Learning Based Road Estimation

Sanna Amirijoo, Karl Niwong

Master's thesis
2023:E44

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Abstract

The interest in autonomous driving has vastly increased, leading to a surge in research and development efforts over the past decades. This technology could enhance road safety, alleviate traffic congestion, and yield numerous environmental and economic benefits. A fundamental prerequisite to developing and integrating autonomous driving is to obtain information about the surrounding environment, particularly in terms of detecting lane geometry. Precise lane detection can be achieved in a number of ways, including with the use of deep learning. This thesis investigates deep learning-based lane detection by developing models that predict the center of the ego lane. The problem is treated as a regression problem and is approached by developing, testing, and comparing several model architectures by using mean squared error as metric. Lane marker detection coordinates were used to generate bird's eye views (BEV) used as input to the different models. Four main models were developed. Model 1 was a convolutional neural network (CNN) and Model 2 was the CNN with a one-dimensional input fused to the feature map of the CNN. Model 3 incorporated a long short-term memory (LSTM) after Model 2 to make use of the temporal information in the data. Furthermore, Model 4 was created which had the same architecture as Model 3 but with another loss function. The experimental results demonstrated a consistent decrease in loss with the introduction of each subsequent model. The conclusions could be drawn that the selection of a metric used to train and evaluate networks was essential in the process of developing a relevant model. From the experience gathered during the thesis, a learning-based approach seemed to hold a lot of potential for estimating road geometry.

Acknowledgements

We gratefully acknowledge Zenseact for their collaboration on this master thesis project, which would not have been possible without their resources and support. The opportunity to visit their office in Gothenburg was invaluable and enabled us to complete this project from Lund. We are especially grateful to our industrial supervisors Junsheng Fu, Philip Granat, and Markus Eriksson, for their active dialogue, insightful contributions, and constant support that enabled us to move forward with the project. Finally, we thank Kalle Åström, our academic supervisor from the Center of Mathematical Sciences at the Faculty of Engineering at Lund University, for always being ready to help if needed.

Contents

1 Introduction

According to the World Health Organisation, approximately 1.3 million people die every year in road traffic accidents [1]. This is one of the reasons why the interest in autonomous driving has vastly increased in the past decades. With the recent developments of high-precision sensors, computer vision, and machine learning algorithms, road vehicles are being implemented with more advanced autonomous driving functionalities. The end goal often is to become fully automated or to enable accurate advanced driving assistance systems (ADAS) [2] [3]. This can result in increased road safety because of the reduction of human error by having constant vigilance and an improved reaction time. Human errors such as sudden braking or improper lane changes can also be reduced and therefore result in less congestion. These advantages would benefit the environment and the economy but also result in new behavioral patterns and inventions with carpools and infrastructure integration. Such technology might also benefit people currently unable to drive [4] [5].

One of the basic requirements for developing and integrating autonomous driving is to obtain information about the surrounding environment [3]. An accurate understanding of the road ahead is especially important in order for the vehicle to plan the route and prevent accidents [2]. It is also important in features such as lane-keeping assist systems, which keep the vehicle in the proper lane and avoid collisions, and autonomous emergency braking and steering, which prevent or reduce the damage in an accident [6]. These ADAS are usually vision based, which is often a good option because cameras provide rich and detailed information about the surrounding environment whilst being relatively cost-effective. As an example, lane detection may be trivial when the road has a well-defined geometry and is clear of obstacles. However, this is not always the case because of natural reasons such as shadows from vehicles or trees, bad road conditions, or poor weather [7].

Precise lane detection can be achieved in a number of ways. An example is the use of high-definition (HD) maps. However HD maps are currently expensive and difficult to implement, and challenges with outdated maps or road work scenarios are problems that need to be addressed [8]. Because of this, it is important to be able to detect the road in real-time by using the outputs from sensors such as cameras, LiDAR, and radar, mounted on the ego vehicle [9]. These sensors can for example be used to detect traffic, pedestrians, or traffic hazards but also to find where the lane markers are located relative to the car.

There are several different methods to estimate the road geometry. Some statistical methods use Hough transforms in combination with a Kalman filter [10]. The benefit of transforming the image into the Hough space is that straight lines can be detected using statistical probability, with a limitation that it requires significant time to obtain an accurate result [7]. Different implementations of random sample consensus (RANSAC) methods have also been used in lane detection tasks to remove outliers [11] [12]. A weak point for tracking methods with strong smoothness assumptions, e.g. Kalman filters, is that some methods assume a stable camera and a flat ground surface. Vibrations from the vehicle and changes in the terrain inject noise into the data, making abrupt changes to the time series, posing difficulties for these kinds of methods [13]. Another limitation with some statistical methods is that it can be challenging to add new inputs to the models because it may require a redefinition of the state variables and a change of the filter's equation [14]. With these limitations in mind, as well as the last decade's improvements, deep learning approaches have become more commonly used in the field of autonomous vehicles.

Deep learning involves training a neural network to recognize complex patterns in data. The advantages of neural networks include being able to detect complex nonlinear relationships between dependent and independent variables, as well as the availability of multiple training

algorithms [15]. Deep learning has also shown promising results for handling image data, where convolutional neural networks (CNNs) are commonly used [16]. In traditional image processing, feature engineering is often used which is a time-consuming process that is not required in deep learning [17]. Furthermore, deep learning can also be used to handle sequential data, such as video data, often with the use of recurrent neural networks (RNNs). With different combinations of CNNs and RNNs, various deep learning architectures have been explored for numerous autonomous driving solutions, including lane estimation, and so far the results have been promising [18].

1.1 Purpose

This thesis investigates learning-based road estimation by developing a model that predicts the center of the ego lane, which is the lane the vehicle is driving in. The problem is treated as a regression problem and is approached by developing, testing, and comparing several model architectures by using mean squared error as a metric.

1.2 Collaboration

This master thesis has been done in collaboration with the software company Zenseact, which develops solutions for autonomous driving and advanced driving assistance systems. Zenseact believes in automation as the solution to human errors in traffic. Currently, their solution aids the driver, but with a future goal of the driver assisting the vehicle [19].

2 Fundamentals

2.1 Artificial Neural Networks

A neural network consists of layers of connected nodes, starting with the input layer and ending with the output layer. In between these, there can be one or more hidden layers of nodes. An illustration of a neural network with two hidden layers can be seen in Figure 1, where each circle represents one node. The network is fully connected, meaning that each node in every layer is connected to all nodes in the next layer [16].

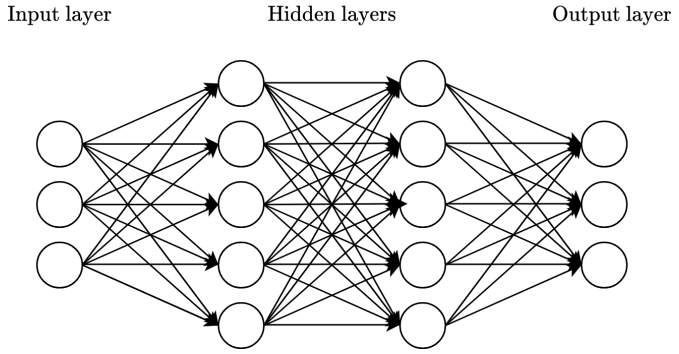


Figure 1: A fully connected neural network with an input and output layer, both with three nodes, and two hidden layers with five nodes.

2.1.1 Nodes

Nodes produce their outputs by applying an activation function to a weighted sum of their inputs. The main purpose of an activation function is to introduce non-linearity into the network computations, allowing the network to capture non-linear patterns, which are often found in real-world data. A node is illustrated in Figure 2.

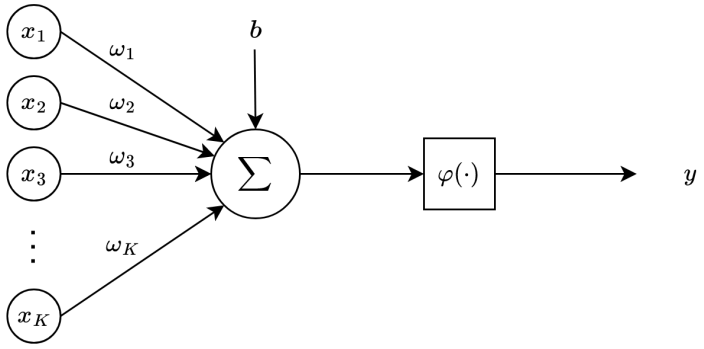


Figure 2: An artificial node. The node sums over the K inputs multiplied with their respective weights, adds the bias, and applies the activation function to produce its output.

The mathematical operations performed in a node can be formulated as,

$$y = \varphi\left(\sum_{k=1}^{K} \omega_k x_k + b\right),$$

where y is the output, φ is the activation function, ω_k is the weight corresponding to the input x_k, and b is the bias. A bias introduces a shift in the activation function to allow for more accurate predictions.

The goal when training a network is for the trainable parameters, which are the weights and the biases, to be chosen by the network such that the network performs sufficiently well on the given task [16].

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are commonly used for image analysis because they are designed to handle spatial relations of the input. The network uses a mathematical linear operation called convolution which is usually denoted with an asterisk,

$$h(t) = (f * g)(t) = \int f(\tau)g(t-\tau)d\tau,$$

where f and g represent the input and the kernel respectively. h is called the feature map. If it is assumed that t can only take on integer values and that f and g are only defined for integer t, the discrete convolution can be defined as,

$$h(t) = \sum f(\tau)g(t-\tau).$$

In machine learning applications, the input and the kernel are usually multidimensional and the convolution is performed over several axes. Using a two dimensional kernel, K, of size $m \times n$ and a two dimensional input image, I, a convolution for pixel (i, j) of the input can be written as,

$$H(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

Every node in each feature map is thus a linear combination of nodes in the previous layer within the receptive field, where the receptive field is defined by the kernel.

In PyTorch, a machine learning framework in Python, which was used for this master thesis, the kernel is redefined such that a convolution is instead implemented as a discrete cross-correlation,

$$\tilde{H}(i,j) = (\tilde{K} * I)(i,j) = \sum_m \sum_n I(i+m, j+n)\tilde{K}(m,n).$$

Using cross-correlation instead of convolution changes the values learned in the network but does not affect the actual result. It is implemented for efficiency. In this thesis, a discrete cross-correlation will be called a convolution in accordance with related work and literature in the area of machine learning [16] [20] [21] [22].

2.2.1 Filters

The combination of a convolution result and an activation function is called a filter. Furthermore, an input may have multiple channels and the filter will have the same depth as the number

of channels. The operations of a filter, F, with multiple channels on a pixel (i, j) can be mathematically represented as,

$$F(i,j) = \varphi\left(b + \sum_c \sum_m \sum_n I_c(i+m, j+n)\tilde{K}(m,n,c)\right),$$

where b is a bias, c iterates over the number of channels and I_c represents the corresponding input channel. The number of filters applied to the input image equals the number of channels in the output. This is illustrated with an example in Figure 3 where four filters of depth two are applied to an input, also of depth two, to produce an output of depth four [16].

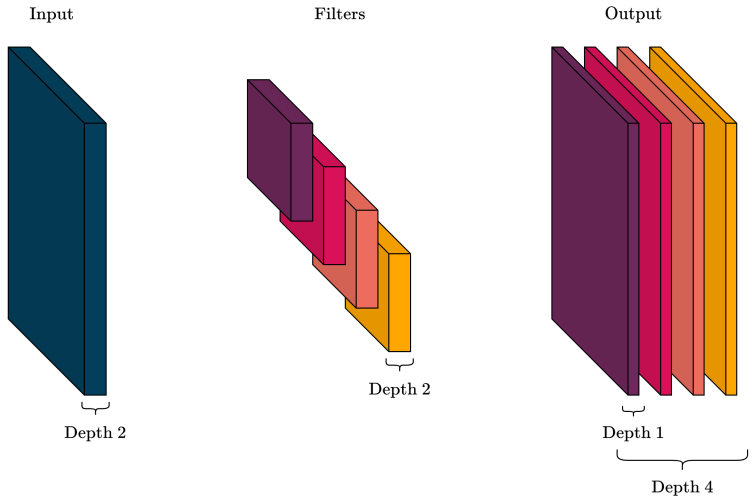


Figure 3: Four filters, with the same depth as the input, produce an output of depth four.

2.2.2 Max-pooling

A max-pooling reports the maximum value within a rectangular neighborhood and is used with a set stride over the input. It is applied to all channels independently. Thus, it does not change the number of channels [16].

2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are neural networks designed to deal with sequential data. Because of this, RNNs can be used in processing speech data, time series data, and text data. An RNN contains feedback connections such that the output from a node can affect the next input to the same node, thus allowing information to persist. Figure 4 shows a simple RNN consisting of one node with a feedback connection.

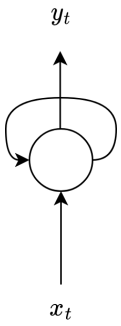


Figure 4: A node with a feedback connection to itself.

In Figure 5 the node in Figure 4 has been unfolded, and it can more clearly be seen why the structure of an RNN is suitable to handle data that comes in sequences. However, in practice, RNNs cannot seem to learn long-term dependencies because of the vanishing gradient problem. This occurs when the weights of the network are no longer updated effectively based on the earlier inputs in the sequence, meaning that the network forgets information from earlier time steps [16].

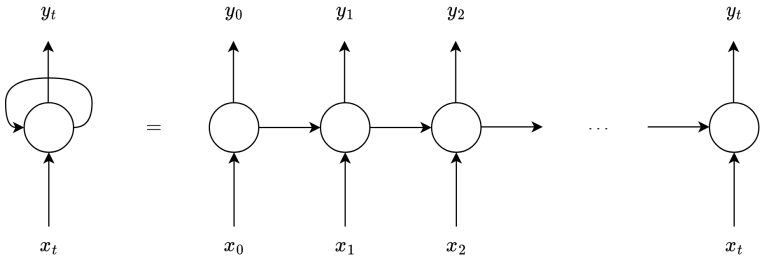


Figure 5: A node with a feedback connection that has been unfolded in time.

2.3.1 Long Short Term Memory

Long short-term memory (LSTM) networks are a subset of RNNs designed to handle long sequences of data. An important aspect of the LSTM is the cell state, C_t, which is the top horizontal line seen in Figure 6.

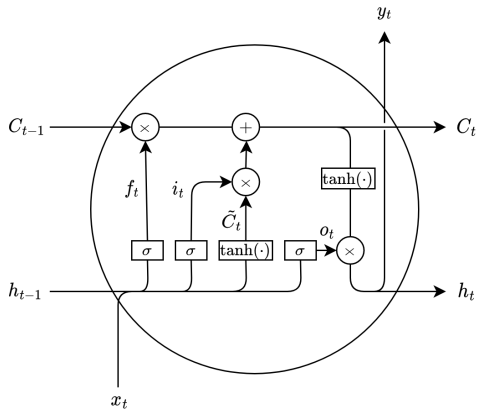


Figure 6: An LSTM cell. It consists of interconnected gates f_t, i_t, and o_t that control the flow of information into and out of the cell.

Information can be added to or removed from the cell state by three structures called gates. They all use the sigmoid function, $\sigma(\cdot)$, which outputs a value between 0 and 1, where 0 means removing all of the information and 1 means keeping all of the information. There is the forget gate, f_t, the input gate, i_t, and the output gate, o_t,

$$\begin{aligned} f_t &= \sigma\left(W_f h_{t-1} + U_f x_t + b_f\right) \\ i_t &= \sigma\left(W_i h_{t-1} + U_i x_t + b_i\right) \\ o_t &= \sigma\left(W_o h_{t-1} + U_o x_t + b_o\right), \end{aligned}$$

where h_{t-1} is the hidden state at time $t-1$ and x_t is the input at time t. b_f, b_i, and b_o are the biases, W_f, W_i and W_o are the weights for the hidden state, and U_f, U_i and U_o are the weights for the input, where the subindices denote the forget, input and output gate.

The information that is to be removed from the cell state is decided by the forget gate. The input gate decides what information in the cell state is to be updated and $\tilde{C}_t$ contains new candidate values that could be added to the cell state,

$$\tilde{C}_t = \tanh\left(W_C h_{t-1} + U_C x_t + b_C\right),$$

where W_C and U_C are the weights for the hidden state and input respectively, and b_C, is the bias. The updated cell state is now given by,

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t,$$

where $\odot$ is the Hadamard product.

The output gate, which is a filtered version of the cell state, decides the final output together with $\tanh(\cdot)$,

$$h_t = o_t \odot \tanh\left(C_t\right).$$

The above equations are formulated for a single LSTM node. In practice, many LSTM nodes are used in a network, meaning that the weights used in the above equations should be replaced by matrices and the gates should be replaced by vectors of values between 0 and 1. Figure 7 shows a network with two layers and hidden size $t+1$. The number of layers describes how many LSTMs are stacked on top of one another, meaning that the input to the second layer is

the output from the first, and the hidden size is the number of features in each layer [16] [23] [24].

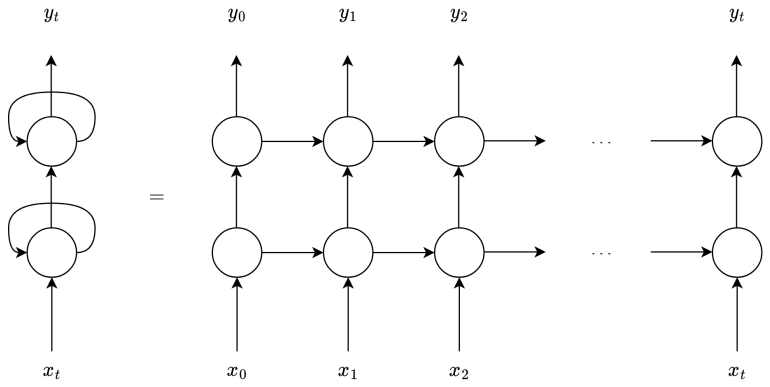


Figure 7: An unfolded RNN network with two hidden layers and hidden size $t + 1$.

2.4 Training Neural Networks

During the training of a neural network, the weights and biases are adjusted to minimize a chosen loss function. Usually, the available data is divided into a training set, a validation set, and a test set. The training set is used for training different networks, whereas the validation and test set are used for evaluating the networks' performances. The validation set is used to compare the performance of networks, and the test set should only be used once, and not until a final network has been obtained. This is to obtain an unbiased evaluation result. After the test set has been used it would be misleading to continue to make adjustments to the network [16].

If the training set distributions do not match the validation and test set distribution it can lead to the network getting a higher loss for unseen data. This is called domain mismatch and can occur because the model learns to recognize patterns and make predictions based on the features that are present in the training domain. Hence, by having data sets with similar distribution, the model can usually perform better for unseen data but also gets evaluated on a wide set of data [25].

2.4.1 Loss Function

A loss function, E, is used to measure how well a network models the data. For regression problems, a commonly used loss function is the mean squared error (MSE),

$$E(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} (y_n - d_n)^2 ,$$

where N is the number of samples in the data set, y represents the output of the network and d is the corresponding target. $\boldsymbol{\theta}$ is all the weights and biases that the output is dependent on [16] [26].

2.4.2 Gradient Descent

An algorithm for minimizing the loss function is called gradient descent. It can be roughly explained as computing the gradient of the loss function for randomly initialized weights and biases, taking a small step in the direction of the gradient, and then repeating until a minimum is reached.

A change of the loss function, ΔE, with respect to small changes in the weights and biases can be written,

$$\Delta E \approx \frac{\partial E}{\partial \theta_1}\Delta\theta_1 + \frac{\partial E}{\partial \theta_2}\Delta\theta_2 + \ldots + \frac{\partial E}{\partial \theta_p}\Delta\theta_p, \tag{1}$$

where θ_i is either a weight or a bias and p is the total number of trainable parameters in the network. The goal is to find parameters such that ΔE becomes negative.

Let a vector of changes in the trainable parameters, $\Delta\boldsymbol{\theta}$, and the gradient of the loss function, ∇E, be defined as,

$$\Delta\boldsymbol{\theta} \equiv (\Delta\theta_1, \Delta\theta_2, \ldots, \Delta\theta_p)^T \tag{2}$$

$$\nabla E \equiv \left(\frac{\partial E}{\partial \theta_1}, \frac{\partial E}{\partial \theta_2}, \ldots \frac{\partial E}{\partial \theta_p}\right). \tag{3}$$

Using Equations (2) and (3), Equation (1) can be written,

$$\Delta E \approx \nabla E \cdot \Delta\boldsymbol{\theta}. \tag{4}$$

To repeat, the goal is to find a change in the parameters, $\Delta\boldsymbol{\theta}$, such that the change in the loss, ΔE, is negative. So choosing,

$$\Delta\boldsymbol{\theta} = -\eta\nabla E, \tag{5}$$

where η is the learning rate, which is a small positive parameter, Equation (4) can be written as,

$$\Delta E \approx \nabla E \cdot (-\eta\nabla E) = -\eta||\nabla E||^2.$$

Since $||\nabla E||^2 \geq 0$, it implies that ΔE will always decrease given that $\boldsymbol{\theta}$ is chosen as described in (5), and taking into account the limitation of the approximation in Equation (4).

To summarize the gradient descent algorithm, the gradient of the loss is calculated and the trainable parameters are updated in the negative direction of the gradient such that the loss decreases until a local or global minimum is reached [27].

2.4.3 Stochastic Gradient Decent

Stochastic gradient descent (SGD) is a way of speeding up the training of a network. The idea is to estimate the gradient by computing the gradient of a small, randomly chosen subset of the training samples instead of computing it for the whole set. This subset is called a mini-batch.

The approximation of the gradient is computed as,

$$\nabla E \approx \frac{1}{M}\sum_{n=1}^{M} \nabla\left((y_n - d_n)^2\right),$$

where M is the number of training samples in the mini-batch.

After the gradient is computed for a mini-batch, the weights and biases are updated in accordance with Equation (5). Then another randomly chosen mini-batch is used to compute the gradient until all training samples have been used. When all training samples have been used, one training epoch has been completed. This is repeated for a set number of epochs.

The speedup happens because instead of applying one gradient and updating the weights one time, N/M gradients are applied and the weights are updated N/M times. SGD is thus a factor N/M faster than conventional gradient descent. In this way, the network can reach similar performance by training for fewer epochs [27].

2.4.4 Adaptive Moment Estimation

Adaptive moment estimation (Adam) optimization is an extension of SGD. It is efficient when working with a lot of data or parameters and requires less memory.

The algorithm works by also considering the past gradients when computing the weight and bias updates for the network,

$$\begin{aligned} \boldsymbol{m}(t+1) &= \beta_1 \boldsymbol{m}(t) + (1-\beta_1)\,\nabla E(t) \\ \boldsymbol{v}(t+1) &= \beta_2 \boldsymbol{v}(t) + (1-\beta_2)\,(\nabla E(t))^2\,, \end{aligned}$$

where β_1 and β_2 are two tuneable parameters, $\boldsymbol{m}(t)$ is the weighted average of past gradients for all parameters at time t and $\boldsymbol{v}(t)$ is the weighted average of squared past gradients for all parameters at time t. With,

$$\begin{aligned} \hat{\boldsymbol{m}}(t+1) &= \frac{\boldsymbol{m}(t+1)}{1-\beta_1^t} \\ \hat{\boldsymbol{v}}(t+1) &= \frac{\boldsymbol{v}(t+1)}{1-\beta_2^t}, \end{aligned}$$

and using Equation (5), the trainable parameters can be updated as,

$$\Delta\boldsymbol{\theta}(t+1) = -\eta \frac{\hat{\boldsymbol{m}}(t+1)}{\sqrt{\hat{\boldsymbol{v}}(t+1)} + \epsilon},$$

where ϵ is a small number to avoid division by zero [16].

2.4.5 Generalization, Overfitting, and Underfitting

A model that can generalize well is able to properly adapt to previously unseen data drawn from the same distribution as the one used to train the model. A model's capability to generalize can be improved by avoiding overfitting during training. Overfitting occurs when the network conforms in too much detail to the training data. When this happens the training loss continues to decrease whilst the validation loss stagnates or increases. Underfitting is the opposite of overfitting and implies that the model has not yet learned enough and the loss is therefore not as low as it could be [28].

In the first epochs seen in Figure 8, the model is underfitted and is still learning, which can be seen by the negative slope of the losses. For later epochs, the network starts to overfit to the training data such that the validation loss starts to increase. The ideal model for the given hyperparameters during the training process can be found when the validation loss is at its minimum, which in the figure is approximately at epoch 25.

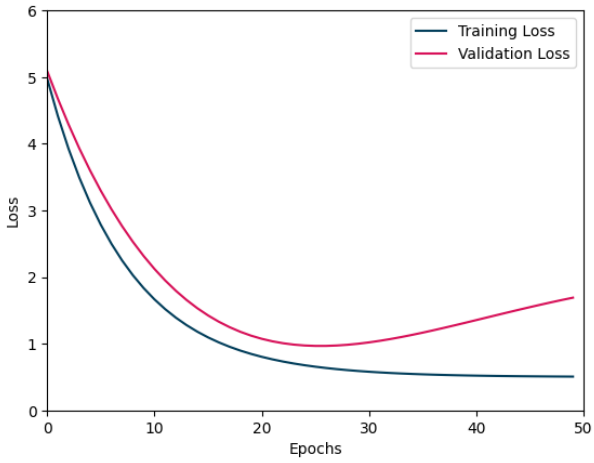


Figure 8: A theoretical learning curve that showcases both underfitting and overfitting to the training data.

2.4.6 L^2 Norm Regularization

L^2 regularisation is a commonly used technique in machine learning to prevent overfitting. The technique involves adding a penalty term to the cost function that the model is being optimized to minimize. This penalty term is proportional to the square of the magnitude of the weights in the model. The effect of the penalty is to penalize bigger weights and therefore yields a model with smaller weights, which can help reduce overfitting by making the model more robust [29] [16].

In essence, L^2 regularisation adds a constraint to the optimization problem, which encourages the model to find a solution that balances between fitting the data well and having small weights. The regularisation term is given by,

$$\Omega = \lambda \sum_{k=1}^{K} \omega_k^2,$$

where K is the number of weights in the network. The trade-off is controlled by a hyperparameter, λ, which determines the strength of the regularisation penalty. Larger values of λ result in stronger regularisation and smaller weights, while for smaller values the opposite is true [16].

2.4.7 Dropout

Dropout is another popular regularization technique in machine learning. It is designed to prevent overfitting by randomly deactivating a set fraction of nodes during training. By introducing dropout regularization, the network becomes more robust and less reliant on specific nodes, leading to improved generalization. [30]

2.4.8 Batch Normalization

A problem that can occur while training a neural network is internal covariance shifting, which refers to the distribution of each layer's input changes during the training when the parameters

get modified. To address this problem, one can normalize the layer inputs as,

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}}\gamma + \beta.$$

The normalization in every layer is done for each training mini-batch. Batch normalization allows for less careful parameter initialization and can also remove the need for dropout. Higher learning rates can also be used, thus speeding up the training process [31].

2.5 Random Sample Consensus

Random sample consensus (RANSAC) is a method to remove outliers. The idea of RANSAC is that if the number of outliers is small, there is a high probability to pick an outlier-free set. An outline of the algorithm can be seen in Algorithm 1 [32].

Algorithm 1 RANSAC Algorithm

1: Randomly select a small subset of the data points.
2: Fit a model to the selected points.
3: For the model, count the number of points that are inbound a predefined threshold.
4: Repeat step 1-3 several times and select the model with the most inliers or the smallest error.

The probability of randomly selecting a set of inliers depends on the proportion of the inliers and the size of the set. If the probability of selecting an inlier is x and the number of points in the set is z. Then the number of iterations, n, needed for the RANSAC algorithm in order to find a set with only inliers with a probability of p, can be calculated as follows.

Firstly, it is assumed that the number of points is large, such that the portion of inliers and outliers do not change when removing a point. The probability of selecting an inlier set can be calculated to x^z. This yields a probability of failing to select an inlier set to $1 - x^z$. To find an inlier set at least once is the complement of failing to find an inlier set all n times. The probability of failing n times is $(1 - x^z)^n$. With this in mind, the number of iterations, n, needed in order to have a probability p to have found at least one set of only inliers can be calculated as,

$$(1 - x^z)^n \leq 1 - p \Rightarrow n \log(1 - x^z) \leq \log(1 - p) \Rightarrow n \geq \frac{\log(1 - p)}{\log(1 - x^z)}. \tag{6}$$

Because of noise, not all inlier sets work equally for estimating the solution. So in practice, it is good to run more iterations than above [32].

3 Related Work

Examples of past successful CNN architectures which have been shown to generalize well to other data sets include U-Net, SegNet, and VGG [33] [34] [35]. U-Net and SegNet are encoder-decoder networks developed for semantic segmentation, which is the classification of every pixel in an input image [33] [34]. VGG was developed in order to investigate the effect of depth on the accuracy of the network. The network was characterized by its simplicity as it used 3×3 filters in 13 convolutional layers, some layers had 2×2 max pooling. The network had 3 fully connected layers before the output. All layers had the rectified linear unit (ReLU) as activation function except for the last one which had softmax [35].

Since CNNs are suitable for handling images, they have been used for road estimation with image inputs [7] [36]. One such paper took a single image, taken from a camera mounted on the front of the ego vehicle as input, and outputted a set of 3D curves as lane descriptors. This was done without assuming a constant lane width or the height or pitch of the camera. The network was evaluated at a set of anchor points, meaning predefined evaluation distances from the car [36]. They also used inverse perspective mapping (IPM) which has been used for lane detection before [37] [11]. The IPM transforms the captured images in the camera perspective to a bird's-eye view (BEV) and enabled mapping between pixels in the image plane and world coordinates [11].

BEV is a 2D representation of information. It can be used to give an intuitive understanding of a surrounding scene or to fuse different kinds of data into a unified representation. It has several times been used in autonomous driving tasks [38] [39] [40] [41] [42] [43]. Furthermore, a BEV representation can also be used to simplify perception problems since it provides a consistent representation of the data [44]. For autonomous driving, the sensors capturing information about the environment do so from the ego vehicle and are often directed parallel to an assumed flat ground. Because of this, the information is captured in a perspective view that is perpendicular to the plane of the BEV. The transformation between the two views is ill-posed and therefore, using BEV in autonomous driving is often a challenge [39].

Many lane detection methods have focused on single images and have thus performed badly on images containing non-ideal conditions, such as heavy shadow or vehicle occlusion. Since lanes are continuous structures, using information from previous frames may improve lane detection in the current frame. To make use of the temporal information, papers have used LSTMs [2] [45] [46].

One paper did semantic segmentation and used an encoder-decoder network structure for the CNN to ensure that the output had the same size as the input. In between the encoder-decoder, they used an LSTM network to recursively predict the lane. In this way, they created an end-to-end trainable network that combined the benefits of using CNNs on images with the time series properties of RNNs. Their proposed networks performed better on almost all their metrics compared to baseline architectures and other networks, which were different variations of U-Net and SegNet [2].

In another paper, a CNN was implemented to detect the lanes. Then an expectation line was constructed and lastly an LSTM was applied to predict the future trajectory of the autonomous vehicle. The two networks were thus trained separately. The results verified the effectiveness of the proposed method [45].

In order to be able to use data that was not optimal to represent as an image, like single values for a frame, the authors created a CNN that took an input image and then concatenated single value parameters, such as BMI, age, and gender, with the image data when the spatial resolution of the feature map was 1 pixel [47].

Other modifications to more conventional CNN architectures have also been tested. Tradi-

tional CNN architectures use square kernels, but it has been found that unconventional kernel shapes can result in improved performance for classification problems compared to square convolution kernels alone [22]. Furthermore, papers have suggested using non-square kernels designed to work with the geometry of the input data [48] [49].

In order for the network to learn, an optimizer needs to be used. Adam is a common optimizer and has previously been used in lane detection tasks [46] [2] [50]. There are suggestions that Adam leads to worse generalization than SGD with momentum [51]. When using adaptive gradient algorithms, L2 regularization, and weight decay are not equivalent. However, by decoupling the weight decay from the optimization steps the original formulation of weight decay is recovered. There has been empirical evidence that the generalization performance improved on image classification tasks compared to only using Adams [29].

4 Method

4.1 Data and Data Preprocessing

The data was collected by special vehicles and saved in logs. Figure 9 shows a picture of a vehicle that collects different kinds of data, such as camera and LiDAR data, which can be used to develop driving-assisted systems in normal cars [52]. It also collects data that can be modified offline to ground truth, GT, used in the company for the development of such systems. The logs used in the project were 30 second splits of longer logs and ordered from when the logs were recorded.

Figure 9: One of Zenseact's data collecting vehicle [53].

The main data for the project consisted of coordinates, (x, y, z), for detected lane markings from highways. These lane markings were detected by another team. For each frame, they output several detections in the ego vehicle's local coordinates. Figure 10 shows the coordinate system which had the origin in the middle of the rear axis of the ego vehicle and the x-axis in the heading direction of the car. The y-axis was an extension of the car's rear axis with a positive direction left of the ego vehicle. Given a positive right-oriented coordinate system, the z-axis pointed up relative to the ego vehicle.

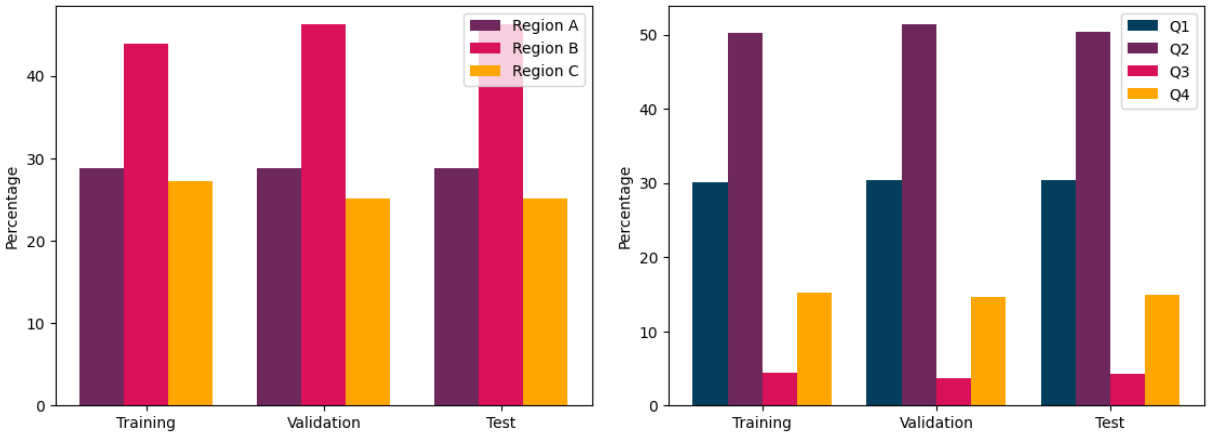
Figure 10: The orientation of the local coordinate system for the ego vehicle with the origin in the middle of the rear axis.

The GT was given in global coordinates and needed to be transformed into the local coordinate system. This was done by extracting the ego vehicle's position and orientation in global coordinates to get a reference point and orientation of the coordinate system. Converting the GT to the local coordinate system was then done by calculating the translation vector between the ego vehicle and the GT points, which represented the displacement of GT relative to the

ego vehicle. To align the local coordinate system, a rotation transformation was applied to the translation vector. This rotation was given by the negative of the ego vehicle's heading angle and yielded that the GT was adjusted to the ego vehicle's perspective, and thereby transformed to local coordinates.

4.1.1 Data Distribution

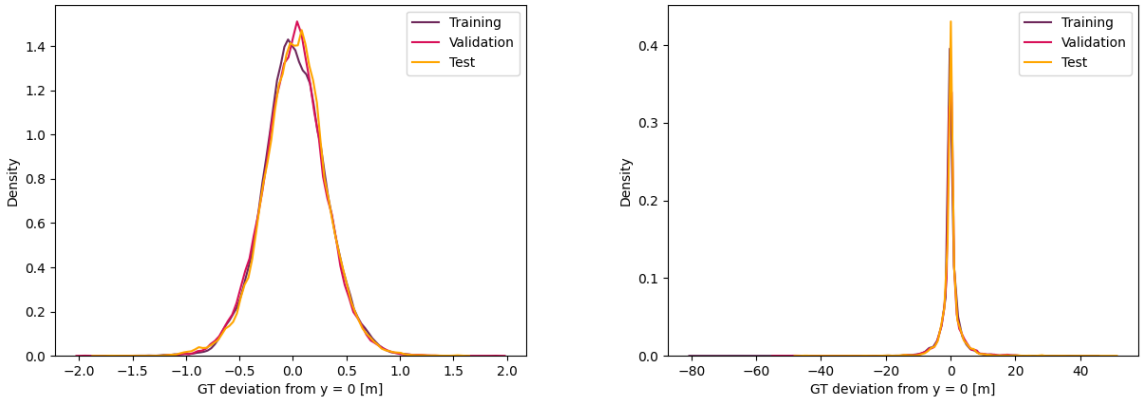
To get an even distribution of data in the training, validation, and test set of where and when the logs were recorded, the first two logs were sent to the training set, the third to validation and the fourth to test, and so on until all logs had been assigned to a set. The distribution of when and where the logs were recorded for each of the sets can be seen in Figure 11. It can be seen that the distributions are similar for the three data sets for these metrics.



(a) **(b)**

Figure 11: The distribution over the location (a) and the time divided in quarters (b) for the split of data sets.

The distribution in deviation from $y = 0$ of the GT points for the anchor point at 0 and 100 meters away from the car can be seen in Figure 12 and shows that they were similar for the three data sets. The distribution for the other anchor points can be seen in the Appendix.



(a) **(b)**

Figure 12: The distribution in deviation from $y = 0$ of the GT points 0 meters (a) and 100 meters (b) ahead of the vehicle.

The variance of the GT in the anchor points can be seen in Figure 13.

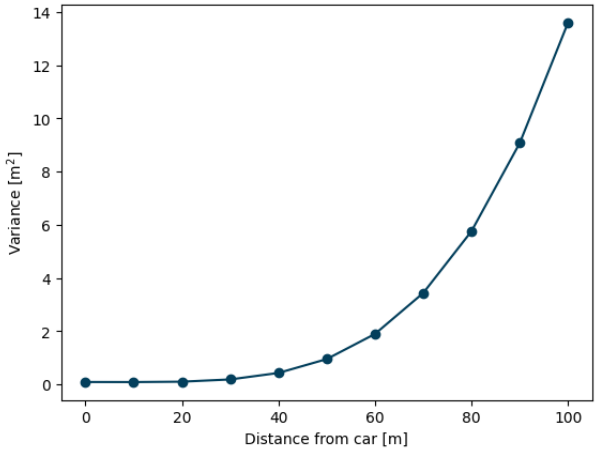


Figure 13: The variance of the GT at the anchor points for all data.

4.1.2 Input

The number of detections in each frame was not constant because of factors such as traffic and weather conditions. Therefore, to represent the data in an understandable way, BEVs were generated. The BEV approach assumed a flat surface and created a grid in local coordinates in front of the car, where each cell of the grid represented an area. If that area had at least one detected lane marking in it, the cell got the value 1, otherwise 0. The BEV used had a cell size of 0.2 m $\times$ 0.2 m and dimension 20 m $\times$ 120 m. The dimension of the BEVs were chosen after doing an analysis of the data, where most of the lane marker detections were in the range of the BEV and it covered all the anchor points. A binary BEV for a frame can be seen in Figure 14.

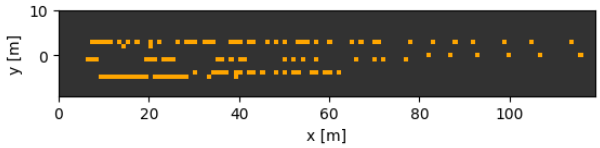


Figure 14: A binary BEV for a frame in local coordinates. Each cell in the grid has the size 1 m $\times$ 1 m for clearer visualization. The cells with at least one detection in them were marked orange, forming a two-lane road.

Since the number of grid cells was dependent on how big of an area was covered by the BEV, as well as how fine the grid was, two more BEVs were created for each frame. These BEVs were also initialized to all zeroes, but every cell that had detections was filled with the mean deviation from the middle of the cell. One BEV for the deviation in the x-direction and one for the deviation in the y-direction. The addition of the BEVs with deviation in x- and y-direction also made BEVs with different cell sizes more comparable, which was interesting because fewer cells would be computationally more efficient.

Another data point used as input to the network was the median of the heading angle of detected vehicles. For the vehicles detected on the opposite side of the road, the angle was rotated π radians before determining the median.

Data that was not included was frames before a lane change. More specifically, all frames where the lane change was 100 m or less in front of the ego vehicle were removed. This was necessary because the GT was only recorded in the ego lane, and therefore the targets would be in the wrong lane if the frames before a lane change were not removed. A removed frame can be seen in Figure 15 where the data collection vehicle changed lanes approximately 60 meters ahead.

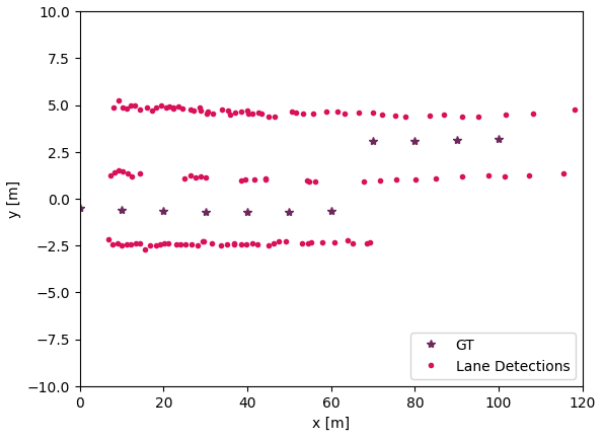


Figure 15: An invalid frame where the GT was not in the ego lane 100 meters ahead of the ego vehicle.

4.2 Baselines

A baseline model is used as a sanity check and as a first comparison. It does not matter how complicated the developed model is, if it cannot beat the baseline it is definitely not a good

model. Usually, a baseline is simple to interpret and implement, creating an understandable reference for the development of new models.

4.2.1 Grid Baseline

With the BEV as an inspiration, an initial baseline was created. The grid baseline was based on a grid around the 11 evaluation distances, as can be seen in Figure 16. For every evaluation distance, there were 2 rectangular areas. The upper rectangle had its limits at $x = \pm 5$, $y = 0$, and $y = 3$. The lower rectangle had the same limits multiplied with -1, generating a symmetric grid reflected in $y = 0$ for every evaluation distance. These limits were chosen because, in the x-direction all relevant lane marker detections would be used and, in the y-direction all detections for the closest lane markers to the left and right would be covered but not the ones in the next lanes. For all lane marker detections in each rectangle, the mean of the y-coordinate was calculated. For a case where there were lane marker detections in the area above and below the evaluation distance, the mean of the y-coordinates was calculated for each rectangular area before calculating the mean between the two values to get a result of where the lane center was estimated to be located at the anchor point. However, if no lane marker detections were in a certain area for a specific anchor point the lane center was estimated by the lane center estimation of the anchor point closer to the car. If there were no anchor points closer to the car, the estimation was that the car was driving in the lane center, i.e. $y = 0$.

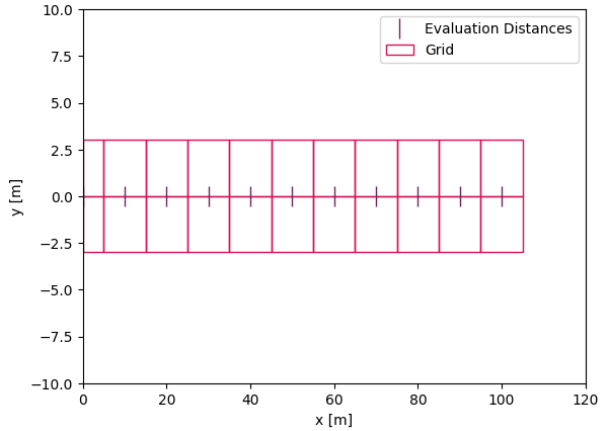


Figure 16: The evaluation distances and the grid for the grid baseline.

4.2.2 RANSAC Baseline

An additional baseline was created. It was a modified RANSAC method that estimated the lane markers closest to the left and right of the ego vehicle which would result in the final baseline, see Algorithm 2. The RANSAC was modified so the fitted lines' intercepts were in certain intervals, making it likely to find the ego lane's lane markers. These intervals were chosen to $[-3, 0]$ for the lower estimate and $[0, 3]$ for the upper estimate with the same argument for the boundaries as for the grid baseline limits in the y-direction.

Algorithm 2 Modified RANSAC Algorithm used as Baseline

1: Randomly select a small subset of the data points.
2: Fit a straight line to the selected points.
3: If the intercept of the line is within the wanted interval, continue. Otherwise, repeat step 2 until a line with the wanted intercept is found.
4: For the fitted line, count the number of points that are inbound a predefined threshold.
5: Repeat step 1-4 several times and select the line with the most inliers.

A rough estimate for the number of iterations needed for the RANSAC method can be calculated from Equation (6). Given 2 lanes, one third of the lane markers would be inliers for a line estimate using the RANSAC baseline, yielding $x \approx 0.3$. To fit a straight line, $z = 2$ points are needed, and with a probability $p = 0.95$, Equation (6) yields an approximation of $n = 26$. However, because of noise, different number of lanes, and the fact that not all inlier sets worked equally well for estimating the lane marker, n was set to 300. For each iteration, the algorithm had 100 chances of fitting a line with an intercept within the desired range.

In an edge case where the RANSAC algorithm did not fit an estimate for one of the lane markers, the lane marker was approximated to be parallel to the other fitted estimate with the car driving in the middle of the lane. If there were no detections, or so few that the RANSAC did not find an estimate for any lane, the middle of the lane was estimated at $y = 0$ for all anchor points.

Different modifications of the RANSAC method were also tried, one included trying to fit several lines to the data points and then find the two closest to the car. The problem with that modification was that it was not trivial to determine the number of lines that should be fitted for each frame and the computation time was higher.

4.3 Network Architecture

4.3.1 Model 1: CNN

Several different CNN architectures were tested. The ones that performed the best were encoder networks. Model 1 was inspired by VGG and the architecture in paper [47]. Both of these networks were designed to handle classification problems but the difference between networks designed for classification and regression is a change in loss function and activation function after the last convolutional layer.

The architecture of Model 1, which had $\sim$ 13 890 000 trainable parameters, can be seen in Table 1.

Table 1: The architecture of Model 1 with $\sim 13\ 890\ 000$ trainable parameters. Every convolutional layer consisted of a convolution, a batch normalization, with $\beta = 0$, $\gamma = 1$ and $\epsilon = 10^{-5}$, and a ReLu activation function.

Layer	Kernel Size	Stride	Padding	Channels	Output Shape
Convolutional 1	5×25	1	0	64	96×576
Convolutional 2	25×5	1	0	64	72×572
MaxPool 1	2×2	2	0	64	36×286
Convolutional 3	5×5	1	1	128	34×284
Convolutional 4	5×5	1	1	128	23×282
MaxPool 2	2×2	2	0	128	16×141
Convolutional 5	5×5	1	1	256	14×139
Convolutional 6	5×5	1	1	256	12×137
Convolutional 7	5×5	1	1	256	10×135
MaxPool 3	2×2	2	0	256	5×67
Convolutional 8	3×3	1	1	512	5×67
Convolutional 9	3×3	1	1	512	5×67
Convolutional 10	3×3	1	1	512	5×67
MaxPool 4	2×2	2	0	512	2×33
Convolutional 11	3×3	1	1	512	2×33
Fully Connected	-	-	-	-	1×11

In accordance with the findings about using non-square kernels mentioned in Related Work, the kernels in the first and second layer of the CNN in Model 1, Model 2, and Model 3 were chosen to have the shapes 5×25 and 25×5 respectively. The size and shape of the kernels were also chosen in regard to the restriction of the number of trainable parameters as well as the size of the input image.

A grid search was done to determine the mini-batch size and learning rate. The final learning rate and mini-batch size were 0.002 and 200 respectively. Taking into consideration the computation time and the memory limits of the computer, but also wanting to reduce both the time and memory consumption, cell sizes for 0.2 m $\times$ 0.2 m, 0.3 m $\times$ 0.2 m, 0.4 m $\times$ 0.4 m and 0.5 m $\times$ 0.5 m were tested. The different cell sizes were tested for a network similar to the one above except for the kernel sizes. The kernel sizes for the two first convolutional layers were 5×15 and 15×5 respectively, and the kernel sizes of the remaining layers were 3×3. Moving forward, cell size 0.2 m $\times$ 0.2 m was chosen and the final network did not allow for a higher cell size than 0.2 m $\times$ 0.2 m. MSE was used as loss function and Adam was used as optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$ and $\epsilon = 10^{-8}$. A high variance of results was observed when training the network for different seeds so L^2 regularisation was used to lower the variance. After testing different weight decays, L^2 regularisation with weight decay 10^{-6} was used. All models were trained for 30 epochs and during the training process, the model with the lowest validation loss was saved.

4.3.2 Model 1 with Weight Decoupling

Inspired by the paper [29], decoupling the weight decay from the Adam optimizer was tested. This was done by using a prebuilt function in Pytorch's library and for several different combinations of hyperparameters including the same as when no weight decoupling was used.

4.3.3 Model 2: CNN + One-Dimensional Input

Some inputs are not well represented as an image, such as single values for a frame. In the provided data, detected vehicles were attached with their coordinates and heading angle. These heading angles were used to create a one-dimensional input to concatenate with the encoded images later in the CNN. Two inputs for each frame were created, a binary input describing if there were any detected vehicles in the range of the BEV, and the median of the detected vehicles' heading angle within the range of the BEV. The binary input was added to be able to distinguish between a 0 because of no detected vehicles and a 0 because of a car that is driving with that heading angle. Vehicles' heading angles detected going in the other direction were rotated π radians before determining the median.

The median of the heading angle of the detected vehicles and the binary input was represented as a 1×1 image of 2 channels and concatenated after the feature map from the last convolutional layer had been flattened. After this, there was one fully connected layer to produce the output. The architecture of Model 2 is shown in Figure 17 where it can be seen that the number of channels increases with two when the one-dimensional input has been concatenated. The number of trainable parameters was $\sim 13\ 890\ 000$.

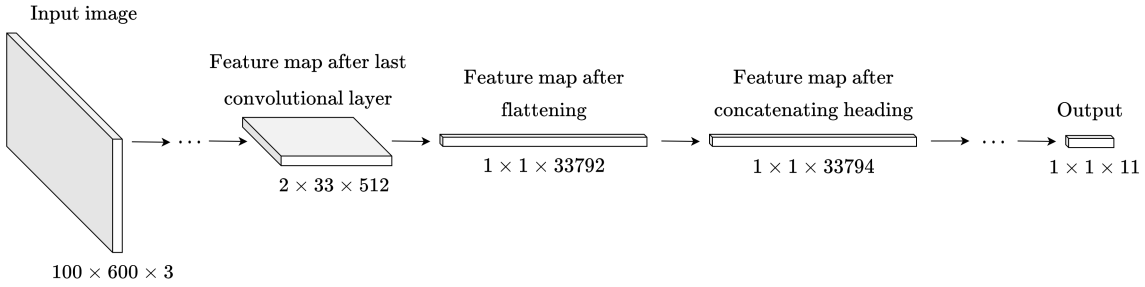


Figure 17: The modification of Model 1 resulting in Model 2. The modification was fusing single value data when the spatial resolution of the feature map in the CNN was 1.

A grid search was done to test different learning rates and batch sizes and were found to be the same as for Model 1. L^2 regularization was used as well but was increased to 10^{-4} to obtain the best validation loss.

4.3.4 Model 3: CNN + One-Dimensional Input + LSTM

To make use of the temporal information of the data, an LSTM was implemented after Model 2 such that the output from the CNN was used as input to the LSTM. Several different network architectures were tested, including 1, 2, and 3 number of layers and hidden sizes 200, 300, 400, and 500. For the most promising networks, 1, 2, and 3 fully connected layers with different combinations of ReLU and Tanh as activation functions were also tested. During the training of the network, sequence lengths of 20, 50, and 80 were tested with different combinations of learning rates and mini-batch sizes. Furthermore, L^2 regularization was used with weight decay 10^{-6}, as well as dropout with a dropout probability of 0.2 between the two fully connected layers.

The final network had an additional $\sim 506\ 000$ trainable parameters because of the LSTM. The LSTM had two layers and a hidden size of 200. It used two fully connected layers with ReLU before the first and Tanh before the last. The network was trained using Adam and MSE loss with batch size 100 learning rate 0.005 and sequence length 50. Similar to the CNN, the

LSTM outputted a vector containing the deviation in y-direction measured in meters. In the cases where the sequence length was not fulfilled, for example, at the beginning of a file, the output of the CNN was used as output from Model 3. The sequence length was reset for each file because of the splitting of the data. This splitting led to the logs being fed into the network were not necessarily consecutive.

Furthermore, experiments combining the CNN from Model 2 and different LSTMs into one trainable network were also conducted.

4.3.5 Model 3 as one Trainable Network

The CNN and LSTM in Model 3 were trained separately. Because of this, it was also tested to train the two networks as one trainable network such that backpropagation is performed for the whole network. Different learning rates were tested.

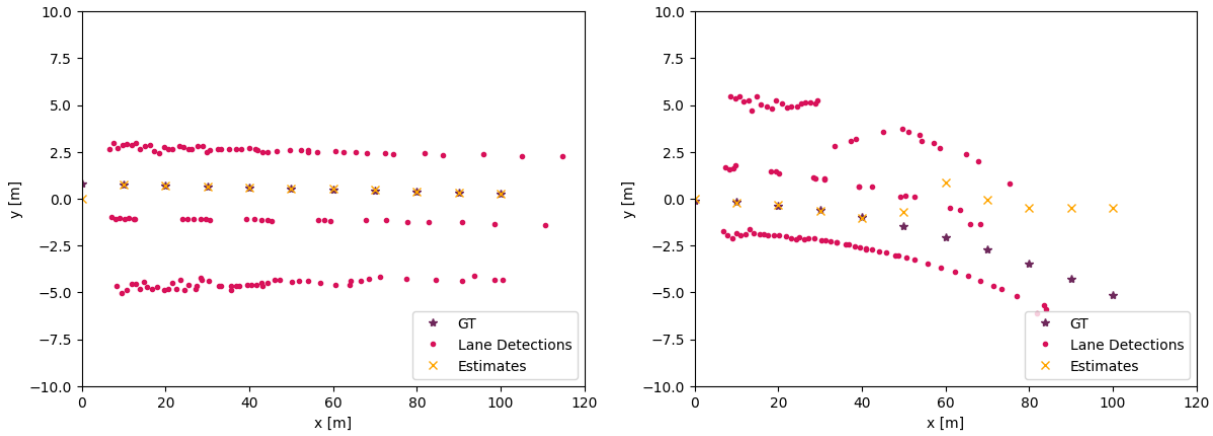
4.3.6 Model 4: Model 3 with Weighted MSE

The metric used to evaluate the performance of the networks was MSE over all evaluation distances which valued all anchor points equally. A weighted MSE was therefore used on Model 3 to increase the importance of the points closer to the car. The LSTM in Model 3 were retrained with the weighted MSE before being evaluated on the same loss as the other models. The weight vector was set to the inverse of the variance of the GT at each anchor point, seen in Figure 13, and then normalized to get the Euclidean length of 1.

5 Results

5.1 Grid Baseline and RANSAC Baseline

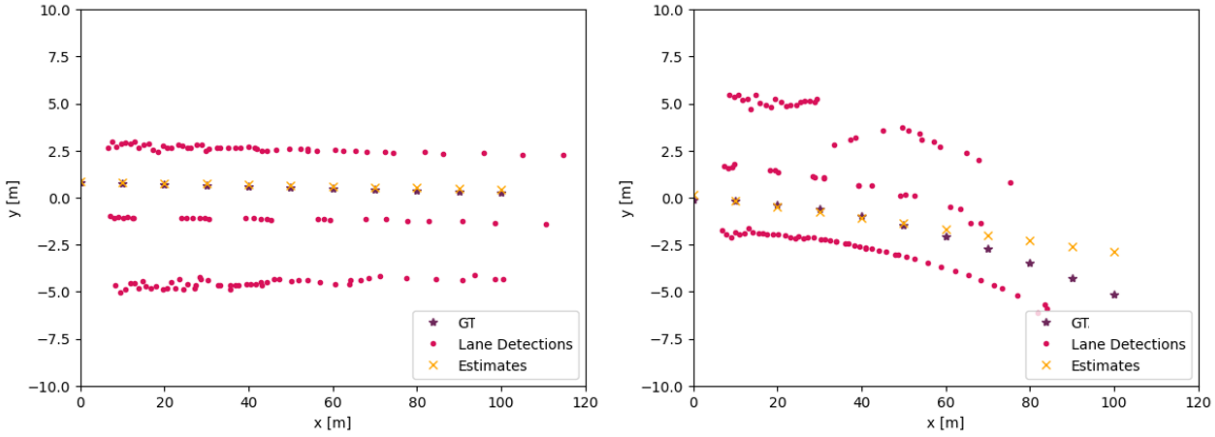
Figure 18 shows the performance of the grid baseline for two frames.



(a) **(b)**

Figure 18: The output of the grid baseline for two frames. In the frame shown in (a), the performance of the grid baseline for a straight lane can be seen. The performance of the grid baseline for a lane that turns a lot can be seen in the frame shown in (b).

The performance of the RANSAC baseline can be seen for the same frames as above in Figure 19.



(a) **(b)**

Figure 19: The output of the RANSAC baseline for the same two frames as for the grid baseline. In the frame shown in (a), the performance of the RANSAC baseline for a straight lane can be seen. The performance of the RANSAC baseline for a lane that turns a lot can be seen in the frame shown in (b).

The difference in results between the grid baseline and the RANSAC baseline was small. The results were measured in RMSE to obtain a more interpretable unit of measure. For the validation data, they had an RMSE of 1.78 m and 1.76 m respectively, which was the lateral error for all the anchor points. Because of the results, as well as the academic literature in the field, the RANSAC baseline was henceforth used as baseline.

5.2 Cell Size

The RMSE results on the validation data when using the different cell sizes are presented in Table 2. Cell size 0.2 m × 0.2 m gave the lowest validation result. An earlier version of the final network was used for these experiments since the network seen in Table 1 did not allow for larger cell sizes.

Table 2: Lowest validation loss when using the same model architecture but different cell sizes.

Cell size [m^2]	0.2 × 0.2	0.3 × 0.3	0.4 × 0.4	0.5 × 0.5
RMSE [m]	0.861	0.937	1.094	1.157

5.3 Model 1: CNN

The loss curve for Model 1 can be seen in Figure 20, where the lowest validation loss was 0.846 m.

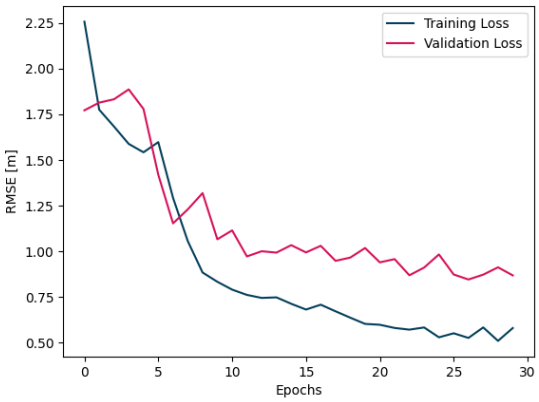
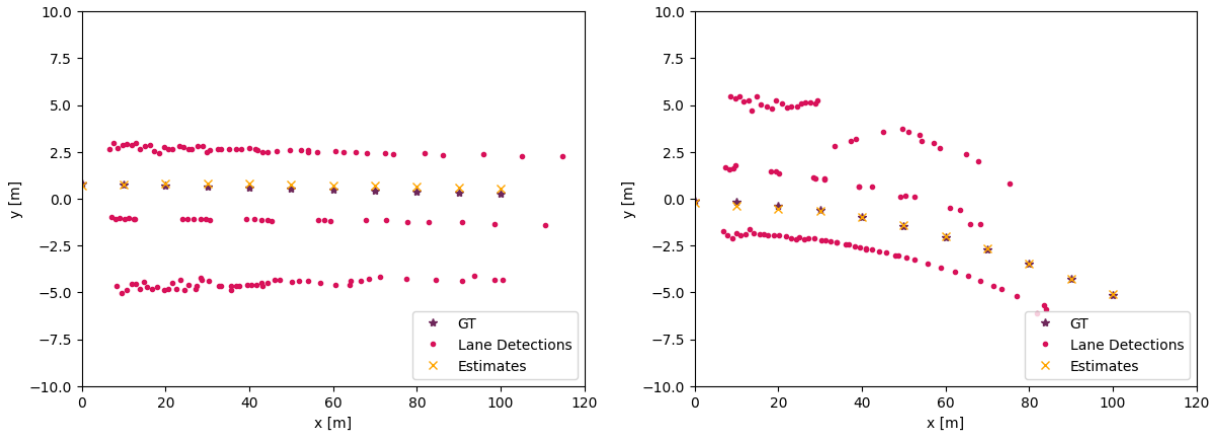


Figure 20: Loss curve for Model 1 with the lowest validation loss of 0.846 meters.

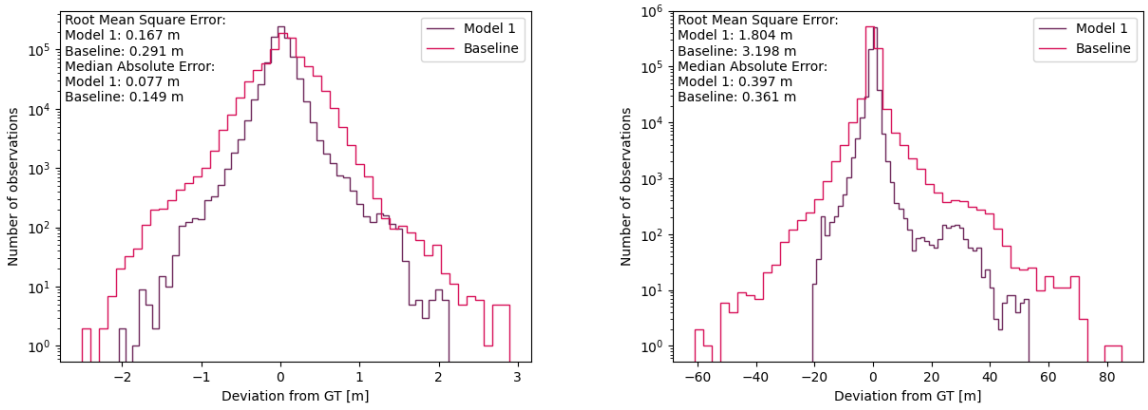
For the two frames, one with straight lanes and one with a turn, the performance of Model 1 can be seen in Figure 21. Since it would be difficult for the naked eye to see any further improvements in these frames, no more model predictions for these frames will be shown.



(a) **(b)**

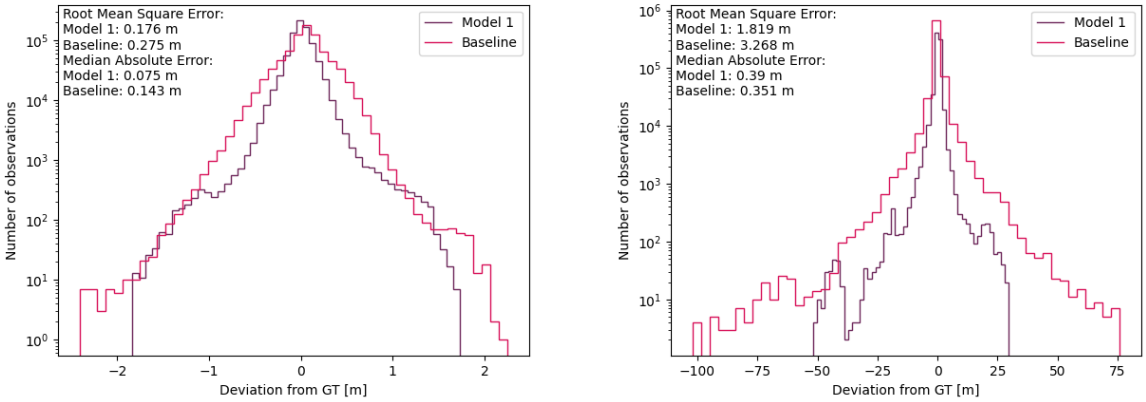
Figure 21: The output of Model 1 for the same two frames as for the baseline models. In the frame shown in (a), the performance of Model 1 for a straight lane can be seen. The performance of Model 1 for a lane that turns a lot can be seen in the frame shown in (b).

For Model 1, the validation errors at each anchor point can be seen compared to the baseline in Figure 22, as well as Figure 32 in the Appendix. The test errors can be seen in Figure 23 and Figure 33. For each anchor point, the RMSE and the median absolute error can also be seen.



(a) **(b)**

Figure 22: Histograms showing the validation errors from Model 1 and the baseline at 0 meters, (a), and 100 meters, (b), ahead of the car. The RMSE and the median absolute error for its respective evaluation distance can be seen in the top left corner.



(a) **(b)**

Figure 23: Histograms showing the test errors from Model 1 and baseline at 0 meters, (a), and 100 meters, (b), ahead of the car. The RMSE and the median absolute error for its respective evaluation distance can be seen in the top left corner.

5.4 Model 1 with Weight Decoupling

Decoupling the weight decay from the optimizer did not lead to any improvements, as can be seen in Figure 24. The lowest validation loss for this network was 0.880 m.

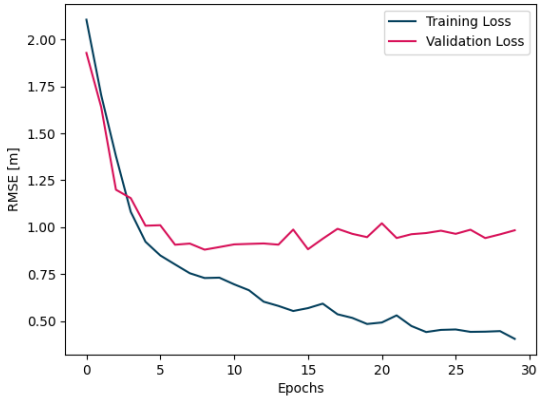


Figure 24: Loss curve for Model 1 with decoupled weight decay and the lowest validation loss of 0.880 meters.

5.5 Model 2: CNN + One-Dimensional Input

Model 2 resulted in improvements in the validation loss compared to Model 1. The lowest loss for the validation set using Model 2 was 0.746 m and the loss curve can be seen in Figure 25.

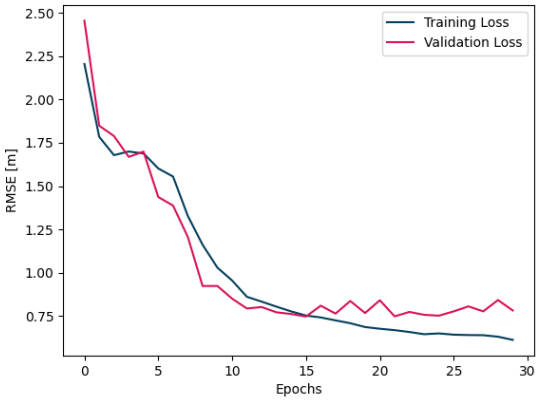


Figure 25: Loss curve for Model 2 with the lowest validation loss of 0.746 meters.

5.6 Model 3: CNN + One-Dimensional Input + LSTM

The loss curve for Model 3 can be seen in Figure 26. The lowest validation loss was 0.691 m, which was a decrease both compared to Model 1 and Model 2.

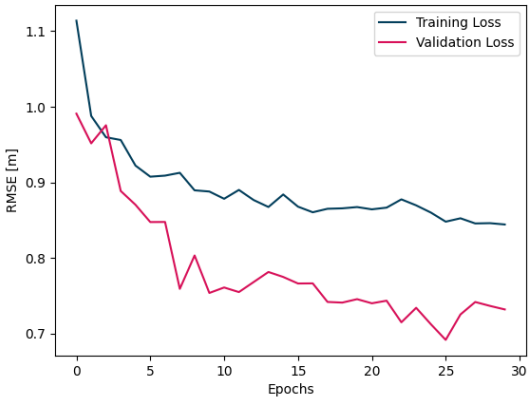


Figure 26: Loss curve for Model 3 with the lowest validation loss of 0.691 meters.

5.7 Model 3 as one Trainable Network

Combining Model 3 into one trainable network and doing backpropagation through both the CNN and the LSTM did not lead to any improvements. The lowest validation loss for the combined network trained as one was 1.538 m and the corresponding loss curve can be seen in Figure 27.

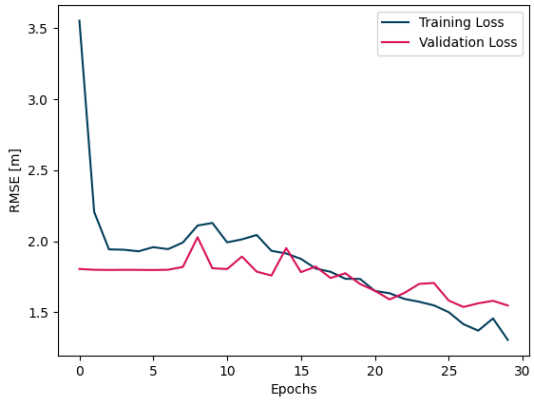


Figure 27: Loss curve for the model using a CNN and an RNN combined and trained as one network. The lowest validation loss was 1.538 meters.

5.8 Model 4: Model 3 with Weighted MSE

The loss curve for Model 4 can be seen in Figure 28. The total validation loss was 0.753 which was higher than Model 3.

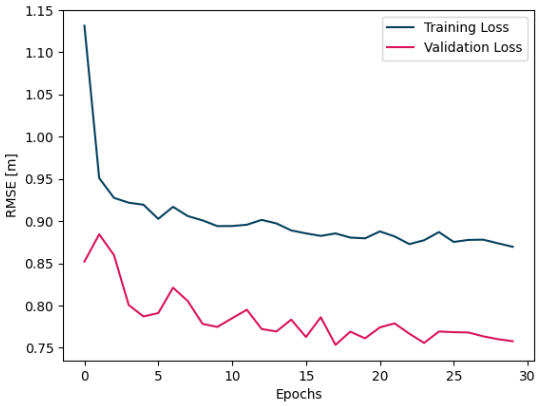
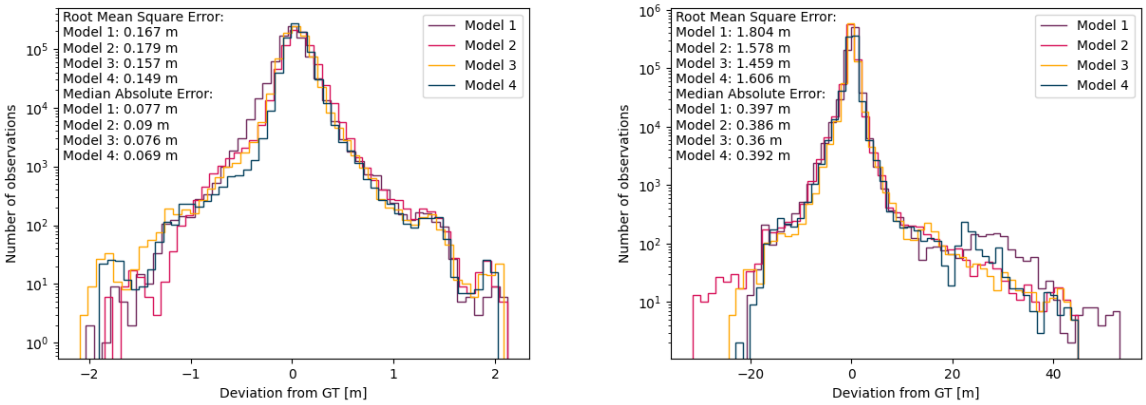


Figure 28: Loss curve for Model 4 with the lowest validation loss of 0.753 meters.

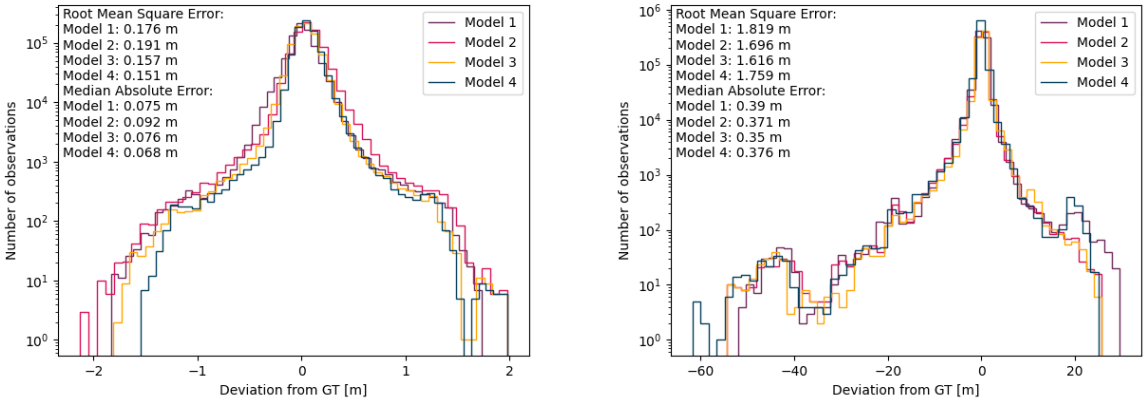
5.9 Summary of Results

The validation and test errors at each anchor point for the baseline and Model 1 can be seen in Figure 22 and 32 and Figure 23 and 33 respectively. Similar error histograms for Model 1, 2, 3, and 4 can be seen in Figure 29 and 34 and Figure 30 and 35. The baseline result was removed from these graphs because of the decreased resolution when including it.



(a) **(b)**

Figure 29: Histograms over the validation errors from Model 1, Model 2, Model 3, and Model 4 at 0 meters, (a), and 100 meters, (b), ahead of the car. The RMSE and the median absolute error for its respective evaluation distance can be seen in the top left corner.



(a) **(b)**

Figure 30: Histograms over the test errors from Model 1, Model 2, Model 3, and Model 4 at 0 meters, (a), and 100 meters, (b), ahead of the car. The RMSE and the median absolute error for its respective evaluation distance can be seen in the top left corner.

All the RMSE results can be seen summarized in Table 3 and 4 for the validation data and test data respectively. The results are presented at a short, middle, and far range of 0 - 30 m, 40 - 60 m, and 70 - 100 m respectively.

Table 3: Validation losses for all models in the short, middle, and far range, as well as the total RMSE. Model 1 consisted of a CNN, Model 2 added a one-dimensional input to the CNN, Model 3 added an LSTM after Model 2, and Model 4 had the same network architecture as Model 3 but trained with a weighted MSE.

	Baseline	Model 1	Model 2	Model 3	Model 4
RMSE 0 - 30 m [m]	0.607	0.149	0.183	0.152	**0.145**
RMSE 40 - 60 m [m]	1.329	0.438	0.414	**0.381**	0.409
RMSE 70 - 100 m [m]	2.594	1.342	1.170	**1.088**	1.189
RMSE Total [m]	1.759	0.846	0.746	**0.691**	0.753

Table 4: Test losses for all models in the short, middle, and far range, as well as the total RMSE. Model 1 consisted of a CNN, Model 2 added a one-dimensional input to the CNN, Model 3 added an LSTM after Model 2, and Model 4 had the same network architecture as Model 3 but trained with a weighted MSE.

	Baseline	Model 1	Model 2	Model 3	Model 4
RMSE 0 - 30 m [m]	0.596	0.178	0.200	0.169	**0.167**
RMSE 40 - 60 m [m]	1.448	0.539	0.517	**0.497**	0.523
RMSE 70 - 100 m [m]	2.671	1.395	1.304	**1.248**	1.353
RMSE Total [m]	1.814	0.894	0.840	**0.803**	0.867

6 Discussion

6.1 Data

The distribution of the data was evaluated in terms of location and time the logs were recorded, as well as deviation from $y = 0$ in the ego vehicle coordinate system. The location was relevant because of weather conditions, but also because different countries could have different color lane markings. The time the logs were recorded was relevant to be able to make sure that all three data sets contained different road conditions. These location and time differences could create variability in the data sets so the data was split in this way to obtain a similar distribution of data in the training, validation, and test set and avoid unwanted cases where, for example, all data in the test set was from the fourth quarter in Region A and all training data from Region B and Region C was from the summer months.

Looking at the deviation of the GT from $y = 0$ at the different anchor points was relevant because this demonstrated the curvature of the lane. It was wanted for all data sets to contain roads with different geometry and not for one set to contain, for example, only straight roads.

6.2 Input

Converting the lane marker detections to a BEV with a set cell size meant that the precise location of the lane marker detections, as well as the number of lane marker detections in each cell, were lost. The bigger the cell size, the more information would be lost. The cell size for the final network was set to 0.2 m $\times$ 0.2 m. A smaller cell size was not feasible due to computational time and memory restrictions. This meant that there was a limitation in the resolution of the input data and thus, a limitation in the performance of the network. However, more information was added as input in terms of the second and third channel of the BEV that described the mean deviation from the center of each cell in the x- and y-direction. There was however an uncertainty in the lane marker detections, which were used to generate the input to the network. The further away from the car the bigger the uncertainty in the detections. This could therefore also have contributed to limiting the performance of the network, especially at further distances.

From Table 2 it can be seen that a cell size of 0.2 m $\times$ 0.2 m outperformed the other cell sizes for that network, which as mentioned, was expected. However, changing the cell size also changed the number of trainable parameters in the network. The smaller the cell size the bigger the input, and the bigger the input, the more trainable parameters in the network. This may have affected the performance of the network since, in general, a large number of trainable parameters could allow a neural network to capture more complex patterns and relationships in the data, which could potentially lead to better performance. Trying to keep the number of trainable parameters constant would however have required a reconstruction of the network architecture. Either way, not all factors could have been kept constant when investigating the effects of different cell sizes. Therefore, it is difficult to draw a certain conclusion as to why there was an improvement in network performance, other than reasoning that this should have been because more information from the input was used, as well as possibly because the network had more trainable parameters.

For Model 2, Model 3, and Model 4 the median of the heading angle of detected vehicles was also used as input. The median resulted in a single value for each frame, removing the need for padding the input. The median was also chosen because detected vehicles could be in the middle of a lane change or in a nonparallel lane, making the median value a more reasonable choice than the mean.

6.3 Baseline

A frame where the grid baseline gave a reasonable result can be seen in Figure 18a, which was expected considering how the grid baseline made predictions. In Figure 18b a limitation of the grid baseline can be seen. When the road turned so that the lane marker detections crossed $y = 0$, the estimates from the grid baseline became bad. Another limitation of the grid baseline was that it estimated the center of the lane straight ahead after the last lane center estimation that was based on lane marker detections instead of taking the shape of the lane up until that point into consideration. This becomes a relevant problem for frames with few lane marker detections since these frames usually only had detections close to the ego vehicle. This meant that the lane center would be predicted straight for several evaluation distances, resulting in a large RMSE, even if the curvature of the lane, based on the detections, suggested otherwise. Furthermore, the grid baseline almost always predicted the center of the lane to be at $y = 0$ for the first anchor point since there were very few detections in that range, as can be seen in Figure 18a.

From Figure 19a it can be seen that the RANSAC baseline, like the grid baseline, gave a reasonable result for the frame with a straight lane. Comparing Figures 18b and 19b it can be seen that the RANSAC baseline outperformed the grid baseline for the frame with a turn.

However, the RANSAC baseline had some limitations too. For example, the fitted polynomial was of degree one. This is not ideal for representing lane markings, especially when the lane is turning because the lane shape cannot accurately be represented in the far field by a straight line. However, not fitting a first-degree polynomial could lead to unwanted behavior far away from the car. One such example occurs when there are only detections relatively close to the car, making a higher degree polynomial fit the points well but further away where there are no detections, making the output deviate significantly from the GT. Another reason for the choice of a first-degree polynomial was that the computational cost increased with a higher-degree polynomial making it more time-consuming as well as needing more iterations of the algorithm due to Equation (6).

The results in combination with the support from other papers using RANSAC in the field of estimating lanes made the RANSAC baseline to be the baseline used for this project. The grid baseline was however used as a sanity check since it was easy to implement and understand.

6.4 Results

6.4.1 Model 1: CNN

From Figure 20 it can be seen that the loss curve for Model 1 looks reasonable. It does however look like the model might still have been able to learn more since the validation loss had not stagnated or begun to increase. Because of this, the model could have been trained for more epochs in order to investigate if the validation loss could have been further lowered.

Looking at Figure 22 and 23 it can be seen that the baseline deviated more from the GT compared to Model 1, both for the validation and the test data.

Model 1 beat the baseline on all distances when measuring in RMSE. This could be explained by the baseline being worse than Model 1 when the road was curved, and that Model 1 was most likely not much worse in the cases it was worse, which is shown when looking at the RMSE and the median absolute error. Furthermore, large errors were penalized more since they were squared in MSE. If the error of the baseline was large, which it would have been for roads with a large curvature, this would have had an even greater effect on the error.

However, Figure 22b and 23b show that at $x = 100$ m the baseline beat Model 1 when measuring in median absolute error. From Figure 32 and Figure 33 in the Appendix it can be

seen that this was also the case for distances $x = 60$ m to $x = 90$ m. This could be explained by the fact that the baseline fitted straight lines to the data and was hence good at estimating the middle of the road when the road was straight. Because the data was collected on highways, the roads were mostly straight, which can be seen in Figure 12, as well as Figure 31. Furthermore, Model 1 did not have the limitation of only fitting straight lines and could thus perform well on both straight and curved roads which can be seen in 21. The fact that it did not perform as well as the baseline on further distances, measured in absolute median error, could be explained by the fact that there were oftentimes very few detections on further distances from the car, which could have made it difficult for Model 1 to make accurate predictions. The baseline was not limited by the fact that there were few detections further from the car since it could fit the straight lines to the detections closer to the car and would thus still make accurate predictions further away from the car, provided that the road was straight. The median was hence a beneficial measure for the baseline when using data consisting predominantly of straight roads.

6.4.2 Model 1 with Weight Decoupling

From Figure 24 it can be seen that the model might have been overfitting since the training loss decreased while the validation loss began to slightly increase. However, this was the best obtained result, measured in validation loss, for the different tested combinations of hyperparameters. It was concluded that for this network architecture weight decoupling might not be suitable to incorporate into Model 1. Because of the results and the time limitation, weight decoupling was only tested for this model.

6.4.3 Model 2: CNN + One-Dimensional Input

From Figure 25 it can be seen that the loss curve for Model 2 looks reasonable. In contrast to the loss curve of Model 1, Model 2 looks as though it had finished training for the set number of epochs since the validation loss had stagnated. Furthermore, comparing Figures 20 and 25, Model 2 seemed to be better at generalizing since the training loss and validation loss were more similar, which might have occurred because of the higher weight decay used. This is a very beneficial property for real-world applications. Moreover, comparing the lowest validation and test loss for the two models, Model 2 outperformed Model 1 with 11.8% on the validation set and 6.04% on the test set, measuring in RMSE. However, looking at Table 3 it can be seen that Model 1 outperformed Model 2 on evaluation distance 0 - 30 m. The addition of the median heading angle was the only thing that differed between Model 1 and Model 2. The heading angle of the other vehicles in the range of the BEV could add information about the geometry of the road at the location of these vehicles. If a heading angle was detected, it was always from a vehicle further down the road, thus, adding information about the lane geometry further from the ego vehicle. Moreover, since the loss used for training the models was MSE, which valued each anchor point equally, the network was not optimized for adding more importance to the anchor points closer to the vehicle. On the contrary, since the largest errors, which affected the MSE the most, were found furthest away from the vehicle, the MSE would decrease more by minimizing these. As a result, it is not unreasonable that Model 2 did not perform as well on closer distances compared to Model 1.

6.4.4 Model 3: CNN + One-Dimensional Input + LSTM

For the LSTM in Model 3, it was not needed that the CNN outputted the location of the lane center at the evaluation distances. The LSTM could have taken a feature map from the CNN of arbitrary size. Because of this, there are many architectures for Model 3 that remain unexplored.

However, the LSTM was added in order to see if the output from the CNN could be improved so it would be counterintuitive to change the architecture of the CNN in Model 3 since it would not be as comparable to the CNNs in Model 1 and Model 2.

From Figure 26 it can be seen that the validation loss was lower than the training loss. This can be explained by the fact that dropout regularization was used during the training process, making it more difficult for the model to fit to the training data. Dropout was not used during validation so the difference in losses is not unreasonable. Another explanation for the training loss being higher than the validation loss could be that the training data was more difficult for the model to adapt to. However, it has been seen that the training, validation, and test set came from a similar distribution when measuring in deviation from $y = 0$, as well as from when and where the data was gathered. Furthermore, a higher training loss than validation loss was not observed when training Model 1 and Model 2, but these did not use dropout. However, there is still a possibility that for this model architecture in particular, fitting the training data was more difficult than the validation data.

Looking at Table 3 it can be seen that Model 3 outperformed all previous models on all evaluation metrics when using the validation data except the closest where Model 1 performed the best. This could be for the same reason as discussed above. Table 3 shows that Model 3 was better than all previous models on all evaluation metrics when using the test data. This was expected since it also makes use of the temporal information in the data. Compared to Model 2, Model 3 performed 7.37% better on the validation data and 4.40% on the test data, measured in total RMSE.

Figure 29 and Figure 30, as well as Figure 34 and 35 in the Appendix, show that Model 3, contrary to Model 1 and Model 2, beats the baseline on all evaluation distances for both the validation and test data when measuring in median absolute error. This further speaks to Model 3 being the best model thus far.

6.4.5 Model 3 as one Trainable Network

The results when combining the CNN and the LSTM in Model 3 into one trainable network can be seen in Figure 27. It can be seen that the training and validation losses were slowly decreasing but were still very high compared to the results from all previous models. It is difficult to draw a conclusion as to why these results were observed other than that it seemed like the model architecture of the CNN was not compatible with the different LSTMs tested when combining them into one network. However, from Figure 27 it seems like the network was slightly underfitted and could improve with more epochs. It would however need to find a loss of more than half the size to outperform Model 3 which seemed unlikely considering that the network did not seem to efficiently learn based on the slight slope of the graph. Because of this, it was decided to not focus more time and computational resources on finding a network that combined a CNN and an LSTM into one trainable network.

6.4.6 Model 4: Model 3 with Weighted MSE

The results raised the question of what a good metric is since, depending on what metric was used, the results could imply different conclusions depending on how they were interpreted. Different metrics also produce models that are optimized for different tasks. It can be argued that, in this case, MSE was a good choice of metric. When using the MSE, the predictions further away from the actual middle of the lane were penalized more because of the square term. For the task of autonomous driving, this could be a good representation of how the penalization would look in reality since estimating the middle of the road a little off would most likely not have any big consequences but a big error in estimation could lead to the vehicle

possibly steering out off lane. Furthermore, MSE valued all anchor points equally, which was not desired when estimating the lane. For the anchors closer to the car it was more critical to have accurate estimates. If the estimates were inaccurate for an anchor point close to the car, it could potentially lead to the car veering of course. Conversely, for an anchor point far away from the ego vehicle, small inaccuracies may not have a significant impact. To account for this a weighted MSE was tested after having developed Model 3, where the errors closer to the car were given more weight than those further away. This ensures that the model was optimized for accuracy where it mattered the most and less sensitive to inaccuracies further away where the car was less likely to be affected.

For Model 4, only the LSTM from Model 3 was retrained with the new loss. This was because when training both the CNN and the LSTM gave bad results. This could be because the learning rate was too low to find a good minimum. Ideally, the hyperparameters should have been retuned, but because it was time-consuming to train both networks since the LSTM needed the output from the CNN, there was no time to tune the hyperparameters and continue to retrain both networks.

Table 3 and 4 show that Model 4 outperformed all models in the close range, which was expected since the closer evaluation distances were weighted heavier. In the middle and far range, the evaluation distances had smaller weights leading to Model 3 outperforming Model 4, which was reasonable. For all models, the losses were higher further away from the vehicle. This was expected because there were fewer and less accurate lane markings at further distances from the ego vehicle.

It is difficult to draw a conclusion as to which model was the best. Model 4 was the best at the close range but Model 3 outperformed Model 4 on the middle and far range, as well as overall. To determine the best model, it is important to take the real-world implementation into account and asses what qualities a good model would have. Furthermore, more testing using several metrics in edge cases, such as highly curved roads or isolated frames with few detections, would be beneficial to more rigorously evaluate the performance of the models.

7 Conclusion

In conclusion, this master thesis proposes different neural network models in order to investigate a learning-based road estimation approach. The results indicate that the more relevant data is used, and the more complex the models are, the better the models perform. The model that used a CNN and LSTM combined with fused one-dimensional data, improved the loss by 10.2% on the test data compared to the model using only a CNN. Furthermore, the selection of a metric used to train and evaluate networks is essential in the process of developing a task-relevant model. At last, from the experience gathered during the thesis, a learning-based approach seems to hold a lot of potential for estimating road geometry.

8 Future Work

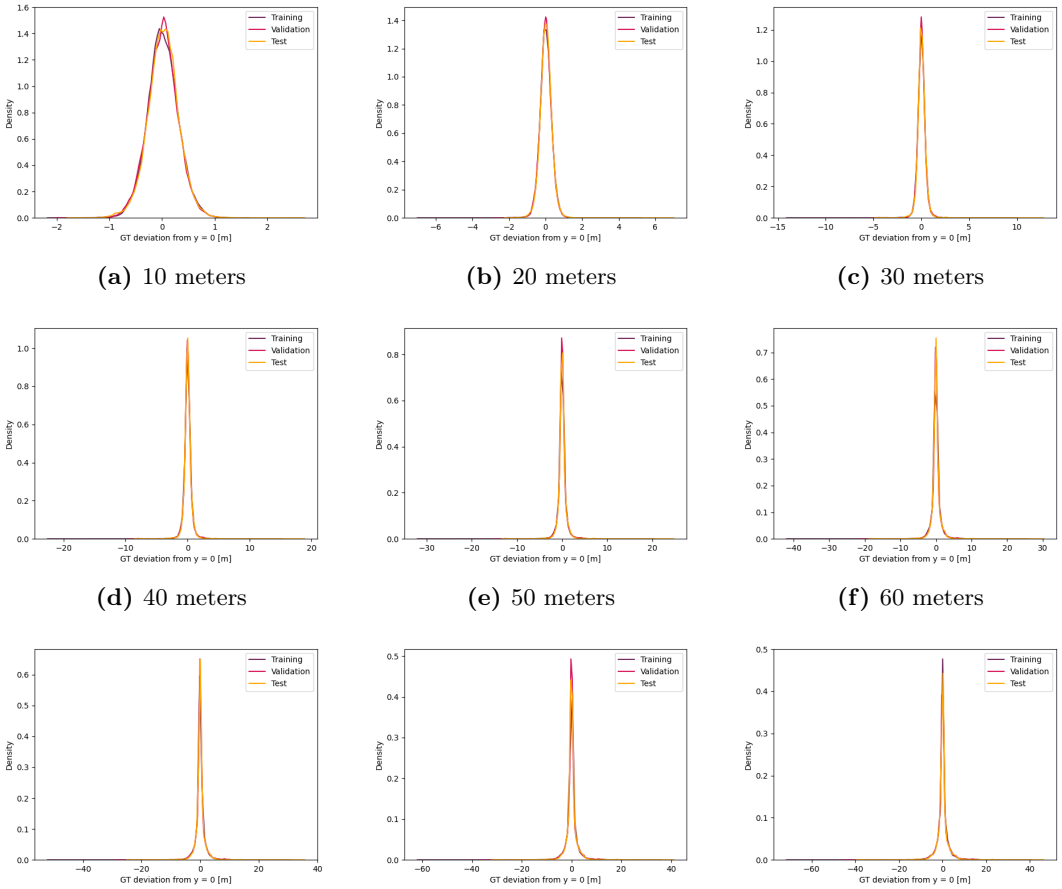
Future work in the field could encompass combining a CNN and an LSTM into one trainable network, instead of training them separately. This would probably require another CNN architecture than the one presented in this project since the current architecture did not seem to be compatible with the LSTMs tested when combining them into one network.

Moreover, the input to the CNN, the BEV, was very sparse which was very computationally inefficient. Because of this, other model architectures such as sparse CNNs, could be investigated. Other approaches could include using only an LSTM and not converting the lane marker detections to a BEV such that no information is lost in the transformation. Alternatively, using sensor data, such as images, directly instead of using them to generate object detection data.

Another interesting addition to the deep learning approach would be to make use of the knowledge that clothoids are a good representation of roads [54]. This would put a limitation on the output of the network, compared to the 11 independent anchor points used in this thesis, and potentially increase the accuracy.

Lastly, transformer models are expected to continue advancing and revolutionizing various fields, including lane estimation. These models, known for their ability to capture complex patterns in sequential data, have the potential to enhance the accuracy and robustness of lane estimation algorithms.

9 Appendix

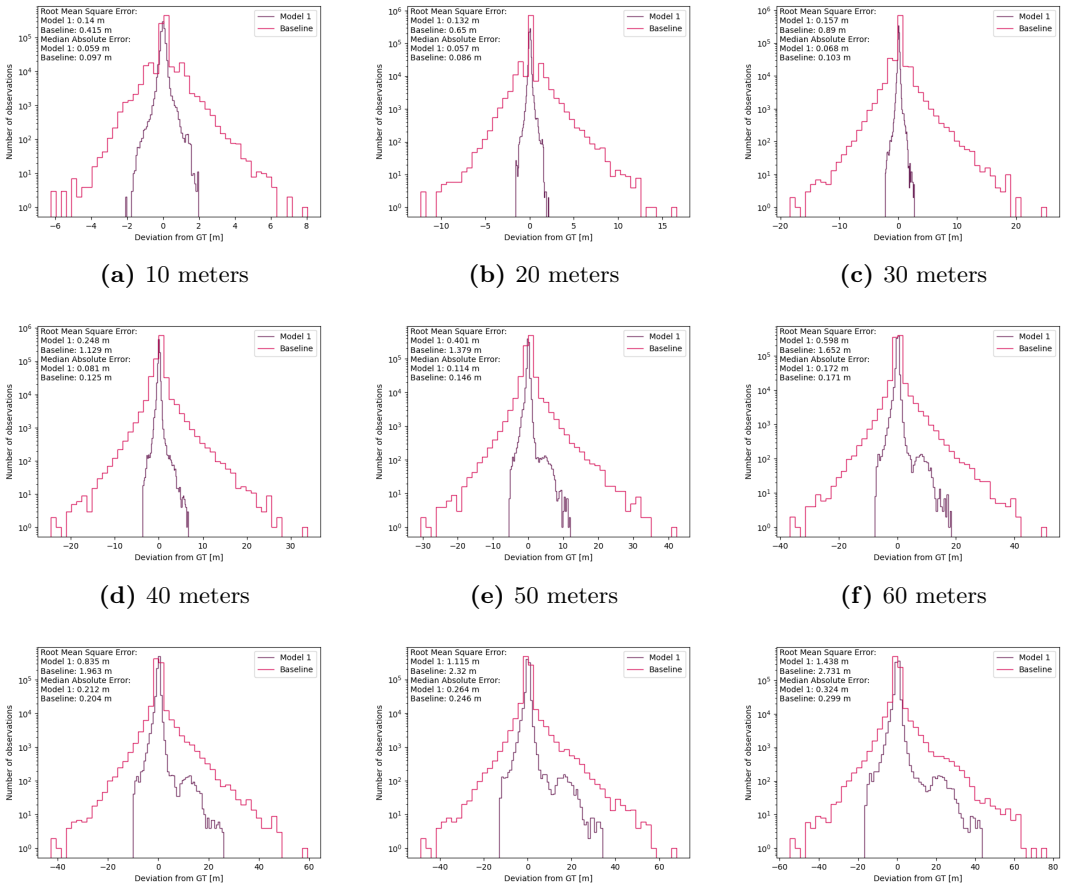


(a) 10 meters **(b)** 20 meters **(c)** 30 meters

(d) 40 meters **(e)** 50 meters **(f)** 60 meters

(g) 70 meters **(h)** 80 meters **(i)** 90 meters

Figure 31: The distribution in deviation from $y = 0$ of the GT points 10 to 90 meters ahead of the car.

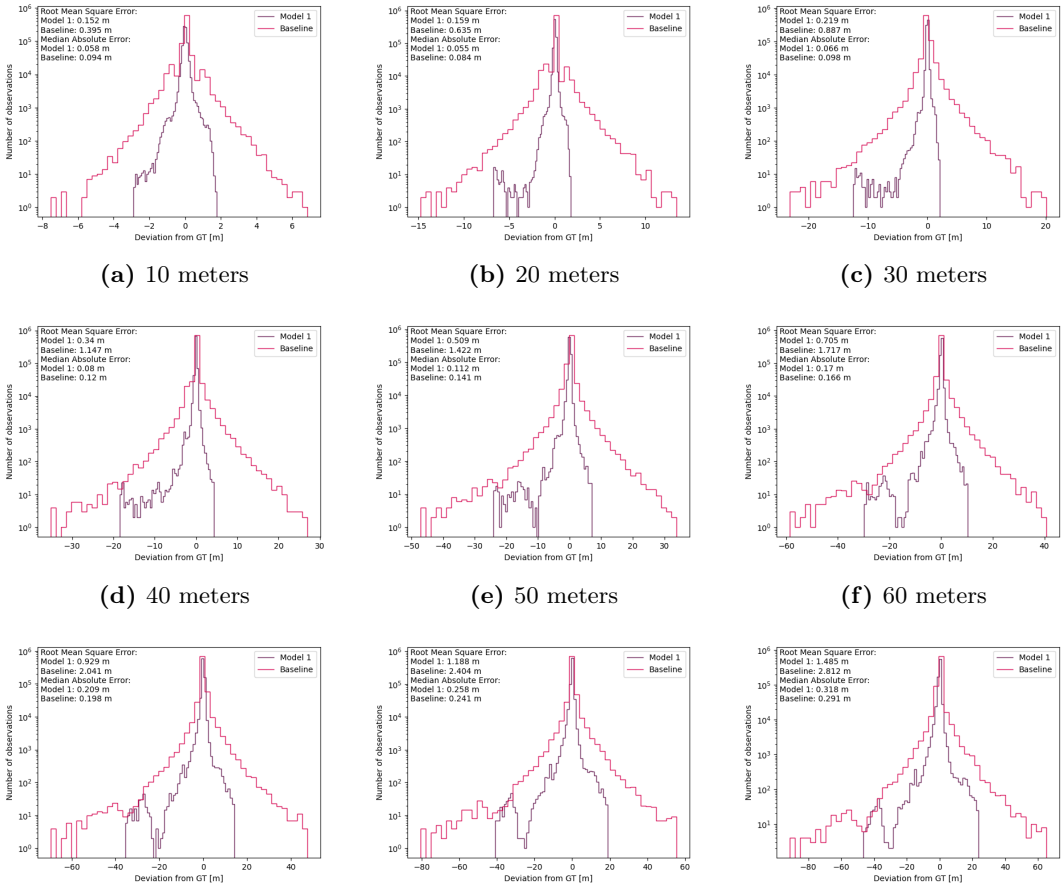


(a) 10 meters **(b)** 20 meters **(c)** 30 meters

(d) 40 meters **(e)** 50 meters **(f)** 60 meters

(g) 70 meters **(h)** 80 meters **(i)** 90 meters

Figure 32: Histograms over the validation errors from Model 1 and Baseline, 10 to 90 meters ahead of the car. The RMSE and the median absolute error for its respective evaluation distance can be seen in the top left corner.

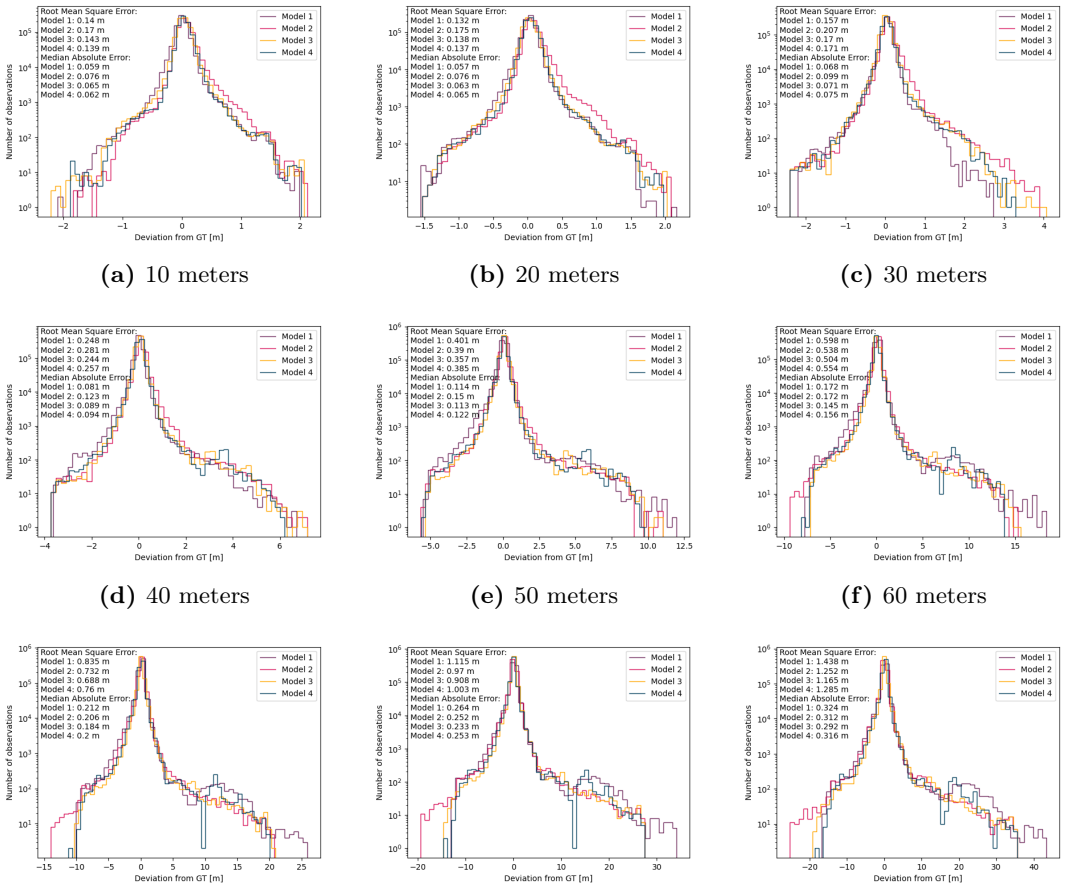


(a) 10 meters **(b)** 20 meters **(c)** 30 meters

(d) 40 meters **(e)** 50 meters **(f)** 60 meters

(g) 70 meters **(h)** 80 meters **(i)** 90 meters

Figure 33: Histograms over the test errors from Model 1 and Baseline, 10 to 90 meters ahead of the car. The RMSE and the median absolute error for its respective evaluation distance can be seen in the top left corner.

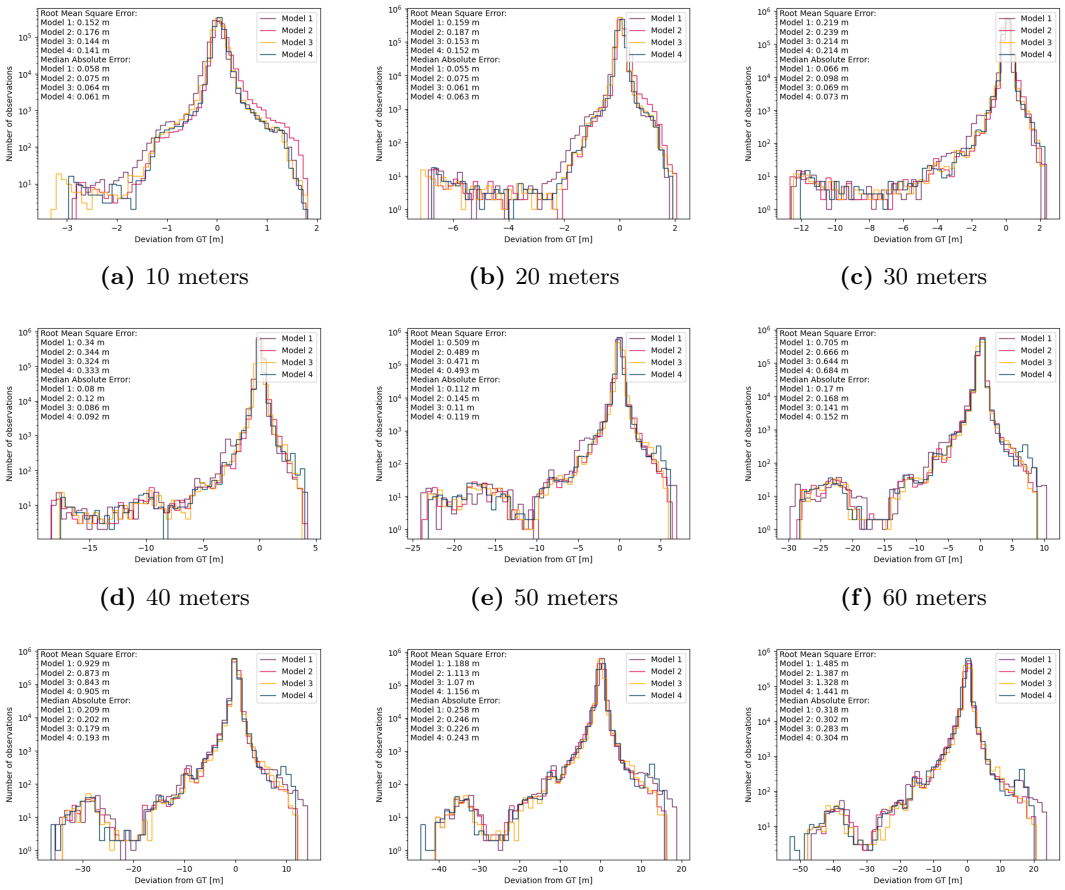


(a) 10 meters **(b)** 20 meters **(c)** 30 meters

(d) 40 meters **(e)** 50 meters **(f)** 60 meters

(g) 70 meters **(h)** 80 meters **(i)** 90 meters

Figure 34: Histograms over the validation errors from Model 1, Model 2, Model 3 and Model 4, 10 to 90 meters ahead of the car. The RMSE and the median absolute error for its respective evaluation distance can be seen in the top left corner.



(a) 10 meters **(b)** 20 meters **(c)** 30 meters

(d) 40 meters **(e)** 50 meters **(f)** 60 meters

(g) 70 meters **(h)** 80 meters **(i)** 90 meters

Figure 35: Histograms over the test errors from Model 1, Model 2, Model 3, and Model 4 10 to 90 meters ahead of the car. The RMSE and the median absolute error for its respective evaluation distance can be seen in the top left corner.

References

[1] W. H. Organization, *Road traffic injuries: Key facts*, 2021. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[2] Q. Zou, H. Jiang, Q. Dai, Y. Yue, L. Chen, and Q. Wang, "Robust lane detection from continuous driving scenes using deep neural networks", *CoRR*, 2019. DOI: 10.1109/TVT.2019.2949603.

[3] Y. Dong, S. Patil, B. van Arem, and H. Farah, "A hybrid spatial–temporal deep learning architecture for lane detection", *Computer-Aided Civil and Infrastructure Engineering*, 2022. DOI: 10.1111/mice.12829.

[4] W. D. Montgomery, "Public and private benefits of autonomous vehicles", 2018.

[5] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations", *Transportation Research Part A: Policy and Practice*, pp. 167–181, 2015. DOI: 10.1016/j.tra.2015.04.003.

[6] S. Patil, "Robust lane detection using deep learning model-literature survey", 2021. DOI: 10.13140/RG.2.2.24871.01445.

[7] Y. Y. Ye, X. L. Hao, and H. J. Chen, "Lane detection method based on lane structural analysis and cnns", *IET Intelligent Transport Systems*, 2018. DOI: https://doi.org/10.1049/iet-its.2017.0143.

[8] B. E. Soorchaei, M. Razzaghpour, R. Valiente, A. Raftari, and Y. P. Fallah, "High-definition map representation techniques for automated vehicles", *Electronics (Switzerland)*, 2022. DOI: 10.3390/electronics11203374.

[9] A. Ziębiński, R. Cupek, D. Grzechca, and L. Chruszczyk, "Review of advanced driver assistance systems (adas)", *AIP Conference Proceedings*, 2017. DOI: 10.1063/1.5012394.

[10] V. Voisin, M. Avila, B. Emile, S. Begot, and J. Bardet, "Road markings detection and tracking using hough transform and kalman filter. advanced concepts for intelligent vision systems", DOI: https://doi.org/10.1007/11558484_10.

[11] A. Borkar, M. Hayes, and M. T. Smith, "Robust lane detection and tracking with ransac and kalman filter", *IEEE*, 2009. DOI: 10.1109/ICIP.2009.5413980.

[12] J. Guo, Z. Wei, and D. Miao, "Lane detection method based on improved ransac algorithm", *IEEE*, 2015. DOI: 10.1109/ISADS.2015.24.

[13] A. Bar Hillel, R. Lerner, D. Levi, M. Trapp, and E. Yechiam, "Recent progress in road and lane detection: A survey", *Machine Vision and Applications*, 2014. DOI: 10.1007/s00138-011-0404-2.

[14] Q. Li, R. Li, K. Ji, and W. Dai, "Kalman filter and its application", *IEEE*, 2015. DOI: 10.1109/ICINIS.2015.35.

[15] T. JV, "Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes", *Clinical Epidemiology*, 1996. DOI: 10.1016/s0895-4356(96)00002-9.

[16] M. Olsson and P. Edén, "Introduction to artificial neural networks and deep learning", 2021.

[17] S. F. Ahmed, M. S. B. Alam, M. Hassan, *et al.*, "Deep learning modelling techniques: Current progress, applications, advantages, and challenges", 2023. DOI: 10.1007/s10462-023-10466-8.

[18] J. Li, X. Mei, D. Prokhorov, and D. Tao, "Deep neural network for structural prediction and lane detection in traffic scene", *IEEE*, 2017. DOI: 10.1109/TNNLS.2016.2522428.

[19] 2023. [Online]. Available: https://zenseact.com/people-at-heart/.

[20] M. Jordà, P. Valero-Lara, and A. J. Peña, "Performance evaluation of cudnn convolution algorithms on nvidia volta gpus", *IEEE*, 2019. DOI: 10.1109/ACCESS.2019.2918851.

[21] 2023. [Online]. Available: https://pytorch.org/tutorials/advanced/numpy_extensions_tutorial.html#parametrized-example.

[22] Z. Wang, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "A multi-objective evolutionary approach for efficient kernel size and shape for cnn", 2022. DOI: 10.1109/IJCNN55064.2022.9892201.

[23] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling", *Proceedings of the Annual Conference of the International Speech Communication Association*, 2014.

[24] 2023. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[25] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Vaughan, "A theory of learning from different domains", *Machine Learning*, 2010.

[26] 2023. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html.

[27] M. Nielsen, "Neural networks and deep learning", 2019. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap1.html.

[28] H. K. Jabbar and R. Z. Khan, "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)", *Computer Science, Communication & Instrumentation Devices*, 2015.

[29] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization", 2019. [Online]. Available: https://arxiv.org/abs/1711.05101.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", 2015.

[32] C. Olsson, "Lecture notes in computer vision", p. 88, 2022.

[33] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", 2015. [Online]. Available: http://arxiv.org/abs/1505.04597.

[34] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation", *IEEE*, 2017. DOI: 10.1109/TPAMI.2016.2644615.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", 2014. [Online]. Available: http://arxiv.org/abs/1409.1556.

[36] N. Garret, R. Cohen, T. Pe'er, R. Lahav, and D. Levi, "3d-lanenet: End-to-end 3d multiple lane detection", 2019. [Online]. Available: https://arxiv.org/abs/1811.10203.

[37] M. Aly, "Real time detection of lane markers in urban streets", *IEEE*, 2014. [Online]. Available: http://arxiv.org/abs/1411.7113.

[38] A. Hu, Z. Murez, N. Mohan, *et al.*, "Fiery: Future instance prediction in bird's-eye view from surround monocular cameras", 2021. [Online]. Available: http://arxiv.org/abs/2104.10490.

[39] Y. Ma, T. Wang, X. Bai, *et al.*, "Vision-centric bev perception: A survey", 2022. [Online]. Available: http://arxiv.org/abs/2208.02797.

[40] "Delving into the devils of bird's-eye-view perception: A review, evaluation and recipe", 2022. [Online]. Available: http://arxiv.org/abs/2209.05324.

[41] A. W. Harley, Z. Fang, J. Li, R. Ambrus, and K. Fragkiadaki, "Simple-bev: What really matters for multi-sensor bev perception?", 2022. [Online]. Available: http://arxiv.org/abs/2206.07959.

[42] Z. Liu, H. Tang, A. Amini, *et al.*, "Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation", 2022. [Online]. Available: http://arxiv.org/abs/2205.13542.

[43] P. Wu, S. Chen, and D. Metaxas, "Motionnet: Joint perception and motion prediction for autonomous driving based on bird's eye view maps", 2020. [Online]. Available: http://arxiv.org/abs/2003.06754.

[44] S. Fadadu, S. Pandey, D. Hegde, *et al.*, "Multi-view fusion of sensor data for improved perception and prediction in autonomous driving", 2022. [Online]. Available: https://arxiv.org/abs/2008.11901.

[45] Z. Wu, K. Qiu, T. Yuan, and H. Chen, "A method to keep autonomous vehicles steadily drive based on lane detection", *International Journal of Advanced Robotic Systems*, 2021. DOI: 10.1177/17298814211002974.

[46] Y. Jeong, "Interactive lane keeping system for autonomous vehicles using lstm-rnn considering driving environments", 2022. DOI: 10.3390/s22249889.

[47] I. Arvidsson, N. C. Overgaard, K. Åström, *et al.*, "Prediction of obstructive coronary artery disease from myocardial perfusion scintigraphy using deep neural networks", *IEEE*, 2021. DOI: 10.1109/ICPR48806.2021.9412674.

[48] J. Luo, W. Zhang, J. Su, and F. Xiang, "Hexagonal convolutional neural networks for hexagonal grids", *IEEE*, 2019. DOI: 10.1109/ACCESS.2019.2944766.

[49] Y. Wu, K. Namdar, C. Chen, *et al.*, "Automated adolescence scoliosis detection using augmented u-net with non-square kernels", *Canadian Association of Radiologists*, 2023. DOI: 10.1177/08465371231163187.

[50] A. A. Mamun, E. P. Ping, J. Hossen, A. Tahabilder, and B. Jahan, "A comprehensive review on lane marking detection using deep neural networks", *Sensors*, 2022. DOI: 10.3390/s22197682.

[51] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, *The marginal value of adaptive gradient methods in machine learning*, 2018. [Online]. Available: https://arxiv.org/abs/1705.08292.

[52] M. Alibeigi, W. Ljungbergh, A. Tonderski, *et al.*, *Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving*, 2023.

[53] Photograph, 2021. [Online]. Available: https://zenseact.com/2021/07/14/.

[54] M. Fatemi, L. Hammarstrand, L. Svensson, and Á. F. García-Fernández, "Road geometry estimation using a precise clothoid road model and observations of moving vehicles", 2014. DOI: 10.1109/ITSC.2014.6957698.

Master's Theses in Mathematical Sciences 2023:E44
ISSN 1404-6342

LUTFMA-3513-2023

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/