Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Max Michael Johnsen Aasbø
Hege Haavaldsen

Autonomous Vehicle Control:

End-to-end Learning in Simulated Environments

Master's thesis in Computer Science
Supervisor: Frank Lindseth
June 2019

NTNU
Norwegian University of
Science and Technology

Max Michael Johnsen Aasbø
Hege Haavaldsen

Autonomous Vehicle Control:

End-to-end Learning in Simulated Environments

Master's thesis in Computer Science
Supervisor: Frank Lindseth
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

In recent years, considerable progress has been made towards a vehicle's ability to operate autonomously. An end-to-end approach attempts to achieve autonomous driving using a single, comprehensive software component. Recent breakthroughs in deep learning have significantly increased end-to-end systems' capabilities, and such systems are now considered a possible alternative to the current state-of-the-art solutions.

This thesis examines end-to-end learning for autonomous vehicles in diverse, simulated environments containing other vehicles, traffic lights, and speed limits; in weather conditions ranging from sunny to heavy rain. Moreover, the thesis further aims to explore some of the uncertainties regarding the implementation of an end-to-end system. Specifically, how the system's overall performance is affected by the size of the training dataset, the allowed prediction frequency, and the number of hidden states in the system's recurrent module.

The thesis proposes an end-to-end architecture combing a traditional Convolutional Neural Network with a recurrent layer, to facilitate the learning of both spatial and temporal relationships. The system is trained using expert driving data from various simulated settings and evaluated by its real-time driving performance in unseen simulated environments.

The results of the thesis indicate that end-to-end systems can operate autonomously in simulated environments, in a range of different weather conditions. Additionally, it was found that using ten hidden states for the system's recurrent module was optimal. The results further show that the system was sensitive to small reductions in dataset size and that a prediction frequency of 15 Hz was required for the system to perform at its full potential.

Sammendrag

I de siste årene er det gjort betydelige fremskritt mot et kjøretøys evne til å operere autonomt. En ende-til-ende-tilnærming forsøker å oppnå autonom kjøring ved hjelp av en enkel, omfattende komponent. Nylige gjennombrudd i dyp læring har økt kapasiteten til ende-til-ende systemer, og de anses nå som et mulig alternativ til dagens løsninger.

Denne oppgaven undersøker ende-til-ende-læring i ulike, simulerte miljøer som inneholder andre kjøretøyer, trafikklys og fartsgrenser – i værforhold som varierer fra sol til kraftig regn. Videre undersøker oppgaven noen av usikkerhetene tilknyttet implementeringen av et ende-til-ende-system. Mer spesifikt undersøker oppgaven hvordan systemets ytelse påvirkes av størrelsen på treningsdatasettet, den tillatte prediksjonfrekvensen, og antall skjulte tilstander i systemets tilbakevendende modul.

Avhandlingen foreslår en ende-til-ende-arkitektur som kombinerer et konvolusjonelt nevralt nettverk med et tilbakevendende nettverk for å lære romlige og tidsmessige relasjoner. Systemet trenes opp ved hjelp av menneskelig kjøredata fra ulike simulerte miljøer, og evalueres på grunnlag av sanntidsytelsen i usette, simulerte miljøer.

Resultatene viser at ende-til-ende-systemer kan operere autonomt i simulerte miljøer, i en rekke forskjellige værforhold. Videre viste resultatene at ti skjulte tilstander for systemets tilbakevendende modul var optimalt. I tillegg viser resultatene at systemet var følsomt overfor små reduksjoner i datasettstørrelsen, og at en prediksjonsfrekvens på 15 Hz var nødvendig for at systemet skulle operere optimalt.

Acknowledgements

We would like to sincerely thank our supervisor, Frank Lindseth, for his guidance throughout the last two semesters. Without his efforts concerning the NTNU Autonomous Perception Lab (NAP-Lab) we would not have been able to choose such an inspiring and exciting research field for our master's thesis.

Table of Contents

Abstract v

Sammendrag vi

Acknowledgements vii

Table of Contents xii

List of Tables xiv

List of Figures xviii

Abbreviations xix

1 Introduction 1

1.1 Background and Motivation 2

1.2 Objectives and Research Questions 3

1.3 Contributions 4

1.4 Thesis Outline 5

2 Background 7

2.1 Artificial Neural Networks 8
2.1.1 Network Structure 8
2.1.2 Training 11
2.1.3 Regularization 14
2.2 ANN Architectures 16
2.2.1 Convolutional Neural Networks 16
2.2.2 Recurrent Neural Networks 19
2.3 Autonomous Vehicle Control 21
2.3.1 Mediated Perception 21
2.3.2 End-to-end Learning 22
2.3.3 Training and Testing 25
2.3.4 Data Augmentation 26
2.4 Approaches in End-to-end Learning 27
2.4.1 ALVINN 28
2.4.2 DAVE 29
2.4.3 DAVE-2 29
2.4.4 End-to-end Driving via Conditional Imitation Learning 31
2.4.5 Adding Navigation to the Equation: Turning Decisions for End-to-End Vehicle Control 33
2.4.6 End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies (2017) 35
2.4.7 Deep Reinforcement Learning 37
2.5 Software and Hardware 38
2.5.1 CARLA Simulator 38
2.5.2 TensorFlow 40
2.5.3 Keras 41
2.5.4 GPU and CUDA 42

3 Methodology **43**

3.1 Environment 44

3.1.1 Choice of Simulator . . . 44
3.1.2 Simulator Controller . . . 45
3.2 Camera Setup . . . 48
3.3 Data Collection . . . 50
3.4 Data Preparation . . . 51
3.5 Model Architecture . . . 55
3.5.1 Problem Definition . . . 55
3.5.2 System Overview . . . 56
3.5.3 CNN . . . 57
3.5.4 LSTM . . . 57
3.5.5 Steering Angle Classification . . . 60
3.6 Experimental Setup . . . 63
3.6.1 Experiment 1: Sequence Length . . . 63
3.6.2 Experiment 2: Size of Dataset . . . 63
3.6.3 Experiment 3: Prediction Frequency . . . 64
3.6.4 Experiment 4: Extensive Real-time Test . . . 64
3.6.5 Training procedure . . . 65
3.6.6 Real-time test . . . 67

4 Results **71**
4.1 Experiment 1: Sequence Length . . . 72
4.1.1 Training . . . 72
4.1.2 Real-time Test . . . 72
4.2 Experiment 2: Size of Dataset . . . 76
4.3 Experiment 3: Prediction Frequency . . . 77
4.4 Experiment 4: Extensive Real-time Test . . . 79

5 Discussion **83**
5.1 Training Data . . . 84

5.2 CARLA simulator 84
5.3 Architecture 85
5.3.1 Steer Predictor 85
5.3.2 Throttle-Brake Predictor 86
5.4 Experimental Results 87
5.4.1 Experiment 1: Sequence Length 87
5.4.2 Experiment 2: Size of Dataset 89
5.4.3 Experiment 3: Prediction Frequency 90
5.4.4 Experiment 4: Extensive Real-time Test 92
5.5 Consistency with Related Work 96
5.6 Fulfillment of Research Questions 98

6 Conclusion and Future Work **99**
6.1 Conclusion 100
6.2 Future Work 101

Bibliography **103**

Appendices **107**

A Preliminary Paper: End-to-end Learning in Simulated Urban Environments **107**

B System Implementation **121**

C System Architecture **125**

D CARLA's prebuilt Cities **127**

List of Tables

2.1 Architecture of the DriveNet's visual encoder. 34

3.1 The collected training data. An observation contains the captured data from a single rendered frame in the simulator. Data were captured with a frequency of 10 Hz. 51

3.2 The different types of data captured from the simulator. 51

3.3 The different routes a model were to drive during a real-time test. 67

3.4 The different types of failures captured by the simulator controller during a real-time test. 69

4.1 Validation loss for models trained with different sequence lengths. 72

4.2 Experiment 1 – The systems' average route completions in Town 2. 73

4.3 Experiment 1 – Total number of failures in Town 2. 74

4.4 Experiment 1 – Number of failures per traveled km in Town 2. 74

4.5 Experiment 1 – Average route completions in each weather condition in Town 4. 75

4.6 Experiment 1 – Total number of failures in Town 4. 76

4.7 Experiment 1 – Number of failures per traveled km in Town 4. 76

4.8 Experiment 2 – Average route completion in each weather condition in Town 2. 77

4.9 Experiment 3 – Average route completion in each weather conditionin Town 2. 78
4.10 Experiment 4 – Average route completions in each weather condition. . . 79
4.11 Experiment 4 – Total number of failures in Town 2. 79
4.12 Experiment 4 – Number of failures per traveled km in Town 2. 80
4.13 Experiment 4 – Total number of failures after driving 55.4 km in Town 4. 80
4.14 Experiment 4 – Number of failures per traveled km in Town 4. 81

List of Figures

2.1 Structure of a Neural Network. 8

2.2 Activation of a neuron. 9

2.3 The effect of the learning rates. Figure (a) illustrates a large learning rate that may lead to suboptimal weight convergence. Figure (b) illustrates a small learning rate that lead to slower training and the risk of being stuck in a local minima. 13

2.4 Underfitting vs. overfitting, (a) illustrates an underfitted model, (b) illustrates an ideal fit, while (c) illustrates of an overfitted model. 15

2.5 Early stopping. The blue graph represents the training loss, while the red graph represents the test loss. The cross indicates the smallest test loss and indicates where the training should stop. 15

2.6 Example of dropout in a neural network 16

2.7 Examples of image transformations 17

2.8 The general architecture of a CNN. 18

2.9 Convolution and depth of output 18

2.10 Example of max pooling. 19

2.11 The structure of a simple RNN. The network consists of an input, a hidden node and an output. The hidden node has a weight connected to itself. This allows the hidden node to receive information from previous time steps. Figure 2.11a shows an illustration of the network, and the right figure 2.11b illustrates the network unfolded over time. 20

2.12 The LSTM architecture. 21

2.13 The ALVINN architecture (Pomerleau, 1989). 28

2.14 The DAVE-2 architecture (Bojarski et al., 2016). 30

2.15 DAVE-2 Training the neural network (Bojarski et al., 2016). 30

2.16 Command Input Network Architecture: Command is processed as an input (Codevilla et al., 2017). 32

2.17 Branched Network Architecture: The command works as a switch that activate the correct sub-module (Codevilla et al., 2017). 32

2.18 Complete architecture of DriveNet (Hubschneider et al., 2017) 34

2.19 C-LSTM Architecture unrolled across time. The CNN extracts features from input images. The feature vectors are sent to a stack of two LSTM layers. The sliding window allows the same frame to be used in different states of the LSTM. Finally, the output is sent to a classification layer that predicts the steering angle (Eraqi et al., 2017) 36

2.20 Encoding the sin wave to a driving steering angle and measuring training error. The blue arrow represent training and the red arrow represent deployment (Codevilla et al., 2017). 37

2.21 Examples of different weather conditions in Town 2 in CARLA. 40

2.22 A simple example of a computational graph. 41

3.1 File structure of a recording. 46

3.2 An example from the controller's overview mode. The green pentagon represents the hero vehicle, while the blue pentagons represent the non-player vehicles. The id of different spawn locations is marked on the road with red numbers. 47

3.3 The camera setup. Five virtual cameras (1-5) are mounted on the roof of the vehicle. 48

3.4 Camera 1-5's captured images from a single observation 49

3.5 Some of the weather conditions used when capturing data. 52

3.6 Gathering of data in rainy conditions using a steering wheel and pedals. . 53

3.7 Examples of data augmented images. 54

3.8 Distribution of the balanced training dataset. 55

3.9 System overview in training phase. 58

3.10 System overview in deployment phase. 59

3.11 The architecture of the CNN 60

3.12 The architecture of the LSTM 60

3.13 The training and deployment procedure of the steering angle prediction. The red arrows and boxes indicate the parts that are only activated during the training phase. 62

3.14 Three data-points generated from a single observation. Shifted steering angles are marked in red. 66

3.15 The structuring of sequences from an array of data-points. In this illustration a sequence length (α) of 5 and a sampling interval (β) of 3 is used. . 66

3.16 The paths of route 1-3 in Town 2. 68

3.17 The paths of route 4-5 in Town 4. 68

4.1 Average route completions for each sequence length in Town 2 from experiment 1. 73

4.2 Experiment 1 – Average route completions for each sequence length in Town 4. 75

4.3 Experiment 2 – Average route completions for each model trained with different dataset sizes in Town 2. 77

4.4 Experiment 3 – Average route completions for each prediction frequence in Town 2. 78

5.1 Comparison of the same turn in a simple (a) and a challenging (b) weather condition. The resolution of the images is equal to the system's input resolution, i.e., 350x160x3. 89

5.2 Examples of a vehicle the system struggled to detect in some weather conditions. Figure (a) shows a black vehicle in *Heavy Rain Noon*, while Figure (b) shows a vehicle in *Clear Wet Sunset*. 94

5.3 Examples where the system struggled to detect lane lines. Figure (a) shows the road in *Heavy Rain Noon*, while Figure (b) shows the road in *Clear Wet Sunset*. 96

D.1 *Town 1* in CARLA is situated in an urban environment and has 2.9 km of drivable road. It consists of roads with two lanes, one in each driving direction, and intersections with traffic lights. Additionally, there are multiple buildings and vegetations in the town. 128

D.2 CARLA's second town, *Town 2*, is also situated in an urban environment, but have different buildings from the first town and 1.4 km of drivable road. Many buildings are taller, creating more complex light conditions. 128

D.3 *Town 3* is in an urban environment with multiple lanes, a roundabout, and uphills and downhills. 129

D.4 *Town 4*, the fourth town is a combination of a highway and an urban environment. The highway has four lanes in the same driving direction. The urban environment can be accessed from the highway by taking on of several exits. 129

D.5 *Town 5* is quite similar to *Town 3*, except there are no roundabouts. . . . 130

Abbreviations

ALVINN	=	Autonomous Land Vehicle in a Neural Network
ANN	=	Artificial Neural Network
API	=	Application Programming Interface
CARLA	=	Car Learning to Act
CCIS	=	Communications in Computer and Information Science
CNN	=	Convolutional Neural Network
CPU	=	Central Processing Unit
CSV	=	Comma-separated values
CUDA	=	Compute Unified Device Architecture
DAVE	=	DARPA Autonomous Vehicle
FC	=	Fully Connected
GPS	=	Global Positioning System
GPU	=	Graphical Processiing Unit
HLC	=	High Level Command
LSTM	=	Long Short-Term Memory
MSE	=	Mean Squared Error
NAIS	=	Norwegian Artificial Intelligence Society
RC	=	Radio-Controlled
ReLU	=	Rectified Linear Unit
RGB	=	Red Green Blue
RMSE	=	Root Mean Squared Error
RNN	=	Recurrent Neural Network
SDG	=	Stochastic Gradient Descent

CHAPTER 1

Introduction

This chapter introduces the thesis by giving a short summary of the motivation and background in Section 1.1. Section 1.2 formulates the research questions and research objectives, while section 1.3 presents the contributions of the work. Finally, Section 1.4 outlines the structure of the thesis.

1.1 Background and Motivation

We are currently at the brink of a new paradigm in human travel: the fully autonomous, self-driving car. Only 50 years ago, cars were completely analog devices with almost no mechanisms for assisting the driver. Over the decades, additional features, controls, and technologies have been integrated, and cars have evolved into exceedingly complex machines.

In recent years, substantial progress has been made towards a vehicle's ability to operate autonomously. Primarily, two different approaches have emerged. The prevailing state of the art approach is to divide the problem into a number of sub-problems and solve them by combining techniques from computer vision, sensor fusion, localization, control theory, and path planning. This approach requires expert knowledge in several domains and often results in complex solutions, consisting of several cooperating modules.

Another approach is to develop an end-to-end solution, solving the problem using a single, comprehensive software component, e.g., a deep neural network. A technique for training such a system is to employ imitation learning. This entails studying expert decisions in different scenarios, to find a mapping between the perceived environments and the executed actions. While some believe that the black-box characteristics of such systems make them untrustworthy and unreliable, others point to recent years' advances in deep-learning and argue that end-to-end solutions show great potential.

An additional topic of discussion entails which type of sensors an autonomous vehicle requires. Some believe that it is necessary to use sophisticated sensors such as LiDARs and radars, alongside regular cameras. Others argue that since humans only use their eyes when driving, a collection of cameras provide a sufficient amount of information. This thesis will explore the possibilities of end-to-end systems using only visual cues from cameras.

1.2 Objectives and Research Questions

The overall goal of this thesis is to research an end-to-end system's ability to drive autonomously in diverse, simulated environments. The proposed system is to be rigorously tested in both urban and high-speed environments, in weather conditions ranging from sunny to heavy rain. Moreover, the system is to be tested at different times of the day, evaluating its ability to handle different light conditions. Thus, the following research goals have been formulated:

RG1: Research an end-to-end system's ability to drive autonomously in urban environments containing other vehicles, speed limits, and traffic-light regulated intersections.

RG2: Research an end-to-end system's ability to operate autonomously on highways alongside other vehicles.

RG3: Research an end-to-end system's ability to operate in challenging weather and light conditions.

Additionally, this thesis aims to explore some of the uncertainties regarding the implementation of an end-to-end system. Specifically, how the system's overall performance is affected by the size of the training dataset, the allowed prediction frequency, and the number of hidden states in the system's recurrent module. The size of the training dataset constrains how many examples the system is allowed to learn from, while the prediction frequency limits how many control-signals the model can produce each second. The number of hidden states regulates how many timesteps the system is able to ”look back”, thus constraining the length of the temporal dependencies the system can utilize. Hence, the following research questions have been formulated:

RQ1: How does the number of hidden states in the system's recurrent module affect the overall performance?

RQ2: How does the size of the training dataset affect the system's overall performance?

RQ3: How does the system's prediction frequency influence the overall performance?

1.3 Contributions

In the early work of this thesis, we tested whether an end-to-end system operating in simulated urban environments could improve its performance by utilizing temporal dependencies between subsequent visual cues. This was done by comparing two end-to-end architectures: a traditional Convolutional Neural Network and an extended design combining a Convolutional Neural Network with a recurrent layer. The results from the preliminary work were presented in (Haavaldsen, Aasboe, and Lindseth, 2019) and submitted to the Norwegian AI Society (NAIS) symposium, which took place on May 27-28, 2019, at NTNU in Trondheim. The paper was accepted as a full paper alongside an oral presentation and is currently in the process of being published in Springer's CCIS series. The paper can be read in Appendix A.

The proposed system in this thesis is a continuation of the work presented in (Haavaldsen, Aasboe, and Lindseth, 2019). The thesis seeks to combine different aspects from recent research to create a versatile end-to-end system able to operate in both diverse environments and challenging weather conditions. A lot of the research done in end-to-end learning for autonomous vehicle control today are limited to specific environments or a narrow range of weather conditions.

Furthermore, we seek to combine the use of navigational commands as network input and the exploitation of temporal dependencies between subsequent images. There have been no attempts - to our knowledge - to combine both techniques in one system. Hopefully, this can lead to a more complete end-to-end system.

Finally, by using an automated evaluation procedure, we hope to quantify the effect from some of the design choices typically made when implementing an end-to-end system for autonomous vehicle control. The proposed systems from related research are, more often than not, presented without explanations regarding some of the important design decisions,

e.g., the number of hidden states in a recurrent layer.

1.4 Thesis Outline

This section includes an overview of how the thesis is structured.

Section 1: Introduction

Contains an introduction to the problem the thesis aims to solve, as well as the research objectives, research questions, and the contributions of the thesis.

Section 2: Background

Covers related work, and introduces relevant theory for this thesis.

Section 3: Methodology

Describes the proposed system architecture, and explains training and testing of the system.

Section 4: Results

Describes the results of the experiments.

Section 5: Discussion

Contains a discussion of the choice of training data, the chosen simulator, and the system architecture. Additionally, the experimental results, consistency with related work, and fulfillment of the research questions are discussed.

Section 6: Conclusion and Further Work

Concludes the work, and proposes areas that could be further explored.

CHAPTER 2

Background

This chapter covers relevant theoretical aspects and history related to the thesis. Section 2.1 introduces the concepts behind artificial neural networks, while Section 2.2 explains some of the different state-of-the-art artificial neural network architectures. Important aspects of autonomous vehicle control and related work are discussed in Section 2.3 and Section 2.4, respectively. Finally, Section 2.5 introduces pertinent software and hardware to the thesis.

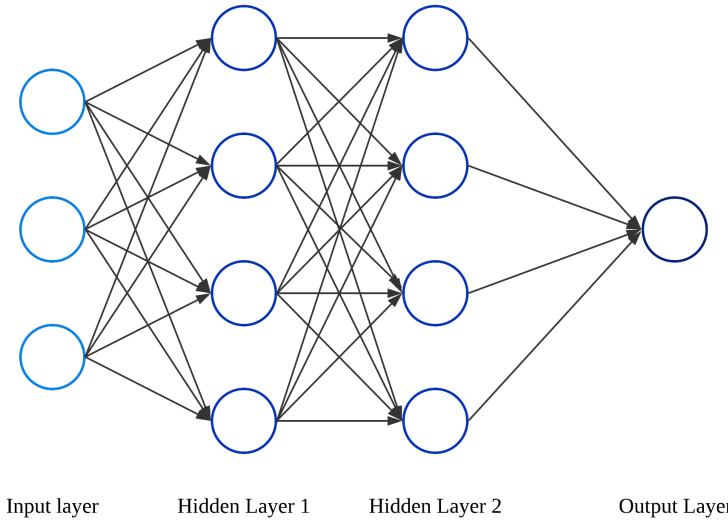


Figure 2.1: Structure of a Neural Network.

2.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational model loosely inspired by the biological brain. The model can learn complex and non-linear patterns by training on labeled data, generally without being programmed with any task-specific rules. The original goal of the ANN approach was to solve problems in the same way that a human brain would. Thus, an ANN can be viewed as a directional and weighted graph in which the nodes represent neurons, and the edges represent axons.

2.1.1 Network Structure

An ANN consist of layers of neurons connected by weighted edges. A network contains one input layer, one output layer, and one or several hidden layers. Each layer can contain any number of nodes. The input layer feeds the network's input into the next layer, the hidden layers transform the input into useful output, and the output layer produces the final prediction. The structure of an ANN is shown in Figure 2.1.

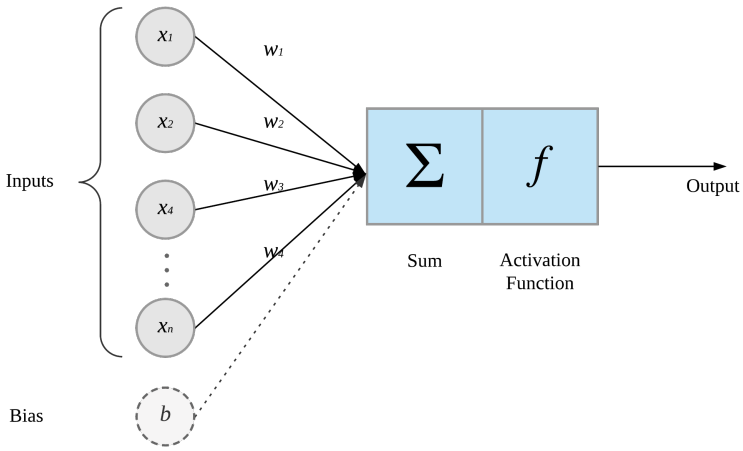


Figure 2.2: Activation of a neuron.

A single neuron can receive multiple inputs and produces a single output. The output is calculated by summing the weighted inputs, and passing the sum through an activation function, as seen in Equation 2.1. The addition of a bias to the sum of the weighted input allows the network to shift the activation function with a constant. The structure of a neuron is shown in Figure 2.2.

$$y = f(\sum_{i=1}^{n} x_i w_i + b) \tag{2.1}$$

Activation functions

To introduce non-linearity to a neural network, activation functions are applied to the output of each neuron. Typically, one wants the function to constrain the output value, be differential, and monotonic. One of the most straightforward activation function is the step-function seen in Equation 2.2. The function sets the output value to zero if the input is negative; otherwise, the output will be one. A drawback to this activation function is that a small change in the weights can result in either a large change, or no change, in the output.

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{2.2}$$

Another alternative is the Rectified Linear Unit activation function (ReLU), seen in Equation 2.3, which is one of the most commonly used activation functions. If the input is smaller than 0 the output becomes 0, else, it remains the same. Opposed to the step function, ReLU has no upper constraints, and it has a smaller chance of vanishing gradients.

$$f(z) = max(0, z) \tag{2.3}$$

The sigmoid activation function, seen in Equation 2.4, is a commonly used activation function that transforms the input value to a number between 0 and 1. It is monotonic and has a simple derivative, making it easy to work with. One drawback with sigmoid is that its derivative has a short range, which can lead to information loss in deeper neural networks. Furthermore, the sigmoid function can result in suboptimal weight convergence during training, one of the reasons being that strongly negative numbers all comes out very close to zero.

$$f(z) = \frac{1}{1 + e{-}z} \tag{2.4}$$

An alternative to the sigmoid function is the Hyperbolic Tangent (tanh) activation function, seen in Equation 2.5. The output of the tanh function looks similar to the sigmoid function, but the output ranges from -1 to 1. Thus, strongly negative inputs will map to negative outputs. Moreover, only input values close to zero will output close to zero values, making it easier for the weights in the network to converge during training.

$$f(z) = tanh(z) \tag{2.5}$$

2.1.2 Training

After a network architecture is decided, the network needs to be trained. During training, the weights between the neurons are adjusted in order to minimize the prediction error on the training data.

Training Procedure

First, the weights and biases are initialized. They can be chosen randomly or by a heuristic, for instance, using a normal distribution. Then, for each input-output pair in the training set, the input is forward propagated through the network to produce a prediction, and a prediction error is calculated using a loss function.

Next, backpropagation is performed to find the optimal weight change for the network. Backpropagation calculates the gradients – that is, the partial derivatives of the loss functions in regards to the weights – in each successive neuron and backpropagates the error throughout the network, adjusting the weights according to the gradients. The specific weight update depends on the chosen optimization function.

One epoch is when all the training examples are passed both forward and backward through the neural network once. Usually, the training procedure performs several epochs before the weights that yield the minimum loss are found.

Loss Functions

A loss function calculates the difference between the output of an ANN and the target value. There exist many different loss functions for different problem types. Regression loss functions are usually used when the output is continuous, while classification functions are usually used when the output is discrete.

MSE Mean Squared Error (MSE) is a regression loss function. It measures the mean absolute value of the difference between the target and the prediction. The MSE equation is shown in Equation 2.6.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.6}$$

RMSE Root Mean Squared Error (RMSE) is similar to the MSE loss function, but takes the square root of the mean squared error, making the scale of the error similar to the target scale. This functionality has implications for the loss functions, namely that the RMSE weights large errors more than MSE.

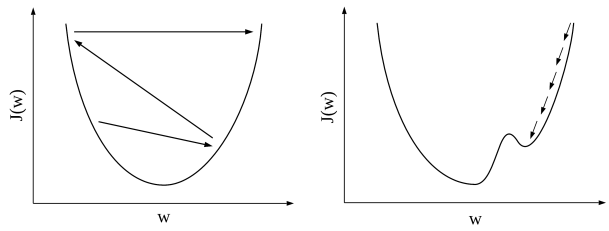
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.7}$$

Optimizers

Optimization algorithms are an essential part of a neural network's training and are mathematical functions dependent on the learnable parameters in the network. The objective of an optimizer is to update the weights such that the loss function is minimized.

Stochastic Gradient Descent The Stochastic Gradient Descent (SGD) optimizer calculates weight gradients and updates the weights for each data point. The gradient of a weight is the partial derivative of the loss in regards to the weight. The weights are then updated with a factor of the gradient, determined by the learning rate. The weight update used in SGD is shown in Equation 2.8

$$w_{i,j}^{t+1} = w_{i,j}^{t} - \alpha\frac{\partial L}{\partial w_{i,j}^{t+1}} \tag{2.8}$$



(a) Large learning rate **(b)** Small learning rate

Figure 2.3: The effect of the learning rates. Figure (a) illustrates a large learning rate that may lead to suboptimal weight convergence. Figure (b) illustrates a small learning rate that lead to slower training and the risk of being stuck in a local minima.

Adam Optimizer The Adam optimizer is an extension of SGD. It calculates an adaptive learning rate based on the first moment (mean) and the second moment (variance) of the gradient. It maintains one learning rate for each parameter. Adam is considered to be computationally efficient, suitable for a large number of parameters, good at handling sparse gradients, and often require little tuning of the hyperparameters. Adam's weight update can be seen in Equation 2.9, where $\hat{m}^t$ and $\hat{v}^t$ is the first and second moment of the gradient, and ϵ is a constant with default value 10^{-8}

$$w^{t+1} = w^t - \frac{\alpha}{\sqrt{\hat{v}^t} + \epsilon}\hat{m}^t \tag{2.9}$$

Learning Rate

The learning rate determines how much the weights are updated during training. If the learning rate is too low, many weight updates are required to reach a minimum, resulting in slow training of the network. Furthermore, it increases the chance of the network being trapped in a local minimum. On the other hand, if the learning rate is too large, the network's weights may never converge. Instead, they oscillate between values as a result of too large weight updates. The learning rate is denoted as α in Equation 2.8 and Equation 2.9, and an illustration of different learning rates can be seen in Figure 2.3.

Momentum

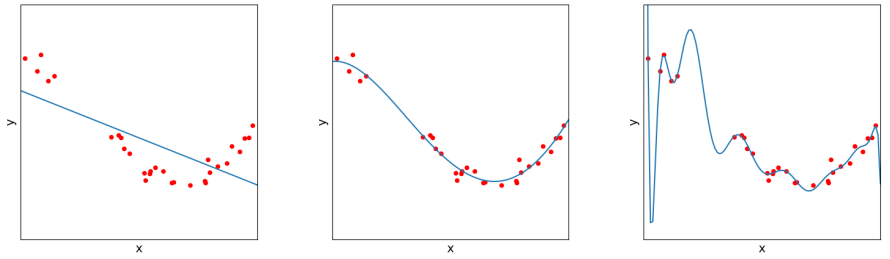
Loss surfaces often contain plateaus, as seen in Figure 2.3. Plateaus can increase the risk of the weights being stuck in a local minimum. Adding momentum to the weight update helps to alleviate this problem. The idea is to use a portion of the previous gradient when updating the weights. This may also speed up the convergence. Weight update with momentum is seen in Equation 2.10, where γ dictates how much the weigth update is affected by the previous gradient.

$$w_{i,j}^{t+1} = w_{i,j}^{t} - \alpha \frac{\partial L}{\partial w_{i,j}^{t}} + \gamma \Delta w_{i,j}^{t-1} \tag{2.10}$$

2.1.3 Regularization

When training is complete, the neural network should be able to perform well on unseen data. However, this may not always be the case. A model that performs well during training but worse on unseen data may be overfitted. Overfitting implies that the network has learned patterns that are too specific to the training data. On the other hand, a model that performs poorly on both the training and test data may be underfitted. Underfitting implies that the model was not able to learn the necessary patterns in the domain. This can be due to poor network architecture or too little training. Overfitting is illustrated in Figure 2.4c and underfitting is illustrated in Figure 2.4a. To acquire good generalization, a network must train enough to make good predictions, but not so much that it overfits, illustrated in Figure 2.4b. There are several regularization techniques to prevent a network from overfitting.

Early Stopping One way to prevent overfitting is to adjust the number of times the dataset is used to adjust the weights. This technique is called early stopping, and the goal is to measure how well a model performs on unseen data. The idea is to split the training data into a training set and a validation set. For each iteration of the training, the training set is used to update the weights, while the validation set is used to measure the network



(a) Underfit **(b)** Ideal fit **(c)** Overfit

Figure 2.4: Underfitting vs. overfitting, (a) illustrates an underfitted model, (b) illustrates an ideal fit, while (c) illustrates of an overfitted model.

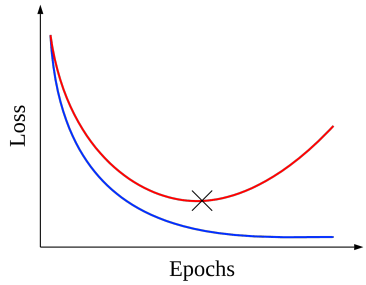
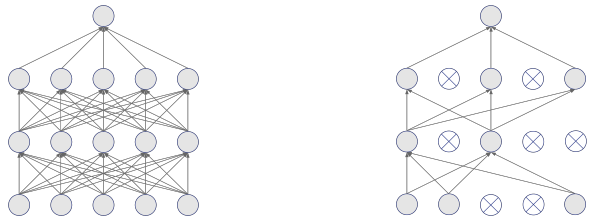


Figure 2.5: Early stopping. The blue graph represents the training loss, while the red graph represents the test loss. The cross indicates the smallest test loss and indicates where the training should stop.

performance. Using a separate dataset to measure the performance, ensures that the model can be evaluated on data it has never seen during training. When the validation loss stops decreasing, the training should stop, such that the final model will be the one with the best validation performance. In Figure 2.5, the cross indicates the smallest validation error, where the training should stop.

Dropout Dropout is another regularization technique. The idea is to drop arbitrary neurons and their connections during training. This results in different thinned versions of the network, as illustrated in Figure 2.6. Dropout prevents internal connections in the network from becoming codependent on each other. During testing, the model uses the original network, where the weights are the average from all the networks from training.



(a) Original network without dropout **(b)** A thinned version of the original network after applying dropout

Figure 2.6: Example of dropout in a neural network

Data Augmentation Data augmentation is a technique to synthetically expand the dataset by adding transformed versions of the data. Data augmentation is a valuable technique in deep learning, where the models require large datasets to perform well. The data samples should be transformed such that it keeps the valuable features in the data but expand the dataset. For instance, a model that classifies animal species should be able to make a prediction regardless of the animal's pose and rotation in the image. In this case, data augmentations as flipping, changing the perspective, and rotating the images can be useful, as seen in Figure 2.7d and Figure 2.7e. Data augmentation is especially helpful when working with images, videos, and text sets, and can make classifications more robust against irrelevant conditions.

2.2 ANN Architectures

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is a particular class of ANNs, often used for image analysis and classification. It uses images directly as input and learns to extract a hierarchy of relevant features that can be used for classification. Where a typical ANN would flatten the input to a one-dimensional vector, a CNN keeps the original structure, preserving the spatial structure of the input. A CNN can learn which features that define an object independently of its location in the image.

(a) Original image

(b) Adjusted lightness

(c) Adjusted hue

(d) Flipped image

(e) Perspective transformation

Figure 2.7: Examples of image transformations

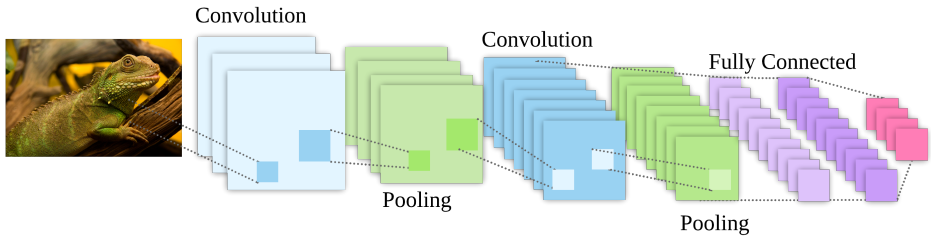
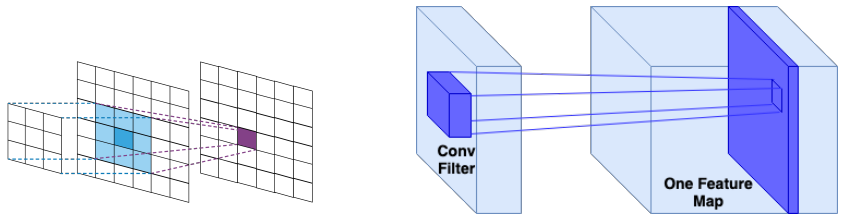


Figure 2.8: The general architecture of a CNN.



(a) Convolution **(b)** Depth of output

Figure 2.9: Convolution and depth of output

A CNN consists of three main components: convolutional layers, pooling layers, and fully connected layers at the end. The architecture of a CNN is shown in Figure 2.8.

Convolutional Layers

The purpose of the convolutional layers is to extract features from the input. It slides a filter across the input while computing the dot product to produce a feature map. The filters act as feature detectors. Each layer can have several different filters, and the number of filters dictates the depth of the output. A convolutional layer is illustrated in Figure 2.9. The filter moves a given amount between each convolution, defined by the stride. Additionally, an input may be padded to prevent a dimension reduction.

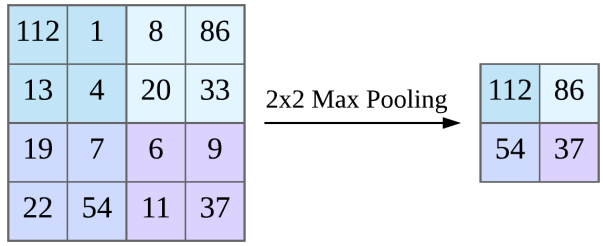


Figure 2.10: Example of max pooling.

Pooling Layer

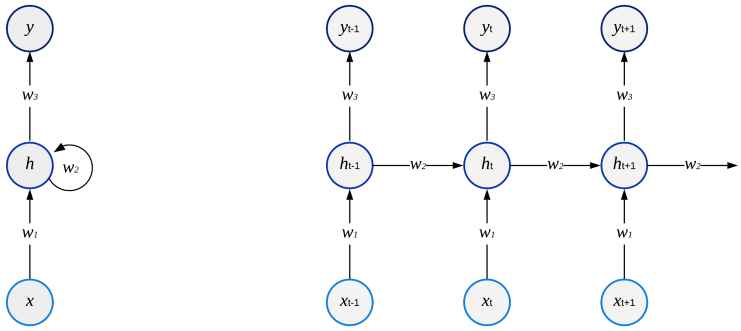
The purpose of the pooling layers is to reduce the dimensions of the input feature maps. This leads to a reduction of parameters in the network, which improves the computational efficiency and controls overfitting. It also improves the network's invariance to small transformations in rotation and position. The pooling is done by sliding a window over the feature maps, producing a single output for each step. There exists different types of pooling: max pooling (selects the maximum value in the region), average pooling (outputs the average value of the region) and sum pooling (outputs the sum of the region). An example of max pooling can be seen in Figure 2.10.

Fully Connected Layer

At the end of the convolutional network, one or several fully connected (FC) layers are added. The output of the previous layers represent the high-level features in the image. The FC layers use these features to classify the input image.

2.2.2 Recurrent Neural Networks

Recurrent Neural Network (RNN) is another subclass of ANNs. Opposed to feed-forward neural networks, RNNs can create connections between units in both directions, as seen in



(a) A folded RNN. **(b)** An unfolded RNN.

Figure 2.11: The structure of a simple RNN. The network consists of an input, a hidden node and an output. The hidden node has a weight connected to itself. This allows the hidden node to receive information from previous time steps. Figure 2.11a shows an illustration of the network, and the right figure 2.11b illustrates the network unfolded over time.

Figure 2.11. This gives the network temporal dependencies, making it able to learn from sequential data.

The RNN can operate on vector sequences in both the input and output of arbitrary weights. It can handle one-to-one, one-to-many, many-to-one, and many-to-many input/output relationships, making it suitable for various applications, such as classifying a text (many-to-one), captioning an image (one-to-many), analyzing a sequence to make a classification (many-to-one), and speech recognition (many-to-many).

Long Short-Term Memory Neural Networks

Long Short-Term Memory (LSTM) Neural Networks is a type of RNN that excels at handling both long-term and short-term dependencies. It follows the general idea of an RNN, but the recurrent module has a unique structure.

The LSTM cell has three inputs: a hidden state, a data input, and a cell state. The cell state holds long-term dependencies. At each time step, important memory is kept, and new relevant information is added. The hidden state interacts with the data and chooses what

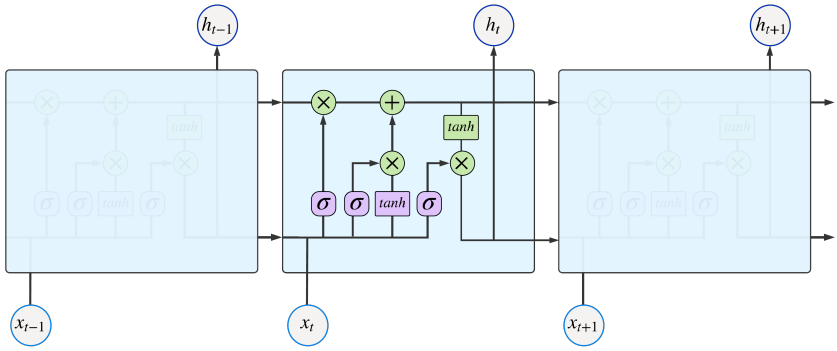


Figure 2.12: The LSTM architecture.

information should be forgotten, and what should be added to the cell state. The output is a filtered version of the new cell state. The architecture of an LSTM can be seen in Figure 2.12.

2.3 Autonomous Vehicle Control

2.3.1 Mediated Perception

Mediated perception is defined by Chen et al. (2015) as an approach that divides the task of autonomous vehicle control into several sub-problems and solves them independently by combining techniques from computer vision, sensor fusion, localization, control theory, and path planning. The results from each component are combined to create a cohesive, high-dimensional, representation of the vehicle's surrounding, that is passed to a decision system that chooses the vehicle's next action. Mediated perception is considered to be the current state of the art approach, and are incorporated in most autonomous vehicles developed by the industry today.

The different sub-problems in mediated perception is often researched separately. Aly (2008) studies real-time detection of lane markings in urban settings, while Felzenszwalb et al. (2010) and Lenz et al. (2011) studied detection of cars and marking them with bound-

ing boxes. These approaches can suffer from false detections, and Geiger et al. (2014) and Zhang et al. (2013) integrate different sources to create a world representation and proposes a probabilistic generative model using various detection results to create the final world representation.

Even though mediated perception is considered the current state-of-the-art approach, there are some limitations to this method. Mediated perception compute a high-dimensional representation of the vehicle's surrounding. However, the task of driving a car only requires manipulation of speed and direction. Taking this into account, along with the fact that only a small portion of the detected objects may be relevant, this world representation may contain redundant information and unnecessary complexity. Moreover, the information from the different components often needs to be converted into the same representation format, which can add noise to the data. Lastly, these approaches require a combination of sensors, e.g., LiDAR, GPS, radars, and very accurate maps, to parse a scene reliably, making it a costly approach.

2.3.2 End-to-end Learning

End-to-end learning is another approach in vision-based autonomous vehicle control which tries to solve the problem using a single, comprehensive software component, e.g., a deep neural network. Such a system automatically learns to map sensor input to actions.

The approach comes with a great advantage, namely the capability to automatically learn which features of the input that is relevant. Deep learning systems' ability to automatically extract important features have shown to result in significant performance boosts in other, similar domains. Moreover, feature extraction is known to be a cumbersome process, and can easily bottleneck a system if not done right.

Even though mediated perception approaches are considered the current state-of-the-art solutions, recent breakthroughs in deep learning have significantly increased end-to-end systems' capabilities, and such systems are now considered a possible alternative.

End-to-end via Imitation Learning

Imitation learning is a supervised learning technique that aims to train a model to mimic an expert's behavior. The model is trained using a dataset containing observations and associated expert decisions. Given an observation o_i, the model predicts an action $\hat{a}_i = F(o_i, \theta)$. The objective is to optimize a set of parameters θ, such that the difference between the estimated action, $\hat{a}_i$, and the expert's action, a_i, is minimized. This can be expressed as Equation 2.11.

$$\underset{\theta}{\text{minimize}} \quad \sum_i L(F(o_i; \theta), a_i) \tag{2.11}$$

This technique assumes that the expert action is solely dependent on the observation. However, this does not hold for all driving scenarios. Imagine a situation where the model should predict a steering angle in an intersection. Without knowing the user's intent, the model has no way of knowing whether to turn right, left, or continue straight ahead. Thus, the model should incorporate the user's intent, h_i in the decision making. This can be expressed as Equation 2.12.

$$\underset{\theta}{\text{minimize}} \quad \sum_i L(F(o_i, h_i; \theta), a_i) \tag{2.12}$$

Limitations in End-to-end Learning

The end-to-end learning approach is a promising research field in autonomous research control, but there exist several challenges when choosing this approach.

Firstly, while its reflexive behavior makes it suitable for the task of autonomic vehicle control, the direct mapping from the input sensors to the predicted output also makes high-level planning more challenging than in mediated perception. For instance, a lane change from the system's perspective would be a sequence of small changes in the steering in one

direction and then in the other direction for some time. This level of abstraction neglects to capture what is truly going on and increases the difficulty of the task.

Moreover, end-to-end approaches utilize deep neural networks. Thus, they are faced with the same challenges deep learning models exhibit today, such as possible convergence into a local minimum and vanishing gradients (Shalev-Shwartz et al., 2017). Some general limitations of deep learning are mentioned below.

Training Efficiency A complex problem often consists of several different and separate subproblems. It can be challenging and inefficient to solve all these problems simultaneously using a single model. (Mnih et al., 2015) used a deep reinforcement approach, where the module responsible for handling visual representation and the module representing the policy were trained together. It can be argued that it would be more efficient to train the modules independently or to start with a pre-trained neural network (Krizhevsky et al., 2012).

The Black-box Problem End-to-end learning can essentially be viewed as a sophisticated curve-fitter, training on examples to estimate a general function. It can be challenging to predict how an end-to-end model will respond when faced with an unexpected outlier value. Additionally, it can be troublesome to reason about the cause of a model failure, thus making it difficult to prevent future failures of a similar nature. This can be a significant challenge in some critical environments, e.g., a medical diagnostic system or an autonomous vehicle.

A Data Hungry Approach For an end-to-end approach to work well, a large and varied dataset is often required. For applications such as autonomous driving, that has a wide area of scenarios, it can be challenging to get a large enough dataset to cover necessary scenarios and conditions.

Hierarchical Structure Several end-to-end models utilize recurrent neural networks to learn temporal dependencies. (Marcus, 2018) finds that recurrent neural networks currently have no natural understanding of hierarchical structure. As an example, while a language model only views sentences as sequences, it is often argued that languages have a hierarchical structure, where larger structures are constructed recursively out of smaller components. (Lake and Baroni, 2017) tested various RNNs' ability to generalize. The models were trained in a many-to-many relationship that generated commands from a sentence. They showed that an RNN can generalize well if the commands in the training data and test data are similar, but that when the network requires systematic compositional skills, an RNN "fail spectacularly". This drawback is likely to apply to other domains as well, e.g., application in planning and motor control. The main problem is that deep learning models learn dependencies in flat structures, and are therefore forced to learn sub-optimal techniques to compensate for this downfall (Marcus, 2018).

2.3.3 Training and Testing

Training and testing models for autonomous driving in the physical world can be expensive and impractical. Gathering a sufficient amount of training data requires both human resources and suitable hardware, and it can be challenging to organize and capture the desired driving scenarios effectively. Moreover, the cost of unexpected behavior while testing a model may be colossal.

One possible alternative is to train models and test models using publicly available datasets. There exist several large open-source datasets specialized for training self-driving cars. The largest one, *BDD100K*(Yu et al., 2018) released by UC Berkely, contains 100,000 video sequences of driving in diverse environments, each approximately 40 seconds. The dataset contains rich annotations such as image tagging, marking of drivable areas, road object bounding boxes, lane markings, and full-frame instance segmentation. The videos and their trajectories may be useful for imitation learning of driving policies, but the annotations can also be used to test a model's ability to solve specific sub-problems of autonomous

driving as object detection, lane marking prediction, and image segmentation. However, this approach does not solve the challenges related to real-time testing of the model in the physical world.

A third alternative is to train and test models in a simulated environment. A simulator can effectively provide the variety of corner cases needed for training, validation, and testing; while removing any safety risks and material costs. The drawback, however, is the loss of realism. A simulation is only an imitation of a real-world system, and a model trained on only simulated data may not be able to function reliably in the real world. Nonetheless, a simulator can give a good indication of a model's actual driving performance and serves well for benchmarking different models. It is also common to run quality assurance tests in a simulated environment before trying the model in the real world.

2.3.4 Data Augmentation

Deep learning is known to be a data-hungry approach, and data augmentation is often used to artificially expand the dataset, as described in Section 2.1.3. In end-to-end learning for autonomous vehicle control, the dataset is comprised of images, making it especially fitting for data augmentation. However, certain domain-specific aspects constrains the use of data augmentation.

When driving, properties such as lane choice, the pose of vehicles in the opposite lane, and traffic rules holds valuable information. If these are lost in the augmentation, the quality of the dataset can be reduced, hurting the end performance of the model. For instance, horizontal flipping of the images will double the dataset and presumably improve a model's ability to follow lanes. However, it may also increase the risk of a model driving into the opposite lane, and it can also lead to a weaker understanding of important traffic rules, e.g., right of way.

Moreover, if the vehicle's camera system are fixed, some image properties – such as the point of view, size, and rotation – holds valuable information. Consequently, it might hurt

the model's performance to train on transformed images that represent different camera angles and skew. However, if a model is designed to apply for different camera setups; data augmentations such as perspective transformations, skew, and zoom can be beneficial.

Even though augmentations such as rotational and perspective transformations can harm the end performance, there exists several data augmentations that are beneficial. A robust model should be able to handle different light conditions and weather conditions, and changing the brightness, adding blur, and generating shadows could help to achieve this. Furthermore, adding images with hue transformations can help the model disregard the color of other non-player vehicles. However, it can also refrain from learning important color features in the data; for instance, the color of the lane lines.

2.4 Approaches in End-to-end Learning

There have been several advances in end-to-end learning for autonomous vehicles over the last decades; the first approach was seen already in 1989. In 2003 a proof-of-concept project called DAVE emerged. The authors trained a neural network that could steer a radio-controlled (RC) vehicle around in a junk-filled alley while avoiding obstacles. DAVE truly showed the potential of an end-to-end approach. Thirteen years later, NVIDIA developed DAVE-2, a framework with the objective to make real vehicles drive reliably on public roads. DAVE-2 is the basis for many end-to-end approaches seen today.

In this section, important advances in end-to-end learning are discussed, especially those of significance to the proposed method in this thesis. The early advances above are discussed, as well as relevant end-to-end approaches that utilize conditional imitation learning, spatial and temporal dependencies.

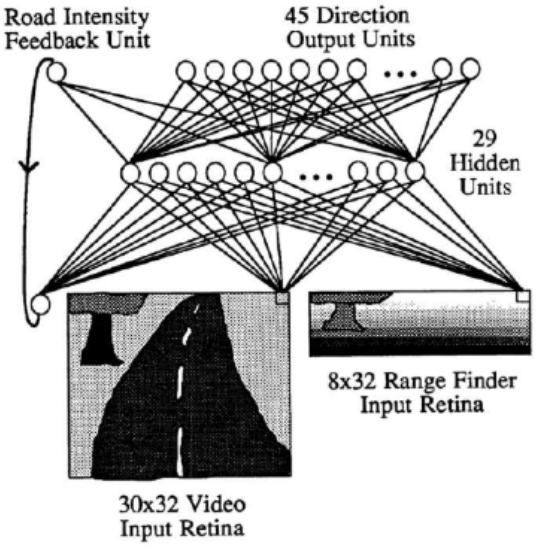


Figure 2.13: The ALVINN architecture (Pomerleau, 1989).

2.4.1 ALVINN

An Autonomous Land Vehicle in a Neural Network (ALVINN) is the first known end-to-end approach for autonomous vehicles and was published in 1989, (Pomerleau, 1989). At that time, vision-based navigation algorithms were considered the state of the art approach. Using a neural network was a novel approach, and ALVINN has inspired many end-to-end approaches in recent time.

ALVINN consists of a fully connected network with three layers, and receives input from a camera and a laser range finder and predicts a steering angle. The network's input layer has 1217 neurons: 960 neurons dedicated to handling road images, 256 neurons dedicated to the laser input, and one neuron that holds information about the road intensity. The last neuron is a feedback neuron and indicates whether the road is lighter or darker than the non-road part of the image. Figure 2.13 presents the complete network architecture.

Results showed that ALVINN is able to follow real roads under certain conditions, with almost as good results as that time's vision-based navigation algorithms.

2.4.2 DAVE

DARPA Autonomous Vehicle (DAVE) is a project sponsored by the US government from 2003. The objective of the project was to create a proof-of-concept of an end-to-end vehicle system. The authors built an RC vehicle with two cameras mounted on each side of the car. The car transmits the videos to a computer, which again controls the car via radio signals.

A neural network, consisting of six convolutional layers, controls the RC vehicle. The network takes an image pair in YUV color space as input and predicts the steering angle as output. It used hours of human driving data and trained the model for 18 epochs (LeCun et al., 2005).

The DAVE system is able to drive through an alleyway filled with junk. On average, it can avoid obstacles and navigate in an off-road setting for 20 meters without crashing (Lecun et al., 2004). The project was considered a success and contributed to the decision of continuing the research.

2.4.3 DAVE-2

DAVE-2 is an effort from NVIDIA to create a reliable self-driving vehicle that can safely drive on public roads (Bojarski et al., 2016). The project bases its work on the findings in (Lecun et al., 2004).

The authors propose using a CNN consisting of nine layers: one normalization layer, five convolutional layers, and three fully connected layers. The network takes images with YUV color space as input and predicts steering commands. Figure 2.14 shows the complete architecture.

DAVE-2 collects data from three cameras mounted on a vehicle and reads the vehicle's corresponding steering commands via a CAN bus. A balanced training set was created by carefully sampling data from 72 hours of human driving data - collected from various road types and weather conditions. Furthermore, the training data was augmented with shifts

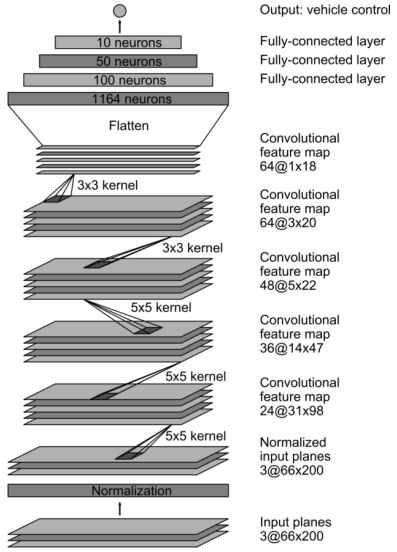


Figure 2.14: The DAVE-2 architecture (Bojarski et al., 2016).

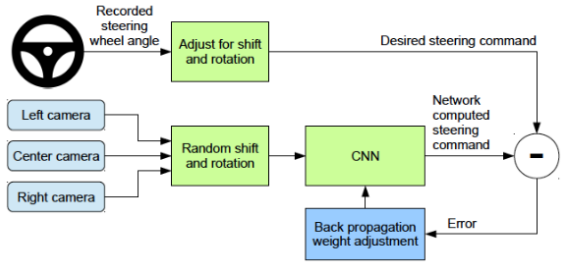


Figure 2.15: DAVE-2 Training the neural network (Bojarski et al., 2016).

and rotations such that the model could learn to recover from mistakes.

During training, images are fed into a CNN to predict a steering angle. The prediction is compared to the actual steering angle, and the weights in the CNN are adjusted with backpropagation to minimize the error. The system architecture is showed in Figure 2.15. When training is complete, the model can generate steering commands from a centered camera.

The results show that the model can drive on trafficked roads with or without lane markings, parking lots, and unpaved roads. It also performs well with a small training dataset.

2.4.4 End-to-end Driving via Conditional Imitation Learning

Current end-to-end approaches require that the action is entirely dependent on its environments; this does not hold when the model needs to make navigational choices. Then, the choice should also depend on the driver's intent. (Codevilla et al., 2017) suggest an end-to-end approach utilizing imitation learning to incorporate the drivers intent.

To solve the limitation of Equation 2.11, the authors introduce *Conditional Imitation Learning*, which frees the model from making navigational decisions. Instead, navigational commands are given as input during training, allowing the model to function in scenarios that require decisions. This is expressed by Equation 2.13.

$$\underset{\theta}{\text{minimize}} \quad \sum_{i} L(F(o_i, c_i; \theta), a_i) \tag{2.13}$$

Here, the approximator F is represented by a deep neural network. Furthermore, the authors introduce a new input $c = c(h)$, where h represents the expert's intentions, and c is a control command given by the expert. The control command given by the expert is dependent on the expert's intention, and is therefore expressed as a function of h. By adding an input, the model should be able to handle more scenarios than before and providing the ability for real-time control of the model.

The network takes an image, the vehicle's measurement, and a navigational command as input, and predicts a steering angle and an acceleration. Two different architectures were tested:

Command Input Network: In this architecture, the inputs are processed independently. A CNN processed the image, and separate fully-connected networks processed the command and measurement inputs. The output from each network is concatenated and sent through a fully connected network. This architecture can handle both continuous and discrete commands, but also introduces the risk of user commands being ignored at test time. The Command Input Network is shown in Figure 2.16.

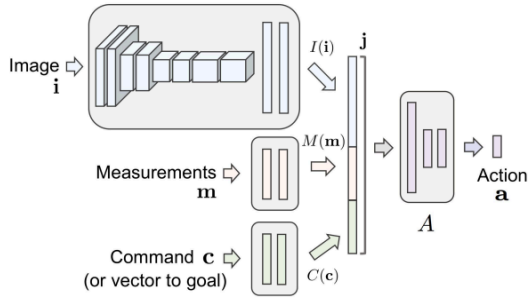


Figure 2.16: Command Input Network Architecture: Command is processed as an input (Codevilla et al., 2017).

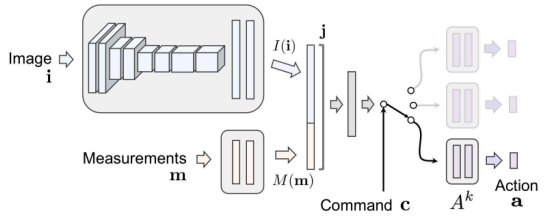


Figure 2.17: Branched Network Architecture: The command works as a switch that activate the correct sub-module (Codevilla et al., 2017).

Branched Network: This network assumes discrete commands $C = c0, ..., ck$. In this network, the image and measurements are still processed separately by a CNN and a fully connected network. However, there is one fully connected network for each command to determine the final action. Each fully connected network is a ”specialist” network for one specific command. The outputs from the image network and measurement network are concatenated and sent to the network that corresponds to the command c. Here the command works as a switch; that way, the command will always have an impact on the results at test time. Figure 2.17 shows the architecture of the Branched Network.

The two approaches were both tested in the CARLA simulator and with an RC vehicle in a residential area. The branched model outperformed the goal-oriented model and the other benchmark models significantly.

2.4.5 Adding Navigation to the Equation: Turning Decisions for End-to-End Vehicle Control

(Hubschneider et al., 2017) examines an end-to-end approach that supports lane following, obstacle avoidance, and has navigation abilities. The paper is inspired by (Codevilla et al., 2017), and also tries to incorporate the user's intent to the model. The authors suggest using the turning signals of the vehicle as input to a neural network. Furthermore, they suggest a modified network architecture to improve driving accuracy. Specifically, they propose using three separate CNNs as encoders that take three following images to exploit spatial dependencies between the frames.

The network takes an image input and a turn indicator as input such that the model can be controlled in real time. To handle sharp turns and obstacles along the road the authors propose using two additional images recorded several meters back to obtain a spatial history of the driving data, and images captured 4 and 8 meters behind the current position is added respectively. The architecture for the CNNs is a modified version of the architecture in 2.4.4, which they call DriveNet and can be seen in 2.18. This model takes three RGB images as input to account for the spatial history. Each image is run through separate encoders simultaneously. The learned features are sent to a stack of fully connected layers. At the second fully connected layer, the turning signals are added as input. To make sure the turn indicators have sufficient effect on the network, a constant factor, λ, is added to scale the weight of the turn indicator. Based on evaluation, the constant is set to $\lambda = 100$.

The authors also experiment with using max pooling instead of stride for dimension reduction to prevent losing features during strides. They also tested using 3x3 convolutions instead of 5x5 convolutions to reduce the complexity but keeping the expressiveness. The final model combines 2x2 max pooling and 3x3 convolutions and resulted in the architecture seen in Table 2.1.

The authors collected 5 hours of driving data, using a single centered camera and the same driver for all the data. The data was captured in various light conditions and street types.

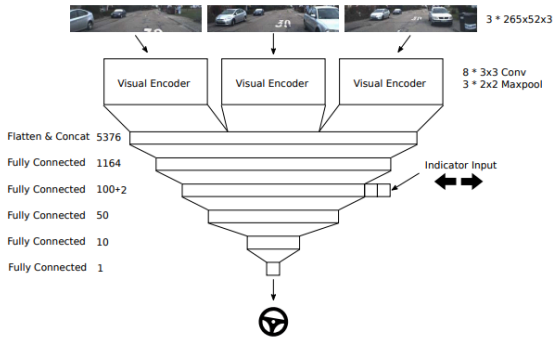


Figure 2.18: Complete architecture of DriveNet (Hubschneider et al., 2017)

Type	Kernel Size	Stride	Output
Input	-	-	52x265x3
Convolutional	3x3	1	50x263x24
Convolutional	3x3	1	48x261x24
Max Pooling	2x2	2	24x130x24
Convolutional	3x3	1	22x128x36
Convolutional	3x3	1	20x126x36
Max Pooling	2x2	2	10x63x36
Convolutional	3x3	1	8x61x48
Convolutional	3x3	1	6x59x48
Max Pooling	2x2	2	3x30x48
Convolutional	3x3	1	3x30x64
Convolutional	3x3	1	1x28x64

Table 2.1: Architecture of the DriveNet's visual encoder.

Images with distortions and shearing were added to extend the dataset. Moreover, various yaw angles were added by cropping input images off-center.

The model was tested with speeds up to 100 km/h and a camera sample rate of 21 Hz, which led to the vehicle moving at most 1.32 m at most between two frames. The paper showed that utilizing three visual encoders had the best accuracy, while their network with one visual encoder had the best performance time. The final model can perform reliable lane following on a highway but struggles to position itself in the middle of the lane when there are shadows on the road. It can perform and lane changes when triggered by a turn indicator. However, it is not able to check if a lane change is safe; it completely trusts the turn indicators. Furthermore, the model can drive around parked vehicles and tackle oncoming vehicles in areas with narrow roads without lane markings.

2.4.6 End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies (2017)

The approaches mentioned in the sections above all used CNNs to map each image input independently into steering commands. While the results were good, none utilized the temporal dependencies in the data.

(Eraqi et al., 2017) tries to utilize the temporal dependencies by combining a CNN with a Long Short-Term Memory Neural Network (C-LSTM). Additionally, the authors proposed to formulate the steering angle prediction as a classification task, imposing a spatial relationship between the output layer neurons. This was done by learning a sinusoidal function to encode the steering angles.

The system takes images from a centered RGB camera and feeds it into a CNN. The CNN learns important features and sends an output feature vector of length 2048 to an LSTM to learn temporal dependencies between the frames. The output from the LSTM is sent to a classification layer that predicts the steering angle. The authors found a sequence length of 5 seconds of driving to be appropriate. Figure 2.19 shows the model architecture. The

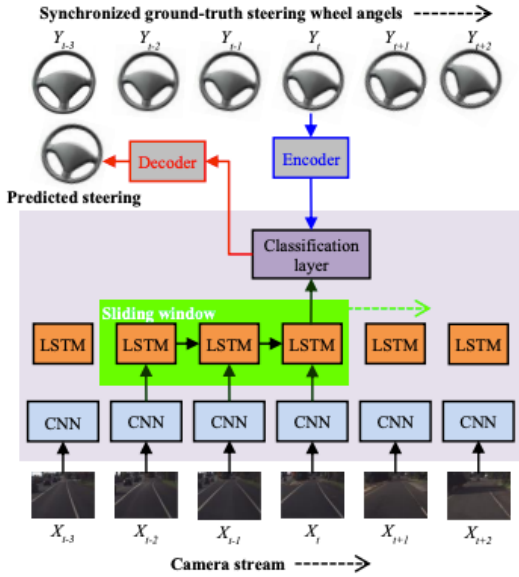


Figure 2.19: C-LSTM Architecture unrolled across time. The CNN extracts features from input images. The feature vectors are sent to a stack of two LSTM layers. The sliding window allows the same frame to be used in different states of the LSTM. Finally, the output is sent to a classification layer that predicts the steering angle (Eraqi et al., 2017)

output from the LSTM is sent to a fully-connected layer with a tanh activation function. This is where the final classification is performed. The authors suggest encoding the output as a sin wave and letting the steering angle corresponding to the phase shift. This encoding is given in Equation 2.14.

$$Y_i = \sin\left(\frac{2\pi(i-1)}{N-1} - \frac{\phi\pi}{2\phi_{max}}\right), \quad 1 \leq i \leq N \tag{2.14}$$

Y_i is the activation of neuron i, and N is the number of classes of the output layer. They use $N = 95$ neurons and a max steering angle of $\phi_{max} = 190°$. Gradual changes in steering angle will then lead to gradual changes in the output layer's activations.

During training, the ground truth steering angle is encoded as a sine wave, as seen in Equation 2.14. The loss is calculated as the root mean square error (RMSE) between the ground

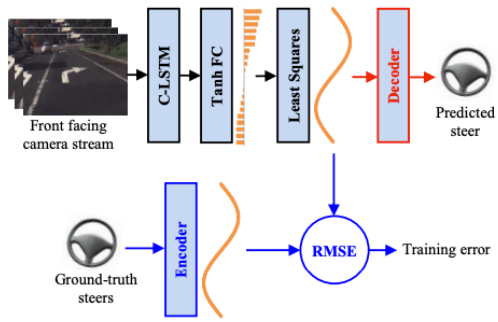


Figure 2.20: Encoding the sin wave to a driving steering angle and measuring training error. The blue arrow represent training and the red arrow represent deployment (Codevilla et al., 2017).

truth sin wave and predicted wave. The loss is used in the backpropagation algorithm to update the weights. During deployment, a decoder transforms the prediction from the Least Squares to a predicted steering angle. This can be seen in Figure 2.20

The model is trained on 7.26 hours of human driving from a database by ”comma.ai” in (Santana and Hotz, 2016). It had data of both highway and city driving in varying light conditions.

The authors compared the results of their C-LSTM model with state-of-the-art CNN approaches, testing the systems on the publicly released database by *comma.ai*. Furthermore, the classification using sine wave fitting was compared to the traditional regression approach. The results showed that the C-LSTM with sine wave fitting as classification improved the angle prediction accuracy by 35 % and the steering stability by 87 %.

2.4.7 Deep Reinforcement Learning

Another hot topic of research in autonomous vehicle control is to develop an autonomous agent using deep reinforcement learning.

Deep reinforcement learning agents solve Markov decision processes. In short, this entails modeling the autonomous driving task as a process where an agent makes actions in an

environment and receives a reward signal for said action. For lane following a reward signal could, for instance, be the vehicle's offset from the center of the lane, combined with the vehicles traveled distance. However, creating such a heuristic can be challenging, as the center of the lane may not always be evident on different types of roads. Additionally, there are many other factors which affect the systems overall performance when operating a vehicle. It is not always easy to represent these factors as heuristic functions.

There has currently been limited research in deep reinforcement learning for autonomous vehicle control, perhaps due to the safety concerns of training and testing deep reinforcement agents. In order for such agents to learn, they need to explore the environment and be able to make mistakes, which can be a dangerous and costly affair. However, Kendall et al. (2018) published an article on this topic, *Learning to Drive in a Day*. The authors chose the agent's reward signal to be the vehicle's forward speed and to terminate an episode if the vehicle drove out of the road. The authors were able to teach a vehicle to perform reliable lane following on a country-side road using one forward-facing camera after only a handful of epochs. More specifically, the final model was able to learn to follow a 250m long road in 11 epochs.

2.5 Software and Hardware

2.5.1 CARLA Simulator

In 2017, the selection of open source simulators suitable for autonomous vehicle research was limited. Furthermore, the few available ones were quite restrictive in terms of customization and control over the environment. Dosovitskiy et al. (2017) saw the need for an open source simulator specifically designed for training and benchmarking autonomous systems, and created CARLA (Car Learning to Act) (Dosovitskiy et al., 2017).

CARLA is an open-source simulator for autonomous driving research and provides digital assets such as urban layouts, buildings, pedestrians, and vehicles. The simulator allows

customization and control of vehicles and provides flexible setup of the environments: such as type of sensors, number of vehicles, and weather conditions. CARLA is implemented as an open-source layer over Unreal Engine 4 and is designed as a server-client system. The server runs the simulation and renders the scene, while the client is an interface that allows interaction between the autonomous agent and the server.

The environments in CARLA consist of 3D models of both static and dynamic objects. The static objects consist of buildings, vegetation, traffic signs, and infrastructure, while the dynamic objects are comprised of vehicles and pedestrians. When CARLA was released it provided two prebuilt cities composed of 40 different buildings, 16 animated vehicle models and 50 animated pedestrian models. Currently, the version used in this thesis has been extended to five prebuilt cities, including urban and suburban environments. Additionally, CARLA supports import of self-made cities.

Town 1 in CARLA is situated in an urban environment and has 2.9 km of drivable road. It consists of roads with two lanes, one in each driving direction, and intersections with traffic lights. Additionally, there are multiple buildings and vegetations in the town. CARLA's second town, *Town 2*, is also situated in an urban environment, but have different buildings from the first town and 1.4 km of drivable road. Many buildings are taller, creating more complex light conditions. Moreover, *Town 3* is in an urban environment with multiple lanes, a roundabout, a tunnel, and uphills and downhills. *Town 4*, the fourth town is a combination of a highway and an urban environment. The highway has four lanes in the same driving direction. The urban environment can be accessed from the highway by taking on of several exits. Lastly, *Town 5* is quite similar to *Town 3*, except there are no roundabouts. Images from the different towns can be seen in Appendix D.

CARLA provides flexible configuration of an agent's sensor specifications, allowing the client to specify the type of sensors and its transformation. The available sensors are RGB cameras, depth cameras, semantic segmentation cameras, and LiDARs. The depth cameras provide a grayscale image where the color intensity represents the distance from an object., while the semantic segmentation provides an image that has classified all items in the image.

(a) Clear day **(b)** Daytime shortly after rain

(c) Daytime rain **(d)** Clear sunset

Figure 2.21: Examples of different weather conditions in Town 2 in CARLA.

CARLA offers seven different weather conditions, at noon and sunset. The different weather conditions consisted of clear sky, cloudy sky, soft rain, medium rain, hard rain, clear wet, and cloudy wet; resulting in a total of 14 different weather/lightning combinations. Figure 2.21 shows some examples of different weather conditions in *Town 2*

The server can be configurated to render various environments, and the client can be used to specify different scenarios. Using a selected combination of sensors, a vehicle can drive around in the environments recording data. This data can be used to train an autonomous system, that can be verified in the simulator.

2.5.2 TensorFlow

TensorFlow, devolped by Google, is a widely used library for implementing machine learning algorithms involving a large number of mathematical operations. The two core components of TensorFlow are *tensors* and a computational graph called a *flow*. A tensor is an N-dimensional vector that can be used to hold N-dimensional data. In the computational

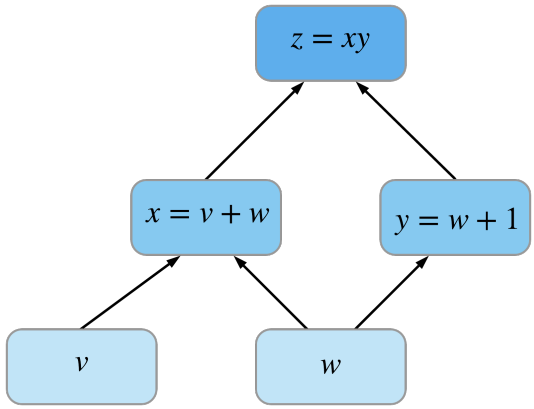


Figure 2.22: A simple example of a computational graph.

graph, each node represents a basic mathematical operation that produces a new tensor. Figure 2.22 illustrates a simple computational graph. The final expression of the graph is formed by traversing the graph in reverse order. Operations in the nodes on the same levels are independent of each other and can be computed in parallel. Thus, the graph can be divided into subgraphs that can be computed independently. The operations of the computational graph are automatically scheduled for parallel computing, allowing users to make use of parallel computing devices to perform operations faster.

2.5.3 Keras

Keras is a high-level Application Programming Interface (API) used to build and train deep learning models and is capable of running on top of several computational backends, including TensorFlow. Keras offers three key advantages. It is user-friendly, modular and composable, and easy to extend. It offers an intuitive set of high-level abstractions that makes it easy to develop deep learning models. Keras supports standard neural networks, convolutional and recurrent neural networks, in addition to other common utility layers like pooling dropout and batch normalization.

2.5.4 GPU and CUDA

A Graphical Processing Unit (GPU) is a single-chip processor mainly used to manage and boost the performance of video and graphics. GPUs have a highly parallel structure, making them more efficient than a general-purpose Central Processing Unit (CPU) for algorithms that can process data in parallel. A GPU can be present on a video card or embedded on the motherboard. CUDA is a parallel computing platform developed by NVIDIA for general computing on their GPUs. CUDA enables developers to speed up computationally heavy applications by utilizing GPUs on the parallelizable part of the computations.

CHAPTER 3

Methodology

This chapters describes the methodology of the thesis. Section 3.1 and Section 3.2 discusses the choice of environment and camera setup, while Section 3.3 and Section 3.4 describes the data collection and preparation process. Section 3.5 introduces the proposed model architectures, while Section 3.6 gives an outline of the experimental setup.

3.1 Environment

Following the arguments presented in Section 2.3.3, it was decided to use a simulator to gather training data and to evaluate the proposed models. The reasoning for this is two-fold. First, it was not feasible to use existing datasets as this thesis' training data required custom annotations (i.e., high-level commands in intersections, and traffic light status). Moreover, for the scope of this thesis, real-world testing was not attainable. A simulated environment provided an excellent testing ground for the different models and allowed us to capture training data from desired driving scenarios efficiently.

3.1.1 Choice of Simulator

When choosing a simulator, three factors were considered: Degree of photorealism, licensing, and completeness out-of-the-box. The degree of photorealism expresses the visual difference between a simulation and real-world. A completely photorealistic simulated scene would appear indistinguishable from real-life. Secondly, only free options were considered. Open-source software was preferred over proprietary software as it allows for customization of the source code. Finally, with a limited time frame, the out-of-the-box completeness of the simulator was vital. The completeness of a simulator refers to how much of the target domain that is modeled. A simulator pre-modeled with intersections, traffic lights, and speed limits can accommodate more complex driving scenarios than, for example, a racing simulator.

After considering several simulators, the CARLA simulator (Dosovitskiy et al., 2017) was chosen. CARLA is an open source simulator built for autonomous driving research and provides an urban driving environment populated with buildings, vehicles, pedestrians, and intersections. Of the several free simulators, CARLA provided the most photorealistic environment, while supporting a range of different weather conditions. Figure 2.21 displays some of the different weather conditions available. To read more about CARLA, see Section 2.5.1.

3.1.2 Simulator Controller

To communicate with the CARLA simulator, a controller software utilizing the simulator's API was written in Python. The controller was used for two primary purposes: to capture training data and to evaluate trained end-to-end models.

Capture Mode

When ran in capture mode, the controller spawns a hero vehicle at a user-specified location in the simulator. The user can then control the vehicle using one of the three supported input sources: Manual control from the keyboard, manual control from a steering wheel and pedals, or autopilot control provided by CARLA. The controller continuously captures and saves data from CARLA, such as the vehicle's current control signal (i.e., steering angle, throttle, and brake), the current speed and location, as well as other relevant information. The controller also annotates the driving data with the correct navigational command automatically. For example, if the user turns left in an intersection, the associated data points are annotated with *Turn Left*.

The data captured from a single, coherent driving sequence is regarded as an episode. When a driving sequence is finished, the controller creates a new directory in the user-specified output folder and saves the episode's captured data to a CSV file inside. Each row in the CSV-file constitutes a single observation. The captured images from the vehicle's vantage point are saved to a separate folder inside the episode folder, while their relative paths are saved to the CSV file. Figure 3.1 illustrates the file structure of recording.

By editing the controller's configuration file, the user can control environmental aspects such as weather conditions, the number of non-player vehicles, and the number of pedestrians. Additionally, the user can define different routes for the autopilot to follow, allowing automatic capturing of data. See Table 3.2 for more information on which kinds of data the controller captures from the simulator.

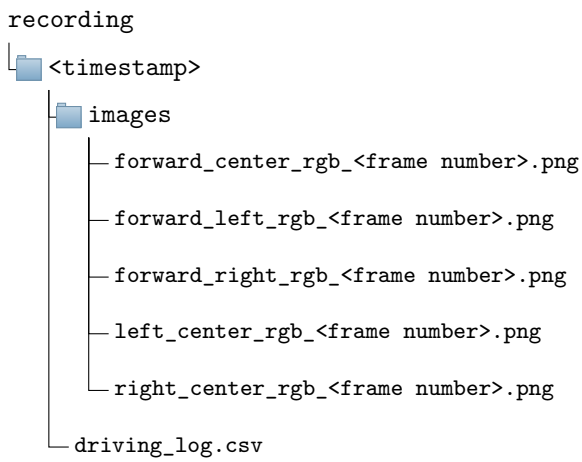


Figure 3.1: File structure of a recording.

Evaluation Mode

In evaluation mode, the controller performs a real-time test on models while recording their performance. A user can specify a folder containing different model files, along with a set of evaluation routes and weather conditions. To evaluate a model on a given route, the interface spawns a hero vehicle at the route's start position, alongside randomly generated non-player vehicles and pedestrians. The current evaluated model is then continuously fed with images from the hero vehicle's vantage point, while the model's predictions are applied as control signals for the hero vehicle. If the hero vehicle approaches an intersection, the controller feeds the correct navigational commands into the model, based on the current route. When reaching the end of the current route, a new route is selected from the set of routes, before the hero vehicle is respawned. The interface evaluates all specified models, on all routes, in all weather conditions; before saving the results and exiting. Read Section 3.6.6 for more information on which criteria the models are evaluated by.

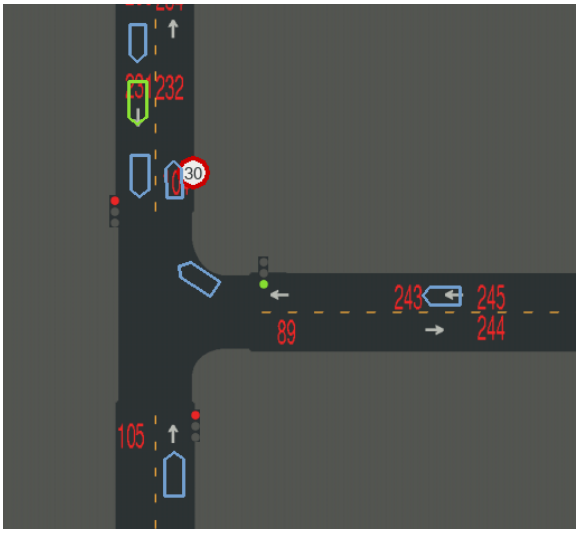


Figure 3.2: An example from the controller's overview mode. The green pentagon represents the hero vehicle, while the blue pentagons represent the non-player vehicles. The id of different spawn locations is marked on the road with red numbers.

Overview Mode

In overview mode, the controller fetches all traffic-related information from the simulator and renders a bird-view, 2D representation of the simulator world. The position of the hero vehicle and all non-player vehicles are visualized and updated real-time. Additionally, the statuses of traffic lights and speed-limit signs are shown. The user may zoom in and out on the visualization to see parts of the world in more details. The overview mode is used to monitor the traffic flow on a macro scale, as well as to ease the process of generating routes. Figure 3.2 shows the controller's overview mode in CARLA's Town 1, zoomed in on a single intersection. The green pentagon represents the hero vehicle, while the blue pentagons represent the non-player vehicles. The id of different spawn locations is marked on the road with red numbers.

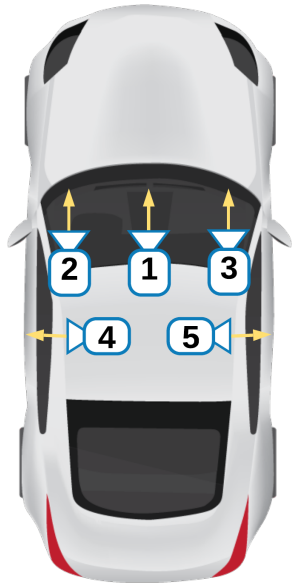


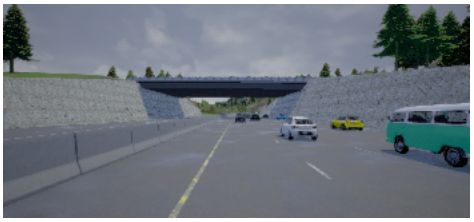
Figure 3.3: The camera setup. Five virtual cameras (1-5) are mounted on the roof of the vehicle.

3.2 Camera Setup

To capture images from a driver's vantage point in CARLA, five virtual cameras were mounted on the roof of the hero vehicle. Each camera produced a 350x160x3 RGB-image. Camera 1-3 were positioned on the center, left side, and right side of the vehicle, respectively, all facing forward. Camera 4 and 5 were positioned at the center of the vehicle, and yaw rotated -90°and 90°respectively. Figure 3.3 illustrates the positioning of the virtual cameras, while Figure 3.4 shows the captured images from a single example frame.

Camera 1 constitutes the actual vantage point of the vehicle (Figure 3.4a), and are positioned at the center of the vehicle's roof. Camera 2 and 3 are used to further expand the captured dataset by simulating the vantage point of a vehicle that is drifting out of the lane (Figure 3.4b and 3.4c). Finally, camera 5 and 6 captures the peripheral view of the vehicle's surroundings (Figure 3.4d and 3.4e).

(a) Forward center (camera 1)



(b) Forward left (camera 2)

(c) Forward right (camera 3)

(d) Left peripheral (camera 4)

(e) Right peripheral (camera 5)

Figure 3.4: Camera 1-5's captured images from a single observation

3.3 Data Collection

When performing imitation learning, the selection-process of training data plays a significant role in a model's ability to perform reliably in different conditions. A model trained only using expert data in ideal environments may not learn how to recover from disturbances. To overcome this, several types of driving data was captured. Expert driving was captured using the CARLA's autopilot control mode, resulting in center-of-lane driving while correctly upholding speed-limits. To capture more volatile data, a randomly generated noise value was added to the autopilot's outgoing control signal. This resulted in sudden shifts in the vehicle's trajectory and speed, which the autopilot subsequently tried to correct. To eliminate undesirable behavior in the training set, only the autopilot's response to the noise was collected, not the noisy control signal. Finally, recovery from possible disaster states was captured by manually steering the vehicle into undesired locations, e.g., the opposite lane, or the sidewalk; while recording the recovery. All data was captured driving 10 km/h below the speed limit to match the velocity of other non-players vehicles.

Two different datasets were gathered, one from Town 1 and one from Town 4. All data were captured in environments without pedestrians, alongside a varying number of non-player vehicles. Some of the training data were captured entirely without any other vehicles, while some of the data were captured with a randomly generated amount (100-200) of other vehicles.

Data were gathered in seven different weather conditions, at noon and sunset. The different weather conditions consisted of clear sky, cloudy sky, soft rain, medium rain, hard rain, clear wet, and cloudy wet; resulting in a total of 14 different weather/lightning combinations. Figure 3.5 shows some examples of the different weather conditions, at both noon and sunset.

Table 3.1 summarizes the gathered datasets. Table 3.2 lists and describes the different types of data contained in each observation. An observation consists of the captured data from a single rendered frame in the simulator, and data were captured using a capture frequency of

Town	No. of observations	Size	Duration
1	87 893	39.6 GB	2.4 h
4	33 757	14.1 GB	1.0 h

Table 3.1: The collected training data. An observation contains the captured data from a single rendered frame in the simulator. Data were captured with a frequency of 10 Hz.

Data	Description	Data Type
Forward Center Image	Image from the vehicle's forward center camera (1).	PNG-image
Forward Left Image	Image from the vehicle's forward left camera (2).	PNG-image
Forward Right Image	Image from the vehicle's forward right camera (3).	PNG-image
Left Peripheral Image	Image from the vehicle's left peripheral camera (4).	PNG-image
Right Peripheral Image	Image from the vehicle's right peripheral camera (5).	PNG-image
Speed	The current speed of the vehicle (mp/h).	int
Speed Limit	The speed-limit at the vehicle's location (mp/h).	int
Location	The vehicle's location, [x-cord, y-cord].	[float, float]
Vehicle Control Signal	The vehicle's control signal, [steer, throttle, brake].	[float, float float]
Autopilot Suggestion	The CARLA autopilot's suggested control signal, [steer, throttle, brake].	[float, float float]
Traffic Light Status	The traffic light status at the vehicles location: Green(1), Red(0)	int
Navigational Input	The activated navigational input: Turn Left(1), Turn Right(2), Straight Ahead(3), Continue Straight(4), Change Lane Left(5), Change Lane Right(6)	int
Environment	The current driving environment: Highway(0), Rural(1)	int
WeatherId	The activated weather condition.	int

Table 3.2: The different types of data captured from the simulator.

10 observations each second. In total, 3.4 hours of training data were captured, 2.4 hours in Town 1, and 1.0 hour in Town 4.

3.4 Data Preparation

It was quickly discovered that data augmentation was essential for good generalization. Specifically, it was possible to simulate several of the varying environmental conditions using different image transformations. For each observation in the training set, a single new

(a) Clear Noon

(b) Clear Sunset

(c) Heavy Rain Noon

(d) Heavy Rain Sunset

(e) Soft Rain Noon

(f) Soft Rain Sunset

(g) Wet Clear Noon

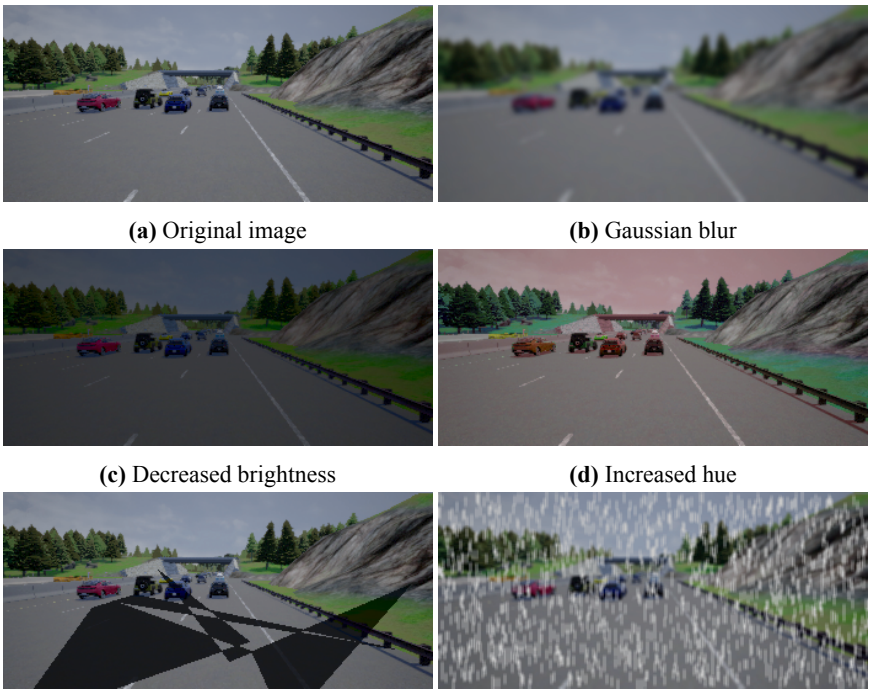
(h) Wet Clear Sunset

Figure 3.5: Some of the weather conditions used when capturing data.

Figure 3.6: Gathering of data in rainy conditions using a steering wheel and pedals.

data augmented sample was generated using one of the desirable transformations picked at random. These included a random change in brightness, random changes in hue, the addition of Gaussian blur, the addition of randomly generated dark polygons differing in position and shape – simulating shadows, and the addition of randomly generated lines simulating rain. Figure 3.7 shows five examples of data augmented images.

Another vital factor in end-to-end learning is the balance of target values within a dataset. A model trained on unbalanced data may incorrectly bias some actions over others. When recording the datasets, the majority of the observations were captured driving straight. To counteract this, observations with a small steering angle were downsampled (by removal), while observations with a large steering angle were upsampled (by duplication). Additionally, the observations corresponding to the different intersection decisions (i.e., turn left, turn right, or straight ahead) were balanced by analyzing the observations' *Navigational Input*-property and downsampling the over-represented choices. Finally, most of the observations where the vehicle was not moving (e.g., waiting for a red light) were downsampled. Figure 3.8 shows the distribution of the balanced training set.



(a) Original image **(b)** Gaussian blur

(c) Decreased brightness **(d)** Increased hue

(e) Dark polygons simulating shadows **(f)** Gray lines simulation rain

Figure 3.7: Examples of data augmented images.

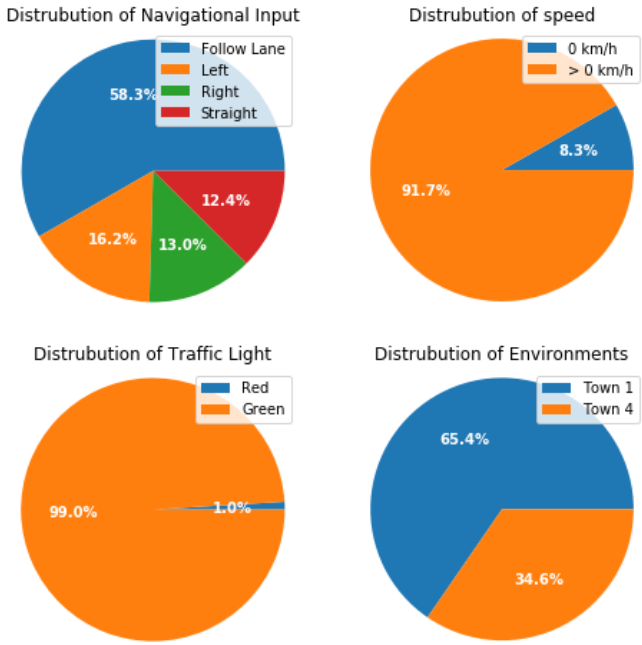


Figure 3.8: Distribution of the balanced training dataset.

3.5 Model Architecture

The overall goal of this thesis was to research an end-to-end model's ability to drive autonomously. This section proposes an end-to-end model based on a deep neural network. Furthermore, a revised imitation learning approach is proposed for training the model to mimic expert driving behavior.

3.5.1 Problem Definition

Section 2.3.2, introduces end-to-end learning via imitation learning. This thesis proposes a revised version, where a model takes an observation o_i, a user's intent h_i, and external state information s_i as input in the decision making. h_i is a one-hot encoded vector representing the user's navigational intent, hereby referred to as the navigational input. The

navigational input can either be: *Turn Left at Next Intersection*, *Turn Right at Next Intersection*, *Continue Straight at Next Intersection*, or *Follow Lane*. Moreover, additional information about the world, such as the current speed limit, the vehicle's speed, and the current traffic light state, are introduced by s_i, hereby referred to as the external state information. The revised imitation learning technique is expressed in Equation 3.1, where a model, F, learns a mapping between the inputs (i.e., o_i, h_i, s_i) and the executed action a_i, by fitting the learnable parameters, θ, to minimize the loss, L.

$$\underset{\theta}{\text{minimize}} \quad \sum_{i} L(F(o_i, h_i, s_i; \theta), a_i) \tag{3.1}$$

3.5.2 System Overview

The proposed system is comprised of two modules, a *Steer Predictor* and a *Throttle-Brake Predictor*. Each module takes a sequence of forward-facing images, navigational inputs, and external state information as input. The images are sent as input to a CNN, where each image is processed independently to extract important features in the images. The CNN structure is further discussed in Section 3.5.3. The output from the CNN is sent to an LSTM alongside navigational inputs and external state information to learn temporal dependencies between the subsequent images. Section 3.5.4, describes the LSTM architecture in more detail.

In the *Steer Predictor*-module, the output of the LSTM is passed through a classification layer, producing a sine steer prediction. During the training phase, the loss between the output of the classification layer and the sine encoded ground truth steering angle is calculated using an RMSE loss function (Equation 2.7) and sent through a backpropagation algorithm to update the weights. After training, when the system is in its deployment phase, the steer prediction is passed through a decoder to produce a steering angle. The *Classifier* in the *Steer predictor*-module is further described in Section 3.5.5.

In the *Throttle-Brake Predictor*-module, the output of the LSTM is passed through a fully

connected layer to predict a throttle and brake value. During training, the loss between the predicted values, and the ground truth, are calculated using an MSE loss function (Equation 2.6). The loss is sent through a backpropagation algorithm to update the weights. During the deployment phase, the output from the Predictor yields the final throttle and brake predictions.

Figure 3.9 illustrates the system in its training phase, while Figure 3.10 illustrates the system in its deployment phase. The blue box represents the *Steer Predictor*-module, and the green box describes the *Throttle-Brake Predictor*-module.

3.5.3 CNN

The proposed system uses a CNN to extract useful features from the input image. The architecture is inspired by the architecture used in NVIDIA's DAVE-2 system (Bojarski et al., 2016). The modified network takes a 160x350x3 image as input, followed by a cropping layer and a normalization layer. The cropping layer removes the top 50 pixels from the image, while the normalization layer scales the pixel values between -0.5 and 0.5. Next follows six convolutional layers, all using a ReLU activation function (Equation 2.3). The first three convolutional layers use a 5x5 filter, while the last three use a 3x3 filter. The first four convolutional layers use a stride of 2, while the last two use a stride of 1. The output of the last convolutional layer is flattened, resulting in a one-dimensional feature layer containing 1024 nodes.

3.5.4 LSTM

The output from the CNN (Section 3.5.3) is concatenated with the current navigational input and external state information, producing a vector of length 1033. The concatenated vector is connected to an LSTM layer with ten hidden states, that uses a sequence of feature extractions over time to produce a control signal. For each time step in the sequence, the LSTM layer sends its output to itself. At the last time step, the output is forwarded to a

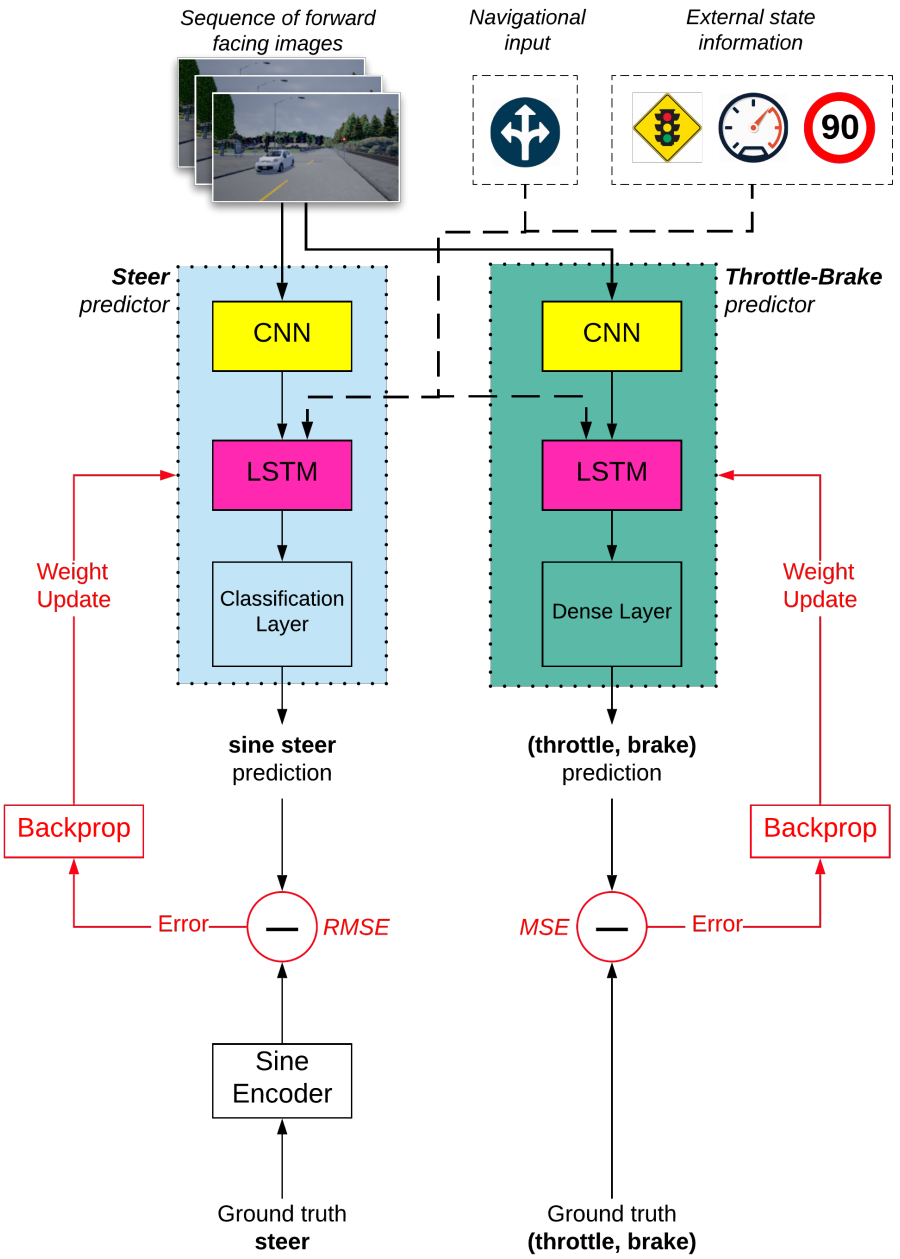


Figure 3.9: System overview in training phase.

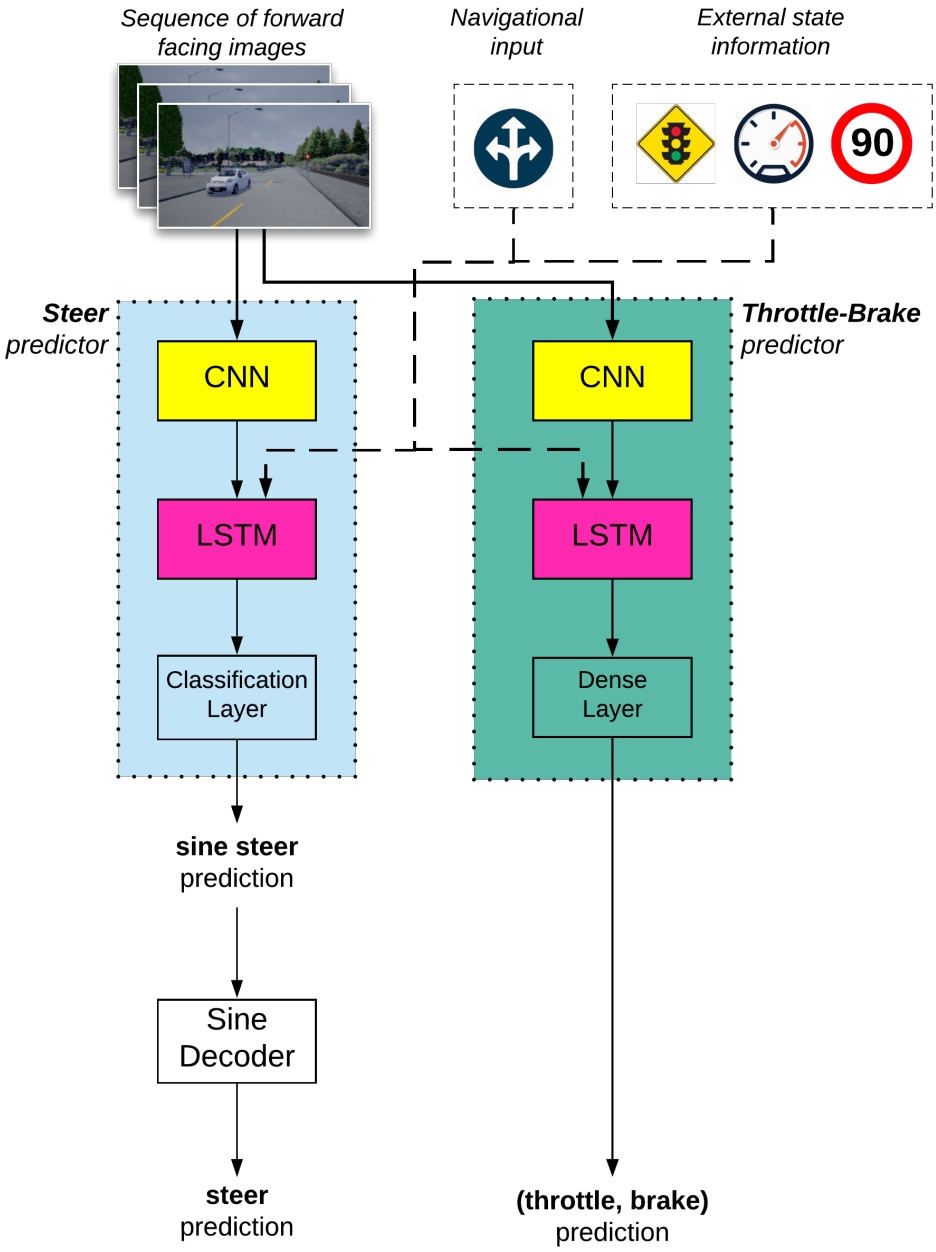


Figure 3.10: System overview in deployment phase.

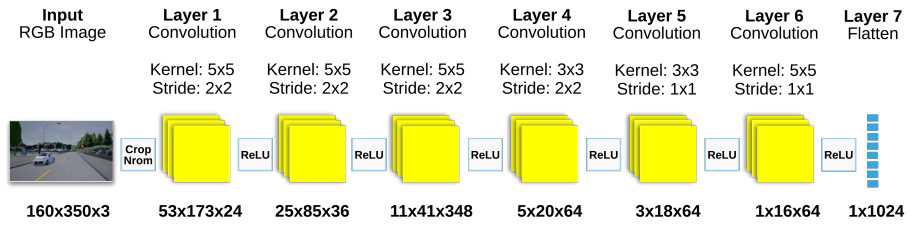


Figure 3.11: The architecture of the CNN

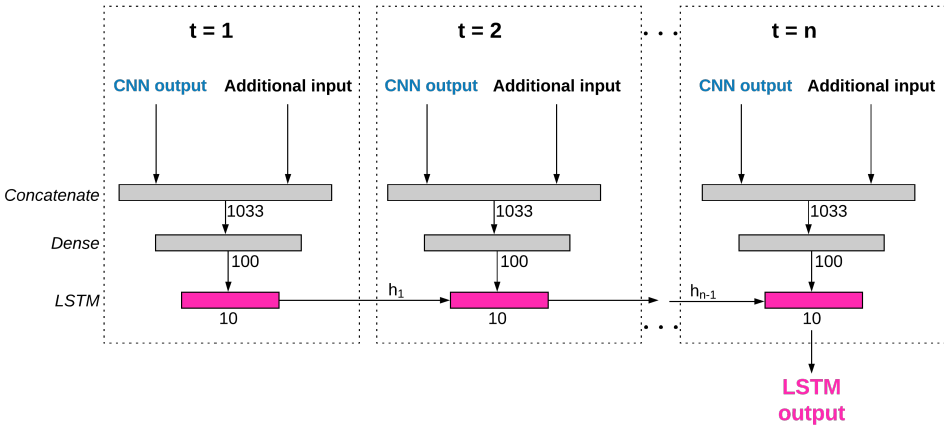


Figure 3.12: The architecture of the LSTM

classification layer in the *Steer Predictor*-module or a dense layer in the *Throttle-Brake Predictor*-module.

3.5.5 Steering Angle Classification

Steering angle prediction in autonomous vehicle control is usually posed as a regression problem, where the system predicts a continuous steering angle from a set of sensory input. This approach can be seen in Pomerleau (1989), Lecun et al. (2004), Bojarski et al. (2016), Codevilla et al. (2017), and Hubschneider et al. (2017).

Another approach is to express the problem as a classification task, where each output node represents an interval of steering angles, and the network predicts the probability of

the target steering angle belonging to each interval. Rothe et al. (2015) argues that posing a problem as a deep classification task can often perform better than a direct regression training in CNNs. However, this approach comes with two challenges.

Firstly, the required amount of training data scales with the number of output classes, i.e., the number of steering angle intervals. Secondly, classification problems assume independence between the output neurons that encode the different classes. Thus, a loss function will neglect the difference between smaller and larger steering angle errors.

Eraqi et al. (2017) argues that the classification formulation can be further improved by introducing a spatial relationship between the nodes in the output layer, i.e., neurons close to each other are more similar than neurons far apart. The objective should be to learn a sine function that encodes the steering angle. This way, a correlation between the output neurons are introduced, thus bridging the gap between the regression problem and the deep classification problem.

Following the same setup as Eraqi et al. (2017), a classification layer containing ten neurons is introduced to the end of the *Steer Predictor*-module. Furthermore, a *tanh* activation is applied to the classification layer allowing the neurons to shape a sine wave with an amplitude of 1. The original steering angle corresponds to the phase shift, ϕ, of the sine wave. During training, the ground truth steering angle is encoded as a sine wave using Equation 3.2. Y_i is the encoded target value for output neuron i, ϕ is the raw steering angle, and ϕ_{max} is the maximum possible raw steering angle. The loss of the prediction is calculated as the RMSE (Equation 2.7) between the predicted waveform and the encoded ground truth waveform. During deployment, the classification layer's output is decoded back to a steering angle. The decoding is done by fitting the classification layer's output to a sine function and returning its phase shift. Figure 3.13 illustrates the training and deployment phase of the steering angle prediction.

$$Y_i(\phi) = \sin\left(\frac{2\pi(i-1)}{10-1} - \frac{\phi\pi}{2\phi_{max}}\right), \quad 1 \leq i \leq 10 \tag{3.2}$$

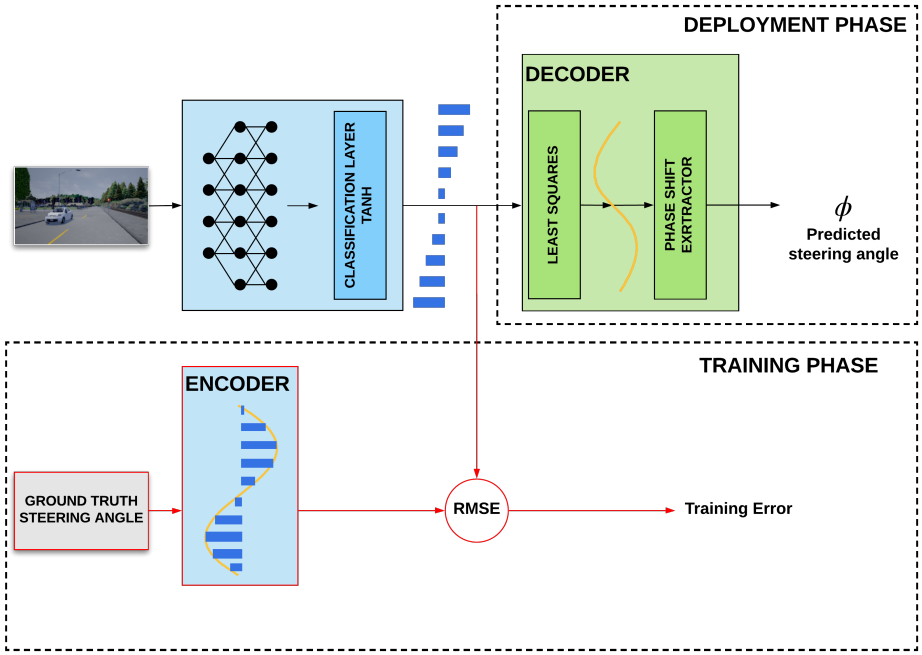


Figure 3.13: The training and deployment procedure of the steering angle prediction. The red arrows and boxes indicate the parts that are only activated during the training phase.

3.6 Experimental Setup

Four different experiments were conducted based on the research questions and objectives introduced in Section 1.2.

3.6.1 Experiment 1: Sequence Length

Goal The goal of the first experiment was to illuminate how the number of hidden states in the LSTM cells affects the system's performance. The number of hidden states in an LSTM controls how many timesteps the network is able to "look back", thus constraining the length of the temporal dependencies the network can learn. Consequently, the number of hidden states decides the sequence length of the input data.

Training Five models were trained using 1, 5, 10, and 20 hidden states in the LSTM cells. The architecture of the neural networks was otherwise similar.

Testing Each trained model was evaluated by their performance from a single run of the real-time test in town 2 and town 4. The real-time test procedure is described in Section 3.6.6.

3.6.2 Experiment 2: Size of Dataset

Goal The goal of the second experiment was to clarify how the size of the training dataset affects the system's performance.

Training Four models were trained using datasets of varying size. The different datasets all originated from the final, augmented and balanced dataset, but with an increasingly

larger portion of the dataset randomly dropped. The portion of data dropped ranged from 20% to 80%.

Testing Each trained model was evaluated by their average route completion on a single run of the real-time test in town 2. The real-time test procedure is described in Section 3.6.6.

3.6.3 Experiment 3: Prediction Frequency

Goal The goal of the third experiment was to determine how the model's prediction frequency affects the performance in a real-time test. The prediction frequency limits how many control-signals the model can produce each second, thus restricting how fast the model is able to react to changes in the perceived environment.

Training The best performing model from experiment 1 is selected for testing without further training.

Testing The model was exposed to town 2's real-time test five times. For each test, the model's allowed prediction frequency was set to 30 Hz, 15 Hz, 10 Hz, 5 Hz, and 1 Hz, respectively. The models were evaluated by their average route completion. The real-time test procedure is described in Section 3.6.6.

3.6.4 Experiment 4: Extensive Real-time Test

Goal The goal of the final experiment was to accurately evaluate the performance of the best model from experiment 1.

Training The best performing model from experiment 1 was selected for testing without further training.

Testing The trained model was evaluated by its average performance on ten runs of the real-time test in town 2 and town 4. The real-time test procedure is described in Section 3.6.6.

3.6.5 Training procedure

The proposed system was implemented in Keras (Section 2.5.3), running on top of the Tensorflow computational backend (Section 2.5.2). All models were trained on a Nvidia 1080 Ti GPU by using the CUDA computing platform (Section 2.5.4).

Before training could start, properly structured data-points had to be generated from the augmented and balanced dataset. The generation of data-points is described in the section below. The generated data-points were split into a training set (80%) and a validation set (20%). All models were trained using an Adam optimizer (Equation 2.9). The models were trained for 15 epochs, with a batch size of 32. The models' weights were recorded after each epoch along with the associated validation error. After the training was complete, the weights associated with the lowest validation error were chosen for further testing.

Generation of data points

For each observation in the augmented and balanced data set, three separate data-points were generated, one from each of the forward facing cameras (left, center, and right). A data-point was comprised of an input and the associated target output. The input encompassed a single image, a scaled vehicle speed, a scaled speed limit, and a one-hot-encoded navigational input. The target output encompassed a vehicle control signal, i.e., a steering angle, brake value, and throttle value. To counteract the positional offset in the data points originating from the left and right forward facing cameras, the associated steering

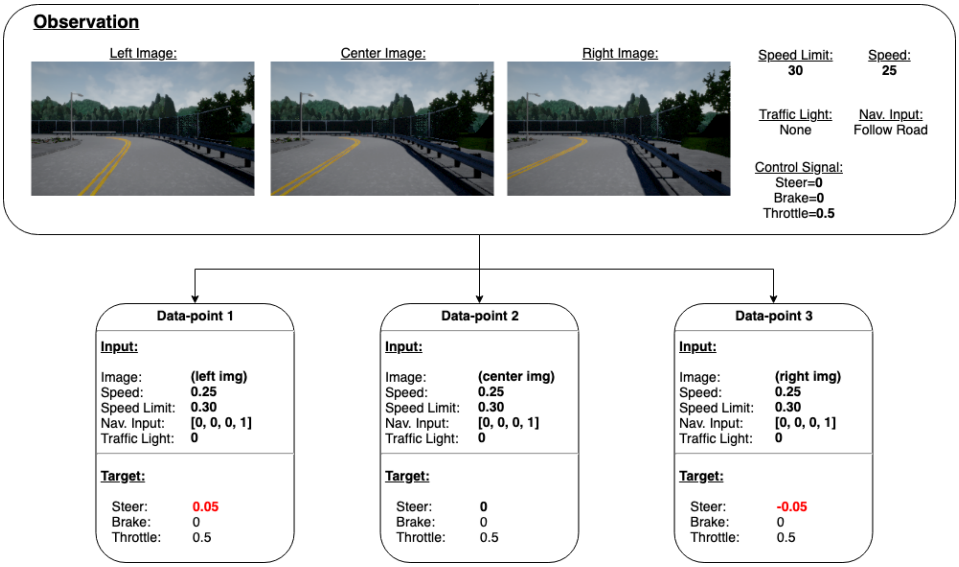


Figure 3.14: Three data-points generated from a single observation. Shifted steering angles are marked in red.

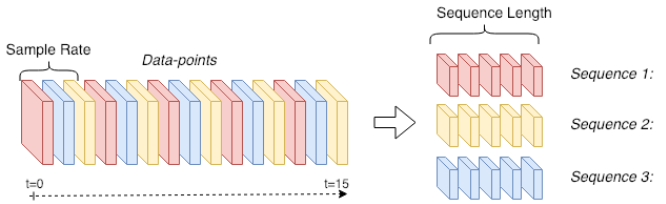


Figure 3.15: The structuring of sequences from an array of data-points. In this illustration a sequence length (α) of 5 and a sampling interval (β) of 3 is used.

angles was shifted by +0.05 and -0.05 respectively. Figure 3.14 illustrates the generation of data-points from a single observation.

The generated data-points were further structured into sequences of length α, using a sampling interval of β. The sequence length determines the number of time steps the LSTM layer is able to look back. The sampling interval controls the period between successive individual time steps within sequences. Figure 3.15 illustrates the structuring of sequences from 15 data points, using a α of 5, and a β of 3.

	Town	Total distance
Route 1	2	355 m
Route 2	2	269 m
Route 3	2	378 m
Route 4	4	1716 m
Route 5	4	921 m

Table 3.3: The different routes a model were to drive during a real-time test.

3.6.6 Real-time test

The real-time performance was tested by letting a model control a simulated vehicle in completely unseen environments. That is, the model was continuously fed with images from a hero vehicle's vantage point, while the model's predictions were applied as control signals for the hero vehicle.

The model was to drive five different predefined routes, in five diverse weather conditions comprised of *Clear Noon*, *Clear Sunset*, *Heavy Rain Noon*, *Soft Rain Sunset*, and *Wet Sunset*. Route 1-3, positioned in *Town 2*, tested the model's ability to operate in urban environments containing traffic, intersections, traffic lights and speed limits. Route 4 and 5, positioned in *Town 4*, tested the model's ability to operate on highways and high-speed arias. The routes are summarized in Table 3.3, and the paths of route 1-3 and 4-5 are illustrated in Figure 3.16 and 3.17 respectively.

In total, the model was to drive through 15 different weather/route permutations in *Town 2*, resulting in a combined distance of 5 010 meters. In *Town 4*, the model was to drive through 10 different weather/route permutations, resulting in a combined distance of 13 185 m.

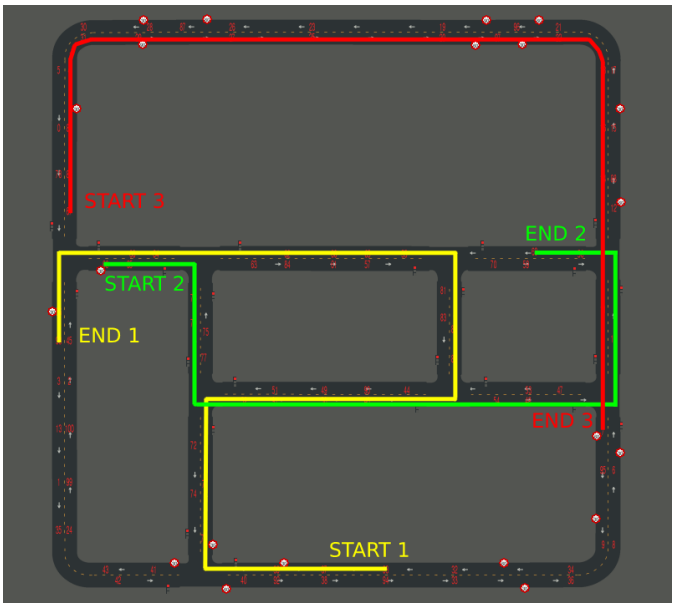


Figure 3.16: The paths of route 1-3 in Town 2.

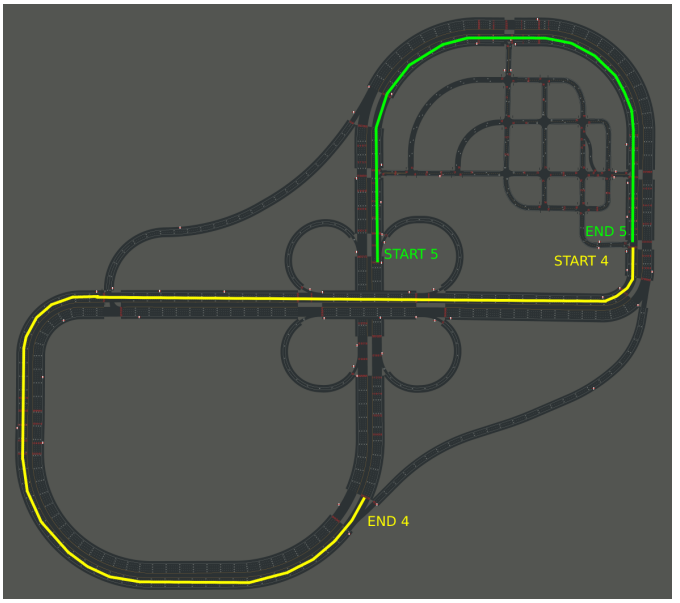


Figure 3.17: The paths of route 4-5 in Town 4.

Failure	Abbreviation	Severity	Route cancelation?
Lane lines touch	Lane L. Touch	Minor	No
Sidewalk touch	Sidew. Touch	Minor	No
Object collision	Object Coll.	Moderate	No
Read-end vehicle collision	Rear-End Coll.	Moderate	No
Route deviation	Route Dev.	Moderate	**Yes**
Front-end vehicle collision	Front-End Coll.	Severe	**Yes**
Entering oncoming lane	Enter Onc. Lane	Severe	**Yes**
Traffic light violation	Traffic Light Vio.	Severe	**Yes**
Loss of vehicle control	Loss of control	Severe	**Yes**

Table 3.4: The different types of failures captured by the simulator controller during a real-time test.

Performance measures

The *Simulator Controller* automatically logged any failures during the real-time test. Eight types of failures of varying severity were defined. Lane touches and sidewalk touches were considered minor failures, while object collisions, navigational input violation, and rear-end vehicle collisions were considered moderate failures. Severe failures were comprised of front-end vehicle collisions, entering the oncoming lane, and traffic light violations. Some failures would result in the cancellation of the current route, e.g., if the model deviated from the predefined route, or entered the oncoming lane without recovery. Table 3.4 summarizes the different failure types.

After the completion of the cancellation of a route, the *Simulator Controller* recorded how much of the current route the model was able to complete.

The defined failure types were slightly changed in *Town 4* to account for the environmental difference between the towns. Firstly, front-end collisions were removed since the crash barrier made it impossible for the hero vehicle to reach the oncoming lanes. Furthermore, *Town 4* do not contain any sidewalks. Thus, sidewalk touches were removed as well. Finally, the high-speed arias of *Town 4* required a new failure type, namely the loss of vehicle control.

CHAPTER 4

Results

This chapter describes the experimental results of this thesis. Section 4.1 and Section 4.2 presents the results from the sequence length test and the dataset size test, while Section 4.3 presents the results from the prediction frequency test. Finally, the results from the extensive real-time test is described in Section 4.4. A preview of the overall performance can be seen in (Aasboe, Haavaldsen, and Lindseth, 2019a). Moreover, additional videos created after the submission of this thesis will be published to (Aasboe, Haavaldsen, and Lindseth, 2019b).

4.1 Experiment 1: Sequence Length

4.1.1 Training

Table 4.1 summarizes the results, showing the best validation losses for steer, throttle, and brake, as well as the training duration. Model B had the lowest steer validation, while model D had the lowest throttle and brake validation loss. The training duration increased with the sequence length.

Id	Seq. Length	Val. Loss, Steer	Val. Loss, Throttle	Val. Loss, Brake	Training Duration
A	1	0.0187	0.0156	0.0034	**0h 43m**
B	5	**0.0161**	0.0094	0.0026	2h 59m
C	10	0.0191	0.0130	0.0024	9h 14m
D	20	0.0169	**0.0092**	**0.0018**	17h 4m

Table 4.1: Validation loss for models trained with different sequence lengths.

4.1.2 Real-time Test

Town 2 – Urban Environment

Table 4.2 shows the models' average route completion in each weather condition. Model A, B, and C were able to finish all routes in *Clear Noon*, *Clear Sunset*, *Soft Rain Noon*, and *Clear Wet Sunset*. In general, model D had the lowest route completions with an average of 75.6%. Figure 4.1 illustrates the average route completions for each sequence length.

The differences became more apparent in *Heavy Rain Noon*, where model C outperformed the other models substantially with an average route completion of 76.6%. Model A, B, and D had an average route completion of 10.6%, 14.1%, and 28.4% respectively.

Id	Seq. Length	Clear Noon	Clear Sunset	Heavy Rain Noon	Soft Rain Noon	Clear Wet Sunset	Avg.
A	1	100%	100%	10.6%	100%	100%	82.1%
B	5	100%	100%	14.1%	100%	100%	82.8%
C	10	100%	100%	76.6%	100%	100%	**95.3**%
D	20	100%	82.4%	28.4%	83.5%	83.5%	75.6%

Table 4.2: Experiment 1 – The systems' average route completions in Town 2.

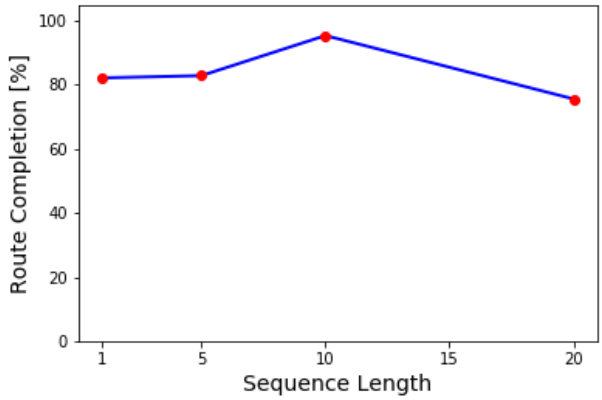


Figure 4.1: Average route completions for each sequence length in Town 2 from experiment 1.

Table 4.3 lists the total number of failures for each model, while Table 4.4 lists the number of failures per traveled kilometer.

On average, model A-D made 0.24, 0.48, 1.26, 3.17 minor failures/km, respectively. Thus, model A considerably outperformed the rest in regards to the minor failures.

Model A-D made 0.48, 0.96, 0.84, 2.90 moderate failures/km respectively. It is important to point out that model B and C's rear endings primarily happened in heavy rain. Model A made severe mistakes early on in heavy rain and was consequently not exposed to the challenging scenarios which led to rear-endings for the other models. With this in mind, the results indicate that model C outperformed the other models with model B closely behind.

On average, model A-D made 0.49, 0.72, 0.42, 1.85 severe failures/km, respectively. The results show that model C outperformed the other models in regards to severe failures, with model B performing similarly.

		Minor		Moderate			Severe		
Id	Seq. Length	Lane L. Touch	Sidew. Touch	Object Coll.	Rear-End Coll.	Route Dev.	Front-End Coll.	Traffic Light Vio.	Enter Onc. Lane
A	1	1	0	1	0	1	0	0	2
B	5	2	0	2	2	0	0	0	3
C	10	6	0	1	3	0	0	0	2
D	20	12	0	1	10	0	0	0	7

Table 4.3: Experiment 1 – Total number of failures in Town 2.

		Minor		Moderate			Severe		
Id	Seq. Length	Lane L. Touch	Sidew. Touch	Object Coll.	Rear-End Coll.	Route Dev.	Front-End Coll.	Traffic Light Vio.	Enter Onc. Lane
A	1	0.24	0	0.24	0	0.24	0	0	0.49
B	5	0.48	0	0.48	0.48	0	0	0	0.72
C	10	1.26	0	0.21	0.63	0	0	0	0.42
D	20	3.17	0	0.26	2.64	0	0	0	1.85

Table 4.4: Experiment 1 – Number of failures per traveled km in Town 2.

Town 4 – Highway

The models were real-time tested on a highway in CARLA's *Town 4* with other vehicles.

Table 4.5 lists the average route completion in each weather condition. The differences between the models were less obvious than in *Town 2*. However, model B excelled with 100% route completion in all routes. Model C also achieved 100% route completions in several weather conditions but experienced one route canceling incident resulting in a 41.3% rout completion in *Clear Wet Sunset*. Figure 4.2 illustrates the average route completions for each sequence length.

Id	Seq. Length	Clear Noon	Clear Sunset	Heavy Rain Noon	Soft Rain Noon	Clear Wet Sunset	Avg.
A	1	100%	100%	41.3%	100%	41.3%	76.5%
B	5	100%	100%	100%	100%	100%	**100**%
C	10	100%	100%	100%	100%	41.3%	88.3%
D	20	100%	52.1%	100%	100%	41.3%	78.7%

Table 4.5: Experiment 1 – Average route completions in each weather condition in Town 4.

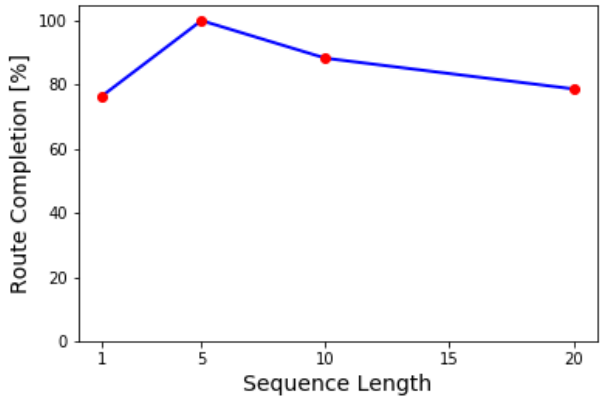


Figure 4.2: Experiment 1 – Average route completions for each sequence length in Town 4.

Table 4.6 lists the total number of failures for each model, while Table 4.7 lists the number of failures per traveled kilometer. On average, model A-D made 0.99, 1.22, 1.72, 0.58 minor failures/km, respectively. Thus, model D performed the fewest number of minor failures. Furthermore, model A-D made 0.30, 0.38, 0.35, 0.59 moderate failures/km respectively. Hence, model A-C performed similarly, while model D had the highest frequency of moderate failures. Finally, model A-D made 0.20, 0, 0.09, 0.01 severe failures/km, respectively. Model A made two severe failures in total, model B made no severe failures, while model C and D made one severe failure. The results show that model B performed the fewest number of severe failures, but that the other models made few severe failures as well.

		Minor	Moderate			Severe	
Id	**Seq. Length**	**Lane L. Touch**	**Object Coll.**	**Rear-End Coll.**	**Route Dev.**	**Traffic Light Vio.**	**Loss of Control**
A	1	10	1	2	0	0	2
B	5	16	0	5	0	0	0
C	10	20	1	3	0	0	1
D	20	6	0	6	1	0	1

Table 4.6: Experiment 1 – Total number of failures in Town 4.

		Minor	Moderate			Severe	
Id	**Seq. Length**	**Lane L. Touch**	**Object Coll.**	**Rear-End Coll.**	**Route Dev.**	**Traffic Light Vio.**	**Loss of Control**
A	1	0.99	0.10	0.20	0	0	0.20
B	5	1.22	0	0.38	0	0	0
C	10	1.72	0.09	0.26	0	0	0.09
D	20	0.58	0	0.58	0.01	0	0.01

Table 4.7: Experiment 1 – Number of failures per traveled km in Town 4.

4.2 Experiment 2: Size of Dataset

Table 4.8 summarizes the different models' average route completions in each weather condition. Model A used 100% of the original dataset and was able to finish all routes expect for *Heavy Rain Noon*, similar to the results in experiment 1. Model B was trained on 80% of the original dataset and was only able to complete all the routes in *Clear Sunset*. Overall, it had an average route completion of 74.1%. Model C was trained on 60 % of the original dataset and was not able to complete all the routes in any weather condition. Additionally, it had a notable performance decrease with an average route completion of 59.1 %. Model D and E were trained on 40% and 20% of the original dataset respectively and performed similarly as model C, with an average route completion of 60.0% and 55.1%.

Figure 4.3 shows the average route completions of each model.

Id	Dataset Size	Clear Noon	Clear Sunset	Heavy Rain Noon	Soft Rain Noon	Clear Wet Sunset	Avg.
A	100%	100%	100%	85.4%	100%	100%	**97.1%**
B	80%	82.4%	100%	36.5%	69.2%	82.4%	74.1%
C	60%	83.5%	66.3%	47.7%	57.5%	40.3%	59.1%
D	40%	91.2%	82.4%	16.1%	57.5%	53.0%	60.0%
E	20%	77.5%	77.5%	7.7%	56.4%	56.4%	55.1%

Table 4.8: Experiment 2 – Average route completion in each weather condition in Town 2.

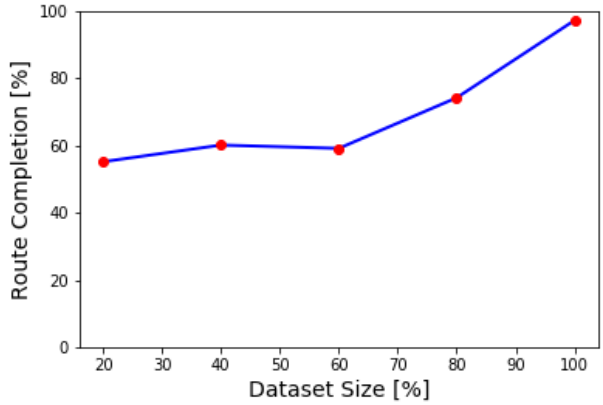


Figure 4.3: Experiment 2 – Average route completions for each model trained with different dataset sizes in Town 2.

4.3 Experiment 3: Prediction Frequency

Table 4.9 summarizes the different models' average route completions in each weather condition. When using a prediction frequency of 30 Hz, the model was able to finish routes in all weather conditions except for *Heavy Rain Noon*, similar to the results from experiment 2. Dropping the prediction frequency to 15 Hz, for model B, did not result in any performance loss. When reducing the prediction frequency to10 Hz, the model

had a small performance decrease in *Heavy Rain Noon* and *Soft Rain Noon*, with a route completion of 80.4% and 95.0%, respectively. Further dropping the prediction frequency to 5 Hz resulted in a performance loss in both *Heavy Rain Noon* and *Soft Rain Noon*. When further reducing the prediction frequency to 3 Hz a performance decrease in *Clear Noon*, *Heavy Rain Noon*, and *Clear Wet Sunset* was seen. Finally, reducing the prediction frequency even further, to 1 Hz, resulted in a poor performance in all weather conditions. Figure 4.4 shows the average route completions in all weather conditions for the different prediction frequencies.

Id	Pred. Freq.	Clear Noon	Clear Sunset	Heavy Rain Noon	Soft Rain Noon	Clear Wet Sunset	Avg.
A	30	100%	100%	85.4%	100%	100%	**97.1%**
B	15	100%	100%	85.4%	100%	100%	**97.1%**
C	10	100%	100%	80.4%	95.0%	100%	95.1%
D	5	100%	100%	56.5%	74.0%	100%	86.1%
E	3	76.4%	100%	38.5%	100%	66.3%	76.2%
F	1	10.5%	76.4%	14.0%	16.8%	21.3%	27.8%

Table 4.9: Experiment 3 – Average route completion in each weather conditionin Town 2.

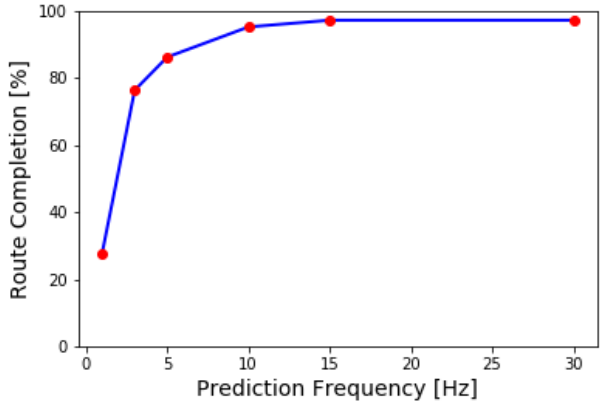


Figure 4.4: Experiment 3 – Average route completions for each prediction frequence in Town 2.

4.4 Experiment 4: Extensive Real-time Test

The model had an average route completion of 95.3% in *Town 2*, which is consistent with the results from experiment 1. In *Town 4* the model was able to complete most routes except for in *Clear Wet Sunset*, where it had an average route completion of 41.3%. The average route completion for all routes were 84.0%, which also complied with the results from experiment 1.

Town	Clear Noon	Clear Sunset	Heavy Rain Noon	Soft Rain Noon	Clear Wet Sunset	Avg.
Town 2	100%	96.5%	80.1%	100%	100%	95.3%
Town 4	100%	100%	83.8%	94.9%	41.3%	84.0%

Table 4.10: Experiment 4 – Average route completions in each weather condition.

In *Town 2* the model performed best during *Clear Noon* and *Clear Sunset*. In total, it made 0.2 moderate failures/km and 0.2 severe failures/km in *Clear Noon*, while it made 0.4 minor failures/km and 0.2 severe failures/km in *Clear Sunset*. The model's performance was slightly worse in *Soft Rain Noon* and *Clear Wet Sunset* where it made 2.0 failures/km and 3.4 failures/km, respectively. As in experiment 1, the model had the worst performance in *Heavy Rain*, where it made 3.5 minor failures/km, 4.5 moderate failures/km, and 2.0 severe failures/km.

	Minor		Moderate			Severe		
Weather Condition	Lane L. Touch	Sidew. Touch	Object Coll.	Rear-End Coll.	Route Dev.	Front-End Coll.	Traffic Light Vio.	Enter Onc. Lane
Clear Noon	0	0	0	1	0	0	0	1
Clear Sunset	2	0	0	0	0	0	0	1
Heavy Rain Noon	14	0	6	10	0	2	0	6
Soft Rain Noon	6	0	0	4	0	0	0	0
Clear Wet Sunset	9	0	0	8	0	0	0	0
Total	31	0	6	23	0	2	0	8

Table 4.11: Experiment 4 – Total number of failures in Town 2.

	Minor		Moderate			Severe		
Weather Condition	**Lane L. Touch**	**Sidew. Touch**	**Object Coll.**	**Rear-End Coll.**	**Route Dev.**	**Front-End Coll.**	**Traffic Light Vio.**	**Enter Onc. Lane**
Clear Noon	0	0	0	0.20	0	0	0	0.20
Clear Sunset	0.41	0	0	0	0	0	0	0.21
Heavy Rain Noon	3.49	0	1.50	2.49	0	0.50	0	1.50
Soft Rain Noon	1.20	0	0	0.80	0	0	0	0
Clear Wet Sunset	1.80	0	0	1.60	0	0	0	0
Total	1.30	0	0.25	0.96	0	0.08	0	0.34

Table 4.12: Experiment 4 – Number of failures per traveled km in Town 2.

In *Town 4* the model had the fewest number of failures during *Clear Noon*, *Hard Rain Noon*, and *Soft Rain Noon*. In *Clear Noon*, it made 0.68 minor failures/km and 0.15 moderate failures/km. In *Soft Rain Noon* in made 0.24 minor failures/km and 0.08 moderate failures/km, and 0.08 severe failures/km. In *Hard Rain Noon* in made 0.63 minor failures/km and 0.54 moderate failures/km, and no severe failures/km.

The model had more occurences of failures in the weather conditions during sunset. In *Clear Sunset* the model made 0.61 minor failures/km and 0.68 moderate failures/km; while it made 1.47 minor failures/km, 1.28 moderate failures, and 0.73 severe failures/km in *Clear Wet Sunset*.

	Minor	Moderate			Severe	
Weather Condition	**Lane L. Touch**	**Object Coll.**	**Rear-End Coll.**	**Route Dev.**	**Traffic Light Vio.**	**Loss of Control**
Clear Noon	9	0	0	2	0	0
Clear Sunset	8	0	7	2	0	0
Heavy Rain Noon	7	0	3	3	0	0
Soft Rain Noon	3	0	1	0	0	1
Clear Wet Sunset	8	4	2	1	0	4
Total	35	4	13	8	0	5

Table 4.13: Experiment 4 – Total number of failures after driving 55.4 km in Town 4.

	Minor	Moderate			Severe	
Weather Condition	**Lane L. Touch**	**Object Coll.**	**Rear-End Coll.**	**Route Dev.**	**Traffic Light Vio.**	**Loss of Control**
Clear Noon	0.68	0	0	0.15	0	0
Clear Sunset	0.61	0	0.53	0.15	0	0
Heavy Rain Noon	0.63	0	0.27	0.27	0	0
Soft Rain Noon	0.24	0	0.08	0	0	0.08
Clear Wet Sunset	1.47	0.73	0.37	0.18	0	0.73
Total	0.63	0.07	0.23	0.14	0	0.09

Table 4.14: Experiment 4 – Number of failures per traveled km in Town 4.

CHAPTER 5

Discussion

This section discusses the results of the experiment, in addition to some of the critical decisions made in the thesis. Section 5.1 discusses the choice of training data, while Section 5.2 review the choice of simulator. The model architecture is discussed in Section 5.3, while the results from the experiments are discussed in Section 5.4. Finally, the consistency with related work and fulfillment of the research questions are reviewed in Section 5.5 and Section 5.6, respectively.

5.1 Training Data

The selection of training data had a significant impact on the models' ability to perform reliably in different conditions. Models trained on data from a single weather condition were sensitive to small changes and performed poorly in unseen environments. Thus, data were gathered in different weather conditions, improving the models' ability to generalize in new environments.

Another critical factor was the balance of the dataset. The vehicle had no velocity in a large portion of the raw data. The models learned little from those examples, and there was little use in keeping all of them. The distribution of navigational commands was also very unbalanced. This led the model to adapt biases towards certain scenarios, such as being better at taking a left turn over right a turn. After balancing the data, the models learned more efficiently and drove more reliably in various scenarios.

Furthermore, data augmentation was critical for the system's ability to generalize. Early on, the system struggled with shadow covered areas in unseen environments and rainy weather conditions. This was solved by extending the dataset with augmented images emulating shadows and rain. The data was further extended by adding hue-transformed images. The hue-transformations made the system better at detecting vehicles regardless of color. After testing the system, it was found that the system struggled with vehicle detection in images with lens flare. It can be argued that adding augmentations simulating lens flare could help improve this.

5.2 CARLA simulator

Using the CARLA simulator for both training and testing worked well for this thesis. The use of a simulator allowed for efficient collection of driving data, resulting in a training set that was both quite large (100.000+ observations) and diverse (captured in several types of weather). CARLA offers several different cities, which made it possible to train the model

in one city, and test it in the other. Testing a model in an unseen environment helps to evaluate its ability to generalize. The simulator also came with out-of-the-box support for other non-player vehicles, speed limits, and traffic-light regulated intersections; making it a good fit for the thesis' research goals.

Another critical factor was the ability to collect expert data using CARLA's built-in autopilot. The performance of an end-to-end model utilizing imitation learning is heavily dependent on the quality of the training data, and controlling the vehicle manually was quite challenging. The autopilot was therefore essential for capturing ideal steering angles. However, not all parts of the autopilot's driving were optimal. The autopilot often braked late and abruptly when approaching stationary vehicles, resulting in small margins to the vehicle ahead. This would, in turn, influence the models to exhibit some of the same behavior, making it unnecessarily difficult to stop for stationary traffic. Moreover, the autopilot tended to perform needlessly sharp turns. This thesis adjusted the braking and steering behavior of the built-in autopilot in order to collect the quality of the training data. Specifically, how the autopilot adjusted changes in speed and smoothing the autopilot's steering.

5.3 Architecture

5.3.1 Steer Predictor

This thesis is a continuation of the work done in (Haavaldsen, Aasboe, and Lindseth, 2019), which compared a CNN-LSTM model to a pure CNN. In the proposed system from the article, the steer prediction was posed as a continuous regression problem. That is, the system predicted a continuous steering angle directly. In this thesis, however, the same problem was posed as a sine wave classification.

By encoding the steering angle as a phase shifted sine wave, a gradual change in steering caused gradual changes in the output layer activations. The results indicated that this re-

formulation affected the system's steering behavior positively. The improvements were especially prominent in high-speed areas, where the system was able to maintain stable, non-oscillating, steering.

The difference in steering behavior can be observed by comparing a video from the previous article, e.g., (Aasboe, Haavaldsen, and Lindseth, 2018), to one of the videos highlighting the results from this thesis, e.g., (Aasboe, Haavaldsen, and Lindseth, 2019a).

5.3.2 Throttle-Brake Predictor

The proposed system in this thesis predicted throttle and brake values directly as two separate, non-related, values. One possible challenge with this approach is that it assumes independence between the throttle and brake output. However, this assumption does not hold in the real world. There are no driving scenarios were the correct action is to both slow down (brake) and speed up (throttle) simultaneously. Thus, there exists a dependency between the predicted brake and throttle value, which the proposed system neglects.

One possible alternative is for the system to predict acceleration instead, as seen in (Codevilla et al., 2017). This may be done by using the training data's changes in velocity to calculate target values. However, learning to predict acceleration may be more challenging than directly predicting throttle and brake. To accurately predict acceleration, the system would have to learn a relationship between the visual sensor input and the vehicle's current velocity. On the other hand, a system predicting throttle and brake values directly may simply learn always to brake if another vehicle is too close, regardless of its current velocity. While continuous braking may be unnecessary if the vehicle's current velocity is zero, it will not hurt the performance of the system.

Thus, it boils down to a trade-off between the potential performance increase from utilizing the dependency between throttle and brake, and the possible performance decrease from the added complexity.

5.4 Experimental Results

5.4.1 Experiment 1: Sequence Length

The goal of the first experiment was to illuminate how the number of hidden states in the LSTM cells affects the system's performance.

In general, model A, using a single hidden state, made the fewest amount of minor failures, suggesting that it was good at staying in the center of the lane and avoiding obstacles. However, its route completions were among the lower ones, meaning that the mistakes it made were more serious ones, such as disobeying navigational input and driving into the opposite lanes.

Model B, using five hidden states, was able to drive well in all weather conditions in *Town 4*. However, the route completions were drastically reduced in *Heavy Rain Noon* in *Town 2*. This led to several moderate and severe failures in *Town 2*

Model C, using ten hidden states, had the highest average route completion in *Town 2*. Even though it drove the furthest in complex weather conditions, it still had the fewest number of severe failures, indicating that model C was the best at handling complex scenarios. It struggled more in *Town 4*, where it had the second highest average route completion and the highest frequency of minor failures.

Model D, using 20 hidden states, had the lowest average route completion in *Town 2* and also made the most minor, moderate, and severe failures. In *Town 4* it made fewer failures but still had a low completion rate.

When evaluating the models, moderate and severe failures were more emphasized than minor failures, e.g., a model that never drove into the opposite lane was preferred over a model that never touched a lane line. Even though model A and B had few occurrences of minor failures, both models made several moderate and severe failures. Consequently, they also had the lowest route completions. The results imply that a sequence length of

five was too small, while 20 was too high. The differences between model B and C were less noticeable. However, model C was better at handling complex weather conditions and made less moderate and severe failures in *Town 2*. For these reasons, the model with a sequence length of 10 was deemed the best model.

Finding 1

Overall, the system performs most reliably when using a sequence length of **10**.

The experimental results showed that a small sequence length could lead to faster reactions. However, the network could only learn short term temporal dependencies and was consequently more susceptible for individually misclassified frames. A longer sequence length, on the other hand, led to smoother behavior, and thus more stable steering predictions. However, it could also increase the probability of learning wrong long-term temporal dependencies.

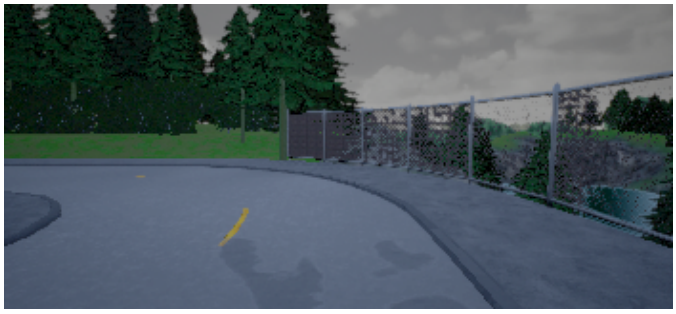
Finding 2

A small sequence length can lead to faster reactions, but the system is also more susceptible for individually misclassified frames.

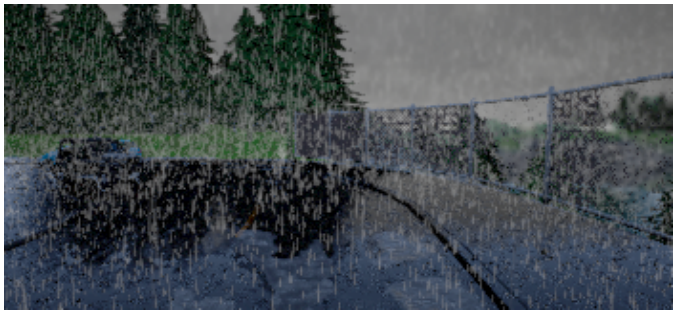
Finding 3

A longer sequence may lead to more stable steering predictions, but can also increase the probability of learning incorrect long-term dependencies.

Model C's real-time performance was not reflected in its validation loss during training. Even though Model C was deemed to have the best real-time performance, it had the highest validation loss for the steer predictions during training. This discrepancy may suggest that the calculated prediction loss during training is not a sufficient indicator of the model's real-time performance.



(a) Cloudy Noon



(b) Hard Rain Sunset

Figure 5.1: Comparison of the same turn in a simple (a) and a challenging (b) weather condition. The resolution of the images is equal to the system's input resolution, i.e., 350x160x3.

Finding 4

The prediction loss during training is not a sufficient indicator for the system's real-time performance.

5.4.2 Experiment 2: Size of Dataset

The goal of the third experiment was to investigate how the size of the training dataset affected the system's performance.

The experimental results showed that a small reduction of dataset size impacted the performance negatively. Reducing the dataset by 20% and 40% resulted in a performance

decrease of 24% and 39% respectively. Thus, the results indicate that the system was sensitive to reduction of dataset size and that the size of the gathered dataset was not excessive.

Finding 5

The system is sensitive to small changes in dataset size.

However, reducing the dataset further did not result in any drastic performance decrease. The average route completion stayed approximately the same after reducing the dataset by 40%, 60%, and 80%. Thus, the results also indicate that the system can operate somewhat successfully, even with a limited training data set.

Finding 6

The system is able to operate somewhat successfully even with a severely reduced dataset (20% of original size).

The results also suggest that the system were more sensitive to changes in dataset size in complex weather conditions. The average route completion in *Clear Noon* and *Clear Sunset* were only decreased by 22.5% after reducing the dataset by 80%. In *Sof Rain Noon* and *Clear Wet Sunset*, the same reduction resulted in a decrease in average route completion of 43.5%. In *Heavy Rain*, the average route completion was decreased by 92.3%.

Finding 7

The system is more sensitive to changes in dataset size in complex weather conditions.

5.4.3 Experiment 3: Prediction Frequency

The goal of the third experiment was to determine how the model's prediction frequency affects the performance in a real-time test.

The experimental results for model A and B revealed that the reduction of prediction frequency from 30 Hz to 15 Hz did not result in any performance decrease. Both models were able to achieve an average of 97.1% route completion. When further reducing the prediction frequency to 10 Hz, a small performance reduction of 2% was observed. However, reducing the frequency to less than 10 Hz resulted in a significant performance decreases. Model D, E, and F used a prediction frequency of 5 Hz, 3 Hz, and 1 Hz respectively, and performed 11%, 20.9%, and 69.3% worse than the best model.

Finding 8

The system perform to its full potential when using a prediction frequency of **15 Hz** or higher.

Finding 9

Using a prediction frequency lower than **5 Hz** results in a significant performance reduction.

The results show that the effect of prediction frequency varied between the different weather conditions. In simple weather conditions like *Clear Noon* and *Clear Sunset* the average route completion was not affected by the reduction of prediction frequency from 30 Hz to 5 Hz. In more complex weather conditions like *Heavy Rain Noon*, however, the average route completion was reduced by 29.9%. One probable explanation is that the system predicts more accurate steering predictions at the beginning of turns in simple weather conditions. Consequently, the system does not need to adjust its trajectory repeatedly during the turn. Thus, the effect of fewer predictions each second is minimized. In more complex weather conditions, the system produces more inaccurate predictions at the beginning of turns and relies on frequent trajectory updates.

Finding 10

The system is more sensitive to low prediction frequencies in complex weather conditions.

5.4.4 Experiment 4: Extensive Real-time Test

The goal of the final experiment was to accurately evaluate the performance of the best model from experiment 1.

Town 2 - Urban Environment

Regardless of weather condition, the system always stopped at red lights, never ignored a navigational input, and followed the current speed limit. Moreover, it performed stable lane following and always tried to be positioned in the center of the lane. When drifting out of the lane, the system corrected its trajectory both quickly and smoothly, never experiencing oscillating steering predictions. In situations where the system had to decrease its speed, it tried to let go of the throttle first. If that was insufficient, the system applied the brakes as well.

Finding 11

The system never ignored a red traffic light or a navigational command and always adjusted its velocity to match the current speed limit.

Finding 12

The system excelled at lane following in urban environments and corrected any trajectory deviations both swiftly and smoothly.

In *Clear Noon*, the system displayed smooth steering behavior, both in low speed and high-speed arias. Overall, it only made one rear-end collision and entered the oncoming lane once.

In *Clear Sunset*, the system almost performed as well as in *Clear Noon*. While it had more lane touches, it did not perform any rear-end collisions. The high-speed area in *Clear Sunset* was darker and had more shadows than in *Clear Noon*. However, the system was

still able to drive just as smoothly. It never experienced any front-end collisions, sidewalk touches, or route deviation in this weather condition.

In *Clear Wet Sunset*, the lens flare from the sun sometimes made it difficult for the system to detect vehicles in front of it, and some of these scenarios resulted in rear-end collisions. Figure 5.2b illuminates this challenging scenario. The system did not perform any other moderate or severe failures in *Clear Wet Sunset*.

Finding 13

Bright lens flare reduces the system's ability to detect vehicles.

In *Soft Rain Noon*, the system sometimes struggled to detect the vehicles in front of it, resulting in some rear-endings. Apart from that, it made no other moderate or severe mistakes. The steering was less smooth in high-speed areas than in *Clear Noon* but the system always stayed within the lane.

The results indicate that *Hard Rain Noon* was the most challenging weather condition. The system displayed a smooth diving behavior on straight stretches, but the steering was uneven in turns. The system had trouble with detecting cars in front of it, mainly black and dark grey cars. The light conditions were poor in *Hard Rain Noon*, which made the dark-colored cars blend more in with the environment. Figure 5.2a shows a black car in *Hard Rain Noon*. Moreover, the system tended to turn a little late in right turns, which resulted in it being closer to oncoming vehicles than what it had seen in the training data. In some cases, this led the system to believe it was about to collide with the oncoming vehicle even though it was still in its own lane. In high-speed areas, the steering was more erratic than in *Soft Rain Noon*, but the system always stayed within the lane.

Finding 14

The system's ability to detect vehicles are reduced in rainy weather conditions.

(a) Hard Rain Noon

(b) Clear Wet Sunset

Figure 5.2: Examples of a vehicle the system struggled to detect in some weather conditions. Figure (a) shows a black vehicle in *Heavy Rain Noon*, while Figure (b) shows a vehicle in *Clear Wet Sunset*.

Finding 15

The system's ability to detect dark-colored vehicles are reduced in low-light environments.

Town 4 - Highway

Regardless of weather condition, the system always stopped at red lights and followed the current speed limit. The system adjusted its distance to the car in front of it by adjusting the throttle and was able to handle traffic congestions in both low-speed and high-speed areas. The system sometimes chose to exit the highway when it drove in the rightmost lane. This was, however, probably an effect of lacking training data.

Finding 16

The system handles traffic congestions on highways.

Finding 17

On highways, the system primarily uses the throttle to adjust its speed according to traffic.

In *Clear Noon*, the system drove evenly in the center of the lane and experienced nine lane touches and two route deviations. Otherwise, it made no severe mistakes. The system displayed a similar driving behavior in *Clear Sunset* but made several rear-end collisions throughout the test. This suggests that the system struggled with vehicle detection in conditions with much lens flare, e.g., sunsets.

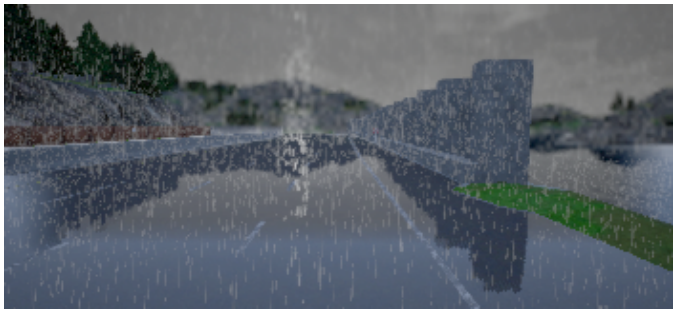
In *Soft Rain Noon*, the system drove smoothly in turns. Moreover, the system made the fewest number of lane touches and apart from *Clear Noon* it had the fewest rear-endings. However, it did lose control of the vehicle due to erratic steering once, resulting in it driving out of the road.

In *Hard Rain Noon*, the roads were shiny, and the environment was reflected on the surface of the road, making it difficult for the system to detect the lane. Figure 5.3a shows an area in *Town 4* during *Heavy Rain Noon*. Consequently, it had the most route deviation in this weather condition, in addition to three rear-end collisions.

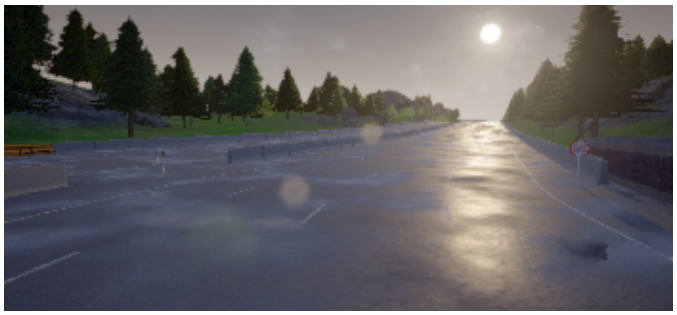
Finding 18

The system's ability to detect lane lines is reduced on wet highways.

Clear Wet Sunset turned out to be the most challenging weather condition in *Town 4*, and the system had its lowest average route completion in this weather condition. The sunlight reflected off the wet parts in the road, making it especially challenging to detect the lane lines, as seen in Figure 5.3b. As a result, the system had more trouble with staying in the lane and drove out of the road four times throughout the experiment. Additionally, the



(a) Hard Rain Noon



(b) Clear Wet Sunset

Figure 5.3: Examples where the system struggled to detect lane lines. Figure (a) shows the road in *Heavy Rain Noon*, while Figure (b) shows the road in *Clear Wet Sunset*.

system experienced four object collisions, two rear-endings, and one route deviation.

Finding 19

The system's ability to detect lane lines is severely reduced on wet highways at sunset.

5.5 Consistency with Related Work

The CNN in this thesis is based on the architecture in Bojarski et al. (2016). The authors were able to use a CNN to drive on trafficked roads with and without lane markings, parking lots, and unpaved roads. This complies with the thesis' results. Even though the implemented models were not tested on unmarked roads or parking lots, they were able to

drive on roads with lane marking, both on roads with and without pavements.

The system was trained to execute navigational commands in intersections. This was also attempted in Codevilla et al. (2017) and Hubschneider et al. (2017). In the former paper, two networks were tested: a *branched network* and a *command input network*. The *branched network* used the navigational input as a switch between a CNN and three fully connected networks, each specialized to a single intersection action, while the *command input network* concatenated the navigational command with the output of the CNN, connected to a single FC network. The authors claimed that the *command input network* performed inadequately when executing navigational commands. This does not comply with the results of this thesis. The proposed architecture takes the navigational command as input after the CNN, in a similar matter as the *command input network*, but was able to execute the given navigational commands with a high degree of success.

In Hubschneider et al. (2017) the turn indicators of the car was used as the navigational commands, which were sent as input to the network. The authors did not use an RNN, but fed three subsequent images to three CNNs and concatenated the output. It was able to perform lane following, avoid obstacles, and change lanes. The navigational commands in this thesis were introduced to the network in a similar matter, and both approaches were able to execute the navigational commands. The proposed system was not tested for lane changes, but seeing achievements in similar approaches indicates that this should be possible.

The system in this thesis used an LSTM to utilize temporal dependencies. A similar approach was attempted in Eraqi et al. (2017). They showed that adding temporal dependencies improved both the accuracy and stability of a model using a single CNN. Furthermore, Eraqi et al. (2017) also posed the steering prediction as a sine wave classification problem, and found that it led to more stable steering. Similar results can be seen in this thesis.

5.6 Fulfillment of Research Questions

The first research question was to examine how the number of hidden states in the system's recurrent module affected the overall performance. It was found that a small number of hidden states could lead to faster reactions. However, it could also affect its ability to learn long-term dependencies negatively. A larger number of hidden state, on the other hand, could lead to more stable steering predictions. Nonetheless, it could also increase the probability of learning wrong long-term temporal dependencies. Overall, it was found that using ten hidden states for the system's recurrent module was optimal for the overall performance. Thus, the first research question has been answered.

The second research question was to examine how the size of the training dataset affected the system's overall performance. It was found that a small reduction of dataset size impacted the performance negatively and that the size of the gathered dataset was not excessive. However, it was found that reducing the dataset further did not result in any drastic performance decrease and that the system was able to operate somewhat successfully even with a limited training data set. Furthermore, it was also discovered that the system was more sensitive to changes in dataset size in complex weather conditions. Hence, the second research question has been answered.

The third research question was to examine how the system's prediction frequency influenced the overall performance. It was found that the system performed to its full potential when using a prediction frequency of 15 Hz or higher. Using a prediction frequency lower than 5 Hz resulted in a significant performance reduction. It was also discovered that the system is more sensitive to low prediction frequencies in complex weather conditions. Consequently, the third research question has been answered.

CHAPTER 6

Conclusion and Future Work

This chapter concludes the work done in this thesis in Section 6.1 and proposes areas for further exploration in Section 6.2.

6.1 Conclusion

The overall goal of this thesis was to research an end-to-end system's ability to drive autonomously in diverse, simulated environments. The thesis proposed a system architecture consisting of two modules: a *Steer Predictor* and a *Throttle-Brake Predictor*. Each module combined a traditional CNN, inspired by NVIDIA's DAVE-2 system (Bojarski et al., 2016), with an LSTM layer to facilitate the learning of both spatial and temporal relationships. The system was trained using expert driving data from various simulated settings and was evaluated by its real-time driving performance in unseen simulated environments.

The proposed system was able to operate successfully in urban environments containing other vehicles, speed limits, and traffic-light regulated intersections. The real-time tests demonstrated that the system excelled at stable lane following, adjusting its trajectory both quickly and smoothly when drifting out of the lane. Furthermore, the system always stopped at red lights, always tried to execute the navigational input, and followed the current speed limit. On highways, the real-time tests demonstrated that the system was able to perform stable lane following in most weather conditions and that it was capable of handling traffic congestions in both low-speed and high-speed areas.

The system was able to operate in a range of different weather conditions. It excelled in dry weather during daytime, making few mistakes. In more challenging weather conditions, like soft rain, the system was more susceptible to failures, which could lead to the occasional rear-end collision. In even harsher weather conditions, such as heavy rain, the system was more likely to perform severe failures, such as entering the oncoming lane or front-end collisions. Nonetheless, the system was able to navigate remarkably well in harsh weather conditions where even humans could struggle.

Additionally, this thesis aimed to explore some of the uncertainties regarding the implementation of end-to-end systems. Specifically, how the system's overall performance was affected by the size of the training dataset, the allowed prediction frequency, and the number of hidden states in the system's recurrent module.

The experimental results showed that the system was sensitive to small reductions in dataset size. However, the system was able to operate somewhat successfully even with a severely reduced dataset, comprised of only 20% of the original dataset. Furthermore, the results suggest that the system performed to its full potential when using a prediction frequency of 15 Hz or higher. Using a prediction frequency lower than 5 Hz resulted in a significant performance reduction. Finally, it was found that while a small number of hidden states could lead to faster reactions, it could also make the system more susceptible to misclassifications. On the other hand, a larger number of hidden states could lead to more stable steering predictions, but could also increase the probability of learning incorrect long-term temporal dependencies. Overall, it was found that using ten hidden states for the system's recurrent module was optimal for the overall performance.

Even though the system experienced several severe failures during testing, its achievements demonstrated great potential in using end-to-end systems to accomplish fully autonomous vehicles.

6.2 Future Work

The first natural continuation of this thesis is to improve the current system further. The proposed system was not finely tuned, and many possible architectural alternatives were left unexplored. A logical extension of the system would be to integrate well known CNN architectures, such as Resnet or InceptionV3, to improve the feature extraction. Additionally, the dataset should be further extended with more scenarios, such that the system can be tested in more complex scenarios, such as environments with pedestrians and roundabouts. In the same manner, the system should be trained and tested in even more challenging conditions, such as snowy roads. Moreover, the system should be tested in scenarios containing several corner cases to investigate the durability of the system.

The system was able to operate fairly well using only visual cues. However, more sophisticated sensors, such as LiDARs, could result in a performance increase. An interesting

research area would be to explore whether the advantages of adding additional sensors outweigh the added hardware and computational cost, in addition to the maintenance such additions would require. Additionally, the proposed system required an input signal representing the current traffic light state. The desired behavior would be for the system to handle light regulated intersections without a control signal. Further work can be done to acquire this skill.

In this thesis, environments containing multiple autonomous vehicles were not researched. CARLA's multi-agent support makes it possible to evaluate how several independent models perform and interact together. This can lead to exciting research areas, such as traffic flow and congestion control in areas containing exclusively self-driving vehicles.

Another exciting field of research would be to extend the system using deep reinforcement learning. A pure end-to-end approach using imitation learning is limited by the training data. Combing the system with deep reinforcement learning would allow the system to learn beyond the examples in the training data, and improve on its own. The recent development of simulators for autonomous driving eases the research of deep reinforcement.

Finally, due to occurrences of potentially dangerous mistakes during testing, more work is required before real-world testing is justified. Furthermore, the models have only been trained on simulated data, and it is not guaranteed that the system can learn the same dependencies from a real dataset. This should be verified before the system is considered ready. When the system is deemed stable enough, the next step would be to train and test it using a more photorealistic simulator, before, finally, testing the system in the physical world.

Bibliography

Aasboe, M., Haavaldsen, H., Lindseth, F., 2018. Preliminary work - autonomous driving using a cnn-lstm: An example of end-to-end learning.
URL https://www.youtube.com/watch?v=ADzEHGmIDQQ

Aasboe, M., Haavaldsen, H., Lindseth, F., 2019a. Final system preview - autonomous vehicle control: End-to-end learning in simulated urban environments.
URL https://www.youtube.com/watch?v=IdzbPRyXaLM

Aasboe, M., Haavaldsen, H., Lindseth, F., 2019b. Playlist - autonomous vehicle control: End-to-end learning in simulated urban environments.
URL https://www.youtube.com/playlist?list=PLuO3qntnejLoxY3UFmTh8JnZXHecr72BF

Aly, M., June 2008. Real time detection of lane markers in urban streets. In: 2008 IEEE Intelligent Vehicles Symposium. pp. 7–12.

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., Zieba, K., 2016. End to end learning for self-driving cars. CoRR abs/1604.07316.
URL http://arxiv.org/abs/1604.07316

Chen, C., Seff, A., Kornhauser, A., Xiao, J., Dec 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In: 2015 IEEE International Conference on Computer Vision (ICCV). pp. 2722–2730.

Codevilla, F., Müller, M., Dosovitskiy, A., López, A., Koltun, V., 2017. End-to-end driving via conditional imitation learning. CoRR abs/1710.02410.
URL http://arxiv.org/abs/1710.02410

Dosovitskiy, A., Ros, G., Codevilla, F., López, A., Koltun, V., 2017. CARLA: an open urban driving simulator. CoRR abs/1711.03938.
URL http://arxiv.org/abs/1711.03938

Eraqi, H. M., Moustafa, M. N., Honer, J., 2017. End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. CoRR abs/1710.03804.
URL http://arxiv.org/abs/1710.03804

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., Ramanan, D., Sep. 2010. Object detection with discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence 32 (9), 1627–1645.

Geiger, A., Lauer, M., Wojek, C., Stiller, C., Urtasun, R., May 2014. 3d traffic scene understanding from movable platforms. IEEE Transactions on Pattern Analysis and Machine Intelligence 36 (5), 1012–1025.

Haavaldsen, H., Aasboe, M., Lindseth, F., 2019. Autonomous vehicle control: End-to-end learning in simulated urban environments.

Hubschneider, C., Bauer, A., Weber, M., Zöllner, J. M., Oct 2017. Adding navigation to the equation: Turning decisions for end-to-end vehicle control. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC). pp. 1–8.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., Shah, A., 2018. Learning to drive in a day.

Krizhevsky, A., Sutskever, I., Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. NIPS'12. Curran Associates Inc., USA, pp. 1097–1105.
URL http://dl.acm.org/citation.cfm?id=2999134.2999257

Lake, B. M., Baroni, M., 2017. Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. CoRR abs/1711.00350.
URL http://arxiv.org/abs/1711.00350

Lecun, Y., Cosatto, E., Ben, J., Muller, U., Flepp, B., 2004. Dave: Autonomous off-road vehicle control using end-to-end learning. Tech. Rep. DARPA-IPTO Final Report, Courant Institute/CBLL, http://www.cs.nyu.edu/~yann/research/dave/index.html.

LeCun, Y., Muller, U., Ben, J., Cosatto, E., Flepp, B., 2005. Off-road obstacle avoidance through end-to-end learning. In: Proceedings of the 18th International Conference on Neural Information Processing Systems. NIPS'05. MIT Press, Cambridge, MA, USA, pp. 739–746.
URL http://dl.acm.org/citation.cfm?id=2976248.2976341

Lenz, P., Ziegler, J., Geiger, A., Roser, M., June 2011. Sparse scene flow segmentation for moving object detection in urban environments. In: 2011 IEEE Intelligent Vehicles Symposium (IV). pp. 926–932.

Marcus, G., 2018. Deep learning: A critical appraisal. CoRR abs/1801.00631.
URL http://arxiv.org/abs/1801.00631

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., Feb. 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.
URL http://dx.doi.org/10.1038/nature14236

Pomerleau, D. A., 1989. Advances in neural information processing systems 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, Ch. ALVINN: An Autonomous Land Vehicle in a Neural Network, pp. 305–313.
URL http://dl.acm.org/citation.cfm?id=89851.89891

Rothe, R., Timofte, R., Van Gool, L., 12 2015. Dex: Deep expectation of apparent age from a single image.

Santana, E., Hotz, G., 2016. Learning a driving simulator. CoRR abs/1608.01230.
URL http://arxiv.org/abs/1608.01230

Shalev-Shwartz, S., Shamir, O., Shammah, S., 2017. Failures of deep learning. CoRR abs/1703.07950.
URL http://arxiv.org/abs/1703.07950

Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., Darrell, T., 2018. BDD100K: A diverse driving video database with scalable annotation tooling. CoRR abs/1805.04687.
URL http://arxiv.org/abs/1805.04687

Zhang, H., Geiger, A., Urtasun, R., Dec 2013. Understanding high-level semantics by modeling traffic patterns. In: 2013 IEEE International Conference on Computer Vision. pp. 3056–3063.

APPENDIX A

Preliminary Paper: End-to-end Learning in Simulated Urban Environments

Autonomous Vehicle Control: End-to-end Learning in Simulated Urban Environments

Hege Haavaldsen[⋆][0000−0001−6372−1989], Max Aasbø[*][0000−0003−3462−165X], and Frank Lindseth[⋆⋆][0000−0002−4979−9218]

Norwegian University of Science and Technology, Trondheim, Norway

Abstract. In recent years, considerable progress has been made towards a vehicle's ability to operate autonomously. An end-to-end approach attempts to achieve autonomous driving using a single, comprehensive software component. Recent breakthroughs in deep learning have significantly increased end-to-end systems' capabilities, and such systems are now considered a possible alternative to the current state-of-the-art solutions.

This paper examines end-to-end learning for autonomous vehicles in simulated urban environments containing other vehicles, traffic lights, and speed limits. Furthermore, the paper explores end-to-end systems' ability to execute navigational commands and examines whether improved performance can be achieved by utilizing temporal dependencies between subsequent visual cues.

Two end-to-end architectures are proposed: a traditional Convolutional Neural Network and an extended design combining a Convolutional Neural Network with a recurrent layer. The models are trained using expert driving data from a simulated urban setting, and are evaluated by their driving performance in an unseen simulated environment.

The results of this paper indicate that end-to-end systems can operate autonomously in simple urban environments. Moreover, it is found that the exploitation of temporal information in subsequent images enhances a system's ability to judge movement and distance.

Keywords: End-to-end learning · Imitation Learning · Autonomous Vehicle Control · Artificial Intelligence · Deep Learning

1 Introduction

We are currently at the brink of a new paradigm in human travel: the fully autonomous, self-driving car. Only 50 years ago, cars were completely analog devices with almost no mechanisms for assisting the driver. Over the decades, additional features, controls, and technologies have been integrated, and cars have evolved into exceedingly complex machines.

⋆ These authors contributed equally to this work.
⋆⋆ Corresponding author.

In recent years, substantial progress has been made towards a vehicle's ability to operate autonomously. Primarily, two different approaches have emerged. The prevailing state of the art approach is to divide the problem into a number of sub-problems and solve them by combining techniques from computer vision, sensor fusion, localization, control theory, and path planning. This approach requires expert knowledge in several domains and often results in complex solutions, consisting of several cooperating modules.

Another approach is to develop an end-to-end solution, solving the problem using a single, comprehensive component, e.g., a deep neural network. A technique for training such a system is to employ imitation learning. This entails studying expert decisions in different scenarios, to find a mapping between the perceived environments and the executed actions. While some believe that the black-box characteristics of such systems makes them untrustworthy and unreliable, others point to recent years' advances in deep-learning and argue that end-to-end solutions show great potential.

However, end-to-end systems cannot make the correct navigational decision solely based on a perceived environment. It is also necessary to incorporate a user's intent in situations that require a decision (e.g., when approaching an intersection). Hence, an end-to-end system should be able to receive and adapt to navigational commands.

The objective of this paper is to investigate end-to-end systems' ability to drive autonomously in simulated urban environments. Specifically, to study their performance in environments containing other vehicles, traffic lights, and speed limits; to examine their ability to oblige navigational commands in intersections; and to explore if a system can improve its performance by utilizing temporal dependencies between subsequent visual cues.

This paper seeks to combine different aspects from recent research in the field of end-to-end learning for autonomous vehicles. Concretely, the use of navigational commands as network input, and the exploitation of temporal dependencies between subsequent images. There have been no attempts - to our knowledge - to combine both techniques in one system. Hopefully, this can lead to a more complete end-to-end system and improved driving quality.

The rest of this paper is organized as follows. Section 2 presents previous related work, while Section 3 addresses the environment in which the data was collected and the experiments were conducted. Section 4 reviews the collection and preprocessing of the data. The model architectures are presented in Section 5. Section 6 and 7 covers the experimental setup and results, while section 8 discusses the results. Finally, Section 9 covers the conclusions.

2 Related Work

There have been several advances in end-to-end learning for autonomous vehicles over the last decades. The first approach was seen already in 1989 when a fully connected neural network was used to control a vehicle [10]. In 2003 a proof-of-concept project called DAVE emerged [9], showing a radio controlled

vehicle being able to drive around in a junk-filled alley and avoiding obstacles. DAVE truly showed the potential of an end-to-end approach. Three years later NVIDIA developed DAVE-2 [1], a framework with the objective to make real vehicles drive reliably on public roads. DAVE-2 is the basis for most end-to-end approaches seen today [2,7]. The project used a CNN to predict a vehicle's steering commands. Their model was able to operate on roads with or without lane markings and other vehicles, as well as parking lots and unpaved roads.

Codevilla et al. [2] further explored NVIDIA's architecture by adding navigational commands to incorporate the drivers intent into the system and predicted both steering angle and acceleration. The authors proposed two network architectures: a *branched network* and a *command input network*. The *branched network* used the navigational input as a switch between a CNN and three fully connected networks, each specialized to a single intersection action, while the *command input network* concatenated the navigational command with the output of the CNN, connected to a single fully connected network.

Hubschneider et al. [7] proposed using turn signals as control commands to incorporate the steering commands into the network. Furthermore, they proposed a modified network architecture to improve driving accuracy. They used a CNN that receives an image and a turn indicator as input such that the model could be controlled in real time. To handle sharp turns and obstacles along the road the authors proposed using images recorded several meters back to obtain a spatial history of the environment. Images captured 4 and 8 meters behind the current position were added as an input to make up for the limited vision from a single centered camera.

Eraqi, H.M. et. al.[4] tried to utilize the temporal dependencies by combining a CNN with a Long Short-Term Memory Neural Network. Their results showed that the C-LSTM improved the angle prediction accuracy by 35 % and stability by 87 %.

3 Environment

Training and testing models for autonomous driving in the physical world can be expensive, impractical, and potentially dangerous. Gathering a sufficient amount of training data requires both human resources and suitable hardware, and it can be time consuming to capture, organize and label the desired driving scenarios. Moreover, the cost of unexpected behavior while testing a model may be considerable.

An alternative is to train and test models in a simulated environment. A simulator can effectively provide a variety of corner cases needed for training, validation, and testing; while removing safety risks and material costs. Additionally, the labeling of the dataset can be automated, removing the cost of manual labeling, as well as the potential of human error.

The drawback, however, is the loss of realism. A simulation is only an imitation of a real-world system, and a model trained on only simulated data may not be able to function reliably in the real world. Nonetheless, a simulator

can give a good indication of a model's actual driving performance and serves well for benchmarking different models. Once a model can perform reliably in a simulated environment, the model can be fine-tuned for further testing in real environments.

In this paper, the CARLA simulator [3] is used to gather training data and to evaluate the proposed models. CARLA is an open source simulator built for autonomous driving research and provides an urban driving environment populated with buildings, vehicles, pedestrians, and intersections.

4 Data Generation

4.1 Data Collection

When performing imitation learning, the quality of the training data plays a significant role in a model's ability to perform reliably in different conditions. However, a model trained only using expert data in ideal environments may not learn how to recover from perturbations. To overcome this, several types of driving data was captured. Expert driving was captured using CARLA's built-in autopilot, resulting in center-of-lane driving while following speed-limits. To capture more volatile data, a randomly generated noise value was added to the autopilot's outgoing control signal. This resulted in sudden shifts in the vehicle's trajectory and speed, which the autopilot subsequently tried to correct. To eliminate undesirable behavior in the training set, only the autopilot's response to the noise was collected, not the noisy control signal. Finally, recovery from possible disaster states was captured by manually steering the vehicle into undesired locations, e.g., the opposite lane, or the sidewalk; while recording the recovery.

For each recorded frame, images from three forward-facing cameras (positioned at the left, center, and right side of the vehicle) were captured, along with the vehicle's control signal (i.e., steering angle, throttle, and brake values), and additional information (i.e., speed, speed limit, traffic light state, and *High-Level Command*). The *High-Level Command* (HLC) is the active navigational command, labeling the data with the user's current intent. Possible HLCs are: *follow lane*, *turn left at the next intersection*, *turn right at the next intersection*, and *continue straight ahead at the next intersection*.

Two different datasets were gathered, one for training and one for testing. The training set was captured in CARLA's Town 1, while the test set was captured in Town 2. Data were gathered in four different weather conditions: Clear noon, cloudy noon, clear sunset, and cloudy sunset. The training set contained driving data captured both with and without other vehicles. The test set exclusively contained driving data alongside other vehicles. All data were captured in environments without pedestrians. Table 1 summarizes the gathered datasets. All expert data was captured driving 10 km/h below the speed limit to match the velocity of other vehicles.

Table 1. The collected datasets. An observation contains the captured data from a single rendered frame in the simulator.

Dataset	Number of Observations	Size [GB]
Training	117 889	31.5
Testing	23 173	8.32

4.2 Data Preparation

For each recorded observation, a data sample was created containing the center image, the vehicle's control signal, and the additional information. Moreover, to simulate the recovery from drifting out of the lane, two new data samples were generated using the observation's left and right images. To counteract the left and right images' positional offset, the associated steering angle was shifted by +0.1 and -0.1 respectively.

For each data sample, a new augmented sample was generated using one of the desirable transformations picked at random. These included a random change in brightness or contrast, the addition of Gaussian noise or blur, and the addition of randomly generated dark polygons differing in position and shape.

When recording the datasets, the majority of the observations were captured driving straight. To prevent an unbalanced dataset, data samples with a small steering angle were downsampled (by removal), while data samples with a large steering angle were upsampled (by duplication). Additionally, the data samples corresponding to the different intersection decisions (i.e., turn left, turn right, or straight ahead) were balanced by analyzing the distributions of HLC-properties in the dataset and downsampling the over-represented choices. Finally, some of the data samples where the vehicle was not moving (e.g., waiting for a red light) were downsampled.

5 Model Architectures

In this paper, two related end-to-end architectures are proposed. The first is a Convolutional Neural Network (CNN) inspired by NVIDIA's DAVE-2 [1] system, while the second extends the CNN with Long-Short-Term-Memory (LSTM) units to capture temporal dynamic behavior.

5.1 CNN Model

The CNN model consists of two connected modules: a feature extractor and a prediction module. The former uses a CNN to extract useful features from the input image, while the latter combines the detected features with the additional inputs (i.e., current speed, speed limit, traffic light state, and HLC) to predict a control signal (i.e., steering angle, throttle and brake values).

The convolutional part of the model is inspired by the architecture used in NVIDIA's DAVE-2 system [1]. The modified network takes a 180x300x3 image

as input, followed by a cropping layer and a normalization layer. The cropping layer removes the top 70 pixels from the image, while the normalization layer scales the pixel values between -0.5 and 0.5. Next follows six convolutional layers, all using a ReLU activation function. The first three layers use a 5x5 filter, while the last three use a 3x3 filter. The first four layers use a stride of 2, while the last two use a stride of 1. The output of the last convolutional layer is flattened resulting in a one-dimensional feature layer containing 768 nodes.

The output from the convolutional layers is concatenated with the additional input containing the speed, speed-limit, traffic-sign, and HLC values. The concatenated layer serves as an input for the predictive part of the model which consists of three dense layers containing 100, 50, 10 nodes respectively. All the dense layers use a ReLU activation function. The last of the dense layers are finally connected to an output layer, consisting of 3 nodes. The complete architecture is shown in Figure 1.

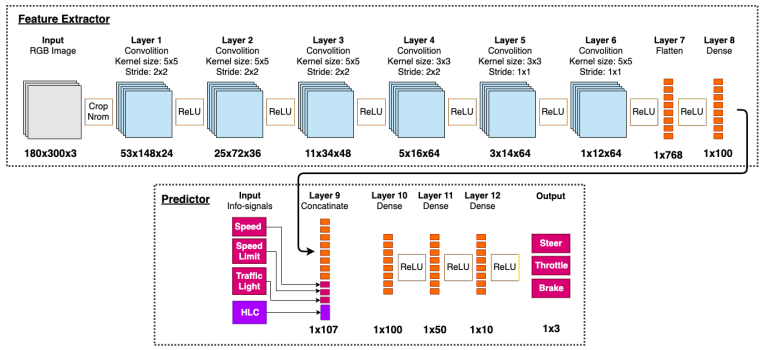


Fig. 1. The architecture of the CNN model. The model accepts a single RGB image and predicts a control signal.

5.2 CNN-LSTM Model

The CNN-LSTM model consists of two connected modules: a feature extractor and a temporal prediction module. The former follows the same architecture as the previous model, shown in Figure 1. The feature extractor is connected to an LSTM layer with 5 hidden states. The model uses a sequence of feature extractions over time to predict a control signal. This allows the model to learn temporal dependencies between time steps.

For each time step, the output of the feature extractor is concatenated with the additional input containing speed, speed limit, traffic light, and an HLC. This is sent through a dense layer containing 100 nodes and fed into an LSTM

layer with 10 nodes. For each time step in the sequence, the LSTM layer sends its output to itself. At the last time step, the output is sent through a dense layer, consisting of three nodes. This is the final prediction. The complete architecture is shown in Figure 2.

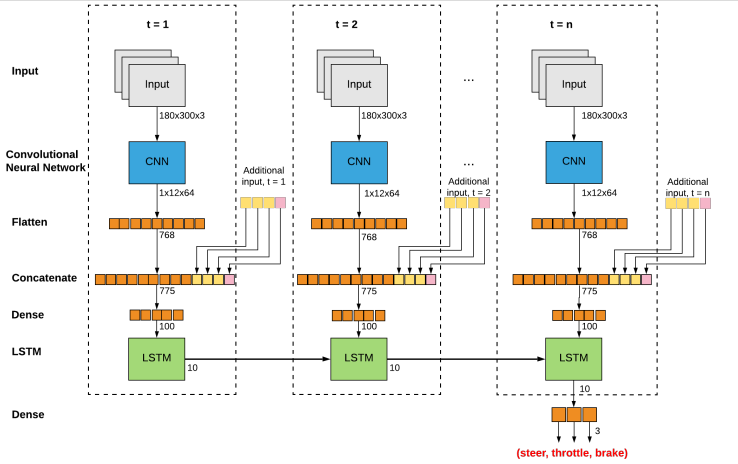


Fig. 2. The architecture of the CNN-LSTM model.

6 Experimental Setup

6.1 Training

Training and validation The dataset was split into a training set (70%) and a validation set (30%).

For the CNN-LSTM, the data samples were further structured into sequences of length five, using a sampling interval of three. The sequence length determines the number of time steps the LSTM layer is able to remember, while the sampling interval decides the period between successive individual time steps within the sequences. Figure 3 illustrates the structuring of sequences from 15 data samples.

Hyperparameters Both models were trained using an Adam optimizer [8] and an Mean Squared Error loss function. The models were trained for a 100 epochs, with a batch size of 32 data-samples. The models' weights were recorded after each epoch along with the associated validation error. After the training was

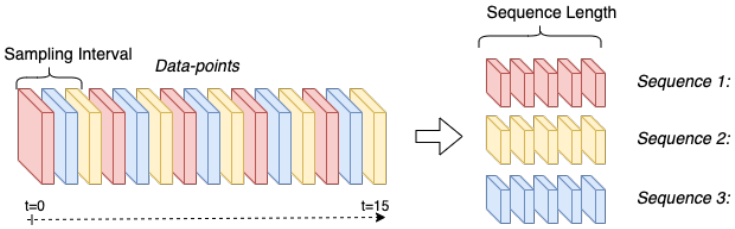


Fig. 3. The structuring of sequences from an array of data samples. A sequence length of five and a sampling interval of three is used.

complete, the weights associated with the lowest validation error were chosen for further testing.

6.2 Testing

After training, each model's predictive and real-time performance was measured. The predictive performance was tested by exposing the models to the unseen test set while calculating the average prediction error. The real-time performance was tested by letting the models control a simulated vehicle in CARLA's second town. Each model was to drive through a predefined route ten times. The test evaluator provided HLCs to the model before each intersection. A model's performance was measured in the number of route completions, the average route completion percentage, and the number of failures. Model failures were recorded according to severity. Touching a lane line was considered a minor failure, while a low-speed rear-ending or an object collision was considered a moderate failure. Object collisions without recovery or an ignored HLC were considered a severe failure. Catastrophic failures consisted of either entering the opposite lane, disregarding a red traffic light, or colliding with oncoming traffic.

7 Experimental Results

7.1 Validation and Test Error

After 100 epochs of training, both models' training and validation loss had stabilized. The CNN and CNN-LSTM had their lowest validation error after epoch 66 and 81 respectively. The models were then evaluated on the unseen test set from CARLA's second town. On the test set, the CNN was able to predict a control signal with an average error of 0.023, a 43% increase compared to its validation error. The CNN-LSTM was able to predict a control signal with an average error of 0.022, a 57% increase compared to its validation error.

Table 2. Test and validation loss during the training of the models.

Model	Validation Loss	Test Loss
CNN	0.016	0.023
CNN-LSTM	0.014	0.022

7.2 Real-time Test in Simulated Environment

Both models were real-time tested in CARLA's second town. To test their ability to handle various driving scenarios in an urban environment, each model drove a predefined route ten times. The results are described below. Two videos were created to show a successful run and some common failures. One demonstrates the CNN model [5] and the other demonstrates the CNN-LSTM model [6].

Table 3. Summary of test results. Each model attempted to drive a predefined route (Figure **??**) ten times. A model's performance were measured in the number of route completions and the average route completion percentage.

Model	Route Completions	Avg. Route Completion
CNN	2	$56\% \pm 39$
CNN-LSTM	5	$81\% \pm 9$

Table 4. Average failures per run. The models' failures were recorded according to severity. Touching a lane line was considered a minor failure, while a low-speed rear-ending or an object collision was considered a moderate failure. Object collisions without recovery or an ignored HLC were considered a severe failure. A catastrophic failure consisted of either entering the opposite lane, disregarding a red traffic light, or colliding with oncoming traffic.

Model	Minor	Moderate	Severe	Catastrophic
CNN	2.10 ± 1.73	0.65 ± 0.99	0.25 ± 0.44	0.23 ± 0.57
CNN-LSTM	2.90 ± 1.72	0.10 ± 0.31	0.05 ± 0.22	0.17 ± 0.38

CNN Out of ten runs, the CNN model was able to complete the predefined route twice, with an average completion percentage of 56% over all runs. The model performed well on lane following and drove reliably in the center of the lane most of the time. The model handled most intersections but had a tendency to perform very sharp right turns. This resulted in 2.1 minor failures (i.e., the vehicle touching the lane line) per run. It always tried to follow the provided HLC, and never ignored a traffic light. It was able to handle complex light conditions, such as direct sunlight and dark shadows in the road. The model hit objects without recovery several times, leading to 0.5 severe failures per run.

The vehicle rarely hit objects outside of the lane, but occasionally struggled to stop for other vehicles, leading to several low speed rear-end collisions. In total, the model had 0.1 object collisions, and 1.2 rear-end collisions per run. Finally, 0.7 times per run it struggled to find the lane after a turn, leading to a catastrophic failure. The model usually held the speed limit but found it hard to slow down fast enough to a speed limit when the speed was high. The results are summarized in Table 3 and Table 4.

CNN-LSTM The CNN-LSTM model was able to complete the predefined route five out of ten times, with an average completion percentage of 81% over all runs. It drove reliably in the center of the lane most of the time but tended to perform sharp turns. This lead to 2.9 minor failures (i.e., lane line touches) per run. The model always tried to follow the provided HLC, and never ignored a traffic light. It managed to handle various light conditions, such as direct sunlight and dark shadows in the road. The vehicle rarely struggled to stop for other objects or vehicles, resulting in 0.1 object collisions and 0.1 rear-end collisions per run. Finally, 0.5 times per run it struggled to find the lane after a turn, leading to a catastrophic failure. The model adapted well to the different speed limits. The results are summarized in Table 3 and Table 4.

8 Discussion

This paper proposed two architectures for an end-to-end system: a traditional CNN inspired by NVIDIA's DAVE-2 system [1], and an extended design combining the CNN with an LSTM layer to facilitate learning of temporal relationships. Both models were able to follow a lane consistently and reliably. Any disturbances or shifts in the trajectory were quickly corrected, without being overly sensitive. The CNN exhibited less volatile steering compared to the CNN-LSTM in the high-speed stretch of the route, but the CNN-LSTM outperformed the CNN in turns following high-speed stretches. Additionally, both models obeyed all traffic lights and always tried to follow the provided HLC at intersections.

When introducing other vehicles, the differences between the models became more apparent. The CNN-LSTM adapted its speed according to traffic, and only rear-ended another vehicle once throughout the whole experiment. In scenarios where another vehicle blocked most of the view (e.g., when following another vehicle closely in a turn), the CNN-LSTM seemed to be able to use past predictions as a guide. The CNN, on the other hand, experienced more trouble when driving alongside other vehicles. Although the model, to some degree, adapted its speed according to traffic, it often failed to react upon sudden changes. This led to frequent rear-endings throughout the experiment.

The difference between the models' performance may be explained in their architectural differences. The CNN-LSTM model used five subsequent observations when making predictions. This allowed it, by all indication, to learn some important temporal dependencies - acquiring some knowledge about the relationships between movement, change in object size, and distance. The CNN

model, however, interpreted each observation independently, which restrained its ability to understand motion. It still learned to brake when approaching a vehicle, but was not able to differentiate between fast approaching and slow approaching objects. Predictions related to distance were solely dependent on the size of objects. Moreover, the CNN could not rely on past predictions when faced with confusing input, which seemed to result in more unreliable behavior.

It should be mentioned that although the CNN model had 28 % less minor failures than the CNN-LSTM model, its completion rate was 32 % lower than the CNN-LSTM. The reduction in minor failures was probably a result of the lower average route completion, not an indication of better performance.

8.1 Consistency with Related Work

The implemented models in this paper are based on the architecture in [1]. The authors were able to use a CNN to drive on trafficked roads with and without lane markings, parking lots and unpaved roads. This complies with this paper's results. Even though the implemented models were not tested on unmarked roads or parking lost, they were able to drive on roads with lane marking, both on roads with and without pavements.

Codevilla et. al. [2] claimed that their *command input network* performed inadequately when executing navigational commands. This does not comply with the results of this paper. The proposed architecture takes the navigational command as input after the CNN, in a similar matter to the *command input network*, but was able to execute the given navigational commands with a high degree of success.

In [7] the turn indicators of the car was used as the navigational commands, which were sent as input to the network. The authors did not use an RNN, but fed three subsequent images to three CNNs and concatenated the output. It was able to perform lane following, avoid obstacles and change lanes. The navigational commands in this paper were introduced to the network in a similar way, and both approaches were able to execute the navigational commands. The proposed system was not tested for lane changes, but seeing achievements in similar approaches indicates that this should be possible.

The CNN model in this paper was extended with an LSTM to utilize temporal dependencies. A similar approach was attempted in [4]. They showed that adding temporal dependencies improved both the accuracy and stability of a model using a single CNN. Similar results can be seen in this paper.

9 Conclusion

The results of the experiments indicates that end-to-end systems are able to operate autonomously in simulated urban environments. The proposed systems managed to follow lanes reliably in varying lighting conditions and were not disrupted by disturbances or shifts in trajectory. They were able to abide by traffic

lights and speed limits and learned to execute different navigational commands at intersections.

Both systems managed to adapt its speed according to traffic, but their ability to respond to sudden changes varied. The CNN-LSTM were, by all indication, able to acquire some insight into the relationships between movement, distance, and change in the perceived size of objects. The regular CNN, interpreting each observation independently, was not able to learn these essential temporal dependencies. Hence, the results suggest that exploiting temporal information in subsequent images improves an end-to-end systems ability to drive reliably in an urban environment.

Even though the systems' performed several mistakes during testing, their achievements demonstrated great potential for using end-to-end systems to accomplish fully autonomous driving in urban environments.

References

1. Bojarski, M., Testa, D.D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., Zieba, K.: End to end learning for self-driving cars. CoRR **abs/1604.07316** (2016), http://arxiv.org/abs/1604.07316
2. Codevilla, F., Müller, M., Dosovitskiy, A., López, A., Koltun, V.: End-to-end driving via conditional imitation learning. CoRR **abs/1710.02410** (2017), http://arxiv.org/abs/1710.02410
3. Dosovitskiy, A., Ros, G., Codevilla, F., López, A., Koltun, V.: CARLA: an open urban driving simulator. CoRR **abs/1711.03938** (2017), http://arxiv.org/abs/1711.03938
4. Eraqi, H.M., Moustafa, M.N., Honer, J.: End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. CoRR **abs/1710.03804** (2017), http://arxiv.org/abs/1710.03804
5. Haavaldsen, H., Aasbø, M.: Autonomous driving using a cnn: An example of end-to-end learning, https://youtu.be/Q37jTFZjK2s
6. Haavaldsen, H., Aasbø, M.: Autonomous driving using a cnn-lstm: An example of end-to-end learning, https://youtu.be/ADzEHGmIDQQ
7. Hubschneider, C., Bauer, A., Weber, M., Zöllner, J.M.: Adding navigation to the equation: Turning decisions for end-to-end vehicle control. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC). pp. 1–8 (Oct 2017). https://doi.org/10.1109/ITSC.2017.8317923
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980
9. Lecun, Y., Cosatto, E., Ben, J., Muller, U., Flepp, B.: Dave: Autonomous off-road vehicle control using end-to-end learning. Tech. Rep. DARPA-IPTO Final Report, Courant Institute/CBLL, http://www.cs.nyu.edu/~yann/research/dave/index.html (2004)
10. Pomerleau, D.A.: Advances in neural information processing systems 1. chap. ALVINN: An Autonomous Land Vehicle in a Neural Network, pp. 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989), http://dl.acm.org/citation.cfm?id=89851.89891

APPENDIX B

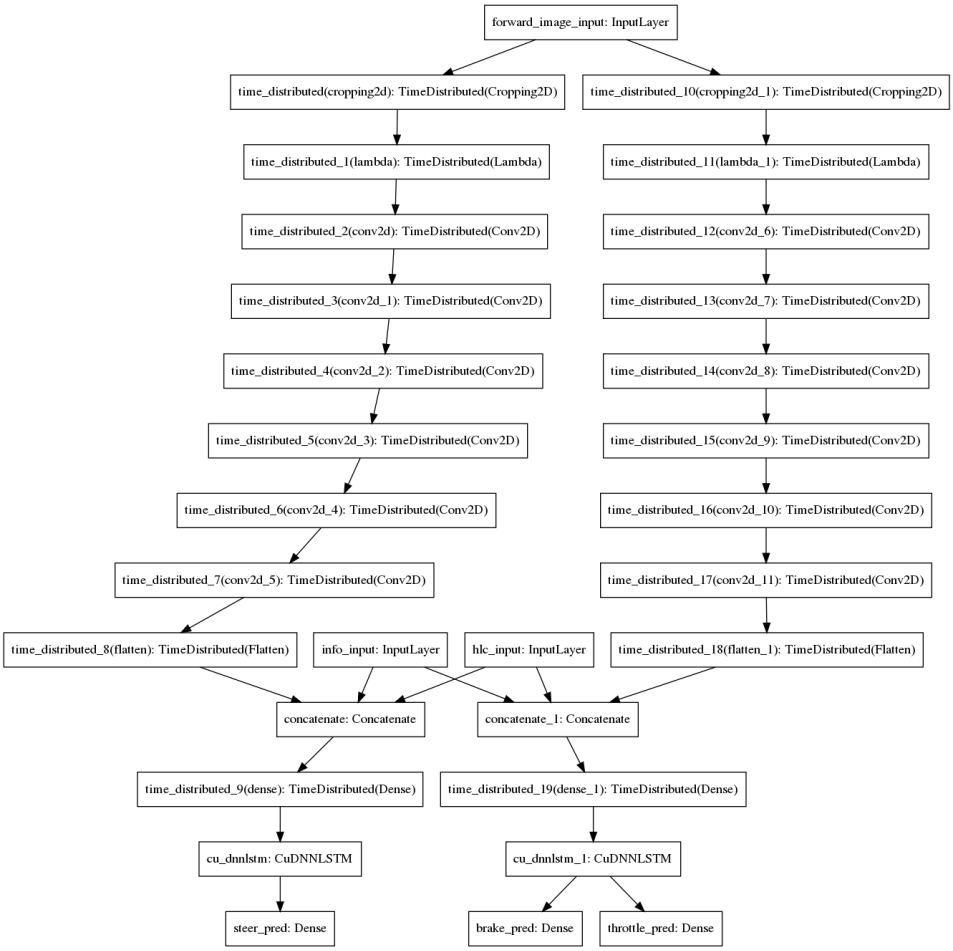
System Implementation

```
forward_image_input = Input(
                        shape=(seq_length,160,350,3),
                        name="forward_image_input"
                      )
info_input = Input(shape=(seq_length,3), name="info_input")
hlc_input = Input(shape=(seq_length,6), name="hlc_input")

x = TimeDistributed(Cropping2D(cropping=((50,0),(0,0))))(forward_image_input)
x = TimeDistributed(Lambda(lambda x: ((x/255.0) - 0.5)))(x)
x = TimeDistributed(Conv2D(24,(5,5),strides=(2,2), activation="relu"))(x)
x = TimeDistributed(Conv2D(36,(5,5),strides=(2,2), activation="relu"))(x)
x = TimeDistributed(Conv2D(48,(5,5),strides=(2,2), activation="relu"))(x)
x = TimeDistributed(Conv2D(64,(3,3),strides=(2,2), activation="relu"))(x)
x = TimeDistributed(Conv2D(64,(3,3), activation="relu"))(x)
x = TimeDistributed(Conv2D(64,(3,3), activation="relu"))(x)
conv_output = TimeDistributed(Flatten())(x)

x = concatenate([conv_output, info_input, hlc_input])

x = TimeDistributed(Dense(100, activation="relu"))(x)
x = CuDNNLSTM(10, return_sequences = False)(x)
steer_pred = Dense(10, activation="tanh", name="steer_pred")(x)

x = TimeDistributed(Cropping2D(cropping=((50,0),(0,0))))(forward_image_input)
x = TimeDistributed(Lambda(lambda x: ((x/255.0) - 0.5)))(x)
x = TimeDistributed(Conv2D(24,(5,5),strides=(2,2), activation="relu"))(x)
x = TimeDistributed(Conv2D(36,(5,5),strides=(2,2), activation="relu"))(x)
x = TimeDistributed(Conv2D(48,(5,5),strides=(2,2), activation="relu"))(x)
x = TimeDistributed(Conv2D(64,(3,3),strides=(2,2), activation="relu"))(x)
x = TimeDistributed(Conv2D(64,(3,3), activation="relu"))(x)
x = TimeDistributed(Conv2D(64,(3,3), activation="relu"))(x)
conv_output = TimeDistributed(Flatten())(x)

x = concatenate([conv_output, info_input, hlc_input])

x = TimeDistributed(Dense(100, activation="relu"))(x)
x = CuDNNLSTM(10, return_sequences = False)(x)
```

```
throtte_pred = Dense(1, name="throttle_pred")(x)
brake_pred = Dense(1, name="brake_pred")(x)

model = Model(
            inputs=[forward_image_input, info_input, hlc_input],
            outputs=[steer_pred,throtte_pred,brake_pred]
        )
```

APPENDIX C

System Architecture


forward_image_input: InputLayer
time_distributed(cropping2d): TimeDistributed(Cropping2D)
time_distributed_10(cropping2d_1): TimeDistributed(Cropping2D)
time_distributed_1(lambda): TimeDistributed(Lambda)
time_distributed_11(lambda_1): TimeDistributed(Lambda)
time_distributed_2(conv2d): TimeDistributed(Conv2D)
time_distributed_12(conv2d_6): TimeDistributed(Conv2D)
time_distributed_3(conv2d_1): TimeDistributed(Conv2D)
time_distributed_13(conv2d_7): TimeDistributed(Conv2D)
time_distributed_4(conv2d_2): TimeDistributed(Conv2D)
time_distributed_14(conv2d_8): TimeDistributed(Conv2D)
time_distributed_5(conv2d_3): TimeDistributed(Conv2D)
time_distributed_15(conv2d_9): TimeDistributed(Conv2D)
time_distributed_6(conv2d_4): TimeDistributed(Conv2D)
time_distributed_16(conv2d_10): TimeDistributed(Conv2D)
time_distributed_7(conv2d_5): TimeDistributed(Conv2D)
time_distributed_17(conv2d_11): TimeDistributed(Conv2D)
time_distributed_8(flatten): TimeDistributed(Flatten)
info_input: InputLayer
hlc_input: InputLayer
time_distributed_18(flatten_1): TimeDistributed(Flatten)
concatenate: Concatenate
concatenate_1: Concatenate
time_distributed_9(dense): TimeDistributed(Dense)
time_distributed_19(dense_1): TimeDistributed(Dense)
cu_dnnlstm: CuDNNLSTM
cu_dnnlstm_1: CuDNNLSTM
steer_pred: Dense
brake_pred: Dense
throttle_pred: Dense

APPENDIX D

CARLA's prebuilt Cities

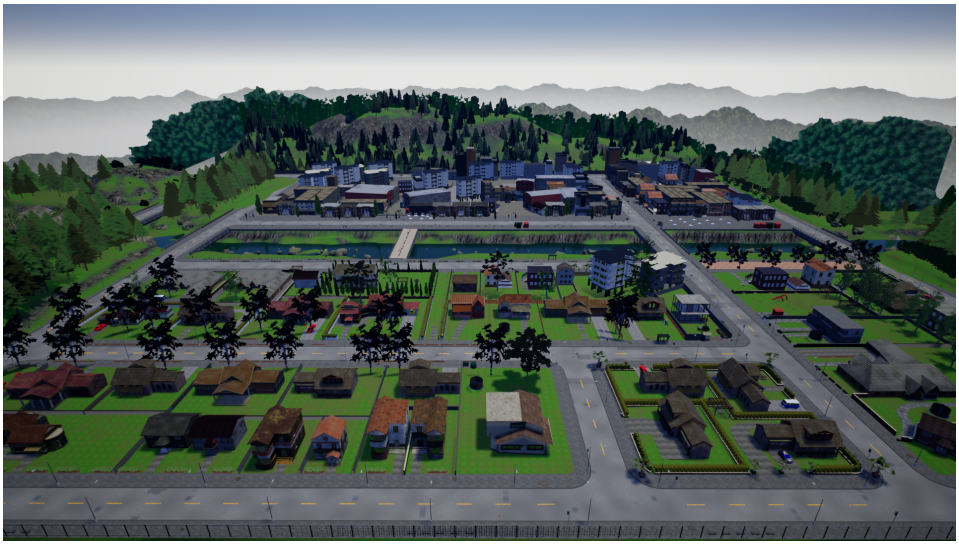


Figure D.1: *Town 1* in CARLA is situated in an urban environment and has 2.9 km of drivable road. It consists of roads with two lanes, one in each driving direction, and intersections with traffic lights. Additionally, there are multiple buildings and vegetations in the town.

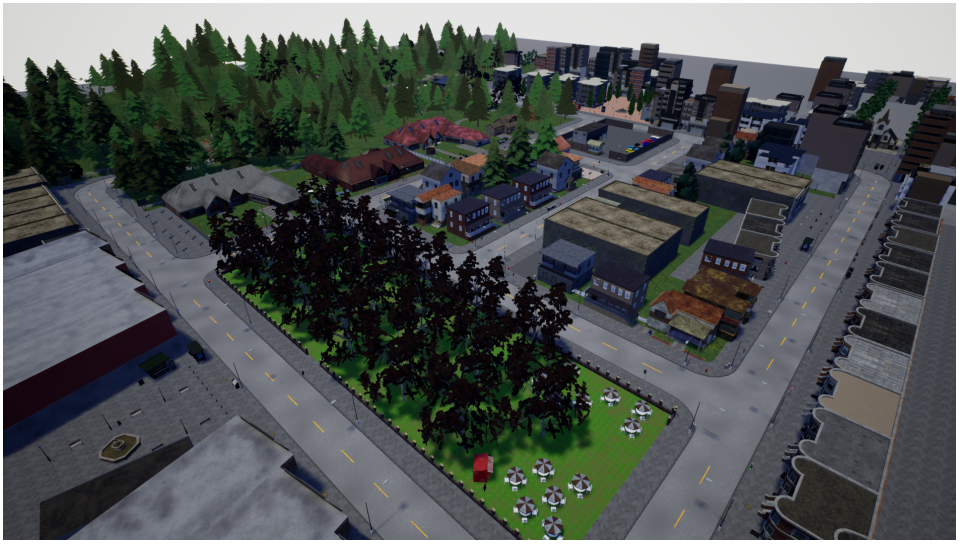


Figure D.2: CARLA's second town, *Town 2*, is also situated in an urban environment, but have different buildings from the first town and 1.4 km of drivable road. Many buildings are taller, creating more complex light conditions.

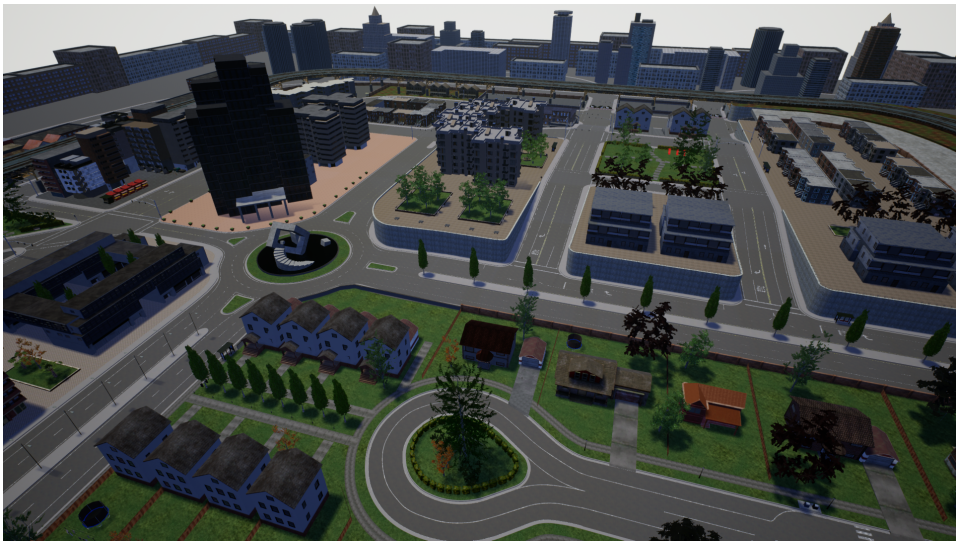


Figure D.3: *Town 3* is in an urban environment with multiple lanes, a roundabout, and uphills and downhills.

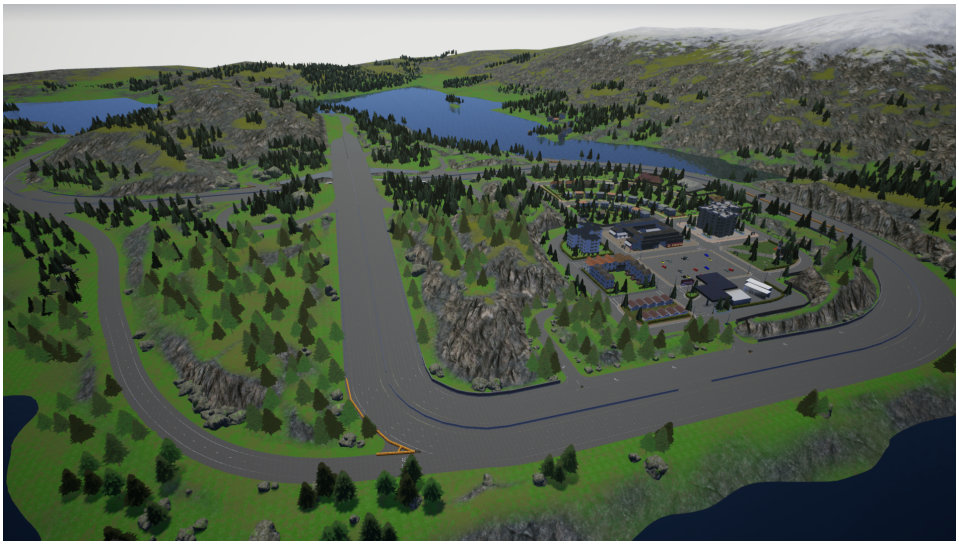


Figure D.4: *Town 4*, the fourth town is a combination of a highway and an urban environment. The highway has four lanes in the same driving direction. The urban environment can be accessed from the highway by taking on of several exits.

Figure D.5: *Town 5* is quite similar to *Town 3*, except there are no roundabouts.

NTNU
Norwegian University of
Science and Technology