CHALMERS
UNIVERSITY OF TECHNOLOGY

Personalized Software in Heavy-Duty Vehicles

Exploring the Feasibility of Self-Adapting Smart Cruise Control Using Machine Learning

Master's thesis in *Complex Adaptive Systems* and *Systems, Control and Mechatronics*

Charlotte De Geer & Alex Matsson

DEPARTMENT OF ELECTRICAL ENGINEERING

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

Master's thesis 2023

Personalized Software in Heavy-Duty Vehicles

Exploring the Feasibility of Self-Adapting Smart Cruise Control Using Machine Learning

Charlotte De Geer & Alex Matsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2023

Personalized Software in Heavy-Duty Vehicles
Exploring the Feasibility of Self-Adapting Smart Cruise Control Using Machine Learning
Charlotte De Geer & Alex Matsson

Supervisor: Robin Karlsson, Volvo GTT
Examiner: Jonas Fredriksson, Chalmers Department of Electrical Engineering

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

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Personalized Software in Heavy-Duty Vehicles
Exploring the Feasibility of Self-Adapting Smart Cruise Control Using Machine Learning
Charlotte De Geer & Alex Matsson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This study aims to explore possible and feasible ways to personalize driving functions for heavy-duty vehicles. The idea is to use machine learning algorithms, specifically focusing on Long Short-Term Memory (LSTM) neural networks and traditional classification algorithms for current state velocity predictions, independent velocity predictions, and driver classification. The goal is to explore potential approaches for enhancing the existing software to improve the vehicle's drivability while not compromising fuel consumption. The research methodology involved collecting relevant data from the heavy-duty vehicle, including various readings using the CAN bus and map-based data. The data was preprocessed and used to train and evaluate the LSTM neural network and traditional classification algorithms. The results obtained were satisfactory for all of the models. The predictions from the LSTM models were adequate. The one-second velocity predictions were favorable when compared to the ten-second velocity predictions. From the training progress, it is possible to see that the model learns and identified trends. Furthermore, the classification accuracy using traditional and LSTM classifiers ranged from 93 % to 99 %. These findings highlight the challenges and limitations of employing LSTM neural networks and traditional classification algorithms for software adaptation. Further research is necessary to explore alternative approaches, such as using sufficient and more suitable data for transfer- and deep learning. The insights gained from this study help comprehend machine learning applications in heavy-duty vehicles and suggest future research efforts to enhance software adaptation and thus improve vehicle performance.

Keywords: Long Short-Term Memory, LSTM, classification, driver behavior, adaptability, machine learning, ML, neural network, time series.

Acknowledgements

We would like to express our deepest gratitude and appreciation to our three supervisors, Klara Palm, Oscar Olesen, and Simon Lillskog. Their support, insightful guidance, and constructive feedback have been invaluable in shaping this master's thesis. We are also grateful for Robin Karlsson, his mentorship, and the opportunities he has provided us during this journey have been essential. We extend our thanks to Jonas Fredriksson for his expertise and critical evaluation of this thesis. Klara, Oscar, Simon, Robin, and Jonas's thorough examination and valuable feedback have undoubtedly enhanced the quality and rigor of our research.

Charlotte De Geer & Alex Matsson, Gothenburg, May 2023

List of Acronyms

Below is the list of acronyms used throughout this thesis in alphabetical order:

ACC	Adaptable Cruise Control
ADAS	Advanced Driver-Assistance System
BLF	Binary Logging Format
BPNN	Backward-Propagation Neural Networks
CAN	Controller Area Network
CHLF	Combined Hierarchy Learning Framework
CSV	Comma-Separated Values
DBN	Deep Belief Network
DBRNN	Deep Bidirectional Recurrent Neural Network
ECU	Electronic Control Units
FP	False Positive
FN	False Negative
GHG	Greenhouse Gas
GPS	Global Positioning System
GRU	Gated Recurrent Units
HF	Heavy Filtering
k-NN	k-Nearest Neighbors
LF	Light Filtering
LIN	Local Interconnect Network
LNG	Liquefied Natural Gas
LSTM	Long Short-Term Memory
MPC	Model Predictive Control
MSR	Multi-target Sigmoid Regression
MF	Medium Filtering
NLP	Natural Language Processing
OL	Overlap

RBF	Radial-Basis Function
RLS	Recursive Least Square
RNN	Recurrent Neural Networks
R&D	Research & Development
RQ	Research Questions
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
spl	Split
TP	True Positive
TN	True Negative
WS	Windows Size

Contents

List of Figures

List of Tables

1

Introduction

Trucks are an essential component of the global transportation system, which means they have a critical impact on the environment and traffic safety. There are several initiatives and ongoing developments to reduce the impact, which of course, include electrically powered trucks. The growth of electrically powered trucks is expected to reach 7% by 2027, but this means that 93% of all trucks will still be diesel-powered [1].

To reduce the negative consequences of diesel-powered trucks, it is possible to improve the efficiency of truck operations. One way to achieve this is by optimizing the vehicle speed adjustment patterns, which can have a major impact on fuel consumption and driving performance [2]. This type of functionality already exists and may be personalized. Taking the driver's behavior into consideration will enhance the driver experience. This might also lead to fewer interruptions of the optimizing software.

Volvo Group has developed an energy-saving system called I-See. The I-See software helps reduce fuel consumption by using the truck's kinetic energy when approaching a hill [3]. With this system, a fuel reduction of 5% is possible compared to a truck without the system [3]. The primary objective of the I-See software is to use the truck's kinetic energy as efficiently as possible by allowing velocities below and above the set speed of the cruise control, hereafter known as underspeed and overspeed. This functionality seamlessly works with the adaptive cruise control (ACC), an advanced driver-assistance system (ADAS) function. The ACC is adaptable to external factors such as vehicles in front but does not adjust the set speed when the scenario, which is not pure topography, demands so.

Machine learning algorithms have rapidly advanced and entered the transportation systems industry. These algorithms can improve the efficiency of truck operations and take driver behavior into account [4][5]. By implementing a machine learning algorithm, driver pattern recognition is possible using sensor data. This type of data can be considered a time series classification problem as data from vehicle sensors sequentially contains temporal information and is suitable for algorithms such as neural networks and classifiers [6].

1.1 Aim

This thesis aims to investigate future improvements in cruise control systems like the I-See system. When the I-See software is active, the underspeed and overspeed might not please the driver, leading the driver to re-assume control over the vehicle and disabling the I-See software. Mapping driver behavior patterns can enable adaptability and ensure the software's core objective, i.e., reducing fuel consumption, is achieved without any overrides. This will contribute to personalized driving behavior, and thus the truck will become self-adapting. To continue to improve this system by making it adaptable, machine learning techniques will be used to identify driver behavior trends. This means the I-See software will be more flexible and adaptable to driver tendencies and behaviors.

The aim of the study has been split into four main research questions (RQ):

- RQ1: What are possible and feasible ways for personalizing a cruise control using machine learning?
- RQ2: What type of data is available, and how can it be extracted? What data will be of interest?
- RQ3: What machine learning algorithms are suitable for adaptability with regard to driver behavior?
- RQ4: Why and how is the self-adapting software method better than the existing solution?

1.2 Literature study

Within the scope of the research questions, related research and studies exist. Personalized driving assistance includes research from predicting a driver's behavior to using driver patterns in functionalities like ACC. Within personalized driver assistance, it is often mentioned that this type of aid can contribute to more effective and safe driving [4][5].

Modeling driver behavior is challenging since it is intuitively a stochastic and highly variable act. In [7], the authors employ a data-centric approach using a large amount of driving data collected from drivers in various traffic conditions. The approach includes statistical machine learning algorithms successfully applied to the data to model driver behavior. Along with successful driver identification, the authors present that the data shows that a driver's pedal position differs from driver to driver, which strengthens the motivation behind this study; driver behavior is personal.

In [8], the authors review personalization approaches for ADAS systems. These systems consider driver preferences and behavior for implicit personalization of the ADAS. Implicit personalization means that the driver does not choose how the software should act, the software adapts itself after the drivers' patterns. The authors

recognize a personalization process that includes three steps. These steps help define how software is adaptable and able to be personalized. Further, the authors discuss studies that have focused on personalized ACCs, which include classifying driver behavior into a handful of categories or driver profiles, using recursive least squares (RLS) to adapt the ACC in real-time to an individual driver and also implementing model predictive control (MPC) to personalize the ACC. This review does not explore machine learning algorithms, even if it is close to the subject, but it does provide an intuitive and simple explanation regarding personalization within ADAS systems.

In [9], the authors investigate a speed control algorithm for an ACC especially designed for scenarios with curved roads. The authors use a self-learning RLS to identify the coefficients of the model. This study also notices the problematic scenario with an active ACC when entering a curve with no vehicle in front. The ACC will not affect the vehicle speed since curvature is not part of the external factors that the ACC takes into consideration. This study is of interest since it shows how a speed profile can be predicted using data from a specific driver. However, the equation expressing the model is not what is sought after since it is based on specific experimental data.

A similar approach is used in [10], where a speed model based on driving styles, road, and vehicle characteristics is proposed. They start with a mathematical model to calculate a safe velocity in curves, which they then expand with a parameter that models a driving style. No machine learning is used in this work, but it still shows a possible way to introduce some personalization.

Machine learning approaches are commonly used in the rapidly developing field of autonomous vehicles. The benefits and future challenges of such methods are discussed in [11] and [12]. They present popular techniques and architectures, such as convolutional neural networks, recurrent neural networks, and deep reinforcement learning approaches. In [13], an artificial neural network predicts gear shifting to improve shifting time, power loss, fuel efficiency, and driver comfort. The prediction is based on throttle percentage, vehicle velocity, and acceleration.

Further, Zhu et al. explore personalization within autonomous driving to avoid conflict between machines and humans when cooperating in [14]. To achieve autonomous cooperative driving, the authors implement a combined hierarchy learning framework (CHLF) based on gated recurrent units (GRU). The authors mainly focus on lane changing within autonomous driving, which is not within the scope of this study, but does present some interesting methods such as recurrent neural networks (RNN), the GRU, to be exact. This type of neural network is advantageous with time series forecasting and multi-step ahead predictions.

Huang et al. investigate how to achieve accurate throttle and wheel loader state predictions using deep machine learning [15]. To achieve the final predictions, the authors use multiple long short-term memory (LSTM) networks along with two backward-propagation neural networks (BPNNs), which use the temporal features

from the LSTM. The researchers focus on feasible ways to accomplish autonomous mobile construction machinery by learning and imitating an expert wheel loader operator's operations. Their study is enticing since it uses deep machine learning to mimic a specific driver's actions, similar to personalizing software based on the drivers' data.

In [16], the authors propose a novel multi-target sigmoid regression integrated with a deep belief network (MSR-DBN) to predict the vehicle's speed and front wheel angle based on driver behavior. The DBN is constructed of several restricted Boltzmann machines, and the prediction is made of MSR. This approach is quite successful; the predicted speeds are almost identical to the test data. It must be mentioned, though, that the prediction is made one time step ahead, which is quite a short horizon and might be why the results are satisfactory.

Since this study will focus on specific driving scenarios, lateral acceleration is an important factor in vehicle dynamics and is used for control and stability. In [17], the researchers examine predicting lateral acceleration using a neural network, namely a low computational complexity feed-forward neural network with a simple structure. Still, their results show that it is powerful and accurate. This shows that simplicity is something to strive for.

On the subject of control and stability, detecting abnormal driver behavior to ensure the driver's safety can contribute to a better and more comfortable driving experience. In [18], the authors use a deep bidirectional recurrent neural network (DBRNN) to detect time series abnormality within driver behavior. The DBRNN learns the correlation between sensory inputs and impending driver behavior, resulting in accurate and high-horizon action prediction. The results show that the system can accurately identify actions such as lane changing and acceleration up to five seconds before the driver commits the action.

Regarding safety, the authors in [19], use machine learning techniques to classify the driver behavior as safe or unsafe to lower the causes of accidental death in traffic crashes. This study presents two approaches, one where two types of classification tools, namely support vector machines and feed-forward neural networks, are used to extract features to detect safe and unsafe driving accurately. The other proposal is a methodology to label driving intervals as safe or unsafe by looking at the relationship between speed and lateral and longitudinal acceleration of the vehicle. The study resulted in an accurate result with up to 90% accuracy for both models. Safety is an important topic, and to be able to classify whether driving is safe or unsafe, especially using a neural network, is within the scope.

The I-See functionality focuses on minimizing fuel consumption, which is why Perotta et al. study on modeling fuel consumption in trucks using machine learning [5], specifically three models using support vector machine, random forest, and artificial neural network, is of interest. The idea is to develop a new fuel consumption model to help fleet managers review the existing vehicle routing decisions. The result shows that all models work but that the RF outperforms the others. It mainly

shows which variables are correlated with fuel consumption, where variables such as gross weight, inclination and curvature are all affecting fuel consumption.

The authors of [20] acknowledge that modeling driving behavior under dynamic driving conditions is complex, which makes it difficult to make a quantitative analysis of the relationship between driving behavior and fuel consumption. As a result, they explore using machine learning to evaluate the fuel efficiency of driving behavior using naturalistic driving data. This includes using an unsupervised spectral clustering algorithm that divides the data into three clusters. Then the dynamic and natural data from driving is integrated to generate a model, which is later used as training data for the deep learning model. The trained deep learning-based model can then be used to predict the fuel consumption associated with different driving behavior. The goal is to offer end-to-end fuel consumption feature prediction, which can be applied in advanced driving assistance systems. This is interesting since the study connects driver behavior to fuel consumption.

1.3 Understanding adaptability, a field study

One must define adaptability to explore whether a system is adaptable. It might seem self-explanatory; it is how something can adapt to a situation. According to [21], adaptability is defined as: "*the ease with which a system or parts of the system may be adapted to the changing requirements*". This aligns with what might seem obvious, but how does one experience adaptability? This is something that needs to be clarified before implementing a solution.

Moreover, an adaptive algorithm will be implemented, which is part of the realm of machine learning. An adaptive algorithm is an algorithm that changes its behavior while it is run according to available information along with a predetermined reward mechanism [22], i.e., the reward needs to be adapted to the driver's preferred driving style. Measuring adaptability is difficult because when absolute adaptability is reached, is it even felt?

To clarify and define vehicle adaptability while driving, the topic was discussed in an interviewing format with R&D employees with extensive truck driving experience at Volvo Group. To maintain consistency during the discussions, several questions were composed.

- What is adaptability for you?
- Can you give an example of adaptability?
- Can you go through your thought process when approaching a specific driving scenario such as an uphill? What circumstances affect you?
- What do you think about current software such as I-See? Do you use it regularly? If not, why?

What was eminent after the interview is that perceived adaptability is not that common, especially not within truck driving. What was mentioned was features such as smart cruise control and wind wiper motion activating when the rain starts to fall. Other examples included a more stiff steering wheel at high speeds and altering pedal resistance based on the driver's preference. In conclusion, there are numerous factors affecting the driver.

There are other fields where adaptability, such as clothes adapting to your body shape and leather shoes stretching to fit your feet better. Adaptability is absent, but your clothes and shoes suddenly fit and feel better. This is the difference, a cruise control adapts after external factors and not the user behavior, whereas clothes and shoes adapt after their user. The second example is what is sought after.

Insightful examples were given during the discussion regarding factors affecting drivers when driving. When approaching a hill or curve, the width of the road, the visibility, and the load are key factors affecting the driver's behavior. Further, factors such as the road condition, the direction of the curve or inclination, and velocity when approaching a driving scenario also influence the behavior.

When answering the last question, all who participated in the interview responded positively. The participants' main objective at Volvo Group is to improve software, such as I-See, further, thus constantly testing it. This means that they both enjoy the software and use it regularly.

1.4 Scope and limitations

This study aims to investigate the adaptability of software that controls the speed of the truck. As the driver's behavior in relation to speed mostly correlates with accelerating and braking actions, other operations, such as gear shifting and steering wheel action, will be excluded. Furthermore, lane changes and the driver's mental state will also be disregarded.

The chosen programming language was Python, and no other languages were considered. Also, the method did not include using any existing software implemented in the truck. The method is, therefore, stand-alone. The data accessible were of different formats, but to simplify the reading and storing of the data, files that were not in the preferred format were neglected. The available data were limited to the data that could be extracted from the internal data collecting system. Data from sensors such as cameras and LiDAR were not covered.

The data must be sequential to catch driver behavior trends, meaning that the input data to a model must be in a window of consecutive order. As a consequence of the sequential format of the data, the implemented models must be able to handle sequential data. Furthermore, as unsupervised machine learning is unsuitable for this study, only supervised learning was regarded.

1.5 Sustainability and ethical aspects

Within the EU, heavy-duty vehicles and buses are responsible for a quarter of the greenhouse gas (GHG) emissions emitted due to road transport, and even though there are improvements in fuel consumption efficiency, the emissions are still rising. The targets right now from the regulations of heavy-duty vehicles from the EU state that CO_2 emissions must be lowered by 15% by 2025 and 30% by 2030 [23].

To reduce CO_2 emissions, Volvo Group has initiatives such as fully electric-powered heavy-duty vehicles and alternative fuels such as liquefied natural gas (LNG) and biogas. The I-See software is also part of their initiatives to lower the fuel consumption within their truck and make it more effective. This study is employed to enhance this effective I-See software further and ensure it is used as often as possible. With a more personalized I-See system, the driver would not be as aware of the system running. Even though all these initiatives are necessary and may help lower the CO_2 emissions, it is important to remember that these initiatives still encourage a maximal consumer-friendly future, which in the end does not help to save the planet.

One must be wary of the consequences when entering the machine learning realm. Machine learning often relies on large amounts of personal data; personal data is essential when implementing personalized software. Privacy concerns arise when individuals' data is collected, stored, and processed without their informed consent or when it is used in ways that compromise their privacy. Proper data anonymization, minimization, and secure handling of personal information are essential to address privacy concerns.

A machine learning model is often called a black box model; it is challenging to understand how they reach their decisions and possesses low interpretability. This lack of transparency can lead to distrust, and efforts should be made to be able to explain the decisions made and develop interpretable models. Further, when implementing machine learning algorithms in a heavy-duty vehicle, the question about accountability and responsibility arises since a machine learning algorithm

makes autonomous decisions. Determining who is accountable for the actions or consequences of machine learning models can be challenging, especially in complex systems. Establishing clear lines of responsibility and accountability is crucial to address potential ethical issues.

1.6 Report outline

The report is outlined as follows, beginning with theory in Chapter 2, existing research on which this thesis is based is introduced. The following chapter, Chapter 3, will focus on the data available and data collection methods. The consecutive chapter, Chapter 4, presents the research design and analysis techniques employed in the study. Chapter 5 exhibits the results combined with an in-depth analysis and discussion. After that, Chapter 6 presents possible further research aspects, and the thesis is concluded in Chapter 7.

2

Theory

The theory chapter starts with a comprehensive exposition of machine learning and continues as it narrows down with specific machine learning algorithms considered and implemented. These algorithms include recurrent neural networks, specifically long short-term memory neural networks, along with common classification and clustering algorithms such as support vector machine and nearest neighbor. The chapter finishes with theory that regards performance measures to evaluate the above-mentioned algorithms.

2.1 Machine learning

Machine learning is a method that lets machines take advantage of data to enhance their performance on a specific task, in other words, methods that enable machines to learn. In [24], Tom M. Mitchell describes machine learning with an intuitive definition as follows: *"A computer program is said to* ***learn*** *from experience* ***E*** *with respect to some class of tasks* ***T*** *and performance measure* ***P*** *if its performance at tasks in* ***T****, as measured by* ***P****, improves with experience* ***E****."*

The core of machine learning is that strategies and algorithms that have had valid results in the past will continue to produce valid results. This means that a learning machine will master generalization from the training experience and can then perform accurately when given unseen data.

Machine learning algorithms are usually implemented when no fully satisfactory algorithm is available or if it is tedious, even impossible, for a human to manually develop the needed algorithms. These algorithms are part of artificial intelligence as a broad sub-field and contain algorithms such as regression analysis, support vector machines for classification, and artificial neural networks. This study mostly focuses on classification algorithms and artificial neural networks, which will henceforth be referred to as neural networks.

Learning is traditionally split into three categories, namely supervised, unsupervised, and reinforcement learning. The first learning category is when the learning machine is presented with the correct output to connect to the input. In the second learning category, there is no correct output to match the given input. Therefore, the learner has to find structure by itself within the input. In the third and last of the traditional

categories, the machine interacts with a dynamic environment where it usually has to perform a task or tasks to reach a goal. As it interacts with the environment, it is fed feedback to maximize its reward based on its actions. This study focuses on the first learning category; supervised learning.

2.1.1 Neural network and types of neural networks

To identify trends or patterns, in other words, to learn, a state-of-the-art solution is to implement a neural network. This computational system is designed to learn from data and make predictions or decisions based on that learning.

This computational system is inspired by the biological structure of an animal brain, mainly focusing on mimicking neurons and their synapses. Within a neural network, a connection is meant to resemble synapses. These connections can broadcast a real number signal to other artificial neurons. The neurons are often distributed into layers; a neural network can consist of several layers with several neurons [25], as illustrated in Figure 2.1 for a visual representation of the simplest form of a neural network, namely a feedforward neural network. In a feedforward neural network, the information moves exclusively in one direction from neuron to neuron, with no cycles or loops of feedback [26].

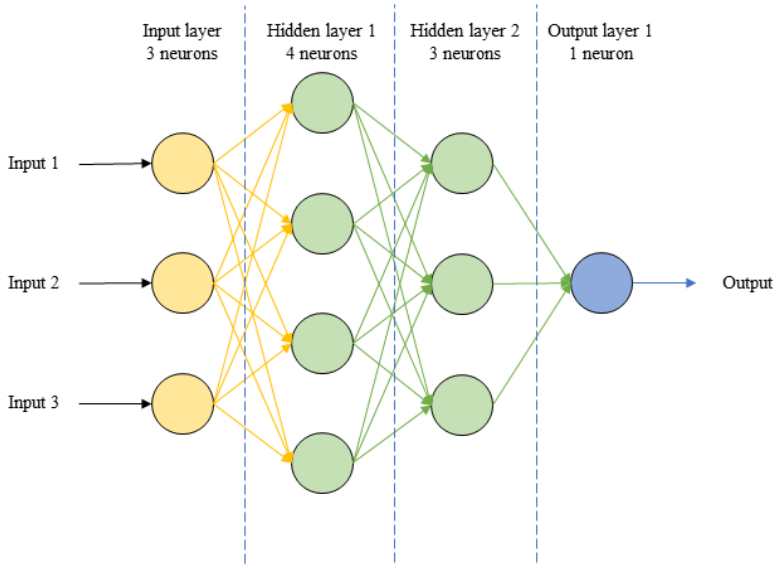

Figure 2.1: A feedforward neural network with two hidden layers, one with four neurons and one with three neurons. This example takes three input features, for example, legal speed, curvature, and inclination, and predicts an output of velocity.

The basic process of a neuron is as follows: a neuron receives signals and processes the information by calculating the weighted sum of inputs, see Figure 2.2. Then an activation function is applied to the weighted sum and the bias, illustrated as the output in Figure 2.2. The neuron proceeds to transmit the results to the next layer of neurons. The bias is a constant value added to the inputs to account for factors

that cannot be represented by the input features alone. The activation function is a mathematical function that can introduce non-linearity and determines the neuron's output. It also has a normalizing effect which restricts the output from becoming too large. The activation function transforms the input signal into an output signal, allowing neural networks to model complex relationships [27].

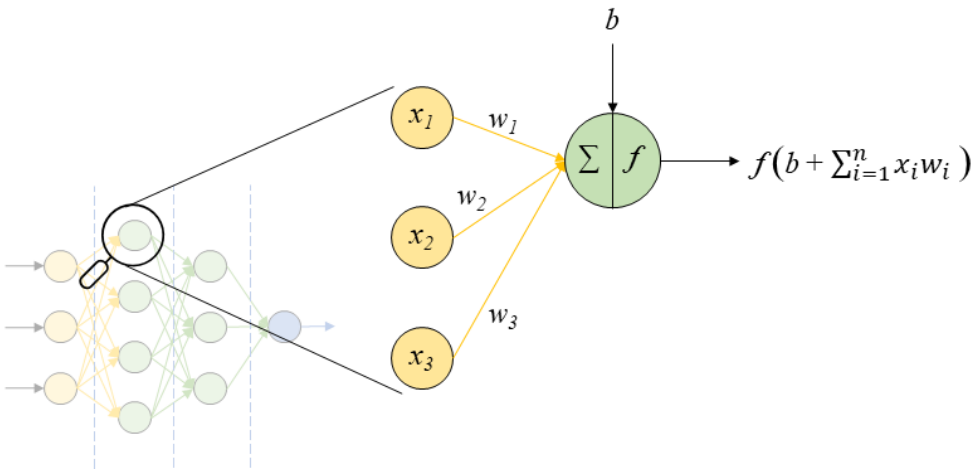

Figure 2.2: An in-depth visualization of a neuron in a simple feedforward neural network. The $\sum$ represents the sum of the input and weights. The f represents the activation function applied to the sum and the bias.

During training the neuron learns to produce the right output by changing the weights. The learning process is the core of neural networks and is typically done using an optimization algorithm called backpropagation. This algorithm can be divided into four steps: forward propagation, loss computation, gradient computation, and parameter update.

The forward propagation produces the predicted output using the given input. Every neuron of each layer takes the weighted sum of its input, applies an activation function, and relays the result to the next layer. The model's accuracy in the loss computation is quantified by comparing the error between the predicted output and the actual output using a loss function. The next step is gradient computation based on the loss with respect to the model's parameters. The gradients demonstrate how sensitive the loss function is to modification of the parameters within the model. The gradients are computed layer by layer using the chain rule, starting at the output layer and moving back toward the input layer. When the gradients are computed it is time to do the parameter update. The gradients indicate which direction the parameters should be updated to minimize the loss function, this is usually done using gradient descent. These four steps are repeated for a number of iterations, or until wanted convergence is reached.

2.1.1.1 Time series

A time series is a number of data points indexed by time: $x_t, x_{t-1}, ..., x_{t-n}$, where n is the number of data points. The data points are often a sequence equally spaced in time. The price of a share on the stock market is one example of a time series, the vehicle speed of a truck sampled each second is another.

2.1.1.2 Recurrent neural networks

A recurrent neural network (RNN) utilizes cycles between neurons to create a dynamic behavior, which means that output from neurons can affect inputs of the same neurons. See Figure 2.3. In other words, the feedback enables that information about the last time step can be stored and influence the processing of oncoming time steps, i.e., employ a memory. This makes RNNs appropriate for sequential data, see, e.g., [28] for a thorough explanation of RNNs.

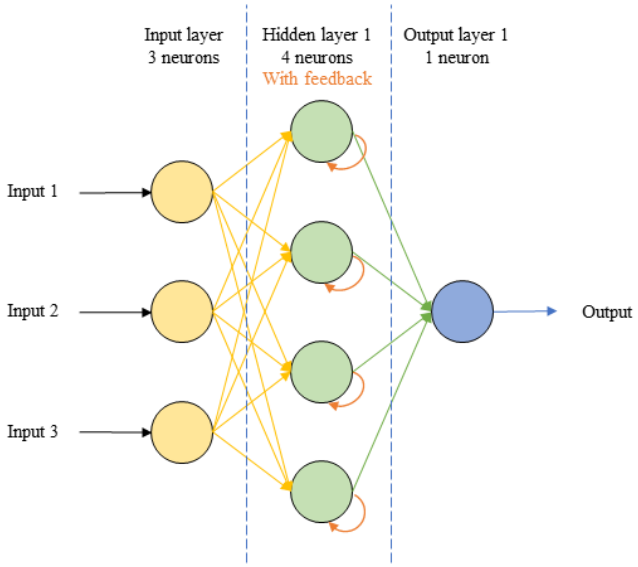

Figure 2.3: A single layer recurrent neural network.

For an intuitive explanation, imagine that the data is the velocity of a truck for five seconds. When trying to guess the velocity for the next second, the sixth second, it would be helpful to know the truck's velocity the second before. The guess would be even easier if all five previous seconds were known. This would provide knowledge, or a memory, such as if the velocity has been constant, increasing, or decreasing.

In an RNN, the input is combined with the hidden state from the previous time step to predict an output. The hidden state is then updated using both the input and the previous hidden state to acquire its memory,

$$a_t = tanh(W_a a_{t-1} + W_x x_t + b_a) \tag{2.1}$$

$$o_t = softmax(W_o a_t + b_o) \tag{2.2}$$

where a_t and a_{t-1} are the current respectively previous hidden state, W_a, W_x and W_o are weight matrices, b_a and b_o are biases and o_t is the output. Hyperbolic tangent (*tanh*) and *softmax* are activation functions.

At each time step, knowledge from previous time steps will affect the predicted output of the current time step, see Figure 2.4 and 2.5. The feedback loop in Figure 2.4 is as long as the sequence fed into the network. Take the example given above with the prediction of the truck's velocity. The feedback loop would then be five time steps long. In each RNN cell within the feedback loop, the output and the current hidden state are computed using the *tanh* activation function. This is done using the input from the current time step and the hidden state from the previous time step.

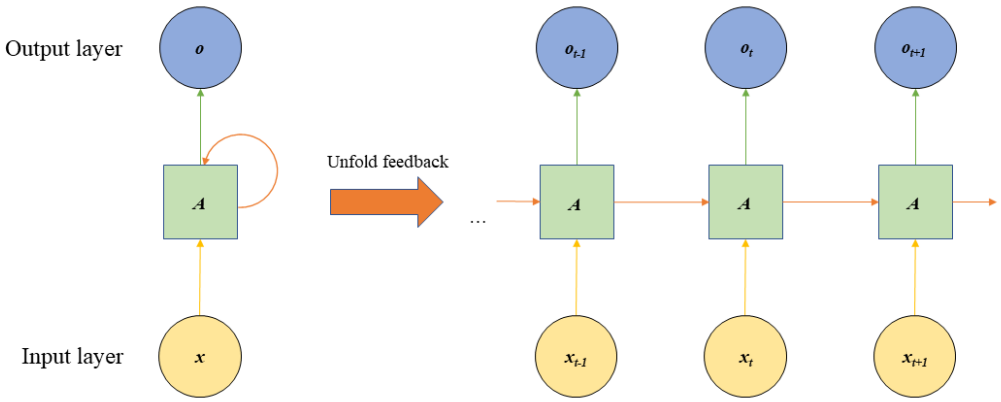

Figure 2.4: The feedback of an RNN neuron and the feedback unfolded. The *A's* represents an RNN cell and should not be confused with the neurons within a hidden layer.

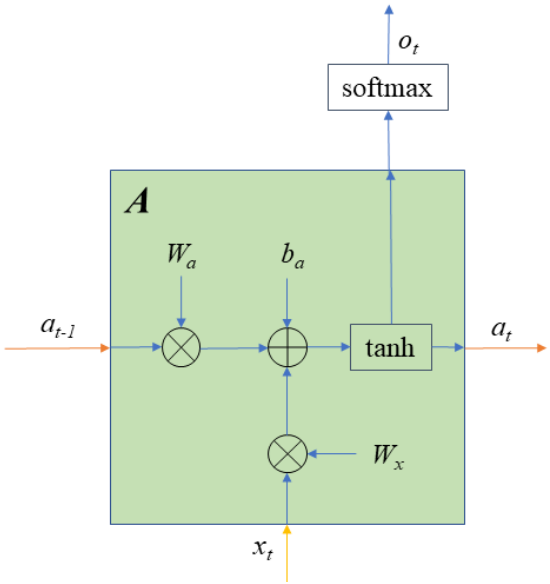

Figure 2.5: The RNN cell with its internal structure. It takes input x_t from the current time step and computes the current hidden state a_t using the previous hidden state with the *tanh* activation function. The current hidden state can then be used in the next time step to predict the current output o_t.

What can be problematic with RNN is that long-term memory might be lost, which means that information at the beginning of the time series might not be considered.

2.1.1.3 Long short-term memory neural network

A possible solution to handle data sequences even better is a long short-term memory (LSTM) neural network. As the analogy suggests, this network type incorporates feedback, meaning it remembers features from past time steps. The LSTM specifically remembers short- and long-term information, as can be interpreted from its name, for more details, see [28].

An LSTM is a type of RNN but with a more intricate structure. The main advantage of LSTMs is that they do not suffer from the vanishing gradient problem, the reason being the cell state that goes through the LSTM unit with just some minor linear interactions. The cell state carries information between time steps, and it is the forget gate that controls how much of the information will be passed on to the next time step. Figure 2.6 presents a simplification of an LSTM unit and shows the three gates that constitute the LSTM. Each gate is built up by one or two neural network layers to interact with each other. In short, the forget gate controls what information should be passed through, the input gate decides what new information to keep, and the output gate controls what to output.

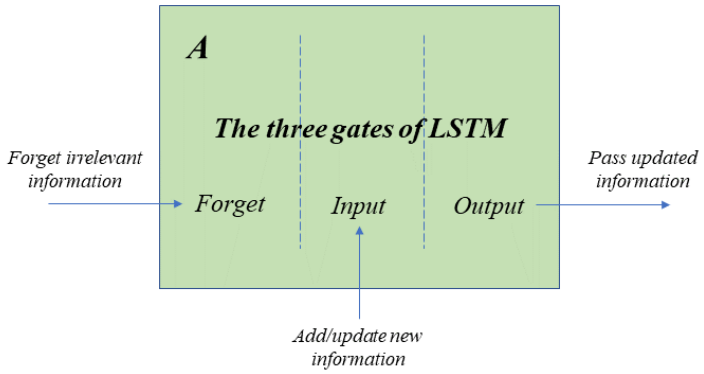

Figure 2.6: Simplified illustration of the LSTM unit. The forget gate has one neural network layer, the input gate has two layers, and the output has one layer.

Now each part of the LSTM will be described in detail, see Figure 2.7 together with Figure 2.8 for a detailed visual representation. The forget gate consists of one neural network layer and uses the hidden state from the last timestep h_{t-1} and the input x_t to decide how much of the cell state from the last timestep C_{t-1} to pass through. The neural network outputs a number between 0 and 1, forgetting or keeping everything. The output from the forget gate is described by

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.3}$$

where σ, W_f, and b_f are the activation function, the weight matrix, and the bias of the neural network. The output f_t is then multiplied with C_{t-1}, deciding how much to keep.

How much to keep from the last time step is decided when it is time to update the cell state with new information, this is done in the input gate. This process is governed by two neural network layers,

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.4}$$

and

$$\bar{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C). \tag{2.5}$$

The processes are almost identical but have different activation functions for the neural network layers, σ and $tanh$. The two layers' outputs are then multiplied and added to the cell state. The interactions with the cell state from the last time step are complete, and C_t is passed on to the next time step. The new cell state is described by

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \bar{C}_t. \tag{2.6}$$

With the cell state updated, the last thing is to decide what to output. This is controlled by the output gate using the hidden state from the last time step, the input, and the new cell state. A neural network layer with a σ activation function uses h_{t-1} and x_t and produces an output, o_t. The newly updated cell state is

passed through a *tanh* function to push the values between -1 and 1, which is then multiplied with the output from the network layer to get the output of the LSTM unit, h_t. This process can be described by

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.7}$$

and

$$h_t = o_t \cdot tanh(C_t). \tag{2.8}$$

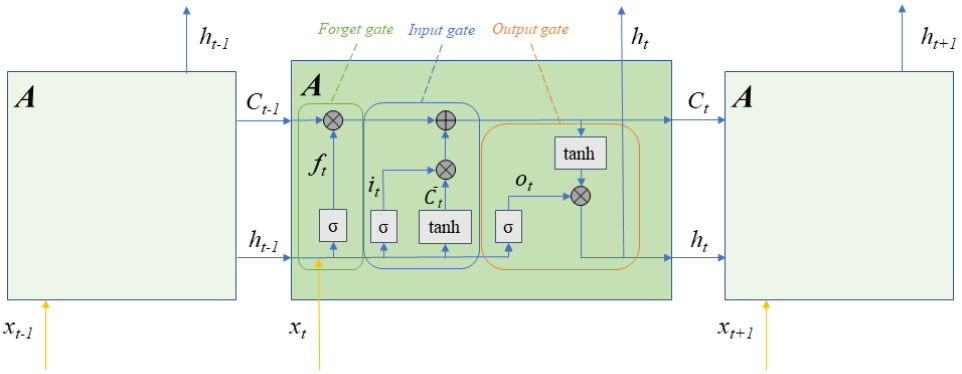

Figure 2.7: Three time steps of an LSTM cell with in-depth visualization of the current time step.

Figure 2.8: Notations used in Figure 2.7.

2.1.1.4 Encoder-Decoder

To handle input of varying lengths, one can use an encoder-decoder. One of the main application areas for RNNs is in natural language processing (NLP) problems. Machine translation, speech recognition, and image captioning are some tasks where RNNs have shown great performance [29]. In such applications, the input can be of various lengths, and in the machine translation case, the input is in one language and the output in another. If a classical neural network approach were used, the varying input length would be problematic since the network would treat the words individually when translating a sentence. To accept varying length inputs, to treat a sentence as a whole, and to find the meaning of the sentence, an encoder-decoder model can be utilized. Inputs are fed to the encoder, often some neural network and the input features are encoded into a fixed-length vector, independent of the input length. If the encoding process works as intended, the intermediate vector will represent the significant parts of the input features. The intermediate vector is

then fed into the decoder, some neural network structure, and the decoder produces an output.

There are many different ways to use an encoder-decoder, and they are not limited to NLP problems but have also successfully been utilized when working with time series data. The encoder is fed a time sequence of data, and the intermediate feature vector works as a summary of the input sequence. The summarized input is then fed to the decoder, which produces a single output. Figure 2.9 presents the structure of an encoder-decoder model used for time series data. The encoder-decoder model is a many-to-one structure that produces a single output.

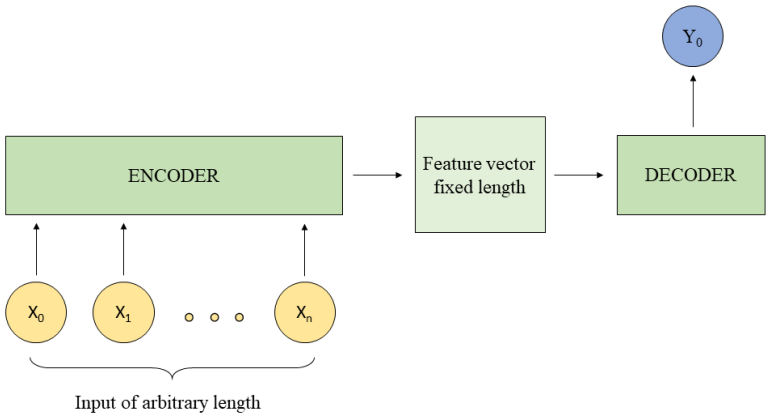

Figure 2.9: Encoder-decoder used for time series data with a many-to-one structure.

2.1.1.5 Hyperparameters

Hyperparameters control the neural network's learning process and are specified manually before training. These parameters define the architecture, behavior, and optimization of the network. Some hyperparameters are almost always used when training a neural network, and some are network-type specific or only used in some cases.

Batch size and learning rate are two hyperparameters used in most neural network training cases. The batch size determines the number of training samples to process before updating the models' trainable parameters. Large sizes will make the gradient steps larger compared to a small size. Widely used values are 32, 64, and 128 [30]. The learning rate affects how much to adjust the model parameters from the calculated gradients. A large value will change the model drastically and may have problems converging. In contrast, a small value will result in a model that changes its parameters slowly, resulting in a slow learning process. To train a neural network, an optimizer algorithm has to be chosen. The optimizer controls how to update the model parameters. Popular optimizers are stochastic gradient descent (SGD) and Adam, which is an extended version of SGD.

The number of layers and neurons in a neural network can also be considered hyperparameters. The number of layers affects the depth, and the number of neurons

affects the capacity and representational power of the network. For an RNN model, the number of stacked layers and each layer's hidden size must be specified before training. More layers and larger hidden sizes result in a deeper network that can capture complex patterns but increases the computational complexity and training time.

When using deep models, the risk of overfitting increases. Overfitting means the model performs well on the training data but fails to generalize to unseen data. This happens when the model starts to memorize the training data rather than learning the underlying structure and patterns in the data. To minimize the risk of overfitting, dropout can be utilized. This hyperparameter is used to deactivate the rate of the RNN units or neurons during training.

2.2 Classification

Classification is used to categorize data or objects based on the features of the data. The objects can, for example, be fruits, and the features to base the categorization on could be the color of the fruits. Classification is a supervised machine learning technique; each object has a class label. The goal is to use the chosen features to classify the objects into the correct class determined by the class label. When classifying objects, different approaches can be used. In recent years, the use of deep learning methods has increased, but more traditional methods are still used. The used traditional classification algorithms in this work will be described in the next three subsections.

2.2.1 Support Vector Machine using radial based function

Support vector machines (SVM) are supervised learning algorithms for analyzing data, usually for classification. This type of classification algorithm is often preferred since it produces significant accuracy with less computational power. This is done by finding a hyperplane in n-dimensions that accurately classifies the input data. The hyperplanes are chosen to maximize the margin between the two data classes. The maximum margin is sought after since it gives the confidence to classify future data points more accurately [31].

The hyperplanes are decision boundaries that classify the data points by which side they fall on of the hyperplane. The n-dimensions are decided by the number of input features. If the number of input features equals two, then the hyperplane becomes a line, see Figure 2.10. If there are three input features, the hyperplane becomes a plane. If the number of parameters exceeds three, then the hyperplane becomes difficult to comprehend. The support vectors are the data points that influence the margin of the hyperplanes, for example, to the right in Figure 2.10 it is the green circle and yellow square, which the blue transparent maximum margin cut through, that establish the margin. Adding or removing data points may therefore influence the maximum margin.

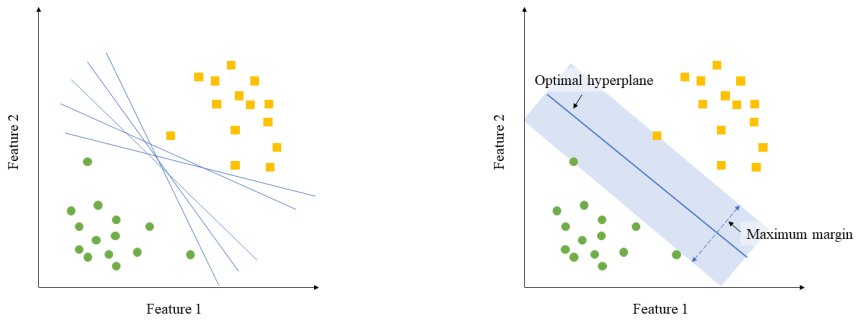

Figure 2.10: To the left, the blue lines represent possible hyperplanes. To the right, the optimal hyperplane is represented with the thicker blue line, also the maximum margin is shown.

When approaching a non-linear problem, it is possible to perform a kernel trick that enables linear learning algorithms to learn a non-linear decision boundary without explicit mapping. A common kernel trick when using a support vector machine is the radial-basis function (RBF) kernel. This function is a real-valued based function whose value depends on the distance between an input point and a fixed point [32]. This kernel is defined as

$$K(x, x^{'}) = e^{-\gamma||x-x^{'}||^2} \tag{2.9}$$

where x and x' are the two points and γ is a hyperparameter that defines the bandwidth of the kernel. The Euclidean distance gives the data points high or low similarity values depending on whether the points are close or not. This is why the RBF SVM model is more suitable since it allows implicit mapping of the input data, which enables an easier separation of classes using a hyperplane for higher dimensions, in other words, more complex and non-linear decision boundaries.

2.2.2 k-Nearest Neighbours classification

The k-nearest neighbors (k-NN) algorithm is a non-parametric supervised learning method used for classification [33]. The input for the algorithm is the k closest training examples, where k is a positive integer, and the output is a class belonging to an object. Its neighbors' class memberships decide the object's class membership via a plurality or majority vote. The dominant class within the neighbors wins, and the object is assigned to that class. See Figure 2.11 for a visual representation of how k-NN work.

Figure 2.11: Input and output to a k-NN algorithm. To the left, the blue box is to be classified with $k = 4$; to the right, the blue box has been assigned the heart-class based on its neighbors.

A Euclidean distance is often used to determine the closest neighbors, but other metrics, such as the Hamming distance, are also common, especially when working with discrete values. In this study, only continuous variables will be considered. Therefore, when using the k-NN algorithm, the Euclidean distance will be the metric to determine the nearest neighbors

$$d = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{2.10}$$

where n represents the dimension and the number of features used in the k-NN algorithm. Since this classification is based on the distance between features, how these are represented is important to remember. If the features come in a great range or are represented in diverse physical attributes, it is important to normalize the training data to improve accuracy.

2.2.3 Random Forest

Random forest combines multiple decision trees to create a more robust model. A decision tree is a machine learning algorithm that, as the name implies, is a tree-based model that repetitively divides the input data into subsets or sub-trees, and as a result, a prediction is given for the input. See Figure 2.12.

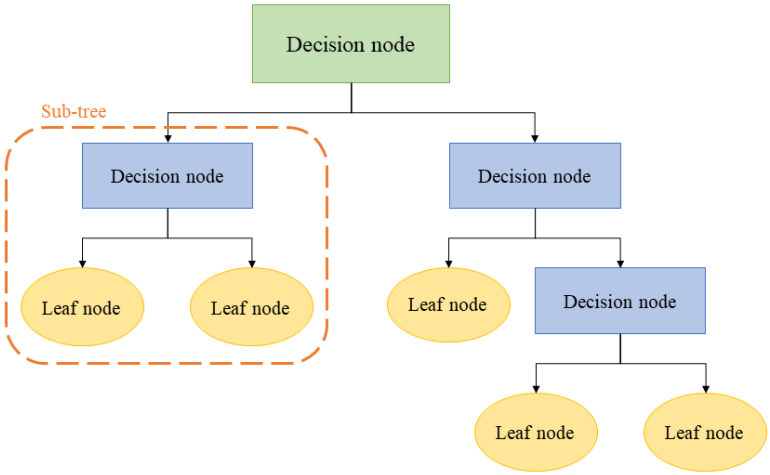

Figure 2.12: An arbitrary decision tree structure with two sub-trees.

The initial decision is made at what is called the root, then the tree branches out using branches and nodes for possible outcomes, respectively, features tests. In the end, there are leaf nodes which is the final prediction.

In random forest, each tree is trained on a subset of data and only considers a random subset of features during the training. See Figure 2.13. This can reduce overfitting, which is common when only using one decision tree. A random forest also helps to introduce diversity, something that may be absent in decision trees.

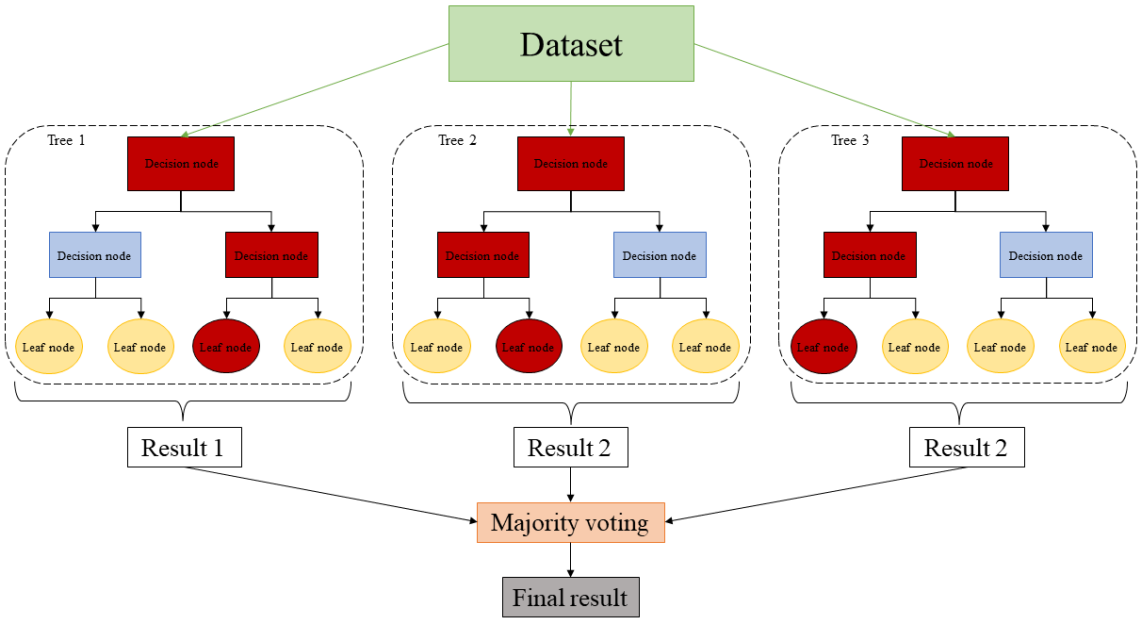

Figure 2.13: A arbitrary random forest for classification structure with three sub-trees.

When using the random forest algorithm, the decision trees are made by separating

the nodes based on the best-performing feature. After the training, the final prediction is decided by evaluating all predictions from the trees, and the class that receives the majority is selected.

2.3 Performance measures

To evaluate the performance of the developed classifiers, metrics are needed. The task in this work is both binary and multi-class classification, and hence some different metrics are chosen. Accuracy and F1-score are the used metrics and can both be calculated using the confusion matrix.

2.3.1 Confusion matrix

In the confusion matrix, one axis holds the true labels, and the other holds the predicted labels. Which axis holds which varies, in this study, the true label will be held on the y-axis and the predicted on the x-axis. Since the elements on both axes are in the same order, the diagonal shows the correctly classified samples. The confusion matrix can be used to identify where the model struggles by observing the columns or rows. If a significant number is outside the diagonal, the model is not performing well. In Figure 2.14, an underperforming and strongly performing confusion matrix of binary classification is illustrated.

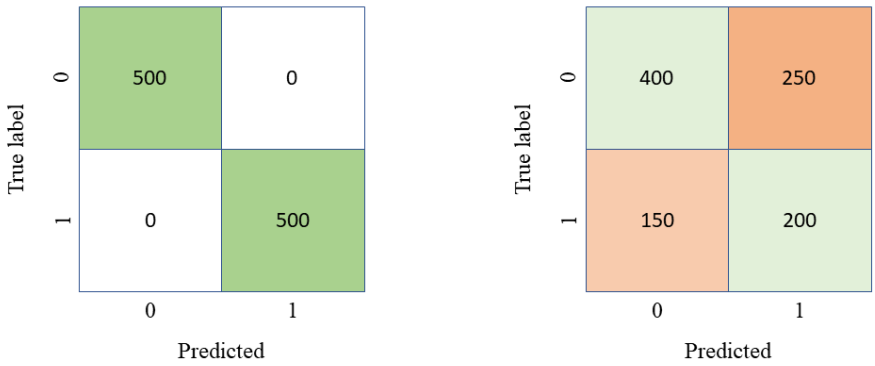

Figure 2.14: To the left, a strongly performing arbitrary model's confusion matrix is visualized. To the right, an underperforming arbitrary model's confusion matrix.

2.3.2 Accuracy

The accuracy is calculated directly from the confusion matrix, as

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \tag{2.11}$$

In the numerator, it is a sum of the true positives and the true negatives. The denominator consists of the sum of all entries in the confusion matrix. When considering one sample, accuracy is the probability that the model predicts the correct output. Accuracy declares the model's performance.

Accuracy is intuitive and works well if the dataset is balanced, meaning that the number of observations across all classes is almost equal. If the dataset is imbalanced, good accuracy can be misleading. If the model performs well for a class that is over-represented in the dataset, it will result in good accuracy even though all samples belonging to a minority class are classified incorrectly. Another drawback with accuracy is that it does not say where the model works worse and which classes it struggles with.

2.3.3 F1-score

F1-score is another measure of the model performance on a dataset. It is used in binary classification tasks and is a combination of the two measures of precision and recall:

$$Precision = \frac{TP}{TP + FP} \tag{2.12}$$

$$Recall = \frac{TP}{TP + FN}. \tag{2.13}$$

Precision is the fraction between true positives and the sum of true positives and false positives. It describes the model's ability to identify all positive instances correctly. Recall, often called sensitivity, is calculated with true positives in the numerator and the sum of true positives and false negatives in the denominator. Recall describes the model's ability to capture all positive instances.

The F1-score is then computed as

$$F1\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.14}$$

and is the harmonic mean of precision and recall. It is a balanced measure between the two and is robust against imbalanced datasets. For multi-class prediction, F1-score can still be used and, in this work, computed with an average for each class.

3

Data collecting and datasets

This chapter focuses on data. Firstly, the data-collecting controller area network (CAN bus) system is introduced and explained. The following section addresses the three datasets utilized in this study, two of them provided by Volvo Group, and the last dataset is a free dataset obtained from the internet. The next section describes normalization methods suitable for the available data. Lastly, motivational data is presented and analyzed.

3.1 Controller Area Network (CAN bus)

A controller area network (CAN bus) is a data bus system that allows devices to communicate. A bus is a system of connections between digital components. CAN is a serial and asynchronous message-based protocol which means that the information is sent as a series of bits, and the messages are only sent if needed. The system was developed by Robert Bosch and officially released at an automotive conference in 1986 [34]. Today, CAN is extensively used in the automotive industry to connect electronic control units (ECUs) and to enable communication between them.

In contrast to the local interconnect network (LIN) [35] system, another extensively used communication protocol, where one ECU is the master and controls all communications, the CAN is a non-hierarchical network. Therefore, several ECUs can send and receive messages simultaneously. If data is transmitted from multiple ECUs simultaneously, the message with the highest priority is sent while the other messages are queued.

When inspecting the messages, or frames as they also are called, in detail, they consist partly of identifiers and partly of data. The identifier holds information about the message's function and the message's prioritization, where a higher number means higher priority. The identifier in the message does not explicitly give the address of the receiving ECUs, but the identifier still decides what ECUs should process the message. These messages are divided into four types: data frames containing data from the sender, remote frames containing a request of a data frame, error frames transmitted from an ECU detecting an error, and overload frames, which are a request to delay data frames and remote frames.

3.2 Existing datasets

To be able to use machine learning, data is essential. When researching data-driven methods, sufficient data is needed. Therefore, datasets and data have been a significant part of this study. Three datasets have been accessible; Volvo Group provided two, and one was accessed from the Internet.

The data that are accessible at Volvo Group are historical in-house testing data. This data are collected when a software release is closing in. Research and development employees test the newly developed software in the vehicle and perform tests to ensure it runs smoothly before sending it off to the field testing vehicles. These test drives are logged via CAN bus, which means they are available to access after the test drive. This dataset will be referred to as the verification dataset in this study. The test drives are not executed by professional truck drivers. Further, the test driving is done using the software to the greatest extent since the software is in focus. This means that test driving might not be equal to the driving performed by professional truck drivers, therefore, not equal to how the truck is usually driven.

The second dataset provided by Volvo Group is from field test data. Professional truck drivers test existing and newly released software, also collected via CAN bus. The data is usually collected from drives driving the same route every day and performed by two or three different drivers. This dataset will be referred to as the field test dataset in this study.

The third dataset is provided and brought forward by researchers to be able to identify drivers for an anti-theft purpose. The dataset contains real driving data processed from an in-vehicle CAN bus with 51 features recorded every second. The data consist of ten drivers driving a passenger car on two round trips each, where the driving path is a total length of 23 km one way. This results in 92 km of driving data per driver, generating 920 km for all drivers [36]. This dataset will be referred to as the security dataset in this study.

3.2.1 Normalization for data-driven methods

The accessible data contain features of highly varying magnitudes, units, and ranges, which can slow down and deteriorate the training in machine learning algorithms [37]. To avoid this, it is possible to normalize the data. Normalization is a common scaling method that scales the data into a suitable range and removes undesirable characteristics from the data [38].

Normalization usually uses minimum and maximum values to proportion the data either between $[0, 1]$ or $[-1, 1]$. Two prevalent normalization methods are min-max scaling and mean normalization. The min-max scaling rescales the data into a range between $[0, 1]$ or $[-1, 1]$, dependant on the nature of the data, using

$$x^{'} = \frac{x - min(x)}{max(x) - min(x)}. \tag{3.1}$$

while the mean normalization method scales the data into the range of $[-1, 1]$ by using

$$x^{'} = \frac{x - \bar{x}}{max(x) - min(x)}. \tag{3.2}$$

Equations (3.1) and (3.2), are both efficient for preparing the data for machine learning algorithms, but when dealing with soft boundaries, as in this case, the min-max scaling is preferred, as argued in [37].

3.3 Motivational data

The existing dataset of historical in-house testing data provides an incentive for the project's aim. To ensure that driver behavior differs, as the researchers in [7] claim, three drivers driving the same route were extracted from the verification dataset for comparison. This showed that the behavior does differ when comparing drivers. Below are the differences between three drivers with different amounts of experience driving a truck on the same path, back and forth. To find driving varieties between drivers, actions such as brake pedal position, velocity, and lateral- and longitudinal acceleration are shown for each driver, visualized using GPS coordinates. The three drivers are all R&D-employees at Volvo Group, and Driver 1 and Driver 2 are not as experienced in driving a truck as Driver 3.

In Figures 3.1, 3.2 and 3.3, a single direction, namely the second direction of the round trip, is shown for all three drivers.

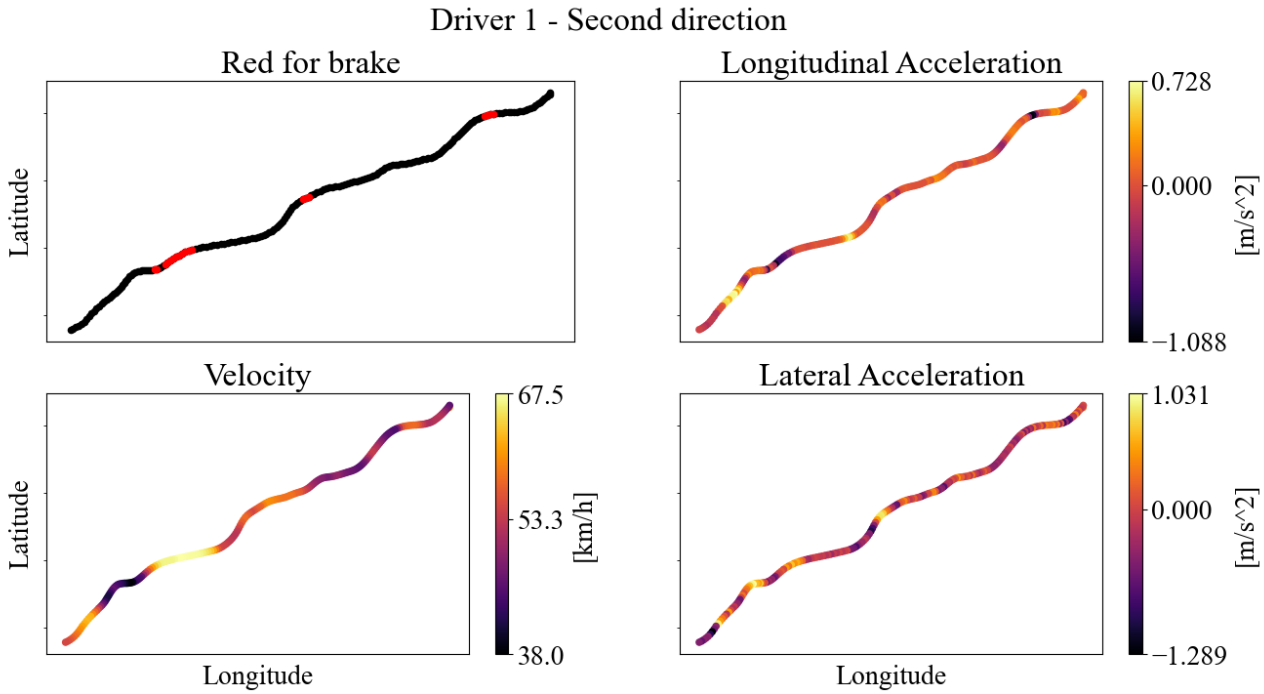

Figure 3.1: Four subplots for brake pedal position, velocity, and longitudinal- and lateral acceleration for Driver 1 using the longitudinal and latitudinal coordinates.

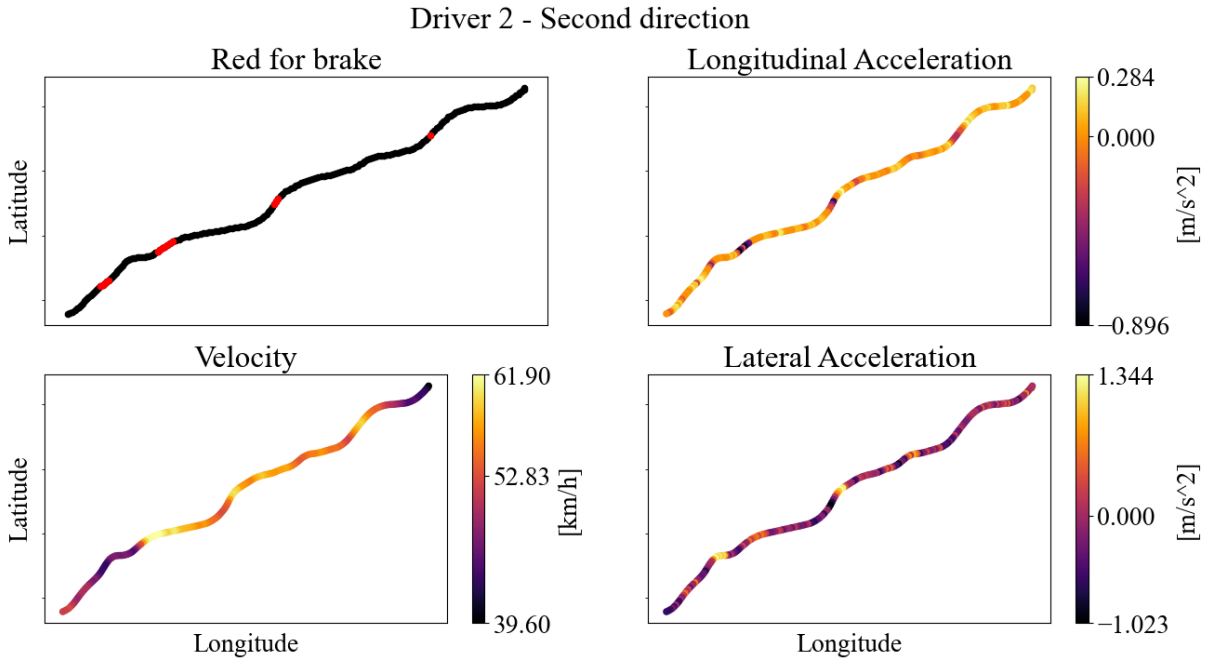

Figure 3.2: Four subplots for brake pedal position, velocity, and longitudinal- and lateral acceleration for Driver 2 using the longitudinal and latitudinal coordinates

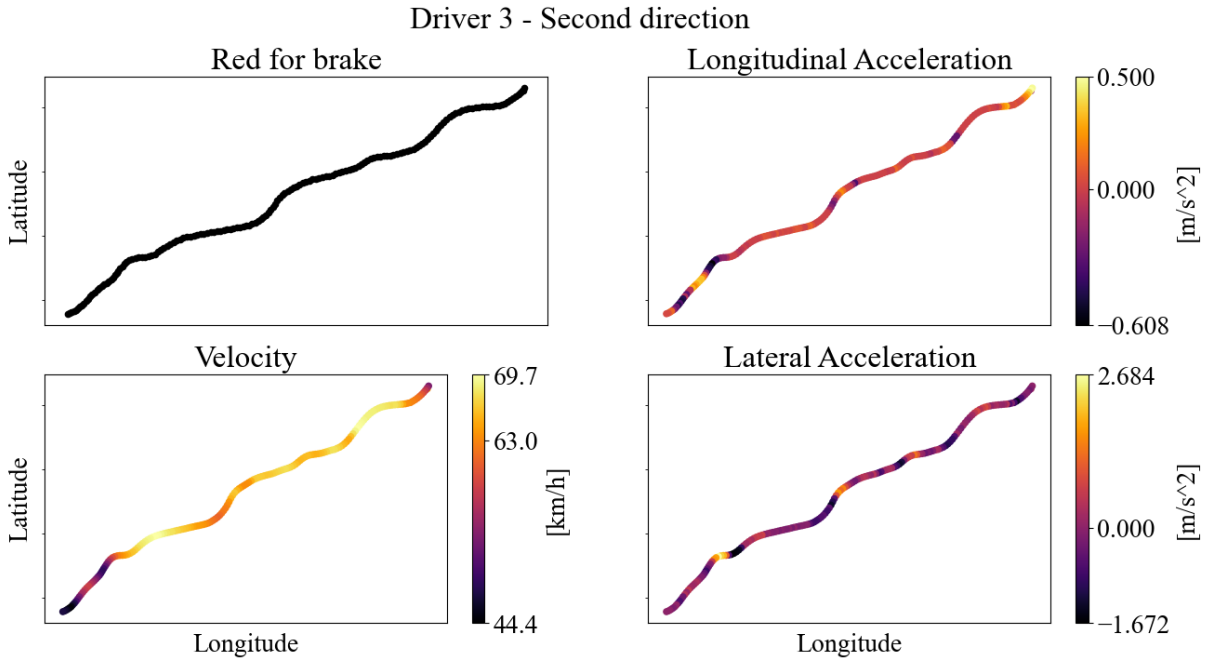

Figure 3.3: Four subplots for brake pedal position, velocity, and longitudinal- and lateral acceleration for Driver 3 using the longitudinal and latitudinal coordinates

As can be observed, the more experienced truck driver, Driver 3, does not hit the brake pedal once during their drive. The velocity is also more even throughout the drive for Driver 3. Driver 1 and 2 fluctuate more in velocity, which is both seen in the velocity plot and lateral acceleration. To dig deeper into the differences, two box plots of both directions with the longitudinal- and lateral accelerations can be seen in Figures 3.4 and 3.5.

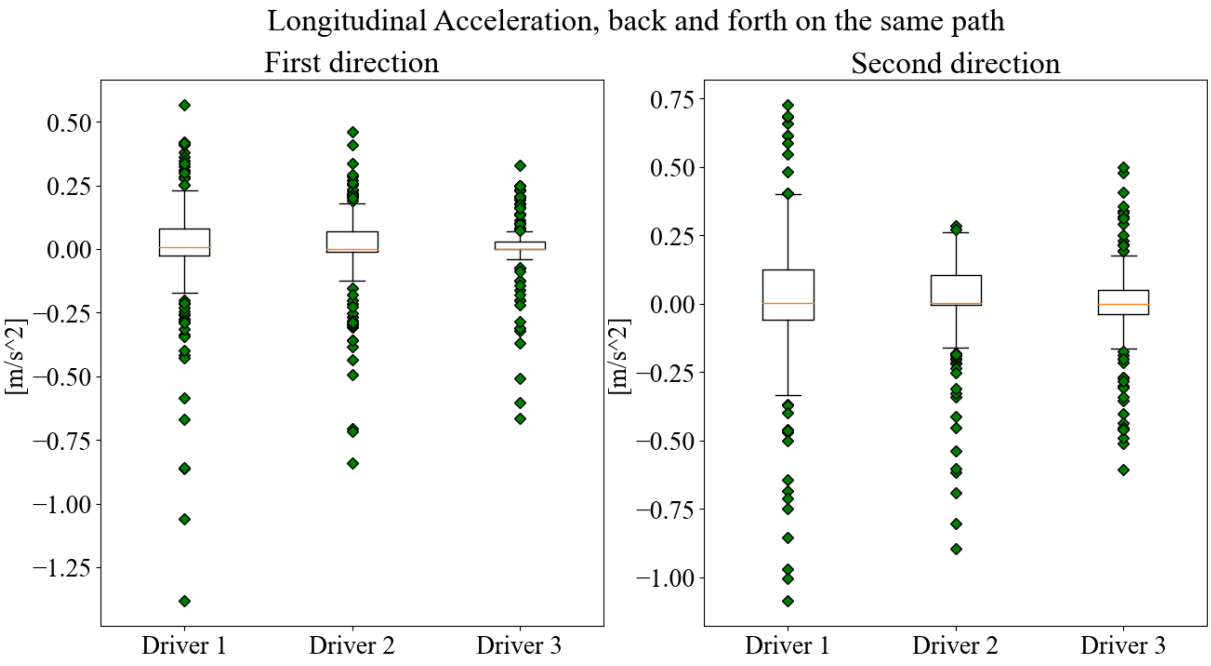

Figure 3.4: Boxplot of longitudinal acceleration for Driver 1, 2, and 3.

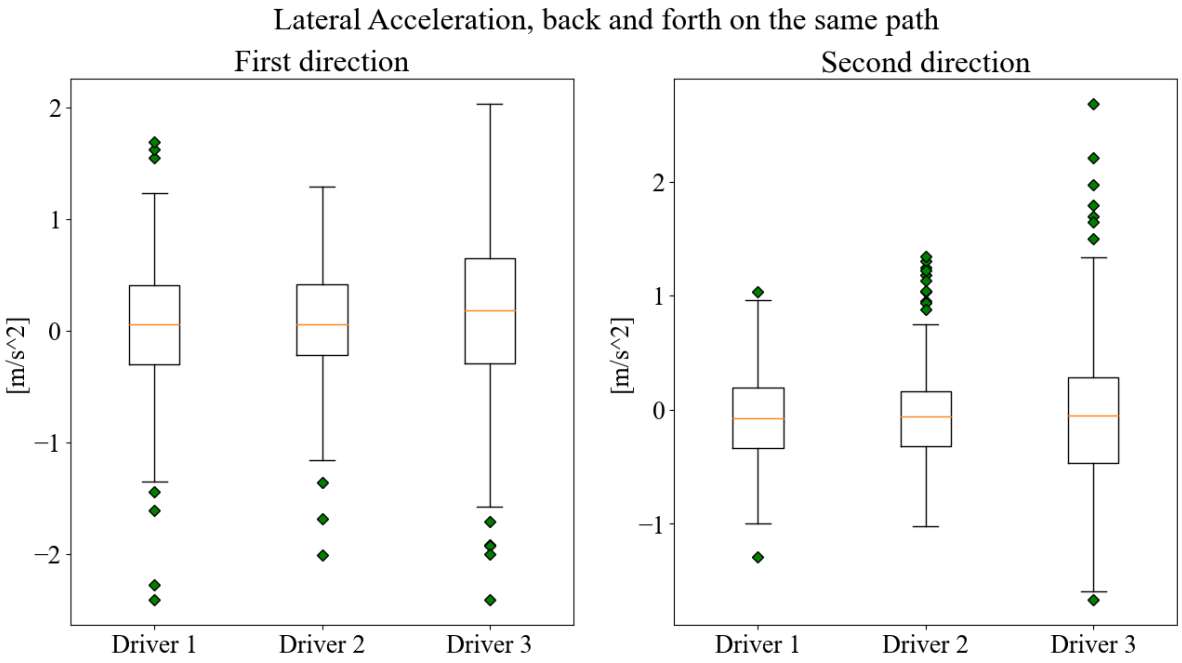

Figure 3.5: Boxplot of lateral acceleration for Driver 1, 2, and 3.

A box plot, or a box-and-whisker plot, graphically presents the statistical summary of the data. The box represents the middle 50% of the data, and the line crossing the box is the median. The line extending from the box are called whiskers. They represent the range of the data but omit outliers. The outliers are, in this case, shown as green diamonds. The discrepancy in driving between the drivers is shown in the box plots, especially in the longitudinal acceleration. The plots above have shown that the two less experienced drivers, Driver 1 and 2, share some trends in their behavior. This is not the case in the first box plot, Figure 3.4, especially when examining the second direction.

4

Method

In this chapter, the implementations of this study are described. First, the existing tools are stated, and the process of understanding the in-vehicle data collected from the CAN bus is discussed. The chapter continues with the preprocessing stage, including filtering and sliding window sequencing. Then the two main concepts implemented are described thoroughly, namely the velocity prediction models and the driver classification models. The details about the training of the models finish the method chapter.

4.1 Research environment

To develop different methods and models, various existing tools were employed. Python and its extension libraries were utilized for all coding implementations. The PyTorch framework was used to develop deep learning models and training algorithms. Other important libraries for preprocessing, classification, performance metrics, and visualization were scikit-learn, pandas, and Matplotlib.

For the initial interpretation of in-vehicle data and to get an impression of the data and what signals could be significant, the program VISION was used. As previously mentioned, the data used comes from CAN bus recordings. To conveniently use these recordings in Python, some in-house resources were used together with Python-can extensions to interpret the data from CAN to pandas DataFrames in Python.

4.2 Extracting the data

Before it was possible to start working with the data, the data had to be in an accessible format that was easy to read and store. Originally the data was in binary logging format (BLF) created by the software that handles output from the CAN bus [39]. The software makes it possible to replay the recordings and observe how the truck's software behaves during a drive. A recording from a CAN bus system can contain information ranging from windshield wiper motion to engine torque. Henceforth, one BLF-file is often called a data log, a coherent recording from a drive. The length of the data log can differ from only a few seconds up to hours. The field test and verification datasets were both in this format.

To make use of the data without using the software, the BLF-files must be read and organized. This was done using the Python-can library and a signal filter that controlled which signals that was read from the BLF-file. The data were not synchronous, as the BLF-files were recorded using the CAN bus. The reading tool ensured that the sampling frequency of each signal was synchronous. The output was a DataFrame, a feature of the pandas' library. The pandas' library is a data manipulation and analytical tool built for Python, and DataFrame is a two-dimensional data structure that can hold the data like a table [40]. The DataFrame structure can be used for several purposes, such as graphical visualization and data administration. To store the data, the DataFrame was saved in a CSV-file, a comma-separated values file that allows the data within the DataFrame to be saved in a table structured format [41]. Since data was continuously added during the study, adding new BLF-files to an existing CSV-file was also made possible by checking that the signals read into the existing file matched the signals in the signals filter for the BLF-file to be added.

4.3 Understanding the data

An essential part of this thesis was working with and understanding the vehicle data. This meant, what signals to use, what parts of the signals were interesting, and what parts needed to be filtered out. The data are the centerpiece when working with machine learning and other data-driven methods. For such a model to work successfully, whether it is about classification, forecasting, or another task, the data are the enabler. If the data are insufficient in some way, the likelihood of the model working as intended decreases significantly.

Choosing what signals to use in the models was a continuous process throughout the study. Understanding what signals could be important was a mix of literature study, discussions with R&D employees with extensive truck driving experience, and data visualization from driving logs. The discussion gave insight into what influences a driver's behavior in different scenarios. Literature and visualization of different signals were helpful tools to find other factors that may affect drivers but are not considered. With several approaches, the possibility of finding useful data increases.

The GPS coordinates from each driving log were used to visualize a drive and to find several drives with the same route. Different signals were plotted with their GPS coordinates to see the evolution of a signal along a route. In Figure 4.1, information about braking pedal usage, whether the cruise control is active or inactive, the velocity and the lateral acceleration during the drive is presented. Other signals were visualized in a similar way. With this methodology, patterns during a drive could be analyzed to identify similarities and differences between drivers on the same route.

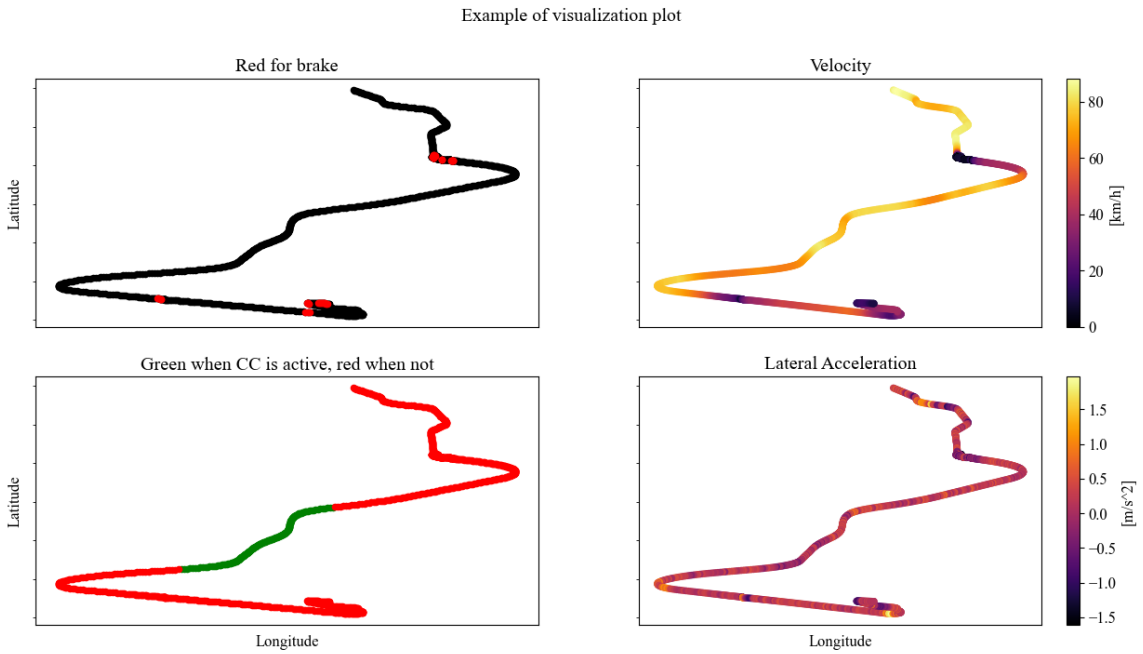

Figure 4.1: Four subplots for brake pedal position, cruise control, velocity, and lateral acceleration during a drive using GPS coordinates.

In addition to providing valuable information regarding what signals could be of interest, this type of visualization gave an understanding of what parts of the data may be filtered out in the preprocessing stage.

4.4 Data preprocessing

Another step in making sure that the data was of sufficiently good quality was preprocessing. The ambition with this stage was to clean the data and shape it into a suitable form. Since the preprocessing results were critical for the performance of the machine learning models, the preprocessing was iterative. If the results were worse than expected, stricter filtering may be needed or the other way around. Before the preprocessing, the field test dataset was labeled with drivers 0 and 1 to prepare it for preprocessing. The whole field test dataset could not be labeled, as extracting all the drivers was impossible. Now each part of the preprocessing will be addressed in detail.

4.4.1 Filtering

As previously mentioned, visualizing signals during several drives yielded valuable information about what data may be less representative. Initially, outliers and obvious measurement errors were removed. These were identified when the data was visualized, for example, when the velocity assumed values above reasonable. The next step was to extract information from the wanted scenario. For example, the driving behavior in an urban environment may differ from the driving behavior on a highway. When driving in a city, there are plenty of things to consider, including

buses, trams, stop lights, pedestrians, etc. On a highway, there are still traffic happenings to consider when driving, but probably fewer. In this study, most urban environments were filtered out using the legal speed limit.

The filtering was split into three categories: light, medium, and heavy. Light filtering meant outliers and unreasonable data were filtered out. Heavy filtering implied that straight and flat roads, urban driving, and data with vehicles in front of the truck were filtered out. The medium filtering lay somewhere in between, as the name indicates.

4.4.1.1 Snippets

Filtering data from a data log means that data points will be removed from the time series structured data. Since the data within a data log is a time series, removing a middle section of the data log and connecting the loose ends would be faulty. The data would then not be subsequent, and, therefore, the loose ends should not be connected. To solve this, subsequent data from drive logs were divided into snippets. Figure 4.2 illustrates the problem. The figure shows a time series illustrated as twelve boxes; each box is a data point. If the enclosed group X_5, X_6, X_7 is filtered out, two snippets with subsequent data points are created instead of connecting X_4 with X_8.

Figure 4.3 presents a driving scenario corresponding to the event in Figure 4.2. The driving scenario is a right turn in an intersection, and the intersection part was not data of interest. Therefore, indicator lights were used to remove such data. The indicator lights were turned on from time step 5, X_5, to time step 7, X_7, and that data was filtered out. The filtering process generated snippets of varying lengths where the data were more suited for machine learning. When a snippet was long enough, longer than 25 seconds, and with alternating road characteristics, such as inclination and curvature, it was chosen as a test snippet and became test data. Four test snippets were chosen as test data.

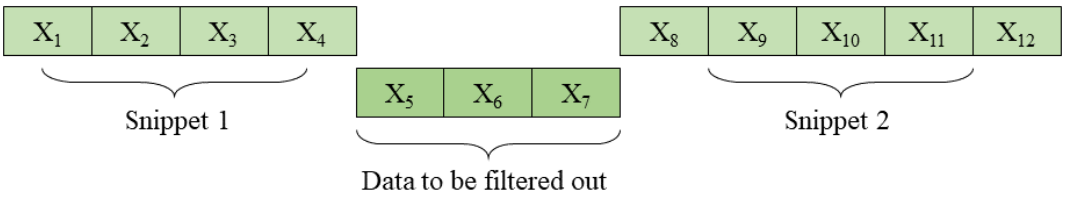

Figure 4.2: $X_1, ..., X_{12}$ is data points from a drive log. X_5, X_6, X_7 will be filtered out, and to prevent disjointed data from being joined together, the two parts are split into snippets of subsequent data.

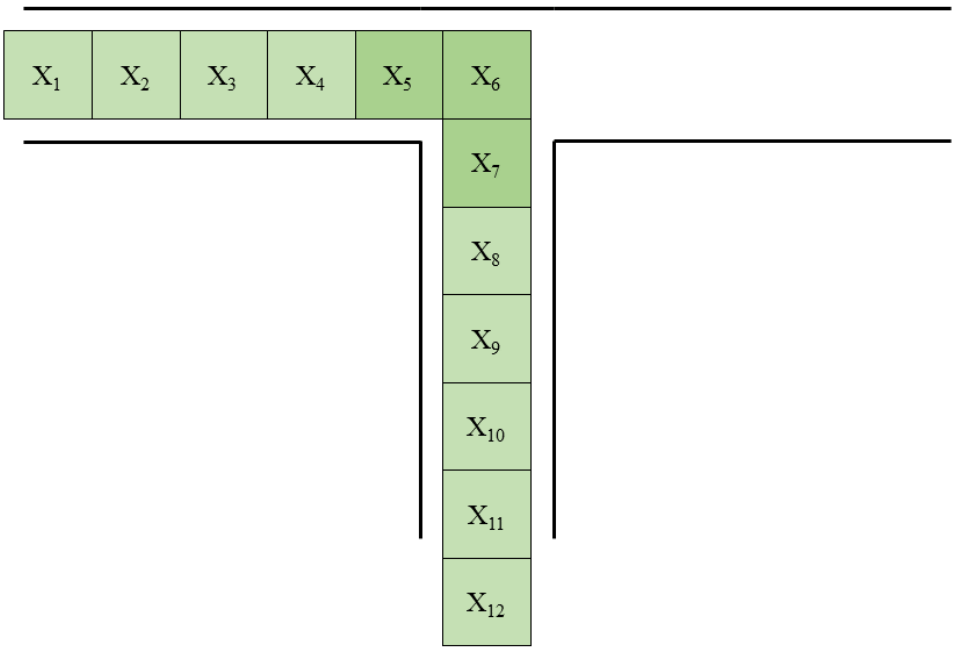

Figure 4.3: $X_1, ..., X_{12}$ is data points from a driving log during an intersection scenario. X_5, X_6, X_7 will be filtered out since the indicator lights were used, $X_1, .., X_7$ will be one snippet, and $X_8, ..., X_{12}$ will be another snippet.

4.4.2 Sliding window sequencing

The next stage in the preprocessing was to divide the snippets of varying lengths into input sequences of equal length and targets. To do this, a technique called sliding window was used, which is a method to divide a sequence of data points into shorter sequences. This was done by extracting a specified data length starting from the left of the time series and, for each new sequence, jumping one step to the right until reaching the last data point. Figure 4.4 illustrates an arbitrary example of the sliding window method. The subsequent data points X_n, where n is the number of time steps, are divided into a sequence of, for this example, length seven with the matching prediction target. As this study utilized supervised learning, the input data to the machine learning algorithm required a target. The target was then used in the learning or training to compare with the predicted output from the model. As mentioned before, this is how a model learns.

The target was chosen as the data point after the end of the sequence. The target could have been any point, depending on what was sought after. In the bottom of Figure 4.4, two sequences are extracted and stored together with their matching targets from this series of data points. The number of generated sequences was dependent on the specified sequence length and the amount of overlap. The overlap is at its maximum, which means just a single time step jump between sequences. With a smaller overlap, the second sequence would not start with X_2. A smaller

overlap will lead to fewer sequences and, therefore, less but more varying training data, which was sought after in some cases.

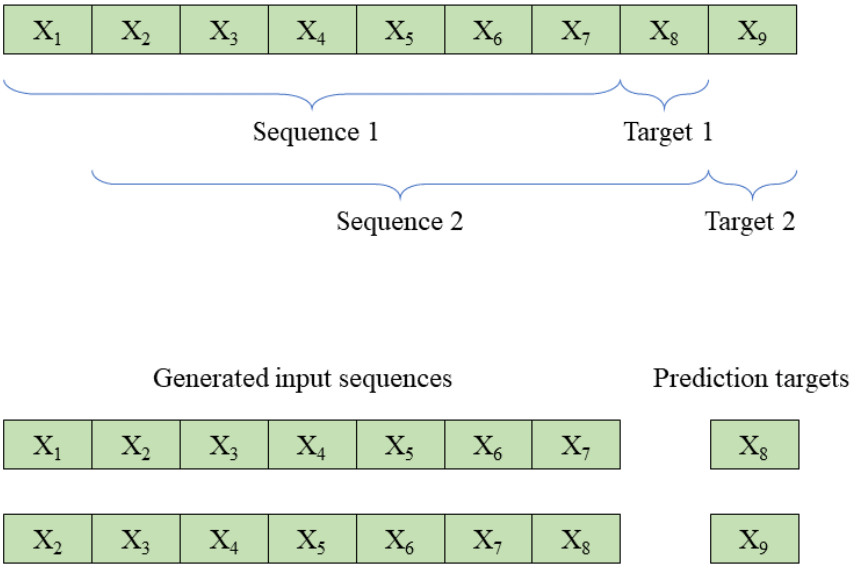

Figure 4.4: $X_1, ..., X_9$ are subsequent data points divided into sequences of length 7 with the sliding window technique. The upper sequence yields two input sequences and their matching prediction targets.

The preprocessing steps were carried out for all implemented models. However, the amount of filtering and sequencing implementation differed slightly. The model-specific preprocessing details will be highlighted together with the implementation descriptions.

4.4.2.1 Normalization

After the preprocessing, the data was in a suitable format for model implementation and training. To improve the training, the data was normalized using the min-max scaling method. This was done after the data had been put into sequences but before it was stacked.

4.5 Velocity prediction

The first category of models implemented in this study were LSTM models that predict the velocity given input data sequences. Two models were developed for this category, and both models use map data to obtain knowledge about the upcoming road. With that knowledge, the models try to predict a velocity, but the prediction procedure differs. One model predicts the future velocity based on the current state of the truck, including the velocity. The other model predicts the velocity

independent of the current velocity, using only map-based data. The main idea behind these two models was to learn the driving behavior of a driver to make personalized predictions. The personalized predictions could then be used in the vehicle to set a velocity in a particular driving scenario, following what that driver would choose.

4.5.1 Velocity-dependent model

The model for predicting velocity using the current velocity was a dual encoder single decoder model, where both encoders and the decoder were LSTMs. The encoders were used to summarize the information from a given sequence, and the summarized information was then passed to the decoder as the initial hidden state. The reason for having two encoders was that they were used to encode information with different numbers of features. The first encoder handled the historical data, data from a few time steps before the current time step. The second encoder handled the map data, providing information about the road ahead. These two types of data had different amounts of features, making it difficult to use a single encoder. Since both the historical data and the map data were sequences, they were called the look-back window and look-ahead window.

The summarized information from both encoders was concatenated and fed to the decoder as its initial hidden state. In addition to encoding the two windows, the decoder used input from the current state, time step (t), to predict the velocity in the next time step ($t + 1$). If the prediction horizon was longer than one time step, the prediction was fed back to the model and used as a part of the look-back window and the input for the next prediction. The entire process, from how data was divided into look back and look ahead windows to the generation of a predicted velocity, is visualized in Figure 4.5.

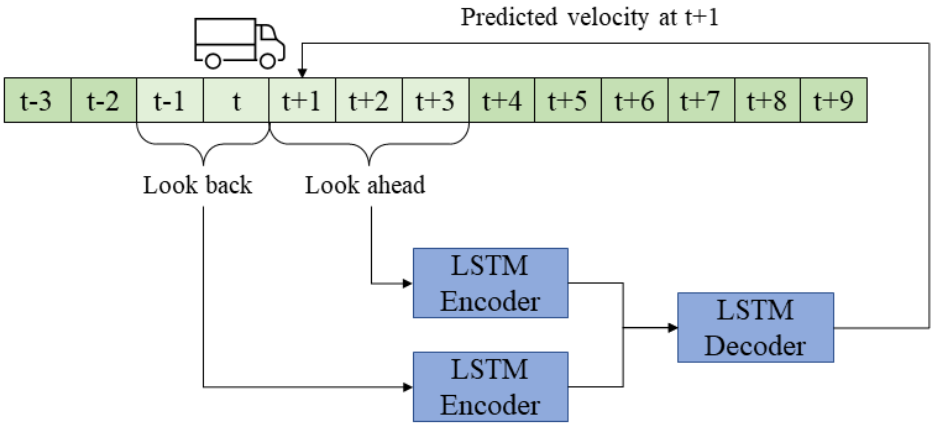

Figure 4.5: Model that predicts velocity based on the current state, previous states, and map data. The different sequences, look back and look ahead, are fed through encoders that summarize the information and output an intermediate vector representation from which the decoder generates an output.

Figure 4.6 presents the overall architecture of this model. Even though both encoders handle different inputs, their structure is the same. Both consist of an LSTM block with three stacked LSTM layers with a hidden size of 100, as illustrated in Figure 4.7. The decoder has the same LSTM block and a final linear layer with 100 neurons to produce a single output, a velocity.

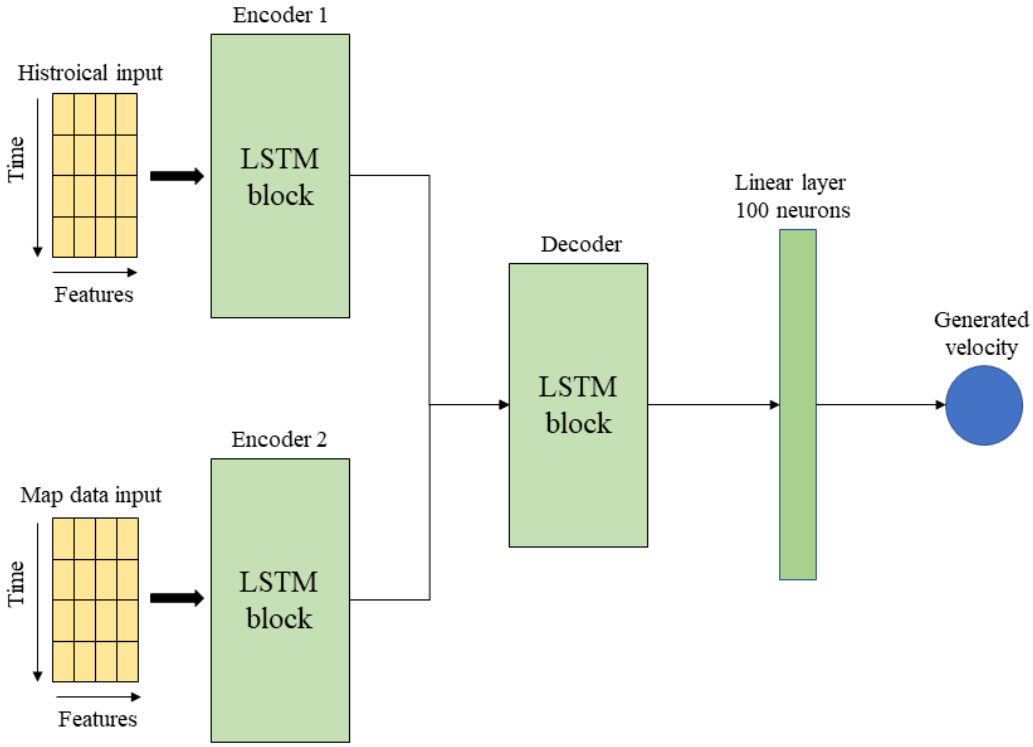

Figure 4.6: The architecture of the dual encoder single decoder model. The LSTM block is illustrated in Figure 4.7.

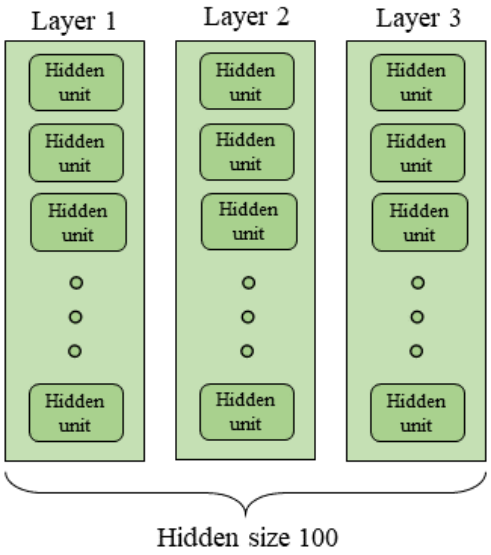

Figure 4.7: The inner structure of the LSTM blocks used in all LSTM models.

For this model, the creation of sequences with the sliding window technique was modified. The generated sequences were split up into two to match the look-back window and the look-ahead window. The target had to be changed as well. Instead of being the velocity at the time step after the generated sequence, it became the time step that came first in the look-ahead window, as seen in Figure 4.5.

4.5.2 Velocity-independent model

The second model to predict velocity had a similar structure but was independent of the current velocity. The first model predicts a velocity based on map data and data from previous time steps. This model predicts a velocity based on the map data alone. Restricting the model to map data without knowing the current state was a limitation. Still, it would make the model flexible and remove any long time horizon problems. Such a model could predict the whole route since the model makes predictions independent of the current state.

This model is illustrated in Figure 4.8 and uses a sequence of map data features describing a part of the road as input to an LSTM neural network which outputs a velocity. For the next prediction, the window is moved one timestep to the right, and a new velocity is made. Each prediction is independent of the previous. The model tries to find a relation between a sequence of map data features and a preferred velocity for a given driver.

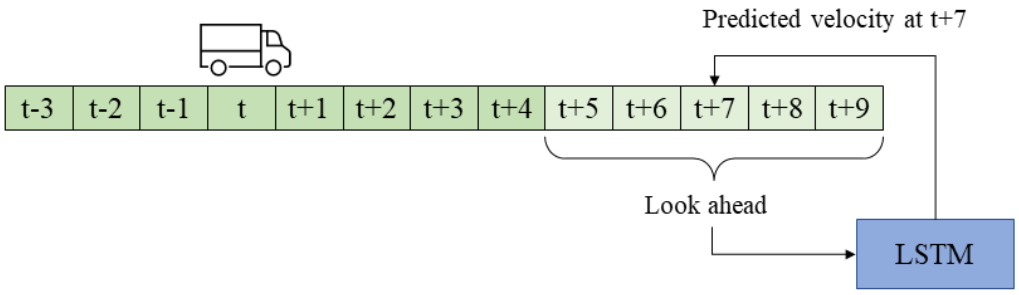

Figure 4.8: Model that predicts velocity based on map data only. One data sequence is fed directly to an LSTM neural network, and the velocity at the middle of the sequence is generated.

In Figure 4.9, the model's architecture is illustrated. It has the LSTM block, seen in Figure 4.7, with three stacked layers and 100 in hidden size, followed by a linear layer that generates a single output.

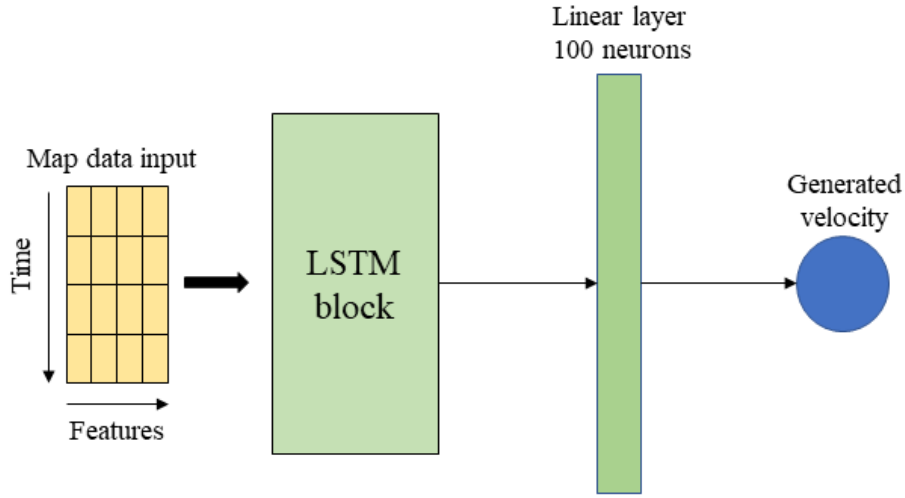

Figure 4.9: The architecture of the independent velocity model.

The only modification to the generation of sequences for this model was regarding the targets. To use the information before the point of prediction and for what comes afterward, the target was changed to the middle of the sequence as for the case in Figure 4.8.

4.6 Driver classification

Along with predicting the velocity for a window of chosen size, the study also aimed to discover whether it was possible to detect who is driving using classification algorithms. Therefore, the second model type developed was focused on driver classification from input data sequences. More traditional methods, such as k-NN and support vector machine algorithms, were used, as well as LSTM models. The input data differed a bit from the velocity prediction models. Information about the oncoming path was essential when predicting velocity, but it was not important when classifying driver behavior. Instead, all signals that can be used to capture driver characteristics were interesting and used as input features.

The main idea with these models was to find differences between drivers in various driving scenarios and to use that information to individually tweak the parameters in the I-See software, making it more personalized. To solve this driver classification task, an LSTM model was developed. As mentioned before, more traditional methods were used. This was to benchmark the LSTM model and see if a neural or deep neural network was needed for this type of classification.

4.6.1 Preprocessing for classification

For the classification part, the preprocessing was modified. The same driver logs were used as in the velocity prediction, but to be able to classify them, labels were needed. Each log was assigned a driver tag, which could subsequently serve as the accurate label for that particular data log.

When using traditional classifiers, the preprocessing was further modified. The sequence for each target was made using the sliding window technique, and the target for that window was simply the driver tag from the data log used to make the sequence. The input to the traditional classifiers was the mean value of the window from each feature. The input was no longer a data sequence but a vector with mean values.

Although the sliding window technique preprocessing is still utilized to modify the input for the LSTM model, the data window size typically increases. Driver behavior may seem indistinguishable when using shorter windows, like a few seconds of data. However, when using larger windows that span one or more minutes, variance in behavior becomes more evident. The target for each window was the same as for the traditional classifiers, thus the driver label of the data log from which the window originated.

4.6.2 Classification using traditional classifiers

Using the security and field test datasets, the k-NN, RBF SVM, and random forest algorithms were employed to classify drivers. Classifying a driver using a classification algorithm like the above-mentioned was fast and straightforward. Such classification could give an insight into if it was achievable and point to which features might affect the classification noticeably.

The Python-based scikit-learn library was used to implement the three traditional classifiers. This free machine learning software library made the chosen algorithms easy and accessible. Scikit-learn allowed tweaking the algorithms to try out which parameters gave the best results.

To test the robustness of the classifiers, the data that was fed into the training was altered. Several training sessions were performed with various filtered data, namely light, medium, and heavy filtering. The setup also varied between the training sessions, this meant altering the window size and overlap.

4.6.3 Classification using LSTM classifier

This LSTM classifier was passed a window of data with driver-defining features and gave a probability as output. Instead of a velocity, the prediction was a probability of what driver that data came from, a class label. Since LSTMs process sequential data and use context, complete data windows passed through this classifier.

The structure of this model was almost the same as the previous LSTM models

except for the model's output size. The output size was the same as the number of classes to predict. If it was a binary classification, the output size was two. Each output is a probability, where the class with the highest value gets assigned to that window. Figure 4.10 presents the architecture of the model. The illustration shows two outputs, referring to a binary classification task.

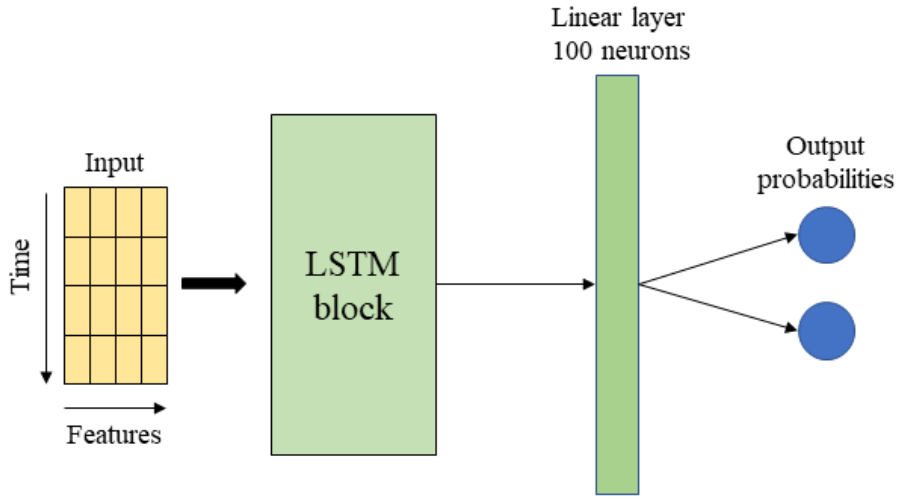

Figure 4.10: The architecture of the LSTM classifier. It has the same LSTM block as the previous models, shown in detail in Figure 4.7.

4.7 Training the machine learning algorithms

The weights within the model must be adjusted, i.e., undergo training, to attain a robust machine learning algorithm model that can deliver highly accurate output. To achieve a robust algorithm, the training utilizes backpropagation, as mentioned in Section 2.1.1. In this study, numerous trainings were executed for the different models. This section will explain the process of experimenting with various parameters that may impact the training.

4.7.1 LSTM - Velocity prediction and classification

The training data affects the model's performance. The data used in the training were the field test dataset. The input for the velocity prediction, dependent on the current state and the binary classification, was from drivers 0 and 1. For the velocity prediction independent of the current state, the data was just driver 1. It was also possible to adjust the hyperparameters to achieve satisfactory outputs.

The data was split into test, training, and validation data. The test data, called test snippets, was extracted during the preprocessing. The training and validation data were constructed by splitting the stacked sequences, excluding the test data, by 90% for training and 10% for validation. The split data was then put into a

PyTorch dataset, which is put into a Dataloader to keep it manageable and help to simplify getting the data to the machine learning algorithm [42].

Varying hyperparameters such as batch size, hidden layers, and learning rate gave varying results. In this study, this was an iterative process that included trying out diverse setups of parameters. Along with altering the hyperparameters, the amount of filtering was also varied. This meant that the training process depended on whether the data was heavily filtered. Heavily filtered data meant that driving scenarios that were concluded not affect personal driving behavior, such as flat and non-winding highways or dense traffic in urban driving, were filtered out. Lightly filtered data included almost all data except for outliers and data that was simply uninteresting, such as the vehicle standing still.

4.7.2 Traditional classifiers

The training of the traditional classifiers underwent the same iterative process as the LSTM classifier. The process included altering the different models' hyperparameters and training the models using different filtered data. In this training, the input data was the field test and security datasets. The training and validation data split varied from 90% training and 10% validation to 50% training and 50% validation.

5

Results and discussion

The results from the two main concepts developed are presented in this chapter. The first concept utilizes machine learning to learn a driver's behavioral pattern to then be able to predict a velocity given a certain road scenario. The second concept presented is also based on machine learning to classify drivers based on their driver patterns. At the end of the chapter, both of the concepts and their belonging results are discussed along with an overall data discussion, as data have had a major impact on this study.

5.1 Velocity prediction

This section presents the results for the two models that aim to predict a velocity from a sequence of input data. Along with the results of the predictions, used input features, window sizes, model architectures, and other important settings will also be described.

5.1.1 Velocity-dependent model

The velocity-dependent model predicted the velocity a few seconds ahead, and to evaluate its performance, it was tested on a snippet from a manually chosen driving log new to the model. The test snippet was a road segment from the verification dataset often used when evaluating new software releases and represented a suitable driving scenario. Four different prediction horizons, one second, two seconds, five seconds, and ten seconds, were used, and the result for each is presented in Figure 5.1 - Figure 5.4. The prediction horizons were shorter than the length of the test snippet, and therefore, several predictions were made. The start of each line indicates the beginning of a new prediction. The predictions correspond to the blue lines and the target velocity to the orange.

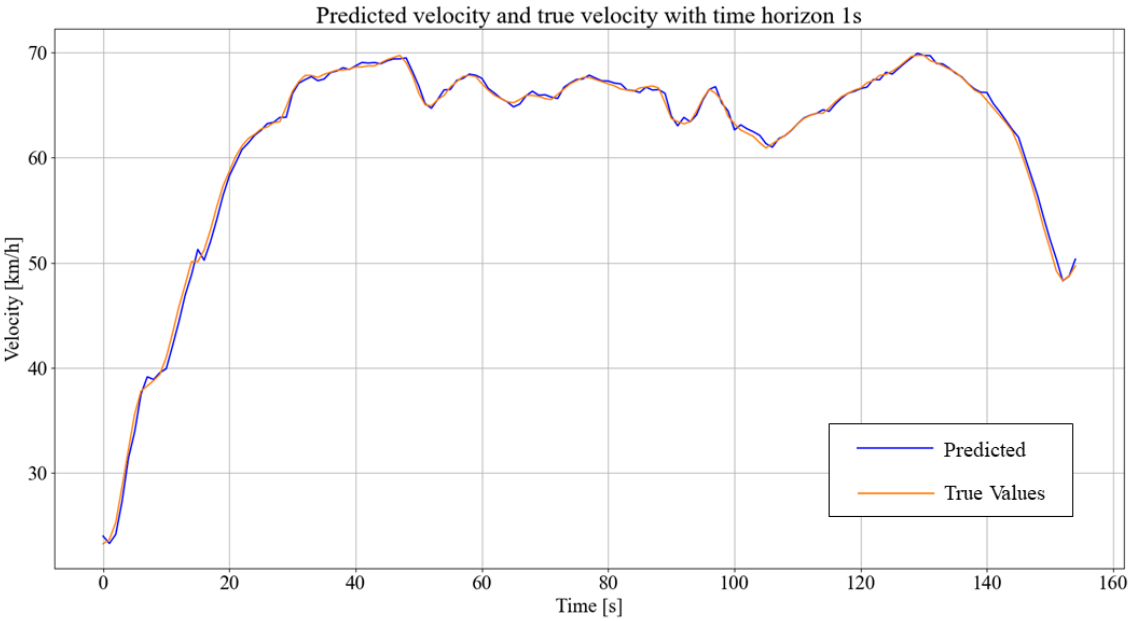

Figure 5.1: Generated velocities, in blue, and the target velocity, in orange, over a test snippet with a horizon of 1 second.

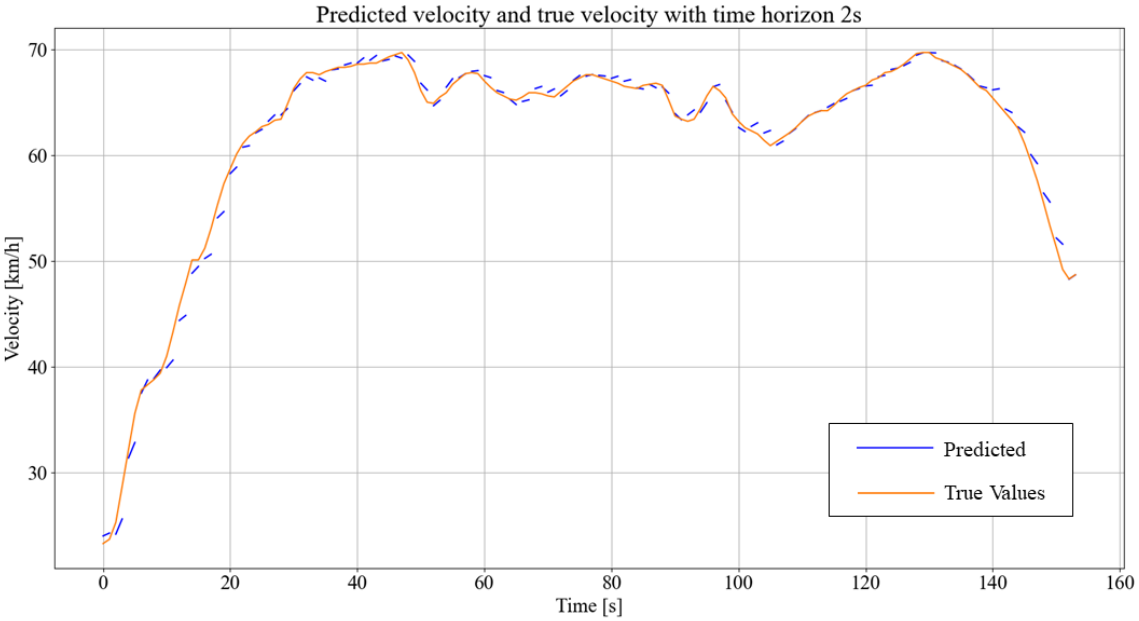

Figure 5.2: Generated velocities, in blue, and the target velocity, in orange, over a test snippet with a horizon of 2 seconds.

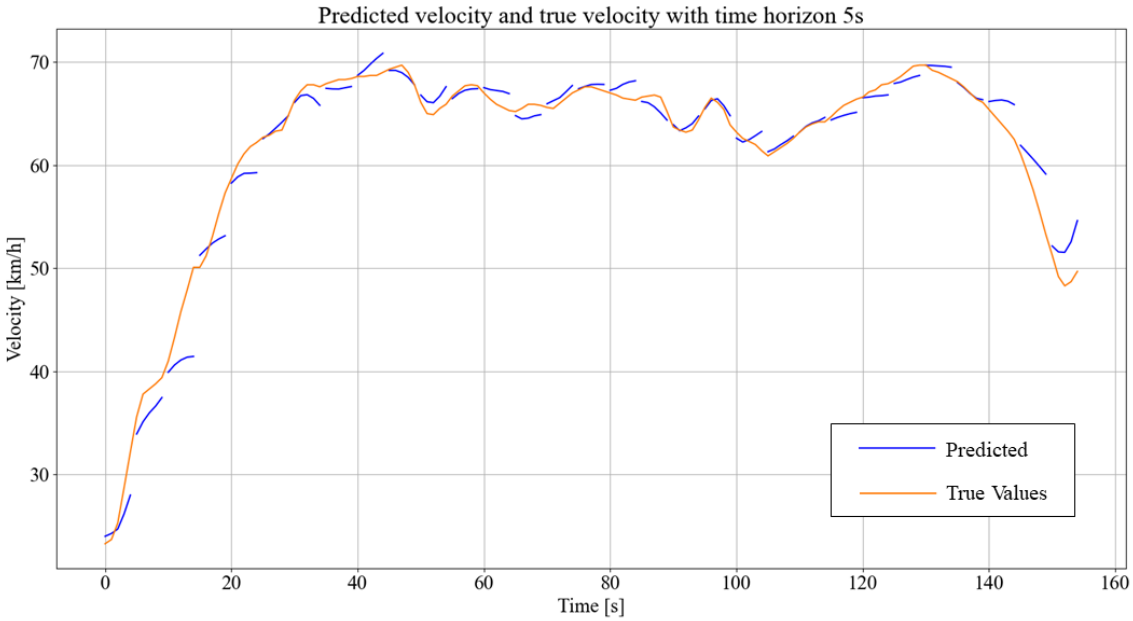

Figure 5.3: Generated velocities, in blue, and the target velocity, in orange, over a test snippet with a horizon of 5 seconds.

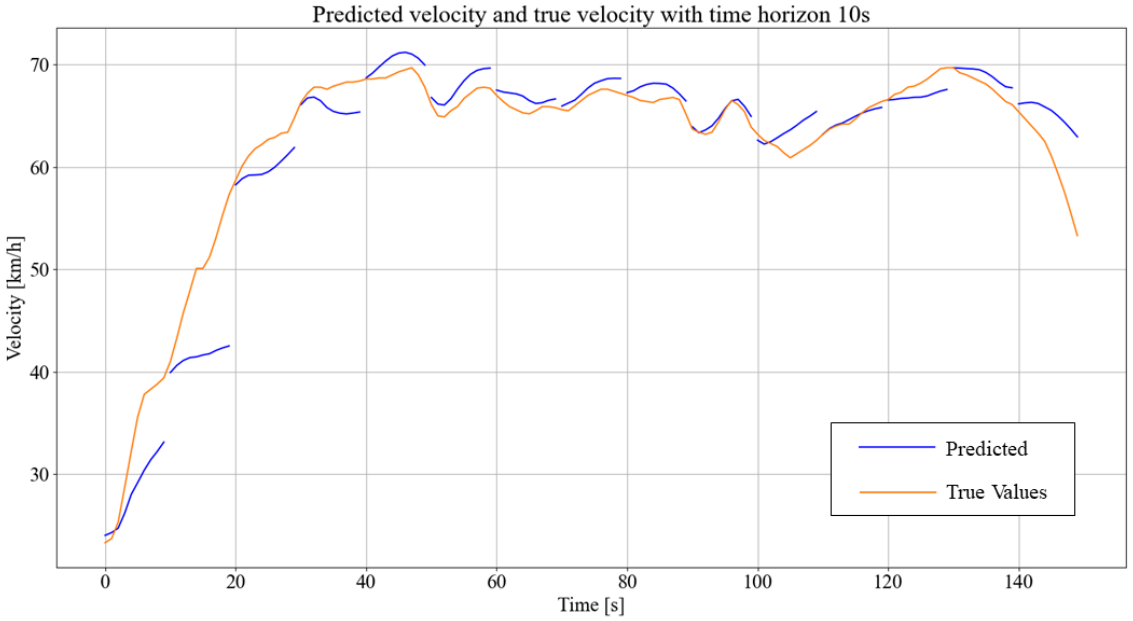

Figure 5.4: Generated velocities, in blue, and the target velocity, in orange, over a test snippet with a horizon of 10 seconds.

In Figure 5.1, the model predicts the velocity one timestep ahead. For this horizon length, the prediction and the target velocity were comparable. A long prediction

horizon was more challenging for the model as the predictions at the end of the horizon were generally further away from the true velocity than the ones in the beginning. This can be seen in Figure 5.4 where the horizon length is ten seconds. It is worth mentioning that with the long horizon, the model reasonably accurately predicted some segments of the route between 40 seconds and 100 seconds, as the target velocity and predicted velocity are in relative proximity. This indicated that the model could learn some of the driving behavior. Figures A.1 - A.4 in Appendix A.1 further demonstrate the learning progress of the model. The model produced an arbitrary output before training, seen in Figure A.1. As the number of epochs increased, the difference between the true velocities and the predictions decreased.

With the architecture presented in Section 4.5.1 and a look-back window set to two and the look-ahead window set to ten, making the total amount of trainable parameters ended up at 1 215 801. The look-ahead features consisted of map-based data, and the look-back features consisted of velocity combined with the map-based data.

5.1.2 Velocity-independent model

The velocity-independent model was evaluated on a similar test snippet as the previous model to ease the comparison between the two, even though this test snippet was extracted manually from a log driven by driver 1. Since this model made predictions without any information about the vehicle's current state, the whole test snippet could be immediately predicted. Figure 5.5 presents the predictions and the true target velocity, and the generated velocities do not follow the target velocity as desired. The differences are clear, and the model failed to capture the chosen velocity.

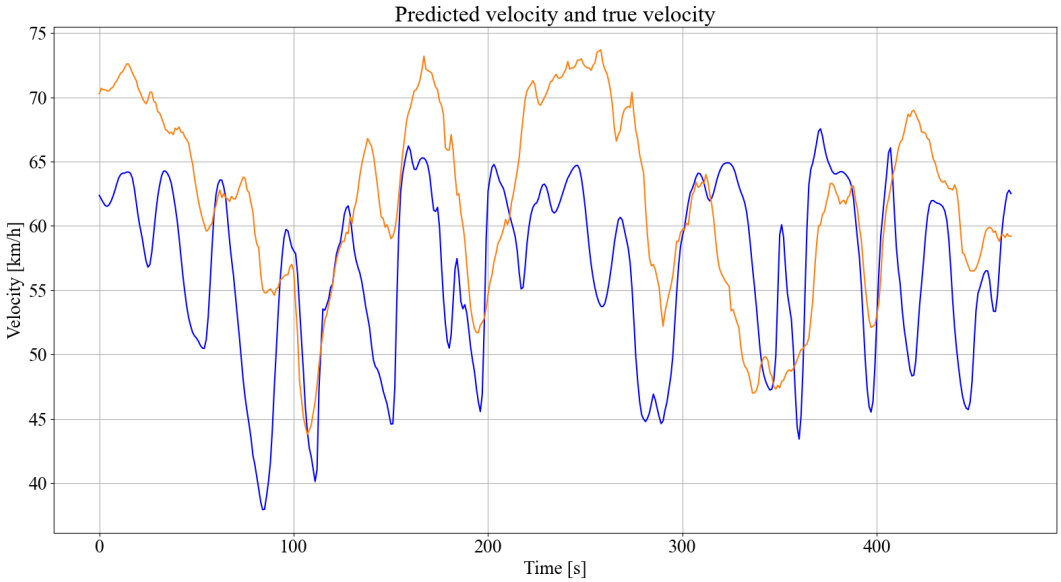

Figure 5.5: Prediction of the independent velocity model. The predicted velocity is in blue, and the target velocity is in orange.

The results for the test snippet were unsatisfactory and probably due to the complexity of the problem and the data, which will be discussed later. However, it does not mean that the model was irrelevant. The model was repeatedly tested on several validation snippets during training to evaluate the learning progress. Such an example can be seen in Figures A.5 - A.9 in Appendix A.2 and in some of these, the model has learned the overall structure of the behavioral pattern, i.e., chosen velocity in a certain scenario. That indicated that a further developed model could potentially learn the driving behavior and generate accurate velocities given only map-based data as input.

The resulting architecture of the model is presented in Section 4.5.2. For the input window, 20 sequential map-based data points were used. In total, the total number of parameters ended up at 817 951.

5.2 Driver classification

In this section, the results from the classification algorithms will be presented. Initially, the results obtained from the traditional classification method will be presented, followed by the outcomes of the LSTM classification. Overall, the classification gave satisfactory results for binary classification with accuracies ranging from 99.3 % to 99.6 %.

5.2.1 Traditional classifications

The results differed based on the input data for the three classifiers used. The training data was from the field test and the security dataset. The classification was either driver one or two, therefore binary for the field test dataset and multi-class classification for the other since it contains ten drivers.

The results have been put into a table to compare the classifiers for the field test dataset. The same driver behavior signals were used for all of the training, namely brake pedal, velocity, and lateral- and longitudinal acceleration, see Table 5.1. The three filtering types are all represented in Table 5.1 along with the setup specification of window size and overlap.

As the window size grows larger and the dataset shrinks, the number of testing data sequences decreases. This is also, intuitively, the case when the filtering is more strict, as in heavy filtering. As can be observed, the traditional classifiers do not deliver accurate classification when presented with a large dataset and a small window size. They perform better when presented with a heavily filtered dataset and a larger window span, especially when observing the k-NN classifier. The best-performing classifier setup is highlighted using bold. The setup with a window size equal to 50 seconds with heavy filtering and split at 10% gave the highest result, including all the classifiers. The medium-filtered data with a window size equal to 200 gave the best overall result of all classifiers at an accuracy of 98.7%. Worth mentioning is that the k-NN classifier still performed well with heavy filtering and

Table 5.1: The result from the traditional classification models using field test data. The setup was as follows: LF is light filtering, MF is medium filtering, and HF is heavy filtering. WS is window size in seconds, OL is the overlap between windows in seconds, and spl is the training and test data split ratio. The − indicates the computational complexity prevented the model from delivering a classification result.

Field test data, four signals	***Classifiers***		
Set up	**k-NN**	**SVM RBF**	**RF**
LF, WS = 10, OL = 1, spl = 0.1	56.8%	–%	–%
MF, WS = 10, OL = 1, spl = 0.1	57.0%	52.1%	–%
HF, WS = 10, OL = 1, spl = 0.1	60.8%	50.5%	56.6%
LF, WS = 20, OL = 5, spl = 0.5	55.9%	55.2%	57.2%
MF, WS = 20, OL = 5, spl = 0.5	54.9%	50.9%	56.6%
HF, WS = 20, OL = 5, spl = 0.5	58.2%	49.2%	55.7%
LF, WS = 30, OL = 1, spl = 0.1	64.2%	54.7%	–%
MF, WS = 30, OL = 1, spl = 0.1	71.3%	53.6%	59.9%
HF, WS = 30, OL = 1, spl = 0.1	85%	56.2%	62.1%
LF, WS = 40, OL = 1, spl = 0.1	67.4%	–%	–%
MF, WS = 40, OL = 1, spl = 0.1	78.8%	53.7%	59.5%
HF, WS = 40, OL = 1, spl = 0.1	**91%**	68.5%	71.6%
HF, WS = 50, OL = 1, spl = 0.1	**96%**	**68.5%**	**74.7%**
HF, WS = 50, OL = 1, spl = 0.4	93.6%	61.1%	72%
MF, WS = 8, OL = 1, spl = 0.5	89%	65.1%	69.7%
MF, WS = 200, OL = 1, spl = 0.4	**98.7%**	65.1%	69.7%

a window size of 40 seconds.

The traditional classifiers were used as benchmarks to indicate whether a driver classification was possible or not. Therefore, several setups were investigated. This was not done for the LSTM classifiers, which will become evident in the next section.

When comparing the three traditional classifiers using the security dataset, the three levels of filtering are not applied. A comparison of the three classification methods for multi-class classification can be seen in Table 5.2.

Initially, the same signals were used except for the brake pedal. This signal was changed for the accelerator pedal. To achieve the same result as in [36], the same 14 signals as the researchers used in that study were used.

5.2.2 LSTM classifications

The LSTM classifier was evaluated to compare the traditional classifiers' performance. This was done using the field test data with light, medium, and heavy filtering, and the signals were brake pedal, velocity, and lateral- and longitudinal acceleration. The results have been put into a table, see Table 5.3. The setup for different types of training is presented in the table. As can be observed, a reduced

Table 5.2: The resulting accuracies from the traditional classification models using the security dataset. The setup was as follows: WS is window size in seconds, OL is the overlap between windows in seconds, spl is the training and test data split ratio, and s is the number of signals used.

Security dataset	***Classifiers***		
Set up	**k-NN**	**SVM RBF**	**RF**
WS = 10, OL = 1, spl = 0.5, s = 4	33.2%	15%	21.5%
WS = 30, OL = 1, spl = 0.5, s = 4	56.2%	16.1%	24%
WS = 40, OL = 1, spl = 0.5, s = 4	63%	16.1%	24.7%
WS = 50, OL = 1, spl = 0.5, s = 4	73.3%	19.1%	26%
WS = 70, OL = 1, spl = 0.5, s = 4	77.2%	19.5%	26.5%
WS = 80, OL = 1, spl = 0.5, s = 4	80.3%	45.7%	26.8%
WS = 40, OL = 1, spl = 0.5, s = 14	94.9%	45.7%	74.4%
WS = 80, OL = 1, spl = 0.5, s = 14	**99.1**%	46.5%	21.5%

Table 5.3: The resulting accuracies from the LSTM classification models using the field test dataset with four signals. The setup was as follows: WS is window size in seconds, OL is the overlap between windows in seconds, and spl is the training and test data split ratio.

Set up, *field test data*	**LSTM Classifier**
HF, WS = 20, OL = 1, spl = 0.5	91.7%
HF, WS = 40, OL = 1, spl = 0.5	98.2%
MF, WS = 80, OL = 2, spl = 0.5	92.9%

amount of setup was tried for this model since window sizes under 40 seconds gave inadequate accuracies.

Only lateral- and longitudinal acceleration was used to determine the potential for classifying drivers using fewer signals. The results are in Table 5.4. Figure 5.6 presents a corresponding confusion matrix when evaluating the best-performing model on 5000 samples. From the confusion matrix, it can be seen that the test data are well balanced between the two drivers and that the model performs equally well at both.

Table 5.4: The resulting accuracies from the LSTM classification models using the field test dataset with lateral- and longitudinal acceleration. The setup was as follows: WS is window size in seconds, OL is the overlap between windows in seconds, and spl is the training and test data split ratio.

Set up, *field test data*	**LSTM Classifier**
HF, WS = 40, OL = 1, spl = 0.5	97.8%
HF, WS = 60, OL = 1, spl = 0.5	99.3%
MF, WS = 80, OL = 2, spl = 0.5	99.5%

What can be observed in Table 5.3 and Table 5.4 is that the model using only two

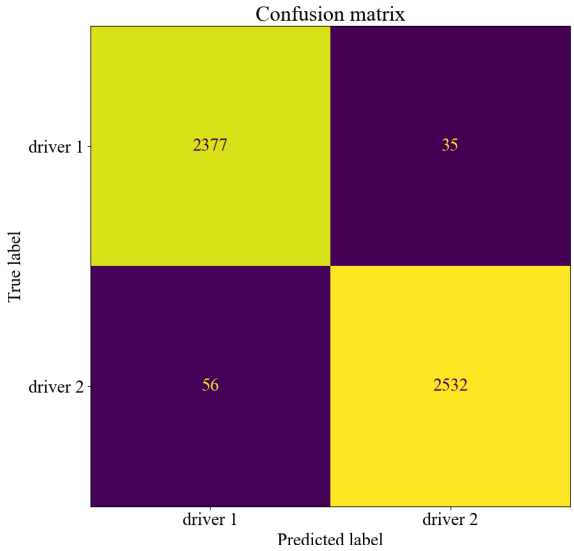

Figure 5.6: Confusion matrix for the binary classification of field test data.

signals gave the best overall results when compared to the model using four signals.

To compare the LSTM classifier to the traditional classifiers, the security dataset was used as well in the training. Four signals were used, namely accelerator pedal, velocity, and lateral- and longitudinal acceleration. The results can be seen in Table 5.5. The belonging confusion matrix in Figure 5.7 shows a bit of an imbalance between the different classes. The model seems to work equally well for all classes.

Table 5.5: The resulting accuracies from the LSTM classification models using the security dataset. The setup was as follows: WS is window size in seconds, OL is the overlap between windows in seconds, and spl is the training and test data split ratio.

Set up, *security dataset*	**LSTM Classifier**
HF, WS = 80, OL = 1, spl = 0.5	98.7%

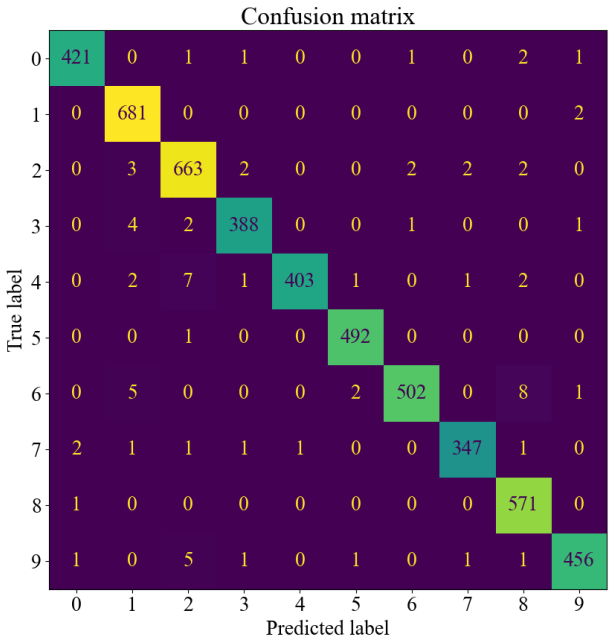

Figure 5.7: Confusion matrix for the multi-class classification of the security dataset.

5.3 Discussion

In this section, the results will be discussed. Firstly, an overall discussion of the data and its difficulties is given. Secondly, the results from the velocity predictions are analyzed. Lastly, the discussion regarding the classifier result is reviewed.

5.3.1 Data discussion

The available data at the start of the study, the verification dataset, were insufficient and not as expected. As mentioned several times, utilizing machine learning algorithms is a data-driven method, meaning insufficient data gives insufficient results. Despite this, the results that were achieved were decent. The verification dataset was collected from R&D employees during test drives of new software. During those test drives, the software is used to the greatest extent, even when it may be working poorly. That means the data do not represent how the driver wants to drive, making it insufficient. With such data, models will learn the software behavior instead of the driver's tendencies. To capture driving behavior and differences between drivers, as much pedal-driving as possible is wanted. The data quality is further reduced because the test drives are performed by R&D employees and not professional truck drivers. A less experienced driver will generally drive inconsistently and be more affected by surrounding and external factors, such as approaching traffic, sight, and the road width.

Developed software for predicting velocity or classifying drivers should work in as many driving scenarios as possible, but when developing such functions, data from

a few drivers on the same route would be preferred. That would simplify analyzing the data to find trends and differences and enable the development to start from a manageable environment. Data from many different drivers on many routes made that challenging. In addition, the driver label from those driving logs was inaccessible, making personalization impossible. Still, the first dataset was valuable for getting to know the structure of the data and when developing tools and methods.

The second dataset, the field test data, was more suitable for this study's methods. Firstly, the field test data is collected from trucks where two or a few professionals drive the same route each day, and secondly, it was possible to extract the driver. This made it easier to comprehend and analyze the data, and driver labels enabled classification and personalized velocity predictions using this data. Unfortunately, the driver labels were only possible to extract from one of the trucks, reducing the possibility of classifying or personalizing the driver predictions for more than two drivers.

What became evident was that the filtering of the data did not always deliver the results that were expected, this was due to imperfections in the map-based data. When extracting a sample of a road that was filtered out due to the road speed restriction, the actual road speed restriction differed from what was given in the map-based data. This type of faulty filtering is inevitable and will contribute to data being unnecessarily discarded.

5.3.2 Velocity prediction

As mentioned, the results for predicting the velocity based on the current state did deliver somewhat satisfactory results. The one-second ahead prediction was almost identical to the true velocity profile, and the ten-second ahead predicted velocity profile was marginally deficient. With one-second predictions, the model is not regenerating a prediction based on a previous prediction, hence avoiding the accumulation of errors. With an increased prediction horizon, the risk of accumulating errors increases as well, which can be seen in the prediction with a longer horizon such as five and ten seconds. The training of the model weights was made for predicting one time step ahead, meaning that the model is a many-to-one and the input is a sequence of data, and the output is a scalar velocity. This will, of course, also contribute to the predictions for long horizons being inaccurate.

Predicting the velocity independently of the current state is a complex problem, even if given information about the oncoming road segment. If given the current velocity and the predicted velocity for the next time step is the current velocity, the prediction could not be that far off. As this model is supposed to be independent of its current state, the predictions are made independently, and each prediction is not based on the previous. This can also contribute to the model predicting incoherent velocities.

For both of the prediction models, the data that was given might not have been ideal. In the discussion session with experienced truck drivers, see Section 1.3, it became

apparent that information about the road, such as road width, is of great importance when adjusting the velocity of a heavy-duty vehicle. Driving a loaded timber truck is unmistakably another experience than driving a non-loaded truck, and the driver's behavior would most likely differ. This is also the case for road conditions and width. A more narrow road covered in water or ice most definitely affects the driver. Likewise, approaching vehicles and traffic are not considered, which, combined with a narrow road, undoubtedly influences the driver's action. These factors were never introduced to the model since they were unavailable, making it impossible to research if they would affect the result.

5.3.3 Classifier

When comparing classifiers, it is indisputable that they all perform well, especially on heavily filtered data and larger window sizes. The traditional classifiers performed poorly on lightly filtered data with window sizes smaller than 40 seconds but were also unable to output any results when the dataset was too large and the window size too small due to computational complexity. When classifying and training the models, various input features were used. This was done to investigate whether a feature impacted the classification and compare features. When fewer features were used in the training, it became evident that the LSTM classifier could still learn to classify binary. However, this was not the case for the traditional classifiers when training on the security dataset, and to achieve high accuracy, 14 signals had to be used. Noteworthy is that multi-class classification is more complex than binary classification. The traditional classifiers did deliver high accuracy when using the field test dataset and four signals, even though a window size of 200 seconds was used to achieve such high accuracy. The LSTM classifier performed at an accuracy above 99% when trained on only two signals, namely lateral- and longitudinal acceleration. But as the LSTM classifier performs well when only trained on two signals, it may also do so when trained on multi-class classification and more signals.

The evaluation of models included both accuracy and F1-score. In the result, only the accuracy is mentioned. The simple explanation is that the datasets were balanced, and the F1-score told inseparable information from the accuracy.

6

Further research

This study has been of an exploratory character with promising results but, more importantly, interesting findings to continue researching and developing further. Some interesting topics to further research will now be discussed.

Since data is the key to machine learning models, it is an area to investigate further. Besides working on the quality and suitability of the data, discovering new input signals that could enrich the model with more information about the scenario would be interesting. With today's technology, information about road width, road conditions, approaching vehicles, and weather would be possible to collect using modern cameras, LiDAR, and other sensors. That data could then be input into one of the presented models. With that extra information, different scenarios may be easier to distinguish and simplify the models' learning. Furthermore, since the type of load can influence driving, information about the load type would also be valuable. This type of information is not accessible via the CAN bus system and might be difficult to extract. If it is possible to understand more about the load, for example, knowing if the load is timber, liquid, toxic, or remarkably heavy, it can give great insight into why and how the driver handles the vehicle the way he or she does.

In addition to enlarging the data with more signals, a lot of data from several drivers would enable exciting possibilities. Several drivers would make it possible to do multi-class classifications, either to classify the data to each driver or also make it possible to divide the drivers into different categories. For example, dividing drivers into an aggressive, normal, or cautious category or different environmental-based categories. A shortcoming in this study was the lack of data from one driver. With more such data, the velocity prediction models could expand and perform better when trained on more suited data. Since the whole idea was to find driver-specific differences, a lot of data, with driver labels, from several drivers is needed.

Another topic that was discussed but not researched due to data problems is transfer learning [43]. Transfer learning is a technique that enables training a model on a large dataset and then retraining or fine-tuning some layers of the neural network model. That could be used in a situation similar to this study's and possibly a topic to discover further. Using the whole dataset from several drivers to train the model initially and then the driver-specific data to fine-tune the model. That would give the model an extensive dataset, to begin with, and may speed up the training

progress since that initially trained model would work as the base model.

From the result section, it can be stated that the second velocity prediction model, which makes predictions without any information about the current state, worked badly. However, the fact that the model's performance increased during training and some behaviors and trends were captured makes it an interesting model to develop further. Higher quality data with more professional pedal driving and more input signals to describe different driving scenarios and factors in a better way may make such a model perform better.

The results from the first velocity prediction model are better, but the second model has some advantages when considering an implemented version of the models in a real situation. Since it makes predictions without information about the current state, a prediction is not limited to a specific horizon. The model can generate an estimated velocity for a whole route, making it possible to analyze the results before using them in a real driving situation. That way, the occasions when the model predicts poorly can be prevented. A faulty prediction cannot be completely ruled out when using a black box model, which neural networks are. Reviewing the outputs from the model makes it more robust and safer to use and hence, easier to implement in a large vehicle where an incorrect speed can lead to serious damage.

Another area to research further could be to investigate how to integrate a model similar to the ones in this study into the existing I-See software. Of course, the existing I-See software has always been in mind when developing, but the focus has been on exploring what possibilities exist and not what is possible to integrate with today's system. For instance, how to integrate a machine learning model with the I-See software is an interesting topic. Further, questions about if the learning and inference should be embedded in the truck or on a central computing cluster will also be of importance. If the model will be embedded, what constraints does that put on the model? These types of questions should be considered when exploring this subject further.

7

Conclusion

To conclude this project, the research questions that were stated in the introduction will be answered.

RQ1: What are possible and feasible ways for personalizing a cruise control using machine learning?

The potential solutions were personalized velocity prediction and driver classification based on driving data, using LSTM neural networks. Since these solutions are based on driving data they deliver output that include trends and patterns that belong to a specific driver. Thus, the solution is personalized.

RQ2: What type of data is available, and how can it be extracted? What data will be of interest?

As promised, there was a substantial amount of data. Unfortunately, this data was not suitable for machine learning used in personalization purposes as it was based on software testing. Luckily, we did get access to Volvo Group's field testing data which is more suitable for this study. This data has a lot of potential and should be the main focus if further research is pursued. Free datasets exist on the internet, such as the security dataset, which can be valuable when verifying software in development.

The extraction and storage of the data was a big part of this study. The extraction was made using in-house CAN processing tools and developed tools in Python. The storing was done using CSV, but most importantly, pandas DataFrames were extensively used. This Python-based library is excellent when working with a lot of data and makes data manipulation and analysis easy.

RQ3: What machine learning algorithms are suitable for adaptability with regard to driver behavior?

What was found is that driving is a continuous act, and patterns within driving cannot be picked up from discontinuous data points, therefore, machine learning algorithms that can handle sequential input are suitable for this study.

RQ4: Why and how is the self-adapting software method better than the existing solution?

It was observed during the study that the I-See software was not always running when driving the truck, especially when examining the data from the field test dataset. This prevents the I-See software from reducing fuel consumption to its full potential. Further, if the I-See software is abrupted, the driver is not satisfied with the software's behavior. A personalized self-adapting software can ensure that the I-See system is running and make the driver more comfortable and therefore elevate the drivability. The result from this study cannot be compared to the existing solution, and this research question can not be answered exactly, but the above mention conclusion is what could be reached after this study.

Summary: The data was not what was expected from the beginning of the study, and due to this, the progress of the study took another turn than expected. This is not abnormal within the research of machine learning and is something to be expected when starting a data-driven study. There is a difference between available data and suitable data, and the data processing part is a study itself. With this in mind, the results reached were valid.

Bibliography

[1] truckinginfo.com. Electric, hybrid trucks to exceed 1.6 million units by 2027. https://www.truckinginfo.com/143229/electric-hybrid-trucks-to-exceed-1-6-million-units-by-2027, 2017. [Online; accessed 17-May-2023].

[2] Volvo Trucks. How to improve truck fuel efficiency: Volvo trucks. https://www.volvotrucks.com/en-en/trucks/fuel-pillar.html, 2023. [Online; accessed 12-February-2023].

[3] Volvo Trucks. Explore the i-see. https://www.volvotrucks.co.uk/en-gb/trucks/features/i-see.html, 2023. [Online; accessed 12-February-2023].

[4] Irene Cara, Jan-Pieter Paardekooper, and T Helmond. The potential of applying machine learning for predicting cut-in behaviour of surrounding traffic for truck-platooning safety. In *25th International Technical Conference on the Enhanced Safety of Vehicles (ESV) National Highway Traffic Safety Administration*, pages 3–8, 2017.

[5] Federico Perrotta, Tony Parry, and Luis C Neves. Application of machine learning for fuel consumption modelling of trucks. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3810–3815. IEEE, 2017.

[6] Benoit Liquet, Sarat Moka, and Yoni Nazarathy. The mathematical engineering of deep learning. https://deeplearningmath.org/, 2023. [Online; accessed 16-May-2023].

[7] Chiyomi Miyajima and Kazuya Takeda. Driver-behavior modeling using on-road driving data: A new application for behavior signal processing. *IEEE Signal Processing Magazine*, 33(6):14–21, 2016.

[8] Martina Hasenjäger and Heiko Wersing. Personalization in advanced driver assistance systems and autonomous vehicles: A review. In *2017 ieee 20th international conference on intelligent transportation systems (itsc)*, pages 1–7. IEEE, 2017.

[9] D Zhang, Q Xiao, Jiangping Wang, and K Li. Driver curve speed model and its application to acc speed control in curved roads. *International journal of*

automotive technology, 14:241–247, 2013.

[10] Duanfeng Chu, Zejian Deng, Yi He, Chaozhong WU, Chuan SUN, and Zhenji Lu. Curve speed model for driver assistance based on driving style classification. *IET Intelligent Transport Systems*, 11, 07 2017.

[11] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):712–733, 2020.

[12] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[13] Zengcai Wang, Yazhou Qi, Guoxin Zhang, and Lei Zhao. Smart shift decision method based on stacked autoencoders. *Journal of Control Science and Engineering*, 2018, 2018.

[14] Bing Zhu, Jiayi Han, Jian Zhao, and Huaji Wang. Combined hierarchical learning framework for personalized automatic lane-changing. *IEEE Transactions on Intelligent Transportation Systems*, 22(10):6275–6285, 2020.

[15] Jianfei Huang, Xinchun Cheng, Yuying Shen, Dewen Kong, and Jixin Wang. Deep learning-based prediction of throttle value and state for wheel loaders. *Energies*, 14(21):7202, 2021.

[16] Lei Yang, Chunqing Zhao, Chao Lu, Lianzhen Wei, and Jianwei Gong. Lateral and longitudinal driving behavior prediction based on improved deep belief network. *Sensors*, 21(24):8498, 2021.

[17] János Kontos, Balázs Kránicz, and Ágnes Vathy-Fogarassy. Neural network-based prediction for lateral acceleration of vehicles. In *2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS)*, pages 153–158. IEEE, 2022.

[18] Oluwatobi Olabiyi, Eric Martinson, Vijay Chintalapudi, and Rui Guo. Driver action prediction using deep (bidirectional) recurrent neural network. *arXiv preprint arXiv:1706.02257*, 2017.

[19] Emanuele Lattanzi and Valerio Freschi. Machine learning techniques to identify unsafe driving behavior by means of in-vehicle sensor data. *Expert Systems with Applications*, 176:114818, 2021.

[20] Peng Ping, Wenhu Qin, Yang Xu, Chiyomi Miyajima, and Kazuya Takeda. Impact of driver behavior on fuel consumption: Classification, evaluation and prediction using machine learning. *IEEE access*, 7:78515–78532, 2019.

[21] Bedir Tekinerdogan and Mehmet Aksit. Adaptability in object-oriented software development: Workshop report. In *Proceedings of the 10th Annual Euro-*

pean Conference on Object-Oriented Programming (ECOOP), 1996.

[22] Anthony Zaknich. *Principles of adaptive filters and self-learning systems*. Springer Science & Business Media, 2005.

[23] European Commission. Reducing CO emissions from heavy-duty vehicles. `https://climate.ec.europa.eu/eu-action/transport-emissions/road-transport-reducing-co2-emissions-vehicles/reducing-co2-emissions-heavy-duty-vehicles_en`, 2023. [Online; accessed 22-May-2023].

[24] T.M. Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997.

[25] Larry Hardesty. Explained: Neural networks. `https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414`, 2017. [Online; accessed 17-May-2023].

[26] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[27] Understanding feedforward neural networks. `https://learnopencv.com/understanding-feedforward-neural-networks/`, 2023. [Online; accessed 17-May-2023].

[28] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, Aug 2015. [Online; accessed 15-April-2023].

[29] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[30] Kangil Lee and Junho Yim. Hyperparameter optimization with neural network pruning. *arXiv preprint arXiv:2205.08695*, 2022.

[31] Rohith Gandhi. Support vector machine—introduction to machine learning algorithms. *Towards Data Science*, 7, 2018.

[32] Jake VanderPlas. Python data science handbook. `https://github.com/jakevdp/PythonDataScienceHandbook/tree/master/notebooks`, 2016.

[33] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[34] Wikipedia contributors. Can bus — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=CAN_bus&oldid=1150654485`, 2023. [Online; accessed 19-April-2023].

[35] Wikipedia contributors. Local interconnect network — Wikipedia, the

free encyclopedia. https://en.wikipedia.org/w/index.php?title=Local_Interconnect_Network&oldid=1141381183, 2023. [Online; accessed 19-April-2023].

[36] Byung Il Kwak, JiYoung Woo, and Huy Kang Kim. Know your master: Driver profiling-based anti-theft method. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 211–218. IEEE, 2016.

[37] Sudharsan Asaithambi and How Why. Why, how and when to scale your features. *Preuzeto od https://medium. com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e*, 2017.

[38] Simplilearn. Normalization vs standardization - what's the difference? https://www.simplilearn.com/normalization-vs-standardization-article.

[39] FileInfo.com. Blf file extension - what is a .blf file and how do i open it? https://fileinfo.com/extension/blf. [Online; accessed 4-May-2023].

[40] ProjectPro. Python pandas dataframe tutorial for beginners. https://www.projectpro.io/article/python-pandas-dataframe-tutorials/405. [Online; accessed 4-May-2023].

[41] Google Ads Help. Csv file: Definition. https://support.google.com/google-ads/answer/9004364?hl=en. [Online; accessed 4-May-2023].

[42] Wherll. How to use datasets and dataloader in pytorch for custom text data. *Towards Data Science*, 2021.

[43] Qiang Yang, Yu Zhang, Wenyuan Dai, and Sinno Jialin Pan. *Transfer learning*. Cambridge University Press, 2020.

A

Appendix - Training progress velocity prediction models

A.1 Model using current velocity

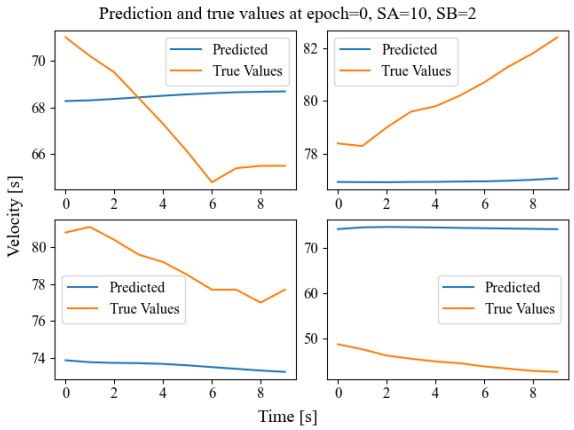

Figure A.1: Ten seconds velocity prediction on four test snippets before training.

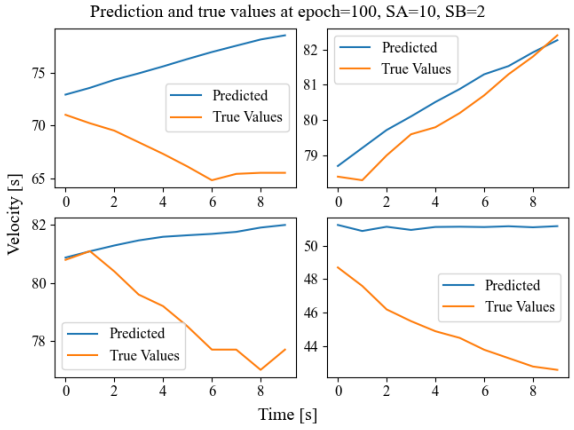

Figure A.2: Ten seconds velocity prediction on four test snippets after 100 epochs.

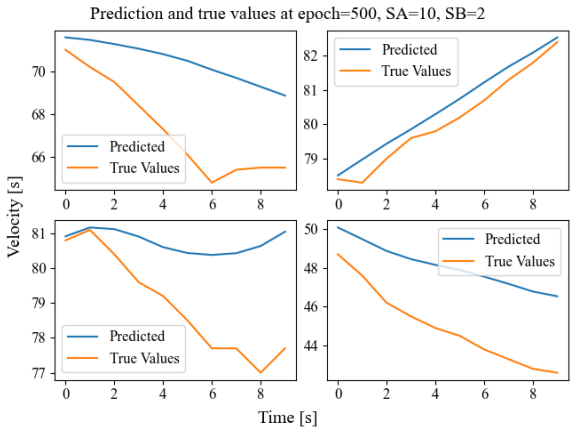

Figure A.3: Ten seconds velocity prediction on four test snippets after 500 epochs.

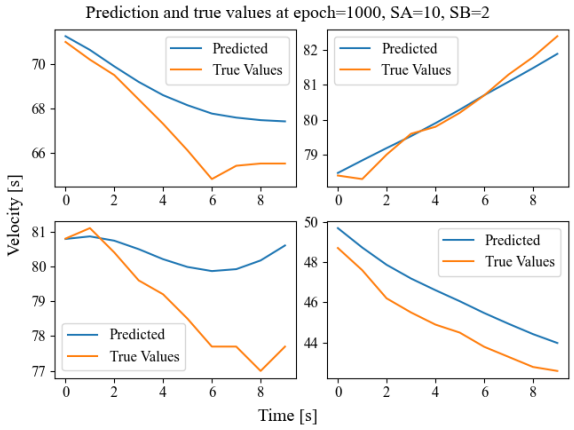

Figure A.4: Ten seconds velocity prediction on four test snippets after 1000 epochs.

A.2 Model independent of current velocity

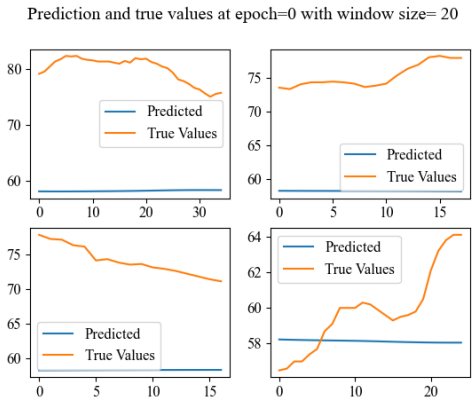

Figure A.5: Velocity prediction on four test snippets before training.

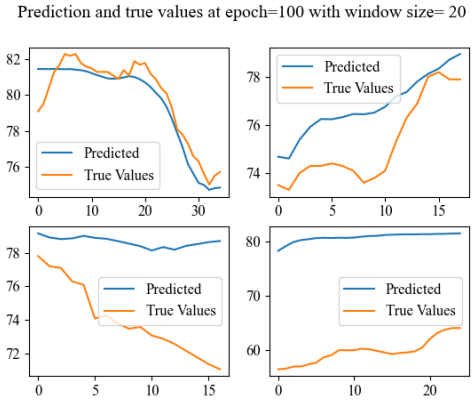

Figure A.6: Velocity prediction on four test snippets after 100 epochs.

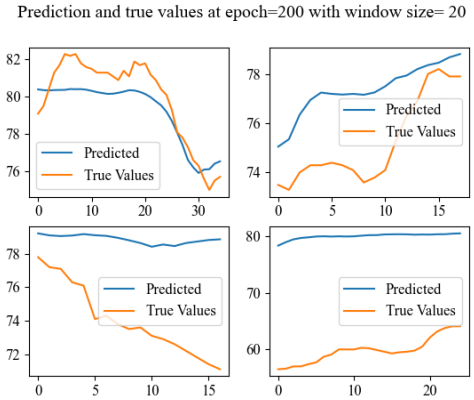

Figure A.7: Velocity prediction on four test snippets after 200 epochs.

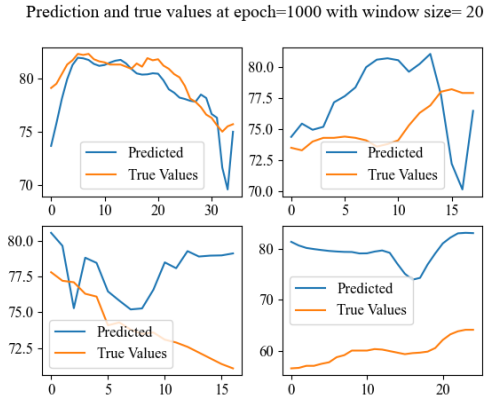

Figure A.8: Velocity prediction on four test snippets after 1000 epochs.

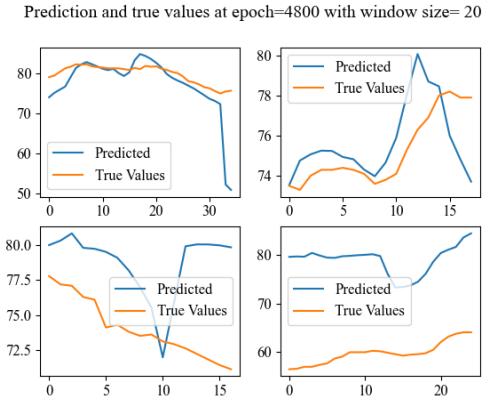

Figure A.9: Velocity prediction on four test snippets after 4800 epochs.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY