study only used the encoder part of the network, since it was used as a backbone for the investigated networks. Therefore, the network which was used as the ENet Backbone corresponds to the ENet network presented in Table 1 in [27], up to, but not including, the first upsampling.

E.2 Network with 2 Skips

Figure E.1 shows the design for the head of the network with 2 skips, which was attached to the ENet Backbone.

E.3 Confidence Network with 2 Skips

Figure E.2 shows the design for the head of the confidence outputting network with 2 skips, which was attached to the ENet Backbone.

E.4 Confidence Network with 1 Skip

Figure E.3 shows the design for the head of the confidence outputting network with 1 skips, which was attached to the ENet Backbone.

E.5 Multi-stream Network with 2 Skips

Figure E.4 shows the design for the head of the multi-stream, confidence outputting network with 2 skips, which was attached to the ENet Backbone.

E.6 Multi-stream Network with 1 Skips

Figure E.5 shows the design for the head of the multi-stream, confidence outputting network with 1 skips, which was attached to the ENet Backbone.

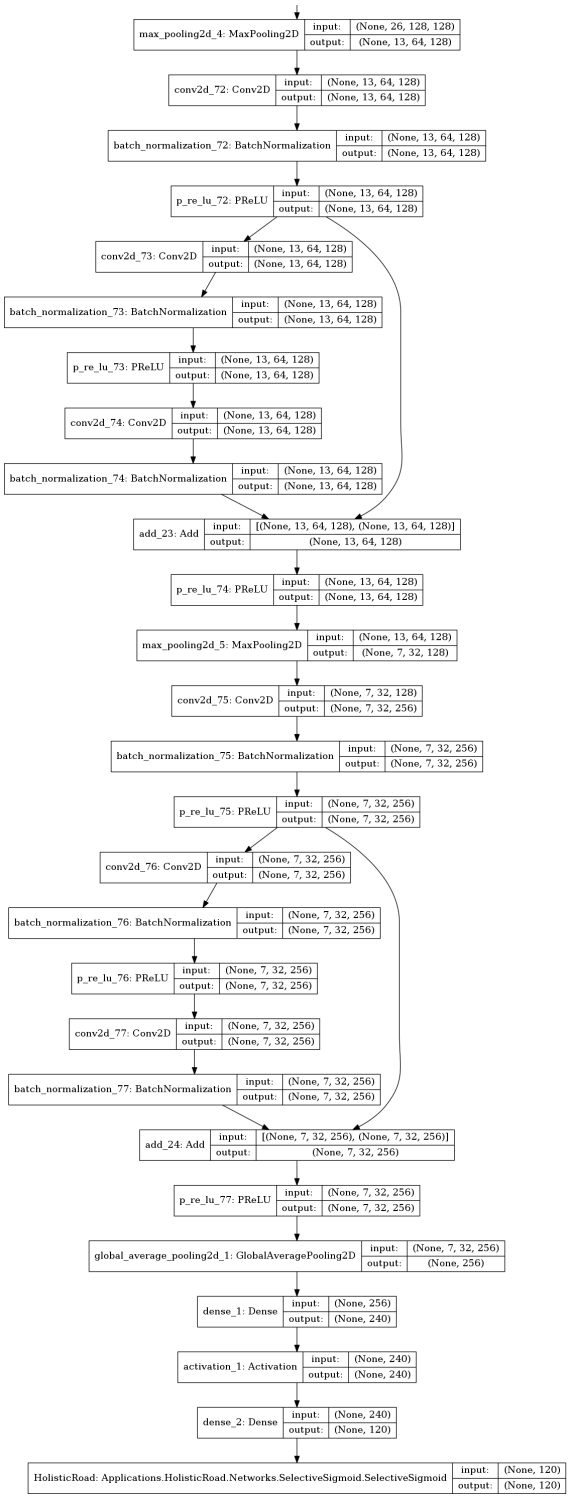


Figure E.1: *Architecture of the head with two ResNet style skips*

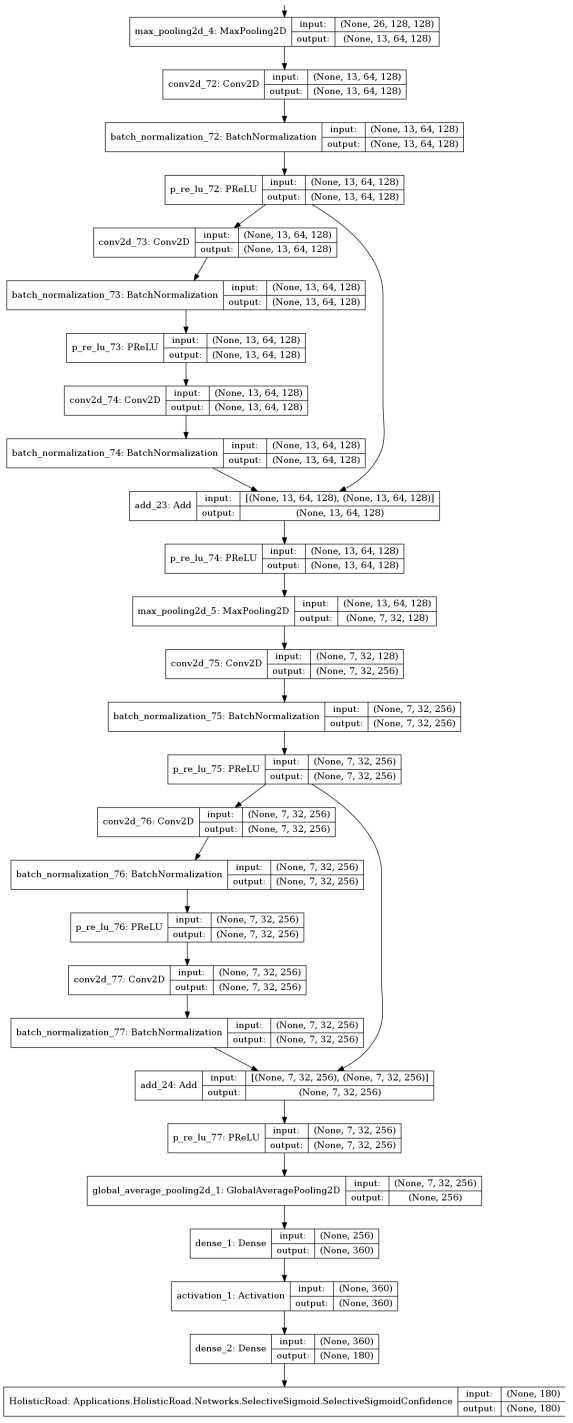


Figure E.2: *Architecture of the confidence outputting head with two ResNet style skips*

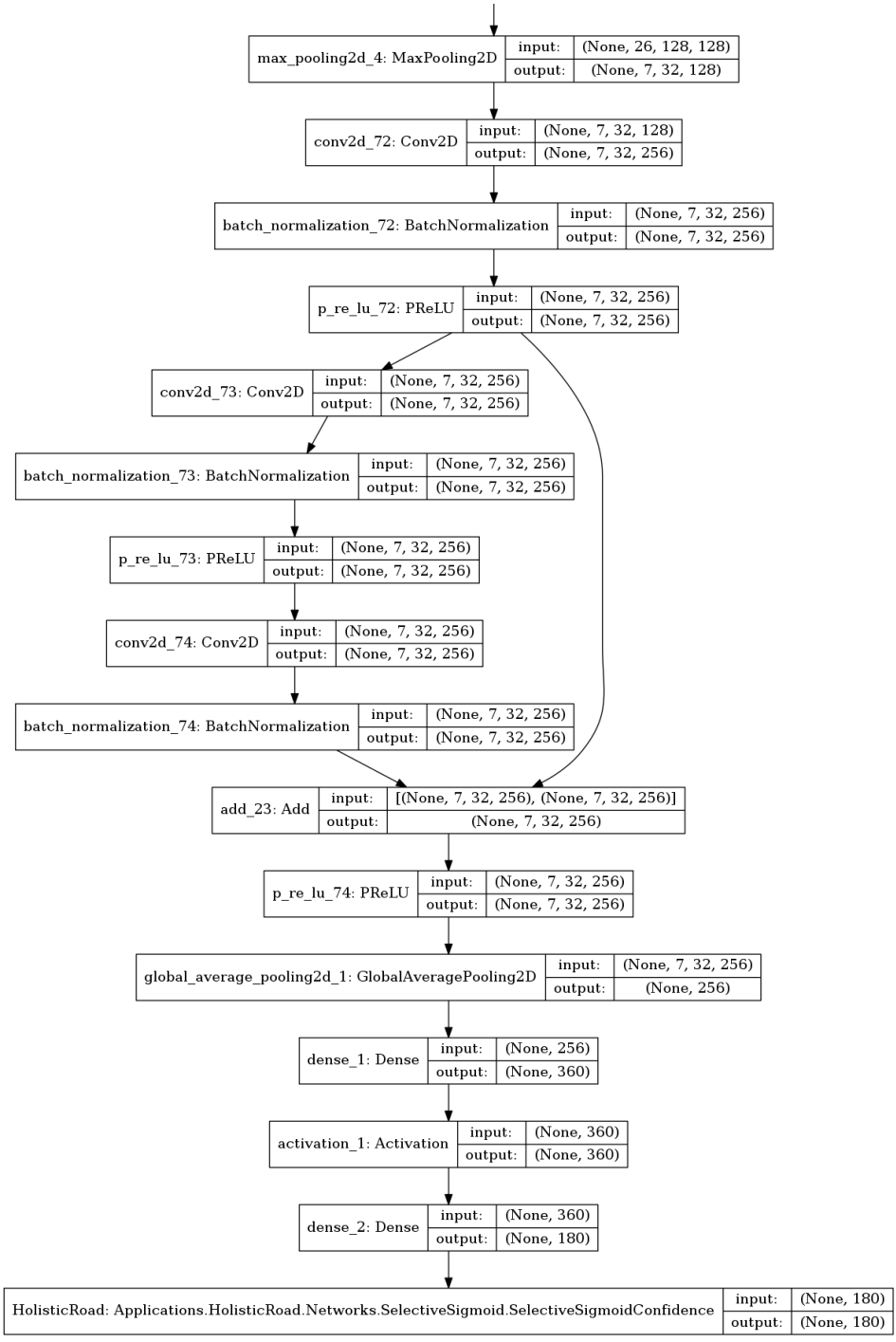


Figure E.3: *Architecture of the confidence outputting head with one ResNet style skips*

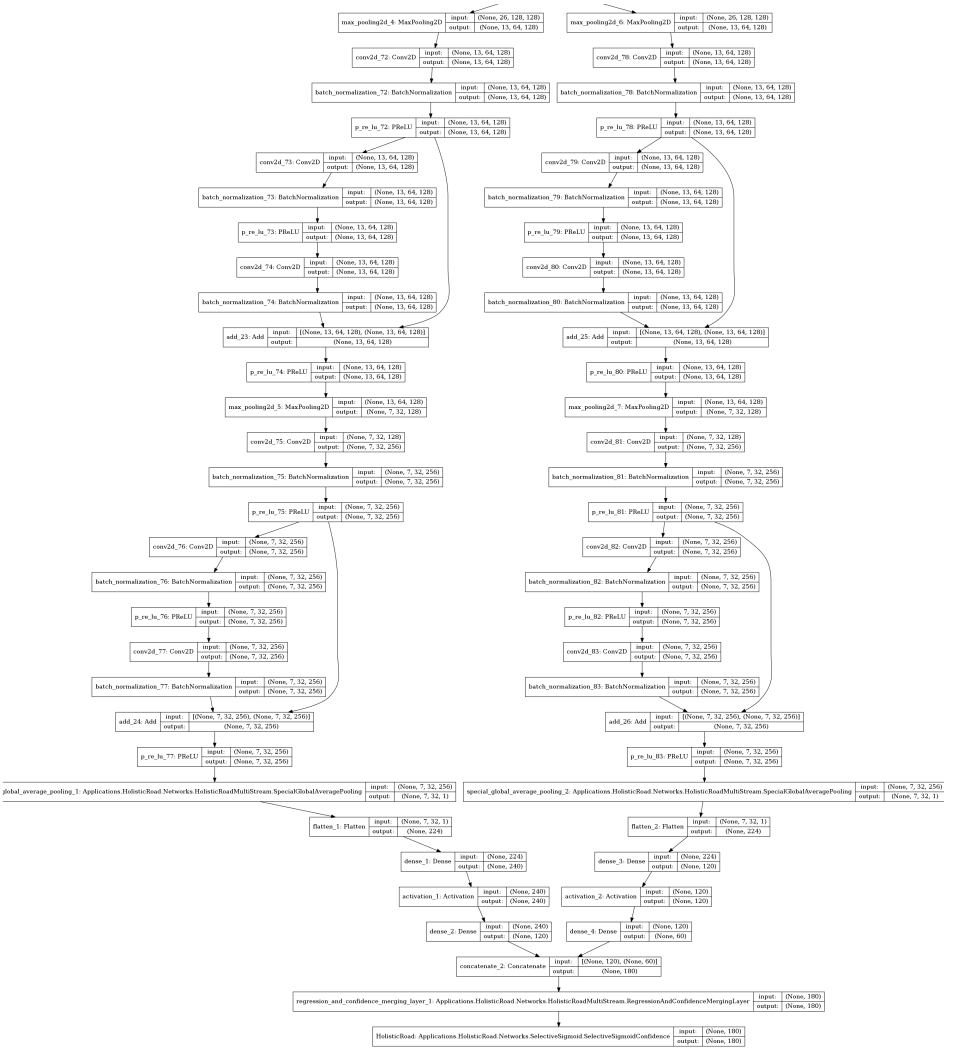


Figure E.4: *Architecture of the multi-stream confidence outputting head with two ResNet style skips*

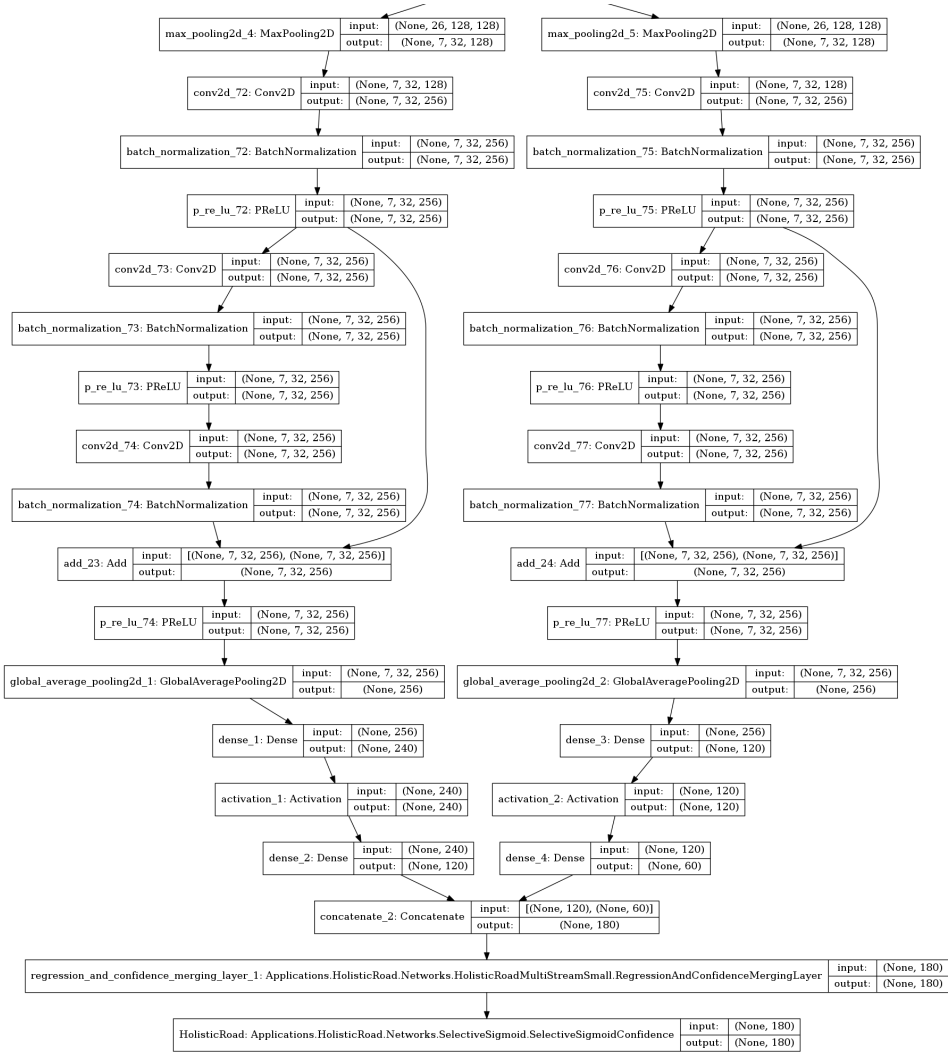


Figure E.5: *Architecture of the multi-stream confidence outputting head with one ResNet style skips*

Bibliography

[1] tuSimple Benchmark. URL http://benchmark.tusimple.ai/. Visited 2018-12-04.

[2] Sina Mokhtarzadeh Azar, Mina Ghadimi Atigh, and Ahmad Nickabadi. A multi-stream convolutional neural network framework for group activity recognition. *CoRR*, abs/1812.10328, 2018.

[3] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *CoRR*, abs/1505.07293, 2015.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL http://arxiv.org/abs/1604.07316.

[5] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:436–44, 2018.

[6] Z. Chen and X. Huang. End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1856–1860, June 2017. doi: 10.1109/IVS.2017.7995975.

[7] François Chollet et al. Keras. https://keras.io, 2015.

[8] Daniel Godoy. Understanding binary cross-entropy / log loss: a visual explanation, 2018. URL https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. Visited 2019-03-08.

[9] Xianzhi Du, Mostafa El-Khamy, Vlad I. Morariu, Jungwon Lee, and Larry S. Davis. Fused deep neural networks for efficient pedestrian detection. *CoRR*, abs/1805.08688, 2018.

[10] European Automobiles Manufacturer's Commision. 'Active' vehicle safety most effective, new analysis of accident data shows, 2018. URL https://www.acea.be/press-releases/article/active-vehicle-safety-most-effective-new-analysis-of-accident-data-shows. Visited 2018-11-26.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947–951, 2000.

[13] Trevor Hastie, Robert Tibsharini, and Jerome Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2009.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Dec 2015. doi: 10.1109/ICCV.2015.123.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[17] L. D. Jackel, B. Boser, H.P Graf, J. S. Denker, D. Henderson Y. LeCun, O. Matan, and R. E. Howard. Vlsi implementations of electronic neural networks: An example in character recognition. *IEE*, 1990.

[18] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5574–5584. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision.pdf.

[19] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[20] M. Lange, D. Zühlke, O. Holz, and T. Villmann. Applications of lp-norms and their smooth approximations for gradient based learning vector quantization. In *22st European Symposium on Artificial Neural Networks*, pages 271–276.

[21] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech and time-series. *The handbook of brain theory and neural networks*, pages 255–258, 1998.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *IEE*, 1998.

[23] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *ICLR*, 2014. URL `http://arxiv.org/abs/1312.4400`.

[24] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach. *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 286–291, 2018.

[25] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach. *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 286–291, 2018.

[26] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018.

[27] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *CoRR*, abs/1606.02147, 2017.

[28] World Health Organisation. Global Status Report on Road Safety, 2015. URL `http://apps.who.int/iris/bitstream/10665/189242/1/9789241565066_eng.pdf?ua=1`. Visited 2018-11-26.

[29] Chaoyun Zhang, Rui Li, Woojin Kim, Daesub Yoon, and Paul Patras. Driver behavior recognition via interwoven deep convolutional neural nets with multi-stream inputs. *CoRR*, abs/1811.09128, 2018.