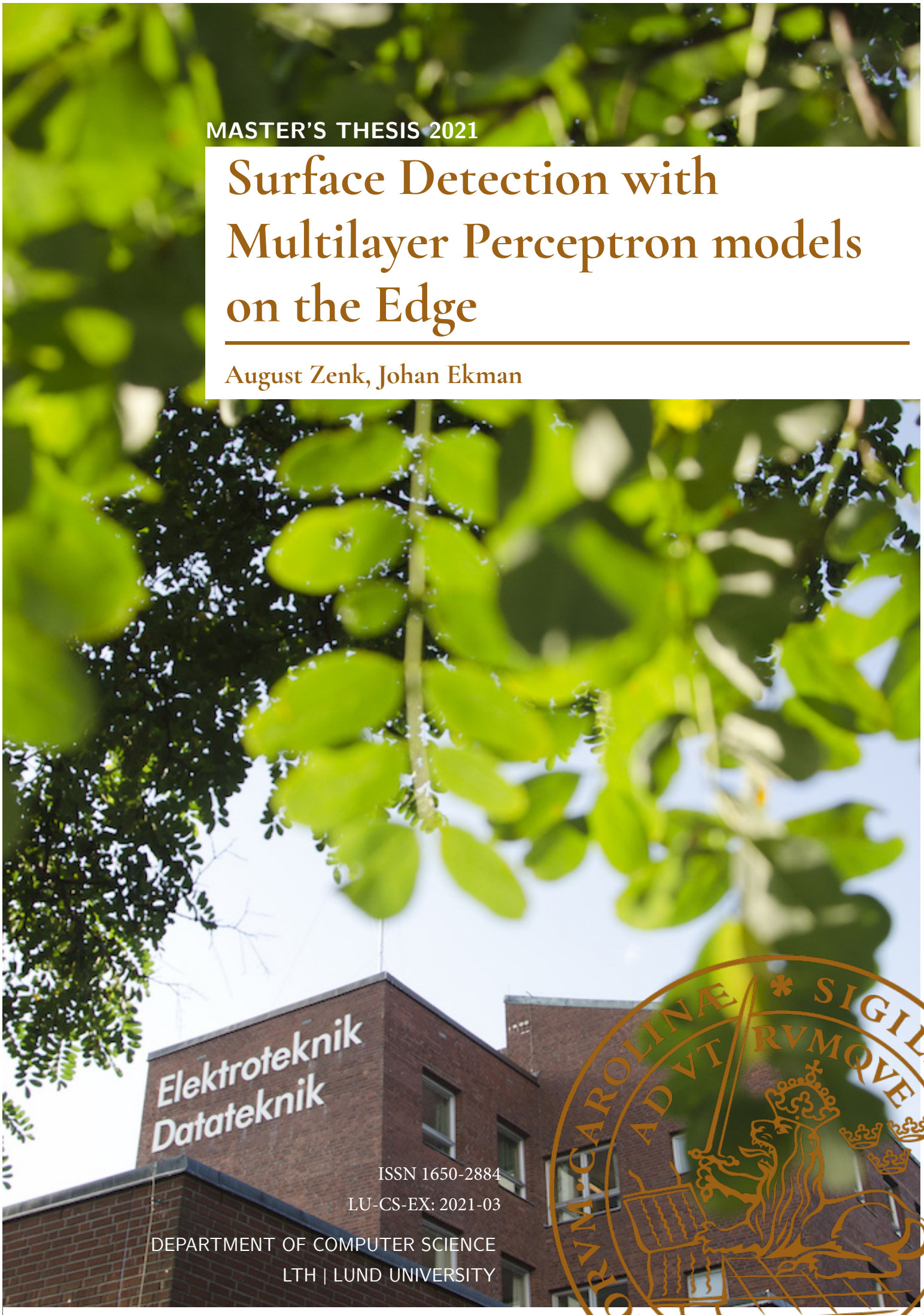


MASTER'S THESIS 2021

Surface Detection with Multilayer Perceptron models on the Edge

August Zenk, Johan Ekman

ISSN 1650-2884
LU-CS-EX: 2021-03

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-03

Surface Detection with Multilayer Perceptron models on the Edge

Underlagsdetektering med Flerlagerperceptronsmodeller på en Mikrokontroller

August Zenk, Johan Ekman

Surface Detection with Multilayer Perceptron models on the Edge

August Zenk
elt15aze@student.lu.se

Johan Ekman
elt15jek@student.lu.se

January 14, 2021

Master's thesis work carried out at Sigma Connectivity.

Supervisors: Arvid Elmér, Arvid.Elmer@sigmaconnectivity.se
Flavius Gruian, Flavius.Gruian@cs.lth.se

Examiner: Jacek Malec, Jacek.Malec@cs.lth.se

Abstract

As the number of IoT-devices increases rapidly, the load that the data communication will put on the networks will become critical. There are different approaches to solve this problem. One way is to increase the capacity of these networks. This solution would still leave the issue of storing all the data. Another way is to reduce the amount of data that are being sent. Instead of streaming data over the internet to make calculations on the cloud, these calculations could be moved and performed directly on the device. This method is called edge computing. Other advantages of this method is a lowered latency and decrease of streamed personal data.

In this project, we have implemented regression-based multilayer perceptrons (MLPs) on a microcontroller and evaluated their performance. With the sensor gateway Ardesco, we solved a use case of detecting the surface a radio car was driving on. Two systems were created, the Mixed system, which used one model for each surface class, and the Hybrid system, which used one model that could classify different surfaces. An accelerometer embedded in the Ardesco captured the vibrations created by the radio car, which the systems then used to predict the surface. This model was able to detect two surfaces; asphalt and grass. The models were trained on two data sets, Raw data and Feature data. They were then evaluated on the accuracy of prediction, memory allocation, and the energy consumption of running it. The system that performed the best in all categories was the Mixed system, for both datasets.

Keywords: MLP, Multilayer Perceptron, Edge omputing, Machine Learning on the Edge, Accelerometer, Surface detection

Acknowledgements

We would like to thank our team at Sigma Connectivity for being a big support during this project. Arvid Elmér, Artur Kobylinski, Jonas Bergh, and Ivan Fulöp has given us all prerequisites for a successful master thesis. It has been inspiring to work with you.

We would also like to thank our supervisor at Lund University, Flavius Gruian. All feedback and expertise have been essential for this work. You have encouraged us to tackle a lot of problems during this project.

Finally, we would like to say thanks to the people at Ekkono Solutions AB for their dedication and availability. We could always count on a fast reply to any question.

Contents

Chapter 1
Introduction

In 1948, Claude Shannon published a paper [25] describing a new communication system. That is considered the start of the Information revolution. The exponential progress in communication technology and digitalization has led to today's ambitions to develop a society built on connectivity.

The Cambridge Dictionary [2] defines connectivity as: *the ability of a computer, program, device, or system to connect with one or more others*. Networks with many connected devices make the communication system efficient. The concept of connectivity in devices is called the Internet of Things (IoT).

Ericsson has estimated that in 2022, 18 billion devices will be connected to some form of a communicative network [19], making the IoT market one of the fastest-growing. Because of this rapid growth, the interest within this field has intensified. McKinsey published a report [14] in 2015, analyzing the value beyond the hype. In 2025, they estimate IoT to have an economic impact of $ 3.9 trillion to $ 11.1 trillion per year. As it highest, it is equivalent to represent 11% of the world's economy.

Sensors and cameras are often embedded in these devices to gather data, which then is sent over the networks to communicate with other devices and servers. A device working as an entry point for a core network is called an edge device. With the estimated billions of devices, there will be an enormous production of data that needs to be processed. This data is getting transmitted over the networks, and trying to manage this traffic creates new challenges. For example, let us say a smart device is programmed to calculate a customer's daily needs and communicate this data over a predetermined frequency. The device would try to send over this channel while competing with all other users of that frequency. The load at the frequency would increase, leading to a negative effect on the latency. That leads to the dilemma of either keeping the efficiency high, spending a lot of money on expanding the bandwidth, or saving money at the cost of performance. Also, transmitting personal data over a big network makes it more accessible for hackers. Personal data is a valuable resource and should be protected. One way to lower the traffic on these networks, make the communication safer, and save on expenses is to perform edge computing. In this project, we have studied edge computing in

the form of the implementations of regression-based multilayered perceptrons on an edge device's microcontroller.

Usually, the edge device sends the raw data via the network to the cloud, where the data could be processed and then sent back to the device. Moving these calculations from the cloud and performing them straight on the edge device would lower the amount of data that otherwise would be communicated over the network. This concept is called edge computing. If these computations are machine learning, it is called machine learning on the edge. By moving the intelligence from the cloud to the edge, the amount of transferred data could be reduced by 96%. That, in turn, could lead to the price being lowered by up to 36% [18].

1.1 Purpose and problem description

The increasing number of connected devices will put a lot of pressure on communication networks, over which more data will be transmitted. One possible solution to lower the amount of traffic over these networks is machine learning on the edge.

The purpose of this thesis was to analyze the result of running a machine learning solution on an edge device. The problem we wanted to solve was to classify the surface a radio car was driving on. To classify these surfaces, we used regression-based multilayered perceptrons models, which we trained using accelerometer data gathered from driving a radio car. We utilized two approaches that we called the Mixed system and the Hybrid system. The Mixed system used several models, each taught to recognize a specific surface, and the Hybrid system used only one model for the classification. These approaches were evaluated based on three criteria:

- **Accuracy,** the percentage of correct predictions.
- **Memory allocation**, how much memory space the systems allocated on the edge device.
- **Energy consumption,** the amount of energy the solutions needed when inference on the device.

As the microcontroller we used had limited memory capacity, we chose to evaluate our models based on their size. The microcontroller was also battery-driven, making it crucial to have a solution that did not drain too much energy. Lastly, we looked at the accuracy as a measurement of how well the models performed.

A subproblem that we investigated in this thesis was if it was possible to preprocess accelerometer data to get a better performance in terms of prediction accuracy.

1.1.1 Reasearch questions

RQ1: What classification accuracy can we achieve with regression-based models on a microcontroller?

RQ2: Will a system with two regression models, one for each surface, reach a higher classification accuracy than a system that uses one regression model? If so, how would this affect the size of the system and its energy consumption?

RQ3: Can we create a surface detection system with higher classification accuracy by applying preprocessing to our data? If so, how will this affect memory allocation and energy consumption?

1.2 Contributions

This thesis contributes to the product value of Sigma Connectivity and Ericsson's sensor gateway Ardesco. It was also one of the first outsourced projects that built around Ekkono Solutions AB:s library Crystal, providing insight to the company regarding its usability. The use case where we classify surfaces based on accelerometer data offers a basis for a method that could map and anticipate maintenance for vibration-sensitive vehicles. For example, if some surfaces were labeled harmful, we could save the amount of time spent driving on these and used that to calculate potential damage.

1.3 Limitations

We evaluated the models on their accuracy, memory allocation, and energy consumption. An aspect that we left out was the latency between decision-making at the edge contra cloud-based machine learning. Although a significant part of machine learning on the edge, it was not possible to measure due to the microcontroller not being able to stream large quantities of data to the cloud at the time of the thesis.

We gathered data from the car in three different states, when it stood still, when it was driving on asphalt, and when it was driving on grass. We trained the models to recognize only these states, with further data gathering as a future goal. We gathered the data when the surfaces were dry, and we did not drive the car up- or downhills.

When evaluating the model's energy consumption, we only measured the consumption during the inference of the systems. Since we trained the models on a computer before we implemented them on the microcontroller, we did not measure the time or energy required during training. We structured the thesis around the Ardesco [9], which limited the use of components to those implemented on the hardware.

The company Ekkono Solutions AB supplied us with two code libraries, Edge and Crystal, that we used to train and convert our machine learning models to edge models. Ekkono Solutions AB is a company that Sigma Connectivity had contacted since they wanted a machine learning solution for the Ardesco. Since the libraries were in an early stage version, our choice of machine learning algorithm was limited to the ones implemented in the libraries at the time of the thesis.

1.4 Related work

A study [28] published in 2018 analyzed the performance of the three algorithms; Random Forests(RF), Support Vector Machine (SVM), and Multilayered perceptron (MLP) implemented on the edge. The accuracy, energy consumption, how fast they trained, and the

inference were all tested in the survey. They chose the algorithms because of their computational ability and since there was a lack of research with different algorithms within edge computing. In the report, the authors tested both classification and regression on all models. When they evaluated the regression models, they used the coefficient of determination as measurement. To find the accuracy, they compared the predicted values to the real ones. In their evaluation, random forests performed best in all aspects, but the authors concluded that MLP also performed well. The main drawback was that the model took a long time to train on the edge. If the training phase could be done on a computer instead and then integrated to the edge, it would be much more efficient. Also, the MLP (and SVM) appeared to be a better option for running on lightweight IoT-devices, but at the cost of having slightly lower accuracy than random forests. Since we trained the models on a computer and there was no implementation of the random forest algorithm in the Crystal library, we decided to use an MLP-model. We also chose to use the coefficient of determination as an evaluating parameter.

The study *Learning from accelerometer data on legged robot*[27] focused on a similar use case of surface detection as ours. In this project, a robot detected which surface it was walking on, using accelerometer data and the machine learning approach, Decision trees. The robot could classify three surfaces, a cement floor, a carpet inside their laboratory, and a carpet on a "RoboCup field." Variances and correlation coefficients calculated from raw data samples of x-, y-, and z-axis filled a sliding window and later on fed the decision tree. The raw data samples were limited to detect values in the range of ±2 g. They presented results that showed that surfaces similar to each other, in this case, carpet inside their laboratory and the carpet on a "RoboCup field," proved to be difficult to classify correctly. The surfaces we started to work with were asphalt and grass because we wanted to see if it was possible to train models with higher classification accuracy if the surfaces were less similar. We set our accelerometer to measure data in the range ±2 g since this was the most sensitive setting, and we deemed that our car had as delicate movement as the robot.

1.5 Outline of the report

Chapter 1, Introduction This chapter contains an introduction to edge computing and a summary of related work within this field. The scope of this thesis is described by the stated purpose and limitations.

Chapter 2, Theory This chapter presents the theory that we used to reach our results during this thesis.

Chapter 3, Method This chapter presents a more detailed description of the use case, including adjustments of MLP-models and the data processing.

Chapter 4, Results This chapter introduces the method of finding the optimal modifications for each model. A comparison in memory use, energy consumption, and accuracy performance between the two systems is presented.

Chapter 5, Discussion This chapter presents a general discussion of the results, important factors affecting the outcome, and improvements.

Chapter 6, Conclusion This chapter presents a summary of the thesis and discuss how the work could be continued in the future. The research questions stated in the introduction is answered.

1.6 Division of work

The table 1.1 shows how the work was divided. The 'x' denotes the person that was involved in that part.

	August	Johan
PRACTICAL WORK		
Collecting data	x	x
Accelerometer adjustments HW		x
HW application	x	
ML implementation	x	x
Enable flash memory on HW	x	
Demo of the application	x	x
REPORT		
Introduction	x	x
1.1		x
1.2		x
1.3	x	
1.4	x	
1.5		x
1.6		x
2.1	x	
2.2		x
2.3	x	
2.4	x	
3.1	x	
3.2	x	
3.3		x
3.4	x	
3.5	x	
3.6		x
4.1	x	
4.2	x	x
4.3	x	
4.4	x	x
5.1-5.8	x	x
6.1-6.2	x	x
References		x
Appendx A	x	
Appendix B	x	x

Table 1.1: The division of work between Johan and August

Chapter 2
Background

In this chapter, the hardware Ardesco is introduced, together with the implemented accelerometer and microcontroller. The machine learning algorithm used in this project, the multilayer perceptron, will be explained. Also, the theory behind the evaluation of the models is found in this chapter.

2.1 Ardesco

The **Ardesco**, **A**pproved **R**eference **D**esign by **E**ricsson and **S**igma **CO**nnectivity, is a sensor gateway (see figure 2.1) created in collaboration by the companies Sigma Connectivity and Ericsson and based on the hardware Thingy91 by Nordic Semiconductor. The Ardesco uses the operative system Zephyr [29]. The goal with this hardware was to create a cheap, short time to market, and afficient product that could be used for different applications.

The device supports the nRF9160 [24] SiP (System in Package). A SiP solution contains several integrated circuits stacked on top of each other. The nRf9160 uses the 64 MHz Arm Cortex-M33 [1] with a 256 kB low leakage RAM. That meant we had to fit our models to be suitable for this environment. The Cortex-M33 is a microcontroller, which is a computer that operates on a single integrated circuit [22] containing a processor core, memory, and programmable input/output peripherals.

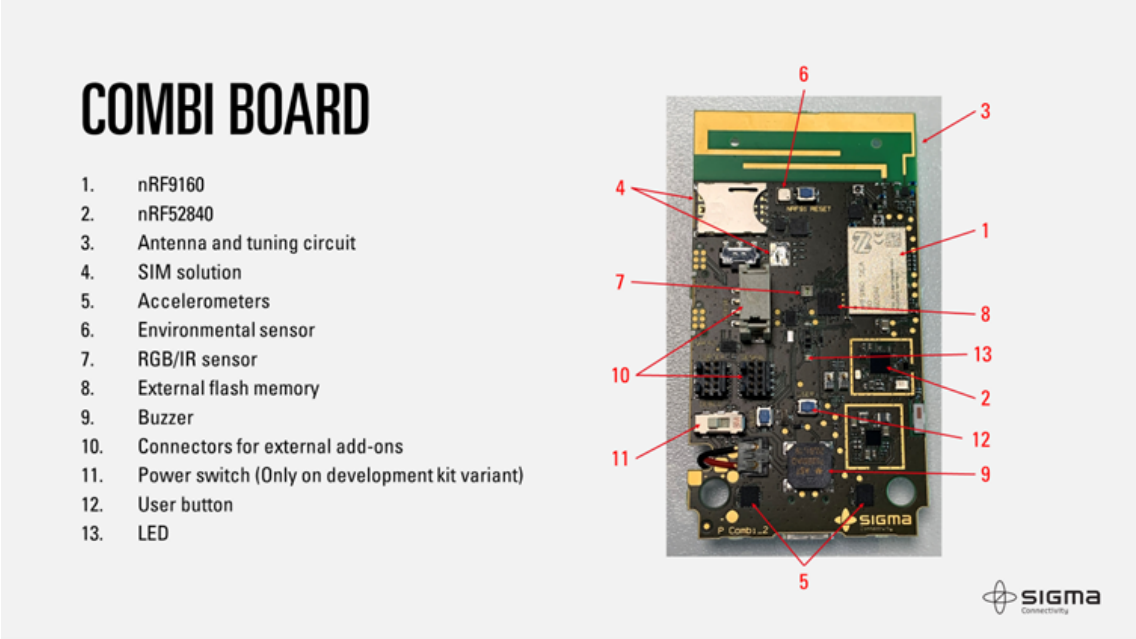


Figure 2.1: Overview of the Ardesco

2.1.1 Accelerometer

The Ardesco hardware support several different sensors, one of them being the accelerometer ADXL362 [7] that we used during this project. The sensor is designed to run with low power consumption and consumes 2μA with a 100Hz output rate and even less if in motion trigger mode. The accelerometer has three axes that measure the acceleration: side-to-side(x-axis), forward(y-axis) and up-and-down(z-axis). The values at rest is x=0, y=0, z=9.82, values measured in m/s^2. One data sample from the accelerometer containing three data points; acceleration of x, y, and z.

The range of measurements has three different settings: ±2g, ±4g, and ±8g, along with three different noise reduction settings: *Normal operation*, *Low Noise*, *Ultralow Noise*. These settings require a certain amount of power, see Table 2.1. During this thesis, we used the settings *Normal operation* within the range of ±2g. In the Ardesco, the sensor ran on 1.8V. Other settings were experimented with without a significant change in the result. That lead us to use the configurations that had the least energy consumption.

The ADXL362 supports different sampling options, including a wake-up trigger that starts measuring at a predetermined acceleration and several different sampling frequencies. During this project, we used a setting that let us sample the data once it was ready. That resulted in a frequency of ≈ 11Hz.

2.2 Ekkono Solutions AB

Ekkono Solution AB is a company that specializes in creating programming libraries for machine learning solutions on microcontrollers. During this thesis, we used two of their

Noise settings	Voltage supply	Noise ($\mu g/\sqrt{Hz}$)	Current consumption (μA)
Normal operation	2 V	550	1.8
Normal operation	3.3 V	380	2.7
Low Noise	2 V	400	3.3
Low Noise	3.3 V	280	4.5
Ultra Low Noise	2 V	250	13
Ultra Low Noise	3.3 V	175	15

Table 2.1: Table over energy consumption with different noise

libraries, Edge and Crystal. Edge is Ekkono's main library and had, during the time of this thesis, a wider variety of methods and functions than the Crystal library. Therefore, we used Edge when training and examine what configurations of attributes and layers fitted our data best. We did this in the python environment, Jupyter Notebook. The Crystal library was designed to work on devices that used the Zephyr operative system, like the Aredesco. This library could be integrated and used on a microcontroller to load and read machine learning models. To use the models that we had trained in Edge on the Ardesco, we used a built-in method to convert the models so that they could be used in a C based environment. The models were saved as hexadecimal vectors that we placed in the main script of the Ardesco. Here the integrated Crystal could read the models and use them for classification.

2.3 Multilayer Perceptron

The algorithm used during this thesis was a regression-based multilayer perceptron (MLP)[12]. An MLP is a popular [23] artificial neural network architecture and is build up by a few layers that all consists of a various number of something called a perceptron. A perceptron is an artificial nonlinear neuron that has been implemented with a specific algorithm. During this project, we used something called supervised data, meaning that the models were trained to map an input to a known output. This data passes through the system by entering through the input layer, continuing through the hidden layer(s), and finally exiting through the output layer. As seen in figure 2.2, all inputs will flow through the system and generate one output).

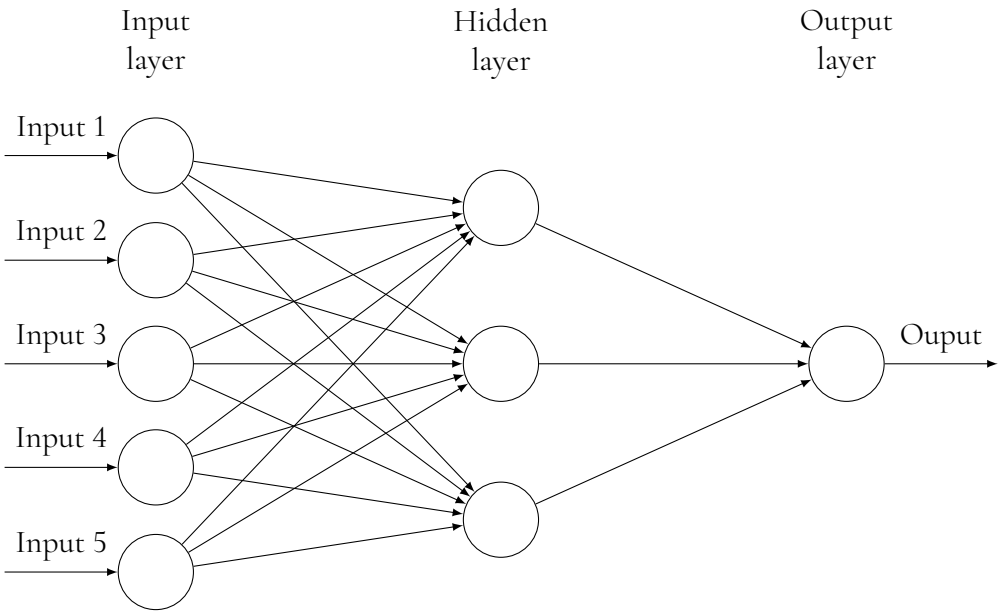


Figure 2.2: The figure shows an example of an MLP-network with one hidden layer. Each circle symbolizes a neuron.

During the training of a model, each neuron in the current layer receives one value from every neuron in the previous layer. This value gets multiplied with a weight value, which shows the importance of the input. To that product, a bias value gets added before the sum then passes through an activation function, see figure 2.3. There are several activation functions to apply. For this thesis, the Crystal library only made it possible to use a nonlinear Sigmoid function. As seen in figure 2.4, the summed value will be the input to the Sigmoid function, which normalizes the sum and gives an output between 0 and 1.

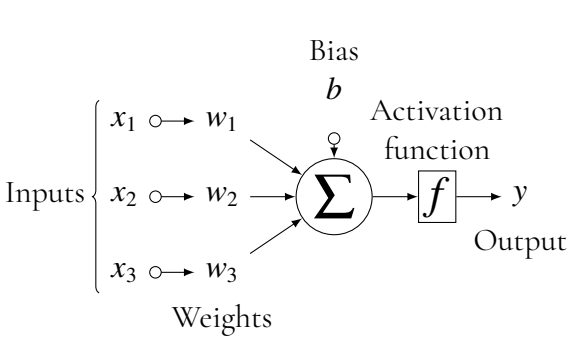


Figure 2.3: Overview of the perceptron algorithm

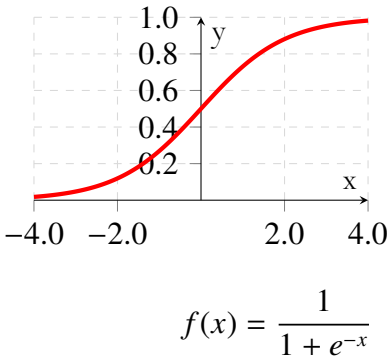


Figure 2.4: Sigmoid function

2.3.1 Input layer

When designing a neural network, the input layer should have one node for each type of attribute in the dataset. For this project, a pipeline, described in 2.3.6 and 5.2, decided the number of nodes in the input layer. The data is then sent from each neuron in this layer to each neuron in the upcoming layer.

2.3.2 Hidden layer

A neural network without hidden layers is just a linear regression model. Adding hidden layers will improve the model's ability to solve nonlinear problems. Each neuron in the hidden layers can partition the input space and with that increase [26] the computational power for the system.

The number of layers and dimensions of neurons in the hidden layers is adjustable so that the network can be fitted to any data. There are different architectures of neural networks. The one used during this project is called dense layers. That means each node in a layer has a connection to all nodes in the previous and upcoming layers. How the network is constructed will have an impact on the results. By adding more neurons to the hidden layers [20], the system will not necessarily be better, but instead, lead to overfitting. Overfitting indicates that the network fits a specific dataset too well. In the right graph in figure 2.5, the effect of this is visualized. The model fits the data samples with perfect accuracy, meaning that it can recognize these points. Introducing new, and the model would not be able to process them correctly. If the network instead had too few neurons in its hidden layers, the model would become underfitted, meaning that there is not enough computational power to extract information and patterns from the dataset. In the left graph in figure 2.5, the effect of this is visualized. The networks recognize no patterns in the dataset. The center graph shows a model where the number of hidden layers and neurons are well adjusted to obtain a balanced performance.

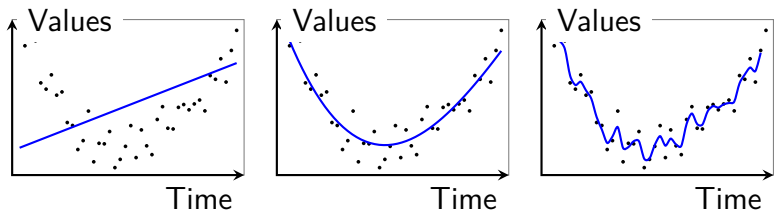


Figure 2.5: Left: underfitted, center: well fitted, right: overfitted

2.3.3 Output layer

The number of perceptrons in the output layer can vary, depending on whether the model is a classifier or regressor. A regression-based model will have one output, while a classifier will have one for each class. The output of this layer is the final prediction.

2.3.4 Error-backpropagation

The MLP model used during this project was trained using an algorithm called error back-propagation. This process includes finding the gradients from the loss function, which in this report is the mean square error (MSE) function, also called L2. This loss function was set in the code library by Ekkono Solutions and was not changeable.

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(Y_{true} - Y_{predicted})^2 \tag{2.1}$$

In equation 2.1 n is the number of predictions, Y_{true} is the true value, and $Y_{predicted}$ the predicted value. The equation sums all the squared errors and divides the result by the number of predictions (in the training step). This signal is propagating backward in the network and gives feedback to the system, adjusting the model to reach a better performance. The bias, weights, and coefficients in the activation functions are modified to find the best configuration for the problem. The analysis of the gradients in the loss function tells whether the weights should increase or decrease in value. This concept is called gradient descent. The goal is to find a local minimum of the error between prediction and true label through these adjustments. During this process, the learning rate is an important parameter. It designates the step length of each iteration to minimize the gradient of the MSE for each weight.

2.3.5 Regression and Classification

Machine learning is useful for two tasks: classification and regression.

A regression model will give a prediction for a continuous value. Problems solvable with a regression model could be, for example, a forecast of price, temperature, or weight, which only requires one output.

For a classifier, the number of outputs could vary. The model is predicting discrete value(s) mapped to a class. An example of this could be a determination of low, average, or high price, cold, cool, or warm temperature, light, medium, or heavyweight.

Our task of detecting the surface on required a classification model with a discrete output indicating standing still, asphalt, or grass. During this thesis, the code libraries that we used had been implemented by Ekkono Solutions to support a few chosen machine learning algorithms. A regression-based multilayered perceptron model was the best option for this use case since it is still possible to solve classification problems with a regression model. We did this by setting thresholds for the continuous output values, e.g. all values within an interval were forced to one class. For this thesis, we implemented two different systems of regression models for solving the classification problem, explained in section 3.3.

2.3.6 Configurations of the pipeline

In the Edge library, we had the option to pass the data through a *pipeline*. In this pipeline, additional attributes could be extracted and used when training our models. Three possible attributes could be added; lags-, deltas-, and backward moving attributes. These attributes could generate more information and patterns of the data since they offered knowledge of how the data changed over time. By adjusting the pipeline, we could add more or less time dependency on our classifications. By introducing attributes before constructing the network, the network itself did not need to find these patterns. According to Ekkono Solutions, this would keep the memory size of the models down.

- **Lag**, compare the current data sample to a sample n steps back.

$$lag(x_t, n) = x_{t-n} \tag{2.2}$$

- **Delta**, calculates the difference between the current accelerometer data point and the data point n steps back, for a certain axis.

$$delta(x_t, n) = x_t - x_{t-n} \tag{2.3}$$

- **Backward moving average (BMA)**, calculates the average accelerometer data point in a n long interval, for a certain axis.

$$backwards\ moving\ average(x_t, n) = \frac{1}{n}\sum_{i=0}^{n} x_{t-i} \tag{2.4}$$

n is the length of the pipeline and could be set to any value.

2.4 Accuracy parameters

The evaluation of our models was based mainly on three categories. Except for the memory allocation and energy consumption, we studied the performance of accuracy, both as a regression model and as a forced classifier. In this section, we present the parameters that we used.

2.4.1 Error calculation, regression evaluation

In this thesis, a regression model has been used to make classifications. The evaluation of regression models is usually based on the errors between the predictions and the correct value. Information about these errors can be found through different parameters.

Mean Absolute Error [16] (MAE): MAE gives a value on the average error between prediction and real value. This error does not provide any information regarding the direction of the error, i.e. if the predicted value was too high or too low.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|Y_{true} - Y_{predicted}| \tag{2.5}$$

Here n is the number of predictions, Y_{true} the true value and $Y_{predicted}$ the predicted value.

Root Mean Square Error [6] (RMSE): It is the root square of the previously mentioned Mean Square Error. In contrast to MAE, RMSE puts more emphasis on large differences between prediction and the correct value. Since the difference is squared, a large error will become even more profound.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(Y_{true} - Y_{predicted})^2} \tag{2.6}$$

Here n is the number of predictions, Y_{true} the true value and $Y_{predicted}$ the predicted value.

Coefficient of Determination [15] (CoD): Also called the R^2-value, the CoD is the squared value of the correlation and shows the linear relationship between predictions and the correct labels. The coefficient ranges between minus one and one, minus one indicating that the predictions are behaving in the opposite way of the real values, and one meaning that they are following each other precisely. The interval zero to one, or zero to minus one, can be divided into three groups. 0-0.3 is weak, 0.3-0.7 is medium, and 0.7-1 is a strong linear relationship.

$$R^2 = \left(\frac{n\sum Y_{true}Y_{predicted} - (\sum Y_{true}\sum Y_{predicted})}{\sqrt{[n\sum Y_{true}^2 - (\sum Y_{true})^2][n\sum Y_{predicted}^2 - (\sum Y_{predicted})^2]}}\right)^2 \tag{2.7}$$

Here n is the number of predictions, Y_{true} the true value and $Y_{predicted}$ the predicted value.

To evaluate regression, we used two different ways to calculate the error between the output of our models and the real value. We calculated the MAE and the RMSE to use as a guideline for the threshold we later set to force the prediction to a class. We used both since they give different information about how the models behave. The CoD was calculated to see how well the trained models were fitted to the data sets. A high value on CoD combined with low accuracy on the test set would indicate that the model was overfitted.

2.4.2 Confusion matrix, evaluation classifier

The confusion matrix [10] is a matrix that shows the performance of a machine learning classifier. The size of the matrix is decided by the number of classes in the problem. In the binary case of True or False, the matrix has a dimension *2x2*, see table 2.2.

		Prediction outcome	
		True Positive	False Negative
Actual value		False Positive	True Negative

Table 2.2: Confusion matrix layout for a binary case

The matrix can be divided into four categories:

True Positive (TP), predicts True and the correct answer is True.

True Negative (TN), predicts False and the correct answer is False.

False Positive (FP), predicts True and the correct answer is False.

False Negative (FN), predicts False and the correct answer is True.

With these, several parameters can be calculated that describe the accuracy of the model.
Accuracy, [4]: Gives a value on the overall performance of the model. Good accuracy often indicates a better model, but the result can be deceiving depending on the use case.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$

Precision, [10]: Shows the prediction accuracy for a specific label and can give more information than just accuracy. If a test set is filled with mostly True values and just some False, then a model that only predicts True would get high accuracy. By looking at the precision of each label, a better evaluation of the model's ability can be done.

$$Precision = \frac{TP}{TP + FP} \tag{2.9}$$

Recall, [10]: Also called True Positive Rate (TPR), the recall gives a value on how many of all the predictions that say True are correctly predicted. The value indicates how well the model predicts outliers.

$$Recall = \frac{TP}{TP + FN} \tag{2.10}$$

F1-score, [10]: It is a combination of precision and recall, often referred to when talking about a model's overall performance instead of referring to accuracy. F1-score is not directly calculated from the confusion matrix but instead uses the values from recall and precision.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.11}$$

We used all these parameters to evaluate the classification. The accuracy and F1-score gave the overall performance of the models. We used precision and recall because we wanted to create models with high classification accuracy for each class.

Chapter 3

Method

In this chapter, the work process is described and motivated. In figure 3.1, the chapter are summarized. Firstly, we write about the gathering of data and how we organized the Raw and Features datasets. Secondly, we describe the systems that solved the surface detection and how we trained our models. Lastly, we present how we evaluated the models.

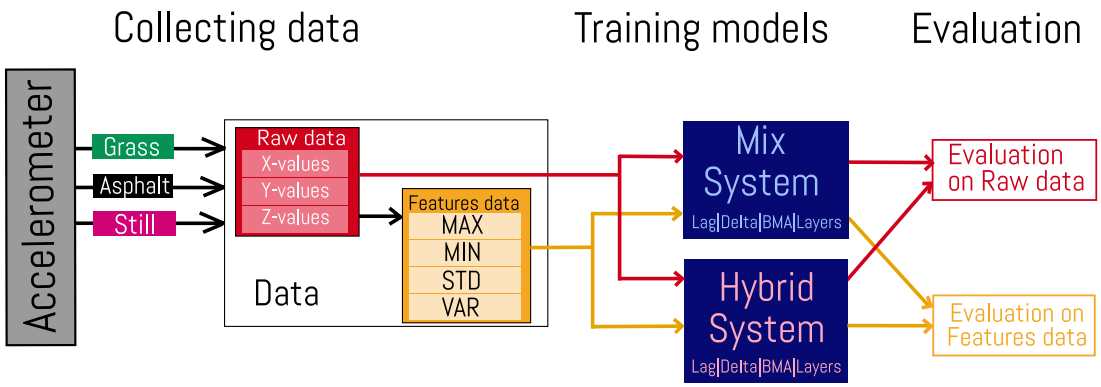


Figure 3.1: Overview of the steps in the method. The paths in yellow follows the Features dataset and the red paths follows the Raw dataset.

3.1 Data

Data is necessary to train a machine learning model. In a high variance case like this, the more data samples the model is trained on, [15] the better it will be at recognizing patterns and

similarities between samples. This section will describe how we collected and processed the data we used to train our models.

3.1.1 Collecting data

Sigma Connectivity wanted this project to show that the Ardesco could be used as an end-to-end solution for machine learning, i.e, that we could both collect data with the sensors integrated on the device and implement and run machine learning models on it. Therefore we used the accelerometer embedded in the Ardesco to gather data (see section 2.1.1). We did this by driving a small radio car over different surfaces, with the Ardesco fastened on the trunk of the radio car. The sensor then measured the acceleration vertically, sideways, and horizontally. In this report, we define vibrations as the difference in amplitude and the frequency of amplitude-spikes of the accelerometer data.

The values of the accelerometer were stored in the memory of the nRF9160, see section 2.1. The memory space allocated for the accelerometer was about 40kB, limiting the number of samples that we collected during each drive to around 1000. Due to this limitation, the datasets that we used were created by merging several samplings of short gatherings with the car. Though a bit tedious, we do not think that using this process affected the result of the data. Some experiments to store data on a flash memory implemented on the Ardesco was done but later not used when we realized that the number of samples gathered in one take would not improve notably.

The accelerometer sensor measured data as float values. The method that read the data from memory could not read float values, which meant that we had to convert the numbers to integers. To be able to get information about decimals, the float numbers were multiplied by 1000. We decided that three decimals were going to be enough to train the models on the difference between the surfaces. Some information about the behavior of the vibrations might have been lost when converting to an integer. It was during this thesis determined neglectable, but further experimenting upon an expansion of the project could be valuable to examine this further.

Vibrations from two different surfaces, asphalt and grass, were chosen to train the models. We wanted to refrain from making the classification problem binary and see if we could use a more complex model on the edge. Therefore, data from when the car stood still was also gathered and used during training. Figure 3.2 presents the number of collected samples used in the Raw dataset. We chose to drive the car on grass and asphalt since we thought the vibrations would generate different enough measurements to make good predictions, see figure 4.1.

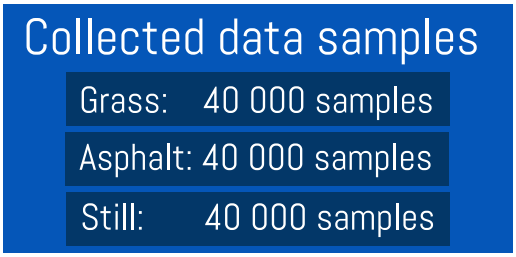


Figure 3.2: Number of collected samples for each surface.

The data that we used for training our models were gathered in the proximities of Sigma Connectivity's headquarters in Lund, making the difference of data samples limited to the streets and lawns close to the building. We made some efforts to differentiate the data by changing the location for each drive, but more changeability could be desirable for further expansion of the thesis.

While we gathered data, we drove the radio car at different speeds, turned, and drove it backward. Figure 3.3 shows the car model that we used. It had a top speed of 33 km/h and quite a sensitive rotating axle, causing it to, sometimes, make steep turns. It was desirable to be able to identify the surface regardless of the speed and how it turned, but driving the radio car at a constant speed and trying to avoid turning could have resulted in more leveled results in regards to vibration. Alternatively, the samples could have been divided further into different speed categories like asphalt fast and asphalt slow, adding more complexity. This approach was not tested but could be studied in a potential expansion of this project.

We only gathered data from when we drove the car on flat surfaces, i.e. we conducted no runs on slopes. We chose to do so since the amount of time we had to collect data samples were limited. If the accelerometer is tilted, as it would be uphill or downhill, the forces measured on the axes would differ, as shown in figure 5.1. Although an examination of the effect of such data would be interesting for future work, we lacked time to investigate this during the thesis.

Figure 3.3: The radio car with an Ardesco on the trunk

3.1.2 Accelerometer data

The data gathered with the accelerometer was processed in two separate ways: the Raw dataset and the Features dataset. We generated the Features dataset by calculating different characteristics from the raw accelerometer data. During the thesis, the Mixed system and the Hybrid system were both trained and compared on both datasets. Since the datasets contained different data, we decided not to compare the models trained on the Raw dataset to models trained with the Feature dataset. Figure 3.1 shows the comparison of the systems. The red lines describe the process of the Raw dataset, and the yellow lines the one for the Feature dataset.

Raw dataset

The Raw dataset contained samples of data points from all three axes; x, y, and z. Initially, we thought that the vibrations created by the car were easiest mapped by analyzing data that had been measured on the z-axis of the sensor. This because the other axis measured acceleration forward/backward and side to side. To test this theory, we plotted the data from each axis in histograms that showed the distribution of the samples from different surfaces, see figure 4.2. We could then see that the result from each axis showed some distinction between the surfaces. The problem was that they all fell within the same distribution, making the samples hard to distinguish from each other. To solve this, we developed a new alternative method of how to represent the collected data and created a dataset that we called Features. To see which of the three axes contained the most information for surface detection, we trained the models with only values from one axis at the time.

Axis evaluation on the Raw dataset

To find which axis on the accelerometer gave the most information regarding a surface, we created a test. Three models were trained, each on data containing only attributes from one axis. The training attributes were the value from the axis, lags from one to five, deltas of one to five, and a backward moving average (BMA) of ten. We choose these values to give the model a better overlook during a switch of two surfaces. Looking and comparing with samples too far back in the pipeline could lead to the model being slow in noticing that the vibration had changed, making the model perform worse. The number of hidden layers used was set to one with eight neurons, as guided by the rule-of-thumb mentioned in section 3.4. The result from these tests are shown in table B.1, B.2, and B.3.

After this, we evaluated the axes performance based on the same parameters as described in section 2.4 and ranked them as to how important they were in further analysis and testing. The result of this test was the basis for the creation of the Features dataset. Also, the test was used to examine if adding more lags -, deltas-, and backward moving average attributes to the best-ranked axis would increase the overall performance of models trained on the Raw dataset. For the second-best ranked axis, fewer attributes would be added, and for the third-ranked, even fewer. These tests is denoted as test #9 and #10 in Appendix B, table B.4

Features dataset

Since the computational power and memory space is limited when running machine learning models on an edge device, we decided to use features from data from one of the axes. As mentioned earlier, we thought that the z-axis would be the axis that was the least dependent on the speed and change of directions of the car. For example, if we trained the model with data from the x-axis, it would have been dependent on the acceleration while turning. Regardless of how the car was driven, differences in the z-axis always occur from surface to surface. Since the axis which contained the most information according to the evaluation test done on the Raw dataset was the y-axis, we compared the result from extracting features from both the y-axis and z-axis.

We calculated features using data from the z-axis and y-axis separately. We did this by dividing the data into intervals with the help of a sliding window. We set the size of the window to 11 due to our sample frequency being 11 Hz, i.e. the number of samples gathered

during a second. We did this based on the result of an article [11] discussing a similar case. We used a window that had an overlap of three samples between the intervals to smooth the transition between the calculated values. We did this based on the previously mentioned article. For each interval of 11, we calculated the standard deviation (STD), the variance (VAR), the maximum value (MAX), and the minimum value (MIN). The results of these calculations were used as attributes to create the data set Features data. We compared the models trained on the datasets created from the y-axis and z-axis and proceeded with the one that gave the highest performance accuracy.

3.2 Model selection

We used a code library that specialized in training and converting machine learning models to make them compatible with edge devices. This library was still being developed by Ekkono Solutions at the time of this thesis. That restricted us to chose from two models that that were implemented in the Crystal library. These were linear regression or a multilayer perceptron model for regression.

As mentioned in 2.3.5, compared to a linear regression model, the MLP could solve more complex problems. We, therefore, chose to work with the MLP.

3.3 The two systems

As mentioned in the theory section, 2.3.5, classification was possible to do with a regression algorithm such as the MLP. To create a model that would be able to make good predictions, we implemented two systems for the use case of classification. The first system, *Mixed system*, consisted of two models, each trained on one of the surfaces. These models will be denoted as *Model Grass* and *Model Asphalt*. The second system, *Hybrid system*, relied on only one model to predict both surfaces. We chose two systems to compare because we wanted to see how the accuracy and memory size changed depending on the system. We hypothesized that the *Mixed system* would have the best accuracy since it had one model for each surface to recognize. Also, we thought that the *Hybrid system* would be much smaller in memory since it only required one model to recognize all surfaces but with a drawback of accuracy.

Mixed system

In addition to the accelerometer values, we used lag-, delta, and backward moving average attributes calculated from the data. Internally, the pipeline was split into two segments; a buffer of samples and samples used for predictions.

In the pipeline, the required segment length for a prediction varied. It depended on how we initialized the lag-, delta-, and backward moving average attributes. A delta attribute that calculated the difference from five samples back required at least that many in the pipeline.

AAfter a prediction, the oldest sample in the segment was replaced with the first sample in the buffer. This process continued until all samples in the buffer had been used. After this, the pipeline was refilled with new samples, and the predictions continued. Both the *Model Grass* and the *Model Asphalt* used the same pipeline for a prediction, see figure 3.4 . Inside a

model, an estimation of a value between *0* and *1* was generated. This value represented the model's prediction of a label (surface) as a float value. We set a threshold to convert the output to a discrete value. If a prediction was greater than the threshold, it was converted to *1* and if it was smaller, to *0*. By this method, we performed classification with regression models. During the training, we set the threshold to 0.5 for all models to make the comparison fair. We considered this limit as the minimum value for a prediction to be transformed into a discrete number of 1. After we had found the final models, we analyzed adjustments of the threshold for best accuracy. MAE indicated how big the mean absolute error was between the prediction and correct label. That gave a good view of what range the predictions correlated to a label of 1 were. With this knowledge, the threshold could be adjusted for improving accuracy.

The output from both models was sent to the function *Decider*, see figure 3.4. This function determined which surface the car was driving on. We then implemented a counter in the decider that keeps track of how many times each model classified a *1*. The model with most counts of *1* within the samples of a pipeline became the final output from the system, indicating what surface the car was driving on. If the counter gets the same amount of ones from each model, the output is the same as the previous classification.

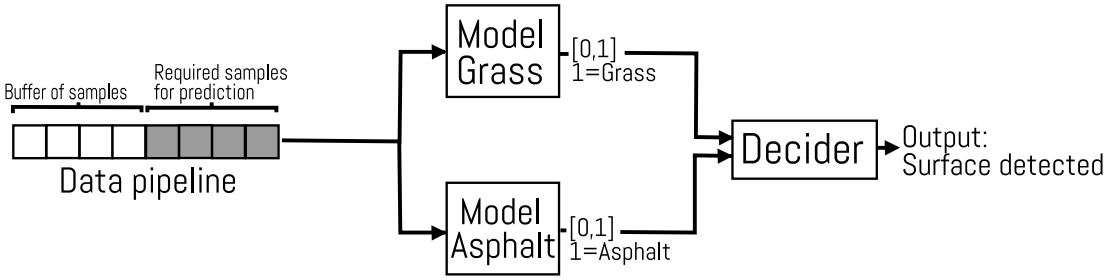


Figure 3.4: A flowchart for the *Mix system*

Hybrid system

For this system, we used the same segmentation of the pipeline as in the *Mixed system*. Instead of two separate models, we implemented one hybrid model, able to recognize both surfaces, see figure 3.5). In the training step, the data samples was labeled as: *0*-car stood still, *1*-asphalt, *2*-grass. Having the labels increasing from still to asphalt and then to grass was the most intuitive approach because that was how the amplitude of the vibrations increased. We conducted a test where grass and asphalt switched labels, but the result did not differ.

The model predicted a float value between zero and two. We then forced that value to a discrete number of 0, 1, or 2 with the help of thresholds set to 0.66 and 1.33. Values below 0.66 were converted to a zero, values between 0.66 and 1.33 converted to 1, and values greater than

1.33 converted to 2. The state of decision then counted what surface had the most occurrences within a pipeline and set the output to that surface.

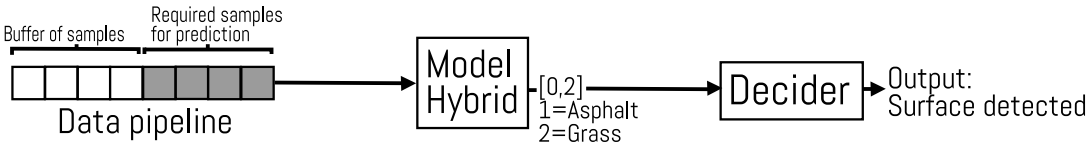


Figure 3.5: A flowchart for the *Hybrid system*

3.4 Training models

The code library that we used to train our models offered functions to add more attributes to the training process than just the raw data. By using the delta, lag, and backward moving average (BMA), see section 2.3.6, the model would get a better understanding of what surface the car was driving on and when it this surface switched to another one. The data set was split into two subsets: a training set that consisted of 80% of the samples and a test set of the remaining 20%.

Depending on the ranking of which axis performed best, see section 3.1.2, the lag-, delta-, and backward moving average attributes were set to different lengths for all axes to generate the best possible models. In these tests, we used the rule-of-thumb that is described later in this section.

When we trained our models, it was possible to change some parameters in the code library. These were *max iterations*, *learning rate*, *early stop iterations*, and the number and size of *hidden layers*. The value of *max iterations* stated how many times the training set was passed through the model. The time for running the training simulations was limited, and the *max iterations* was set according to that restriction. Since the Features data contained fewer samples, we could set a higher value for *max iterations*. We deemed setting different values for the systems fair since we did not compare the systems against each other. When we had found the final models, we increased *max iterations* to see if that would lead to improvements. If the model did not improve during an interval of iterations, we could set the parameter *early stop iterations* to trigger and stop the process. During each iteration, the model was adjusted to fit the data. How large this adjustment was, was determined by the *learning rate*. If the *learning rate* was too big, the model ran a risk of being overfitted. Too low, and it would take too long to find the best fit. The learning rate was set to 0.1, as it has been proven to give the best output of a neural network [3]. The parameters values we used are presented in the table 3.1.

	Max iterations	Learning rate	Early stop iterations
Raw data-models	1000	0.1	500
Features data-models	5000	0.1	500

Table 3.1: Chosen parameters for the different models

The last parameter that we could change was the configuration of *hidden layers*. There are no known ways that best decide the quantity of *hidden layers* and how many neurons these should consist of to get the best result from a multilayer perceptron model. It is generally regarded [20] that most problems can be solved using only one hidden layer, one exception being discontinuous data, e.g. saw formed wave structure. The figures in section 4.1.1 show that the gathered data has such a wave structure. There is no research proving a hidden layer structure with more than two layers is needed for an MLP [20]. Because of this, we tested up to two layers during training. Regarding the number of neurons, there are some rule-of-thumb guidelines when designing neural networks:

I The number of hidden neurons should be between the size of the input layer and the size of the output layer

II The number of hidden neurons should be less than twice the size of the input layer

III The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer

Apart from those, the best way is a trial and error approach[20].

To find a combination of hidden layers and neurons that would fit our datasets, we conducted a test similar to the exhaustive search approach. We trained with up to 50 neurons in each layer with a step size of 5 neurons for every increment. The number of neurons in the second layer was never more than the neurons in the first layer. Before the simulations, we tested this and always obtained higher accuracy with more or equal neurons in the first layer. The test started with a model using one hidden layer consisting of five neurons. We evaluated the model on its precision, described in section 2.4, and the memory size of the model. We trained all models with the same seed to get a fair view of the performance and find any deviation.

The memory we could use on the Ardesco device was limited to around 40kB. That meant that in the *Mixed system*, each of the models could not exceed 20kB and that the *Hybrid system* was limited to a maximum size of 40kB. The results of each test can be found in Appendix A.

When we determined the best model, accuracy and memory size were the decisive parameters.

Summary of the steps in training process for Raw dataset:

1. Rank the axes which generates the best accuracy performance, section 3.1.2.
2. Test different combinations of lag-, delta-, and backward moving average attributes on the axes to see which performs best in accuracy, with regard to the ranking from step 1. Best model will proceed to step 3.

3. Test different number and sizes of the hidden layers. The best model was based on accuracy and memory size. After this step, the best model for *Model Grass*, *Model Asphalt*, and *Model Hybrid* were determined.

Summary of the training process for Features dataset:

1. Test which of z-axis and y-axis generated the best accuracy performance. The best axis proceeded to step 2.
2. Test different configurations of MIN-, MAX-, VAR-, and STD attributes. The best performing model will proceed to step 3.
3. Test different numbers and sizes of the hidden layers. The best model was based on accuracy and memory size. After this step, the best model for *Model Grass*, *Model Asphalt*, and *Model Hybrid* were determined.

We chose this order since Ekkono Solutions recommended us to add the attributes before elaborating with the layers. That would keep the size of the models small and less complex as described in 2.3.6.

3.5 Energy consumption

After testing different configurations of layers, we selected the ones that had the highest performance to be used on the Ardesco. We used the process described in section 2.2 to implement the models on the device. To measure the energy consumption, we used an external battery to drive the Ardesco that we could connect to our computers via a USB cord. The software Otii [21] was then used to measure the amount of Wh that the battery consumed. We did not take into account the energy consumption the Ardesco needed during its start-up process since we only wanted to see the impact of our different systems. The energy was measured during a two-minute time interval. During this period, the measurements were recurrent and showed no diverging energy spikes. The Energy consumption was measured for the two systems inferencing on the Raw dataset and Features dataset.

3.6 Software and Hardware specification

What follows are the specifications of the software and hardware that were used during this thesis.

Software

- Ekkono SDK v.20.10 [8]
- Python v3.7.9
- Windows OS: 18362.11139
- Windows version: 1903

Hardware

- Ardesco Combi DP3 [9]

Chapter 4
Experimental Evaluation

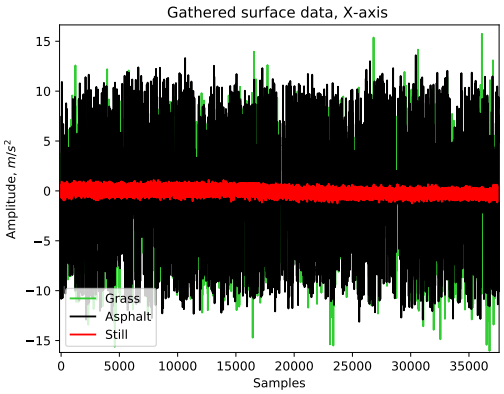
In this chapter, we present the result of all the different tests that were conducted during this project. We here want to show the nature of the accelerometer data and the importance of the data analysis that led to the idea of the Feature dataset. We present the models from each dataset that we considered to perform best based on the evaluation parameters.

4.1 Data

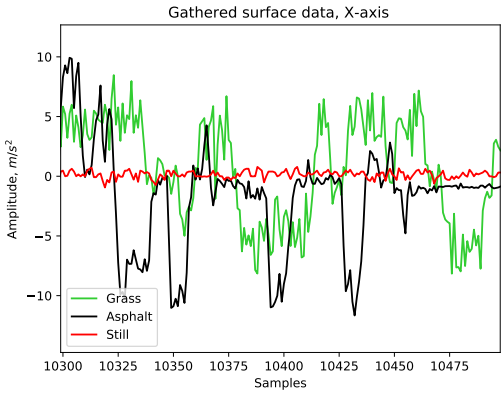
We used two sets of data during this project. The first was called the Raw dataset and contained data straight from the axes of the accelerometer. The second was the Features dataset and was created by features calculated on intervals of 11 samples of the raw data from the z-axis. A Features dataset that was build from raw data gathered on the y-axis was also examined, but the resulting models performed worse than the one trained on the z-axis feature data.

4.1.1 Collection of data

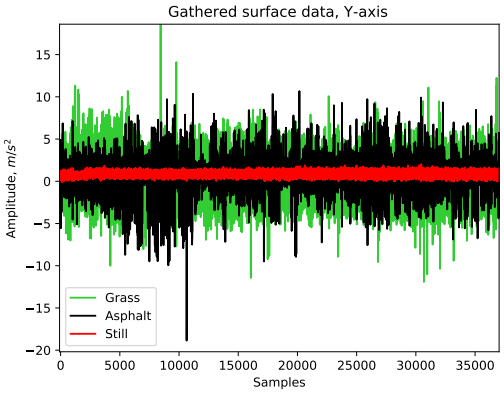
Figure 4.1 shows the data gathered on the different axes of the sensor. The amplitude varies the most between the surfaces for values measured on the z-axis. In figure 4.1a, we can see that the acceleration side-to-side was similar on both grass and asphalt. That is why, when we later tested which axis contained the most information on its own, the models trained on only the x-axis had a hard time separating the two surfaces from each other. Figure 4.1 also shows that the data from when the car stands still oscillate as well. The accelerometer had a high sensibility and therefore picked up small disturbances even when the car stood still. We considered this the variation of amplitude neglectable. Due to the sensor being affected by the earth's gravitational pull, the values gathered from the sensor's z-axis all average around 9.83, see figure 4.1. Also noticeable is that the amplitude for both the asphalt data and grass data sometimes go as far down as negative 9. An explanation for this is that the car sometimes hit something and bounced, or even flipped over, due to reckless driving.



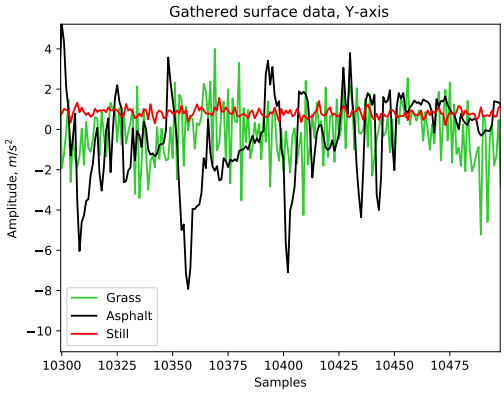
(a) Accelerometer data x-axis, asphalt and grass



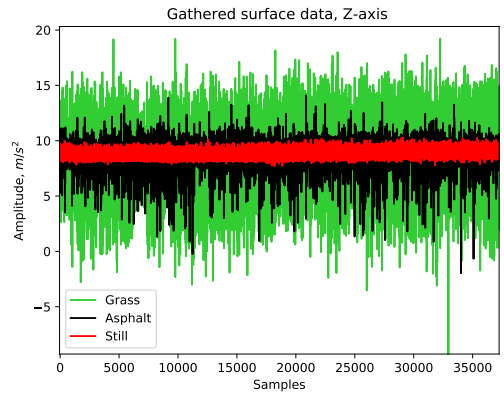
(b) Accelerometer data x-axis zoomed in, asphalt and grass



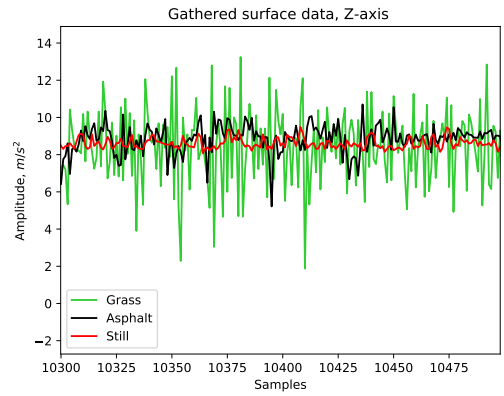
(c) Accelerometer data y-axis, asphalt and grass



(d) Accelerometer data y-axis zoomed in, asphalt and grass



(e) Accelerometer data z-axis, asphalt and grass



(f) Accelerometer data z-axis zoomed in, asphalt and grass

Figure 4.1: The figures show the values measured on the accelerometer axes on different surfaces.

4.1.2 Accelerometer data

As shown in figure 4.1 and 4.2, the Raw dataset gave rise to distributions around the same center value. That makes separating samples close to this center hard for our models since it lays within all distributions. Therefore, we chose to create a second dataset to increase the separability between the surfaces. Both datasets were used to find the optimal models for our Mixed and Hybrid systems.

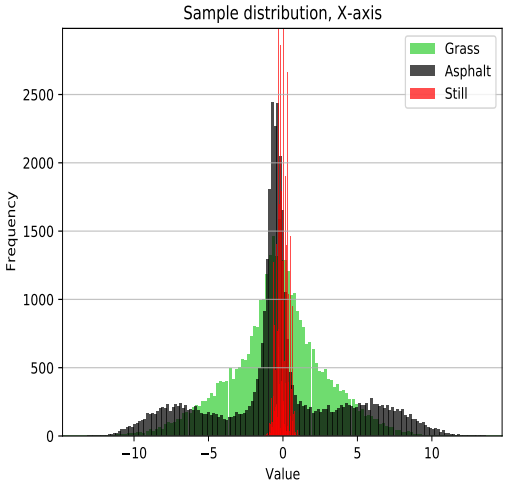
Raw dataset

The figures 4.2 shows the surface's sample distributions measured on each of the axes of the sensor. The models that performed the worst on separating the classes from each other were the ones trained on data from the z-axis. As figure 4.2c shows, the distribution of asphalt all lay within the distribution of grass. That means that samples close to 9.82 would be hard to classify since they could come from any of the surfaces. The data from the y-axis gave the models with the best results. In figure 4.2b it is shown that the distributions of the two surfaces were similar, with one difference between the two. The one for asphalt had two peaks, while the one for grass had one. The second spike can be explained by the car having better friction on asphalt so that it could accelerate with a jump start. The difference in grip is also shown in figure 4.2a. The asphalt distribution has two peaks next to the center. These indicate strong acceleration when turning, something that appeared more often on asphalt due to the wheels of the car having a better grip.

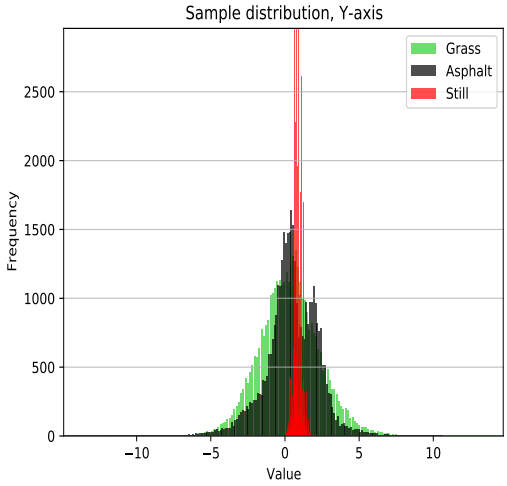
Features dataset

The idea to process the data before training came from studying the lack of separability of the distributions in figures 4.2. We found an article [11] that had a similar problem. They used an approach where they extracted information by calculating characteristics from the accelerometer data to train their models. We adopted this approach, and after testing several different features, we narrowed it down to the ones mentioned in section 3.1.2. These gave the most defined difference between the surfaces and proved to be efficient.

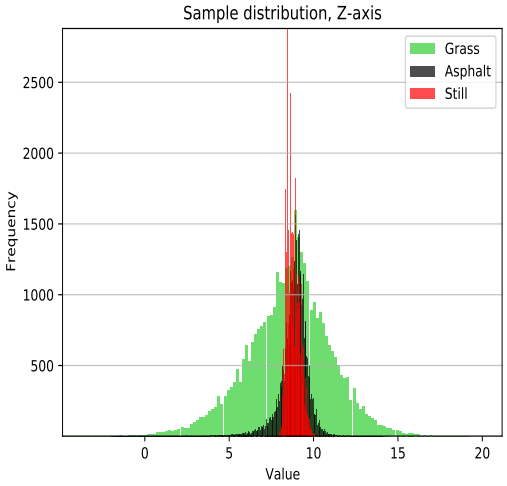
The figures 4.3 and 4.4 shows the behavior of the feature Standard Deviation calculated from both samples from the z-axis and the y-axis. In both the data plot and the histogram, the feature visibly separates the surface samples. When studying the figures, there is a clear distinction between grass and asphalt in 4.3b, showing feature from the z-axis, while the plots are more overlapping in 4.3a, feature y-axis.



(a) Distribution of values from the x-axis



(b) Distribution of values from the y-axis



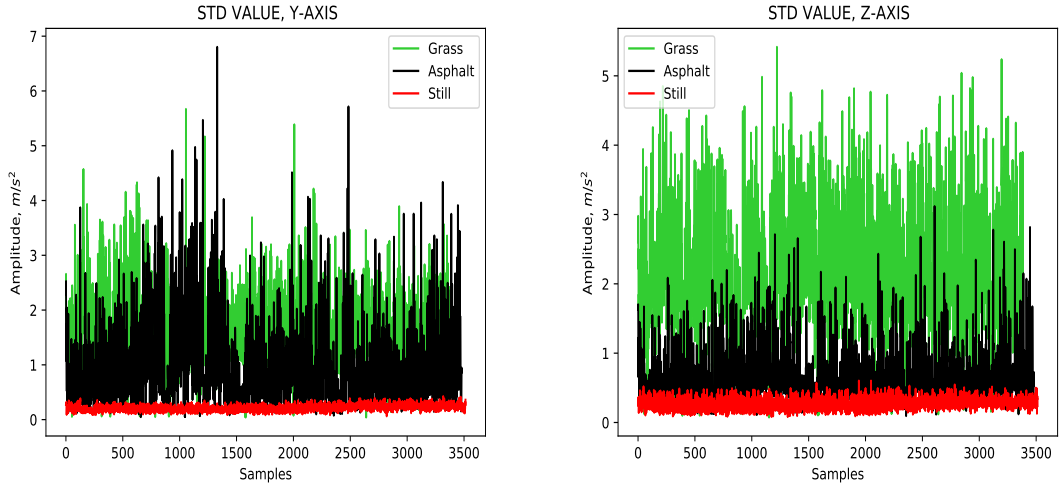
(c) Distribution of values from the z-axis

Figure 4.2: Histogram showing the sample distribution of the Raw dataset gathered on asphalt, grass, and when standing still.

4.2 Training models Raw data

The evaluation test of the axis was a way to get a better understanding of which axis of the accelerometer measured the data that contained the most information regarding the surface. In figure 4.5 it is shown that all the models trained on a single axis performed poorly in classification. The y-axis was shown to be the most stable of the three.

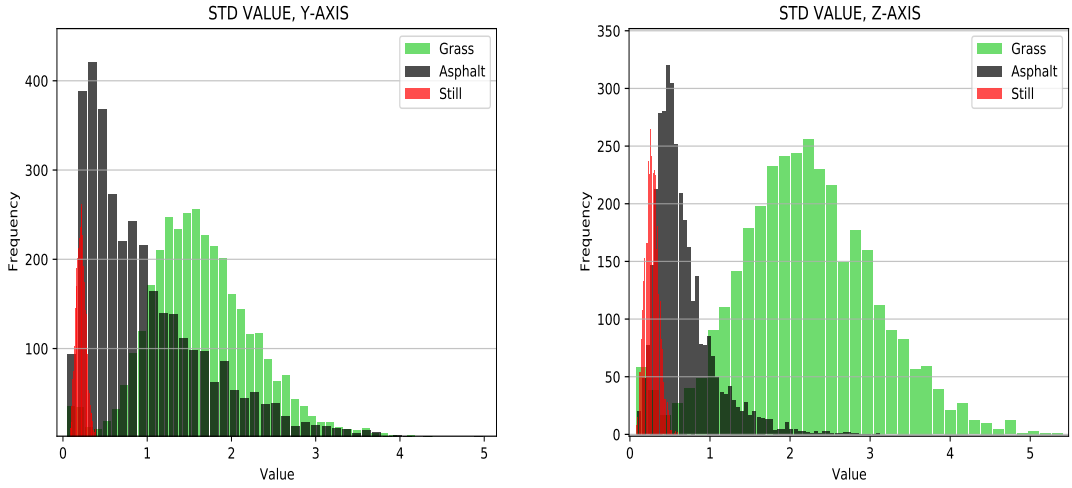
The result from testing different configurations of lags-, deltas-, and backward moving average attributes on the Raw dataset is presented in the tables B.5, B.6, and B.7 in Appendix B. The attribute arrangements that gave the highest classification score are in the tables marked with red. We used these during the exhaustive search for the layer combination that would optimize the model's performance in terms of precision. Figure 4.7 shows how the F1-score



(a) Feature: STD value, y-axis.

(b) Feature: STD value, z-axis.

Figure 4.3: Plots of samples from the Feature dataset, left side shows feature extracted from data from the y-axis, right side shows feature extracted from data from the z-axis.

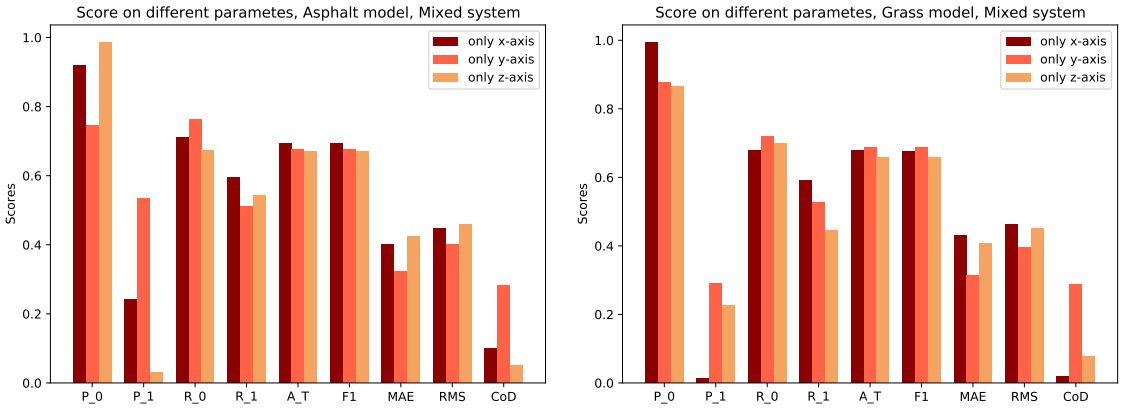


(a) Distribution of the standard deviation, y-axis

(b) Distribution of the standard deviation, z-axis

Figure 4.4: Histogram of features. Left side shows samples from y-axis, right side samples from z-axis

and the precision of each class changed when increasing the numbers of neurons in one layer. The values of these parameters converge, but the size of the model increases linearly. This pattern was recurring during the training of all the models, as is shown in table B.8, B.9, and B.10 in Appendix B. The models that we considered to be the best in terms of accuracy and memory size are marked with red. Before we implemented the models on the Ardesco, we retrained them one more time with a larger *max iterations* to see if their performance could be enhanced further. In table 4.1 the final result for these models are presented.



(a) Performance on Asphalt

(b) Performance on Grass

Figure 4.5: Bar graph of the performance of the models trained on only one of the axis at a time, Mixed system

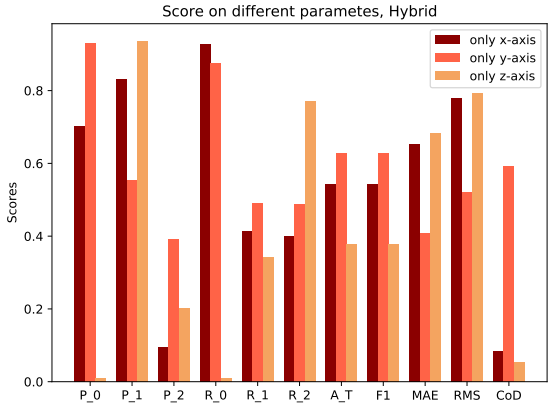


Figure 4.6: Bar graph of the performance of the models trained on only one of the axis at a time, Hybrid system

In table 4.1 and 4.3 $\boldsymbol{Mem}$ is the size of the model in kB, $\boldsymbol{P_i}$ is the precision for label i, $\boldsymbol{R_i}$ the recall for label i, $\boldsymbol{A_T}$ the total accuracy, $\boldsymbol{F1}$ the F1-score, $\boldsymbol{MAE}$ the mean average error, $\boldsymbol{RMS}$ the root mean square error and $\boldsymbol{CoD}$ the coefficient of determination. The values are presented in a normalized state, with the exception of $\boldsymbol{MAE}$ and $\boldsymbol{RMS}$ which shows the error between the real values and the predicted.

Figure 4.8 and 4.9 shows the correlation between the real labels and the predicted for a sample of the test set. Table 4.1 revealed that the Hybrid system had a higher CoD value than the Mixed system, which is visible in the figures. The prediction of label zero seems to be difficult for the Mixed system.

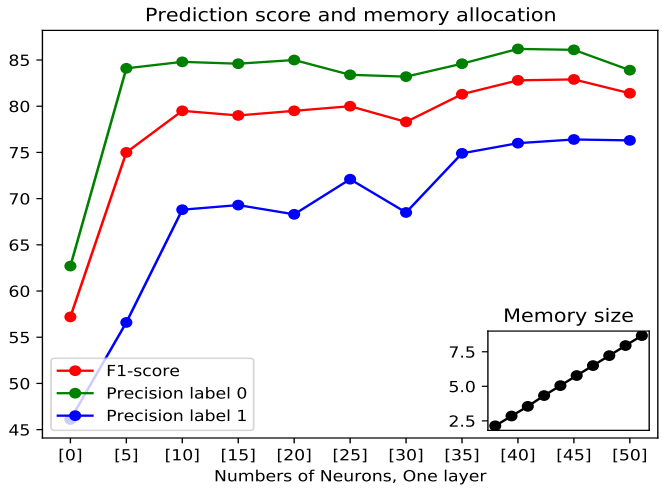


Figure 4.7: Performance and memory size for asphalt model with one layer

Best Model	HL	Mem	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F_1	MAE	RMSE	CoD
Asphalt	[20,20]	6.01	94.8	87.8	-	94.0	89.3	-	92.4	92.5	0.105	0.198	0.825
Grass	[35,20]	8.80	96.0	85.0	-	93.0	91.0	-	92.3	92.4	0.072	0.179	0.854
Mixed System	-	14.81	97.8	86.0	85.9	87.2	91.2	92.0	89.9	89.9	0.116	0.274	0.887
Hybrid System	[50,35]	15.56	96.0	84.8	88.3	95.8	86.7	86.5	89.7	89.7	0.096	0.210	0.934

Table 4.1: Table of the performance of the two systems trained on the Raw dataset. These were later implemented on the Ardesco. The Mixed System is the combination of the Asphalt and Grass models.

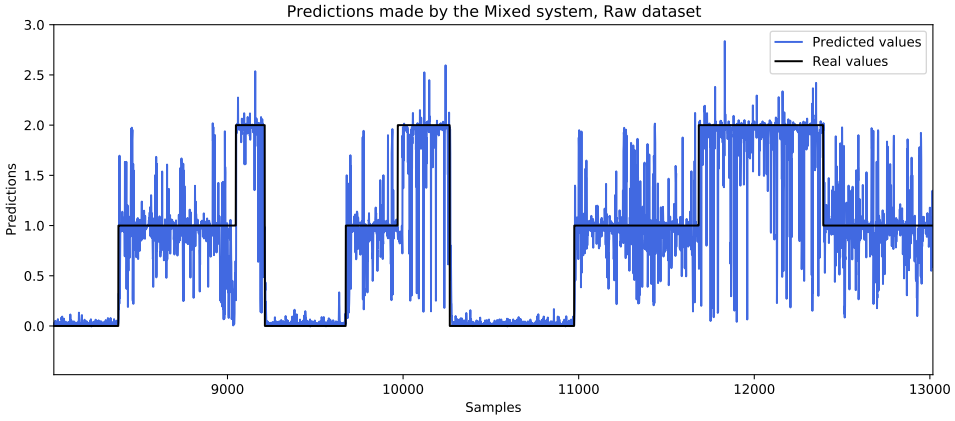


Figure 4.8: Correlation between predictions and real values, Mixed system, Raw dataset

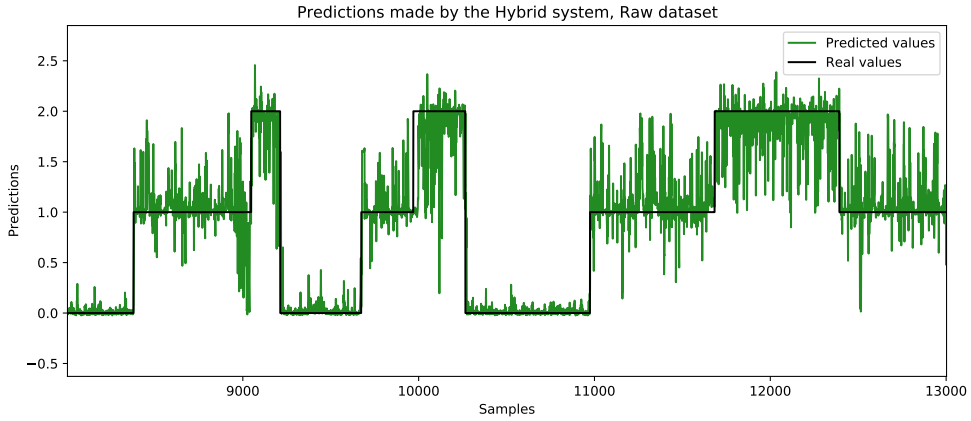


Figure 4.9: Correlation between predictions and real values, Hybrid system, Raw dataset

4.3 Training models Features data

After studying the plots and the sample distributions shown in figures 4.4, 4.3, and Appendix A, we concluded that features extracted from the z-axis divided the surfaces better than the ones from the y-axis. We confirmed this by conducting the same attribute test on both the Features datasets for the Hybrid system. In table 4.2, the models that performed the best from this test is shown. The full result of the test is presented in tables **??** and B.14 in Appendix B.

Best Model	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F_1	MAE	RMSE	CoD
y-axis	100.0	85.1	90.3	94.2	95.4	81.9	91.7	92.1	0.152	0.243	0.914
z-axis	99.1	91.8	95.3	95.8	95.0	96.9	95.4	95.9	0.095	0.178	0.951

Table 4.2: The two best performing models of the Hybrid system during the attribute test, Features dataset from the y- and z-axis.

We used the Features dataset based on data from the z-axis during our exhaustive search. The models that we found which had the best performance are presented in table 4.3. These models were also retrained before implemented on the Ardesco. In figure 4.10 and 4.10 the correlation between predictions by the Mixed and Hybrid systems and the real values are plotted.

Best Model	HL	Mem	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F_1	MAE	RMSE	CoD
Asphalt	[10]	2.41	97.7	93.6	-	93.4	97.8	-	95.1	95.6	0.086	0.164	0.869
Grass	[5]	1.86	99.0	98.6	-	99.4	97.8	-	98.5	98.9	0.028	0.113	0.942
Mixed System	-	3.27	98.7	96.6	97.0	96.4	96.6	98.8	97.4	97.4	0.077	0.204	0.936
Hybrid System	[25]	4.44	98.7	97.1	95.1	97.8	93.4	98.9	96.4	96.8	0.094	0.182	0.949

Table 4.3: Table of the performance of the two systems trained on the Features dataset from the z-axis. These were later implemented on the Ardesco. The Mixed System is the combination of the Asphalt and Grass models.

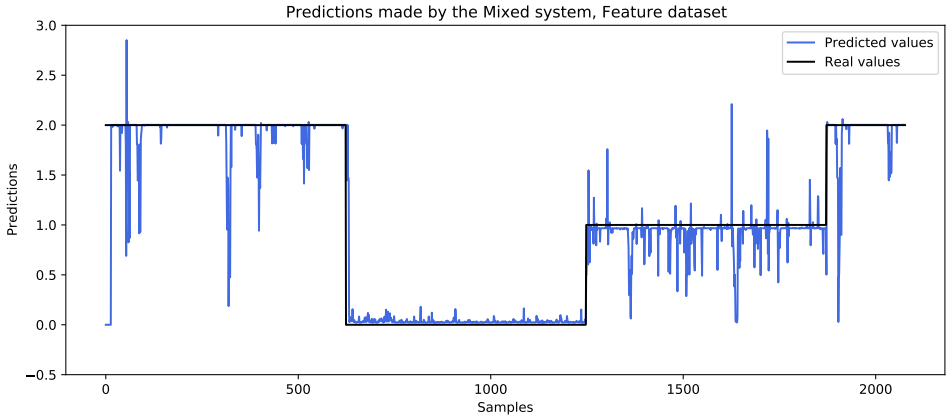


Figure 4.10: Correlation between predictions and real values, Mixed system, Features dataset

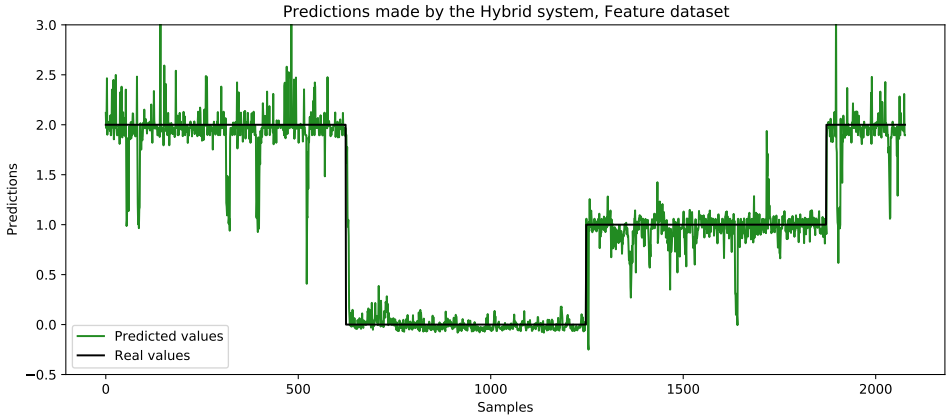


Figure 4.11: Correlation between predictions and real values, Hybrid system, Features dataset

4.4 Energy Consumption

The systems presented in table 4.1 and 4.3 was in turns implemented on the Ardesco, where we measured the energy needed to run them. We did this for both the Mixed and Hybrid systems trained on the Raw and Features dataset, measuring the energy consumption in Wh/min. The result is presented in table 4.4.

Energy Consumption, Measured in $\mu Wh/min$		
	Mixed System	Hybrid System
Raw dataset	343.5	342.5
Features dataset	340.5	343

Table 4.4: The energy consumption for each system implemented on the Ardesco.

Chapter 5
Discussion

In this chapter, we will explain why this thesis is highly relevant to our host company. We also discuss why the y-axis performed surprisingly well, commenting on the attributes and hidden layers. Furthermore, we review the thresholds and patterns in the dataset affecting the predictions. We observe external factors for the system. Finally, we look at the ethical perspectives of the project.

5.1 Collecting data

To create an end to end solution centering the Ardesco, we gathered the data that we used to train the models using an accelerometer sensor embedded in its hardware, inferring machine learning algorithms. The idea to identify surfaces was something that our host company wanted to investigate, as they saw the potential for a use case in the vehicle industry. We chose to measure the vibrations from different surfaces using a radio car since our host company had successfully tried that before. Also, if we could train the Ardesco to separate vibrations created by the radio car, which was light and had good shock absorption, it would most likely work for other cases too.

5.2 Accelerometer data

Our hypothesis that only using the values from the sensor z-axis would work proved to be wrong. Figure 4.2 showed us that there is some correlation between the surface and the other axis values. The width of the histogram bars differed between the classes, indicating that the samples from asphalt, black in figure 4.2, were more concentrated around its mean value, while the ones from grass, green in figure 4.2, was more spread out.

To find out which axis contained the most information, we conducted the test described in . That showed us that models trained on only the y-axis performed surprisingly well. The

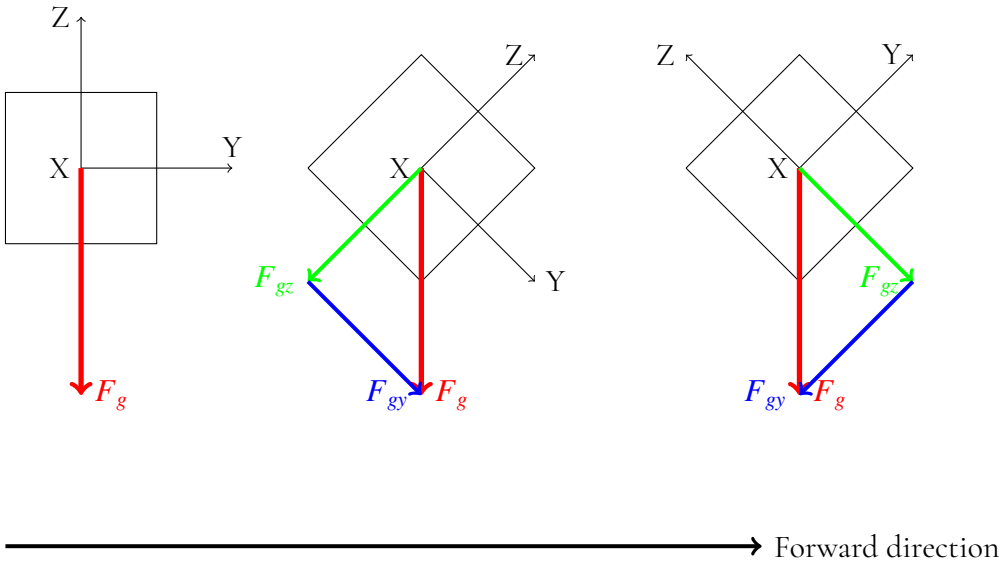


Figure 5.1: The figure illustrates the axis of the sensor if the car tilted. The car was here represented by a square. $\boldsymbol{F}_g$ is the gravitation and $\boldsymbol{F}_{gz}$ and $\boldsymbol{F}_{yx}$ the forces divided over the sensors axis. The division could be done with acceleration forward.

y-axis of the sensor was faced in the direction the car was pointing, meaning that if the car was to hit a bump, i.e. vibrate, the axis would momentarily point in another direction, see illustration in figure 5.1. Since the grass surface was more uneven than asphalt, the values measured by the y-axis were more diversified.

The histograms in 4.2 show that the samples gathered from different surfaces had different distributions, indicating some separability between them. The results differed when we measured the prediction accuracy of using only one axis at the time, as can be seen in figure 4.5a, 4.5b, and 4.6. The models trained on data from the z-axis proved to perform poorly when trying to predict asphalt. The values measured all fell within the distribution of the ones collected on the grass surface, which explains why the prediction rate was low.

Another hypothesis why models trained on data from the y-axis performed better than models trained on the other axes is the difference in the car grip between the surfaces, as mentioned in *Raw dataset* in section 4.1.2. The friction between the wheels and the ground was higher on asphalt compared to grass and would allow the car to accelerate faster. Since the y-axis measured the forward and backward acceleration, this axis would contain information regarding this.

5.3 Frequency domain

After reading other studies [5] [17] that successfully trained machine learning models with accelerometer data processed in the frequency domain, we tried to analyze our datasets with the same approach. We transformed our data with a Fast Fourier Transform to see if a particular frequency stood out in the frequency domain of our dataset. Even though the accelerometer data in the time domain shows an oscillating signal, see 4.1d, we got no useful

output after transforming to the frequency domain. The spectra showed no distinct recurrence that we could use as information about the surface the car was driving on. Since we drove the car on a random route on an inhomogeneous surface, the vibrations contained too many frequencies to be able to see any patterns. After this observation, we limited our project to only work in the time domain.

5.4 Training

The result improved when we added lag-, delta-, and backward moving average attributes to the models. Even just looking two or three steps back lifted the F1-score with a few percentage points. There were no limitations on how many lags, deltas, and backward moving average we could use, resulting in an infinite amount of different configurations. We did not have the time to test all the combinations, so we set up some constraints. When adding these attributes, we noticed that the complexity and size of the networks increased. Therefore, when working with raw data, we chose not to let the number of samples in the pipeline exceed 15. That meant that the maximum number of inputs would be:

$$15 \times 6\ lags\ and\ deltas + 3\ BMA + 3\ axis\ values = 96\ inputs$$

For the models trained on the Features dataset, the constraints on the attributes were set even harsher. Here, we did not exceed more than ten samples in the pipeline. We chose to do so since we calculated one value of the Features dataset from raw data gathered during one second, meaning that it would take ten seconds to fill the pipeline with ten feature values. Adding more attributes would increase the time even further, giving us a model with high latency.

The result from the test of the configurations of attributes, shown in table B.5, B.6, and B.7. showed that, when training on the Raw dataset, a lag and delta of five on each axis combined with a backward moving average of 15 gave the best overall score of the models in the Mixed system. The Hybrid system performed best on the same lag and deltas as the Mixed, but instead of 15 on BMA, it used 10. The length of the BMA seemed to be important on both the Raw and Features dataset. The attribute test conducted on the Features dataset, shown in table B.12, B.13, and B.15, showed that a BMA of 10 gave the best performances. Also, the asphalt and grass models used in the Mixed system gave the best result when we trained them without any deltas. That indicates less correlation between each sample.

Continuing to increasing the number of adding attributes did not increase the classification accuracy, and if we used too many, it would start to go down. As mentioned before, these look-backs in the pipeline were to help the model recognize a switch of surfaces and to be able to find patterns in how the data changed over time. For these tests, the F1-score, precision of label 1 (and label 2 for hybrid), and CoD were the decisive parameters.

During experimenting with the numbers of neurons of each layer, we noticed that the performance on the test data increased as we added neurons even after the rule-of-thumb recommendations. As overfitting usually is shown by a good result on the training set but with large errors in the test set, we did not think that the extra neurons led to an overfitted model. There was some correlation between the numbers of neurons and the coefficient of determination (CoD), as they increased collectively to a point. That indicated that the model's predictions followed the real values well, which in turn could mean that the model

was overfitted, but since the performance on the test set did continue to rise, we deemed that this was not the case. We still had the rule-of-thumb in mind as we selected the best models, but as long as the CoD and the performance on the test set correlated, we felt safe continuing to increase the numbers of neurons.

When training the models on the Features dataset, we noticed that using more than one layer did not increase the performance. As seen when we plotted the features in histogram 4.4 and A.2, the distributions are almost divided from each other. If there had been only two labels, say 1 and 2, these could have been separated by a linear model drawing a line between the distributions. Since we had three, adding one layer was enough to obtain a model with good performance.

5.5 Correlation prediction

One way to get a better correlation between prediction and real value could have been to have a dataset with longer intervals of one kind of data type, meaning the car was driving on the same surface for a long time before a switch to a new one. We created our datasets to simulate several intervals of varying lengths and random switches of surfaces. We tried training with a dataset that had a more uniform design, i.e. the size of each interval was the same. The issue was that the model learned the length of these intervals and anticipated a switch of the surface, making the correlation between prediction and real value better but creating a model based on an unlikely reality since, in our case, we drive the car randomly.

5.6 Mixed vs Hybrid system

The Mixed and Hybrid systems both performed well based on the evaluation parameters. The main idea of creating the Hybrid system was that we thought this method would be smaller in memory. It turned out that a network that could identify two different surfaces needed to be more complex to reach the same classification accuracy as the Mixed system. That led to the one model used in the Hybrid system to be larger than the two used in the Mixed system combined.

Before implementing the models on the Ardesco, we examined the thresholds that set a prediction to a label. As the Mixed system on Raw data had larger error gaps between predicted and real value and had a hard time distinguish standing still from driving on asphalt, the threshold for what was labeled 0 and 1 was the biggest issue. In all other cases, the models obtained the best results when we divided the precision spectrum evenly, meaning we set everything below 0.66 to 0, between 0.66 and 1.33 to 1, and above 1.33 to 2.

After concluding which thresholds gave the optimal performance, we implemented and ran the systems on the Ardesco to measure the energy consumption. The hypothesis was that the calculations for extracting features from the accelerometer data would increase consumption. We also thought that the Mixed system that ran two models would need more power. None of our theories came true, as is shown in table 4.4. Neither the system nor what data we used had any effect on the usage of energy. Worth mentioning is that the approach we used to measure the consumption was only valid when the Ardesco ran continually. If we wanted to save battery, we could implement a sleep function after a predetermined number

of classifications. By doing so, we could argue that the models trained on the Raw dataset would consume less energy since they have faster latency. Also, if we extended the numbers of classes, e.g, adding a model to the Mixed system, the time it would take for the Ardesco to load these might prove to affect the consumption.

5.7 Different car and external factors

One of the drives to collect accelerometer data on asphalt was conducted when it had been raining, making the surface wet. The data that was gathered was then added to the asphalt dataset and plotted together with the rest. We then noticed a divergence in the behavior. As can be seen in 5.3, the data gathered on the wet surface (blue) had somewhat weaker vibrations than the dry asphalt (black). Unfortunately, there was not enough time to investigate to what extent the effect of this had on the performance of the models. But this added weight to our theory regarding how the friction was a parameter used during classification.

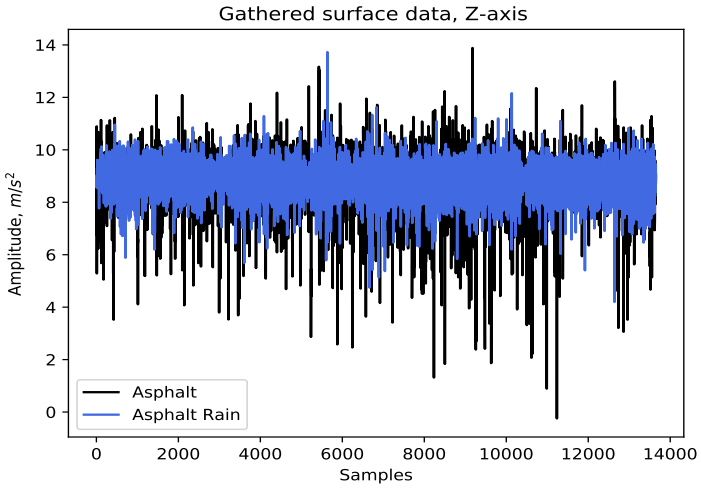


Figure 5.2: The effect on the vibrations on the z-axis of wet asphalt

We wanted to investigate how dependent the accelerometer data was on the car used to gather the samples, as this would be relevant for the future use of our models. We questioned if the Mixed and Hybrid systems we created could be used for surface detection on other cars or if they were limited to the original radio car that gathered the data. To test this, we drove with a different car model [13] and collected data on both surfaces. This car had four-wheel drive and did not have the same dampening system as the first one. The original was back-wheel driven and larger wheels. We used the new dataset as a test set for our models and got the results presented in table 5.1.

The systems both seemed to have problems with classifying vibrations created on an asphalt surface. When this gathering of data occurred, it had been humid outside, making the surface slippery, as shown in figure 5.3. That might have affected the system's precision. Regardless, compared to the results for the original car, shown in table 4.11, 4.9, 4.10 and 4.8, all systems performed worse on the alternative car. The results prove that the nature of the accelerometer data depended on the model of the radio car. That the models performed better

System	Dataset	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F_1	MAE	RMSE	CoD
Mixed	Raw	93.1	68.3	94.1	91.2	74.3	93.4	89.8	89.8	0.289	0.507	0.698
Hybrid	Raw	91.7	71.9	78.3	94.2	72.9	74.8	80.4	80.5	0.243	0.419	0.744
Mixed	Features	97.9	67.4	83.4	86.1	85.0	78.0	82.9	82.9	0.195	0.404	0.755
Hybrid	Features	100	77.2	83.0	94.5	85.0	80.3	85.7	86.7	0.201	0.387	0.789

Table 5.1: Table of the performance of the two systems on Raw (red) dataset and Features(yellow) dataset, gathered with the alternative car.

on the same data that they had been trained on was no surprise. However, the difference in classification accuracy between the two kinds of data was larger than we expected. Retraining the systems and including data from the new car in the dataset could lead to better results.

Figure 5.3: The alternative car with Ardesco on the trunk.

5.8 Ethical perspective

The whole idea of machine learning is to teach computers to find patterns in the data and draw conclusions from these. To be able to do so, there needs to be a sufficient amount of data for the model to learn. That is why data right now is such a valuable resource and why personal data is so desirable for companies. With that, they can map behaviors, pinpoint interests, and personalize commercials. The demand for this information makes people more aware and protective of what digital trace they leave behind, which leads us to the ethical perspective of this thesis. Maybe mapping what surfaces an individual is driving on is a breach of their freedom. To make the model even better, one could argue on the usage of GPS data that would make it possible to compare the prediction with the surface that is there on the

map. Would that information be worth sharing for the possibility to foresee maintenance? Will it be possible to live in a society with 18 billion connected devices without sharing your personal data information? We have to wait and see.

One thing that speaks for the usage of machine learning on the edge is that it is more secure than cloud computing. Data is at its most exposed state when it is being transmitted. When streaming business-sensitive or personal information over the internet to be processed in the cloud, this information could be a target for hackers. By moving the data processing to the edge, it will be easier to control and decide what is being sent.

Chapter 6
Conclusion

We have in this project investigated the performance of regression models implemented on a microcontroller. The goal was to train and run models that could classify surfaces based on the vibrations measured by an accelerometer. Two different datasets have been used in two systems. The first dataset consisted of raw accelerometer data, and the second contained features calculated from short intervals of raw data. These were used to train three different regression models on each dataset. Two models could only classify one surface each. These were later combined and referred to as the Mixed system. The last model could classify both. This model was referred to as the Hybrid system. These two systems were compared based on accuracy, memory allocation, and energy consumption.

6.1 Research questions

RQ1: What classification accuracy we can achieve with regression-based models on a microcontroller?

The highest classification accuracy we attained with any of the models was with the grass model trained on the Features dataset. This model got an F1-score of 98.9 %. The highest classification accuracy we obtained by a system was the Mixed system based on the same dataset, which reached an F1-score of 97.4 %.

RQ2: Will a system with two regression models, one for each surface, reach a higher classification accuracy than a system that uses one regression model? If so, how would this affect the size of the system and its energy consumption?

The two systems obtained similar classification accuracy and drained the same amount of energy when running on the Ardesco. What separated the two from each other was the size of the systems. The Hybrid system that we thought was going to require less memory space needed more than the Mixed one.

RQ3: Can we create a surface detection system with higher classification accuracy by applying preprocessing to our data? If so, how will this affect memory allocation and energy consumption?

By applying preprocessing to the accelerometer data, we manage to extract features that separated the samples from the surfaces. By doing so, our systems could reach a high classification accuracy while requiring less complexity to classify each label. The Mixed and Hybrid systems performed well, allocated less memory, and used the same amount of energy as the ones trained on the Raw dataset while being used on the edge device.

6.2 Future work

If we were to continue this research, it would be interesting to expand on the number of surfaces that the systems could classify. We experimented with adding a third surface but lacked time to gather enough data and train additional models. Increasing the number of classes would put more restrictions on the memory size of the models. We would have had to set harsher limits when training the models. It is also likely that the memory size of the Mixed and Hybrid system would scale differently, making the Hybrid system smaller when adding several surface classes. It would also be fascinating to test how using another sampling frequency would affect the systems. There was not enough time to restart gathering data using a higher or lower sample frequency and then retrain the models. But since the sampling frequency used in the robot experiment, see section 1.4, was 125Hz, this could be of high relevance to the results. A 10x faster sample rate could generate more useful information about how the car moves on a surface. The drawbacks of a higher sampling frequency are the increment of energy consumption and memory allocation, which is critical when running on the edge. We also discussed testing to add more decimals to the accelerometer data. The sensor we used had high resistance to noise, and we could probably experiment with using five or six decimal values instead of three.

As mentioned in section 3.1.1, there are several aspects during the gathering of data that could affect the performance of the systems. We could elaborate on many of these by extending the size of the dataset that we used. By gathering data from more diverse locations, driving uphill and downhill, and on dry and wet surfaces, we could make our systems more versatile.

Bibliography

[1] *Arm Cortex-M33*. Product Information. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m33.

[2] *Connectivity Meaning In The Cambridge English Dictionary*. Dictionary. Camebridge University Press, 2020-07-30. http://dictionary.cambridge.org/dictionary/english/occupation.

[3] P. Agostino Accardo and S. Pensiero. *Neural network-based system for early keratoconus detection from corneal topography*. Research Article. Journal of Biomedical Informatics 35 (2003) 151–159, December 2001. https://www.sciencedirect.com/science/article/pii/S1532046402005130.

[4] A. Balasch, M. Beinhofer, and G. Zauner. *The Relative Confusion Matrix, a Tool to Assess Classifiablility in Large Scale Picking Applications*. Research Article. 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020. http://www.lewissoft.com/pdf/ICRA2020/0204.pdf.

[5] E. P. Carden and P. Fanning. *Vibration Based Condition Monitoring: A Review*. Research Article. Sage Publications, December 2004. https://www.researchgate.net/publication/245381989_Vibration_Based_Condition_Monitoring_A_Review.

[6] T. Chai and R. R. Draxler. *Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature*, volume 7. Research Article. Geoscientific Model Development, 06 2014. https://www.researchgate.net/publication/272024186_Root_mean_square_error_RMSE_or_mean_absolute_error_MAE-_Arguments_against_avoiding_RMSE_in_the_literature.

[7] Analog Devices. *Data Sheet ADXL362*. Product Information. https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf.

[8] Ekkono. *The Product*. Product Information. https://ekkono.ai/the-product/.

[9] Ericsson. *How to quickly develop cellular IoT devices*. Product Information Ardesco. https://www.ericsson.com/en/blog/2020/10/iot-device-development.

[10] M. Gharib and A. Bondavalli. *On the Evaluation Measures for Machine Learning Algorithms for Safety-Critical Systems*. Research Article. 2019 15th European Dependable Computing Conference (EDCC), 2019. https://ieeexplore.ieee.org/document/8893310.

[11] H. Gjoreski and M. Gams. *Accelerometer data preparation for activity recognition*. Conference paper. International Multiconference Information Society – IS, 2011. https://www.researchgate.net/publication/259340203_ACCELEROMETER_DATA_PREPARATION_FOR_ACTIVITY_RECOGNITION.

[12] S. Haykin. *Neural Networks: A Comprehensive Foundation, 2nd Edition, chapter 4*. Book. Pearson Education, 2005.

[13] Remo hobby. *Overview SMAX*. Product Information. http://www.remohobby.com/product/show/1401.html.

[14] McKinsey Global Institute. *The internet of things: Mapping the value behind the hype*. Executive Summary. McKinsey & Company, June 2015. https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world.

[15] J. Jordan. *Evaluating a machine learning model*. Research Blog Post. Jeremy Jordan, 2017-07-21. https://www.jeremyjordan.me/evaluating-a-machine-learning-model/.

[16] V. Kardahri. *How to evaluate regression models?* Research Blog Post. Acing AI. ://medium.com/acing-ai/how-to-evaluate-regression-models-d183b4f5853d.

[17] A. Mannini and A. M. Sabatini. *Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers*. Research Article. MDPI Journals- Sensors, February 2010. https://www.researchgate.net/publication/51970052_Machine_Learning_Methods_for_Classifying_Human_Physical_Activity_from_On-Body_Accelerometers.

[18] M.G. Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. *Machine Learning at the Network Edge: A Survey*. Research Article. Cornell University, 2020-01-29. https://www.researchgate.net/publication/334866807_Machine_Learning_at_the_Network_Edge_A_Survey.

[19] Ericsson P. Wendi, External Communications. *Ericsson Mobility Report: 5G subscriptions to reach half a billion in 2022*. Technical Report. Ericsson, 2016-11-15. https://www.ericsson.com/49e095/assets/content/76c8b70e7a1d4654a5436dbf10cd07b6/2016-11-16-emr-en.pdf.

[20] G. Panchal, A. Ganatra, Y. P. Kosta, and Devyani Panchal. *Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers*. Research Article. International Journal of Computer Theory and Engineering, Vol. 3, No. 2, April 2011. http://www.ijcte.org/papers/328-L318.pdf.

[21] Qoitech. *Otii ARC*. Product Information. https://www.qoitech.com/otii/.

[22] S. R. Rizvi. *Microcontroller programming : an introduction, First edition s. 90*. Book. CRC Press Taylor Francis Group, 2012.

[23] M. Rocha, P. Cortez, and J. Neves. *Evolution of Neural Networks for Classification and Regression*. Research Article. Universidade do Minho, October 2007. https://www.sciencedirect.com/science/article/abs/pii/S0925231207001567#aep-article-footnote-id9.

[24] Nordic Semiconductor. *nRF9160*. Product Information. https://infocenter.nordicsemi.com/index.jsp.

[25] C.E. Shannon. *A matchematical Theory of Communication*. Research Article. Nokia Bell Labs, 1949. https://ieeexplore.ieee.org/abstract/document/6773024.

[26] D.S Touretzky and D.A Pomerleau. *What's hidden in the hidden layers?* Research Article. BYTE Magazine, 1989. https://www.cs.cmu.edu/afs/cs/user/dst/www/pubs/byte-hiddenlayer-1989.pdf.

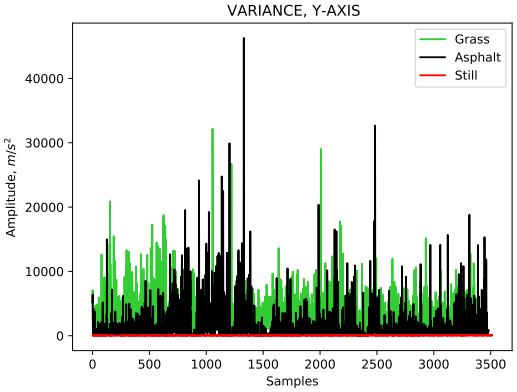
[27] D. Vail and M. Veloso. *Learning from accelerometer data on legged robot*. Research Article. Carnegie Mellon University, July, 2004. https://www.researchgate.net/publication/228531625_Learning_from_accelerometer_data_on_a_legged_robot.

[28] M.T. Yazici, S. Basurra, and M. M. Gaber. *Edge Machine Learning: Enabling Smart Internet of Things Applications*. MDPI Journals- Big Data and Cognitive Computing, 2018-09-03. https://www.mdpi.com/2504-2289/2/3/26.

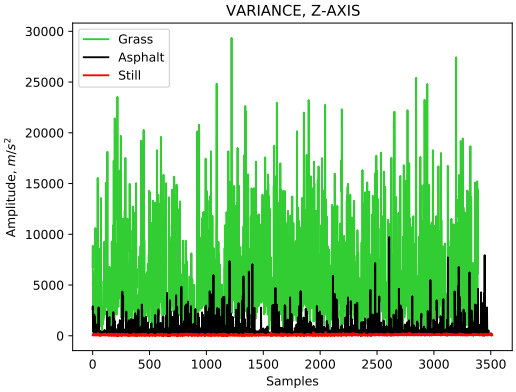
[29] Zephyr. *Zephyr 2.4.0*. Product Information. https://docs.zephyrproject.org/latest/releases/release-notes-2.4.html.

Appendices

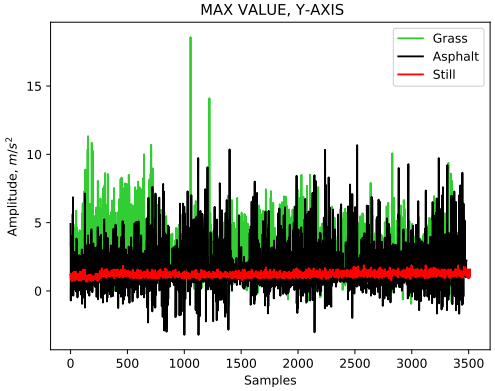
Appendix A
Feature plots



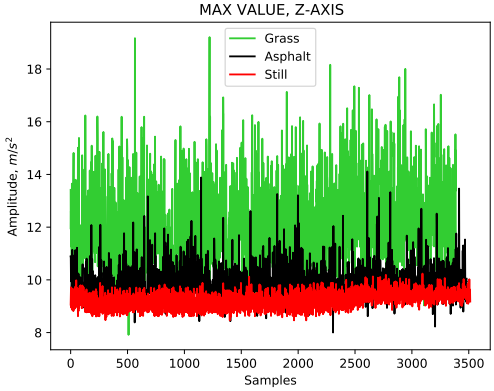
(a) Feature: Variance, y-axis



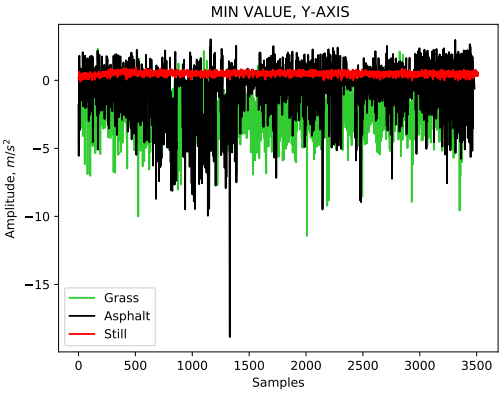
(b) Feature: Variance, z-axis.



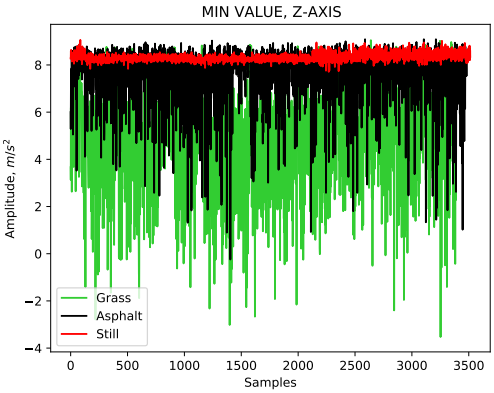
(c) Feature: Maximum value, y-axisfig:YMAX



(d) Feature: Maximum value, z-axis.

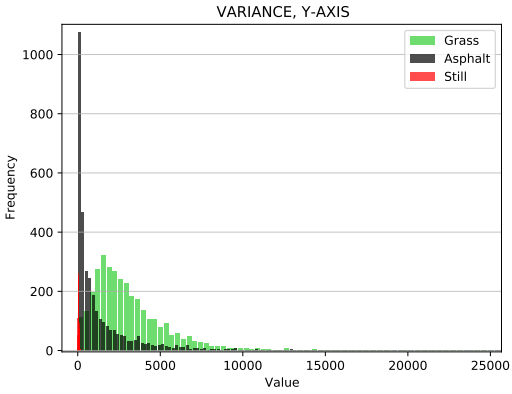


(e) Feature: Minimum value, y-axis

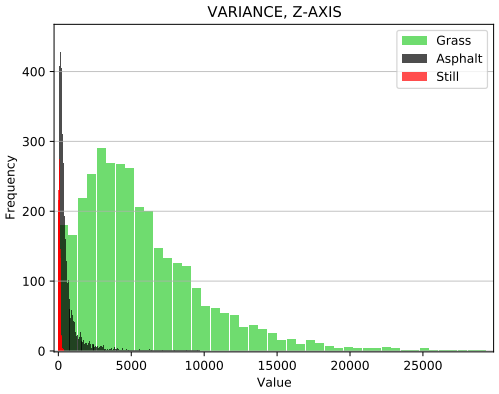


(f) Feature: Minimum value, z-axis.

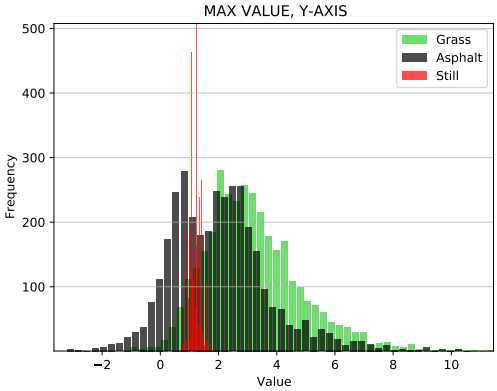
Figure A.1: Plots of samples from the Feature dataset, left side shows feature extracted from data from the y-axis, right side shows feature extracted from data from the z-axis.



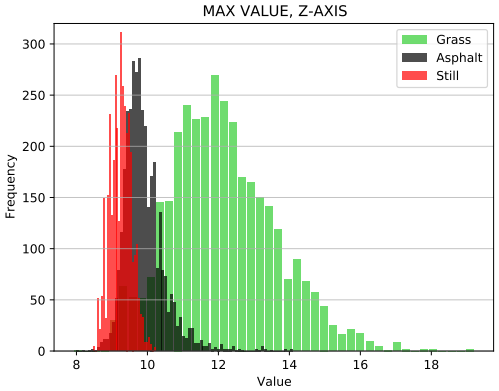
(a) Distribution of the variance, y-axis



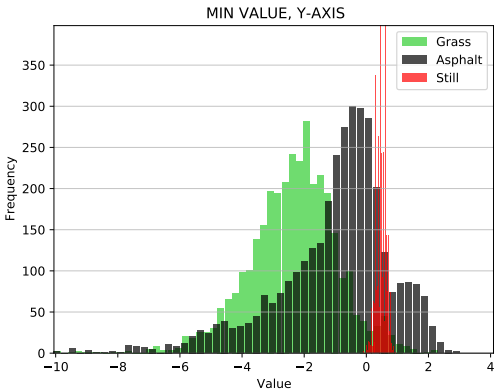
(b) Distribution of the variance, z-axis



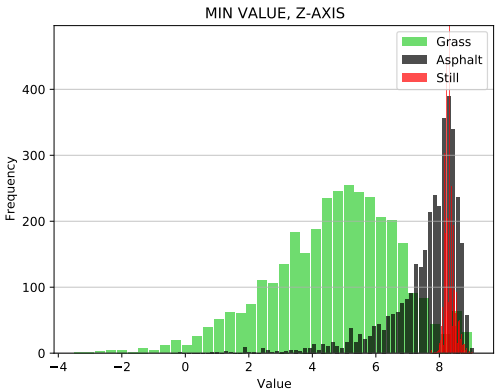
(c) Distribution of the maximum values, y-axis



(d) Distribution of the maximum values, z-axis



(e) Distribution of the minimum values, y-axis



(f) Distribution of the minimum values, z-axis

Figure A.2: Histogram of features. Left side shows samples from y-axis, right side samples from z-axis

Appendix B
Tables

This appendix presenting the results for all simulations on raw data and features data.
The evaluation parameters will in this appendix be denoted as following :
HL - Hidden layer configuration, [first layer, second layer],
P_0 - %, Precision for label zero
P_1 - %, Precision for label one
P_2 - %, Precision for label two
R_0 - %, Recall for label zero
R_1 - %, Recall for label one
R_2 - %, Recall for label two
A_T - %, Accuracy in total
$F1$ - %, F1-score
MAE - Mean Absolute Error
RMS - Root Mean Square Error
CoD - Coefficient of determination
Mem - Memory allocation of model, kB

B.1 Raw Dataset

The tables are presented in the order that they were created; importance of axes, test of attributes, and layers construction. The models that we considered to be the best are marked in red.

B.1.1 Axis test

Mixed System

Asphalt Model

Axis	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD	Ranking
X	91.8	24.2	71.0	59.4	69.4	69.4	0.402	0.448	0.099	2
Y	74.5	53.5	76.4	51.0	67.5	67.5	0.324	0.400	0.281	1
Z	98.7	3.1	67.3	54.4	67.0	67.0	0.423	0.460	0.052	3

Table B.1: Test different axes on the Raw dataset, Asphalt

Grass Model

Axis	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD	Ranking
X	99.5	1.4	67.8	59.0	67.8	67.7	0.431	0.464	0.020	3
Y	87.6	29.0	72.0	52.7	68.6	68.6	0.314	0.396	0.287	1
Z	86.4	22.7	70.0	44.5	65.8	65.8	0.406	0.450	0.078	2

Table B.2: Test different axes on the Raw dataset, Grass

Hybrid System

Hybrid Model

Axis	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F1	MAE	RMS	CoD	Ranking
X	70.3	83.1	9.6	92.8	41.4	40.1	54.3	54.3	0.652	0.779	0.085	2
Y	93.1	55.4	39.3	87.4	49.0	48.9	62.7	62.7	0.409	0.520	0.593	1
Z	0.0	93.7	20.3	0.0	34.2	77.2	37.9	37.9	0.682	0.792	0.054	3

Table B.3: Test different axes on the Raw dataset, Hybrid

B.1.2 Attribute test

Attribute Test, Raw Dataset

TEST#	HL	Lag X	Del X	BMA X	Lag Y	Del Y	BMA Y	Lag Z	Del Z	BMA Z
1	[12]	1-2	1-2	3	1-2	1-2	3	1-2	1-2	3
2	[20]	1-4	1-4	5	1-4	1-4	5	1-4	1-4	5
3	[22]	1-5	1-5	10	1-5	1-5	10	1-5	1-5	10
4	[22]	1-5	1-5	15	1-5	1-5	15	1-5	1-5	15
5	[12]	1-5	0	10	1-5	0	10	1-5	0	10
6	[12]	0	1-5	10	0	1-5	10	0	1-5	10
7	[44]	1-10	1-10	10	1-10	1-10	10	1-10	1-10	10
8	[62]	1-15	1-15	15	1-15	1-15	15	1-15	1-15	15
9	[44]	1-10	1-10	10	1-15	1-15	15	1-5	1-5	10
10	[44]	1-5	1-5	10	1-15	1-15	15	10	1-10	1-10

Table B.4: The test for finding the best configurations for attributes on Raw dataset. Lag represent a lag attribute, Del represent a delta attribute, and BMA represents a backward moving average attribute.

Mixed System

Asphalt Model

TEST#	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD
1	81.8	61.3	81.0	62.5	75.0	75.0	0.302	0.382	0.347
2	82.6	69.1	84.3	66.3	78.1	78.1	0.271	0.355	0.435
3	84.4	72.1	85.9	69.7	80.3	80.4	0.253	0.338	0.487
4	84.9	72.7	86.2	70.4	80.8	80.8	0.246	0.330	0.511
5	84.0	66.5	83.5	67.3	78.1	78.2	0.273	0.355	0.436
6	83.7	64.9	82.8	66.4	77.4	77.5	0.273	0.358	0.427
7	83.0	70.7	85.2	68.5	79.5	79.5	0.707	0.247	0.329
8	83.2	67.7	83.9	66.6	78.0	78.1	0.253	0.333	0.502
9	80.6	73.1	85.8	65.1	78.1	78.1	0.248	0.329	0.516

Table B.5: Test of best attribute configuration for the asphalt model on the Raw dataset

Grass Model

TEST#	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD
1	86.9	46.5	77.3	63.0	73.9	73.8	28.9	37.6	35.8
2	87.2	49.0	78.1	64.7	74.8	74.9	26.4	35.3	43.3
3	87.8	54.7	80.2	68.1	77.0	77.1	24.6	33.2	49.9
4	89.3	54.0	80.2	70.6	77.8	77.9	23.4	32.4	52.3
5	87.6	50.3	78.7	66.1	75.5	75.5	25.2	34.3	46.3
6	86.7	51.2	78.8	64.9	75.2	75.2	25.4	34.5	46.0
7	88.2	52.0	79.3	67.8	76.4	76.5	24.4	33.0	50.4
8	85.9	50.9	78.5	63.4	74.5	74.6	25.3	33.7	48.3
10	87.2	51.5	79.0	66.0	75.6	75.7	24.9	33.2	50.0

Table B.6: Test of best attribute configuration for the grass model on the Raw dataset

Hybrid System

	Hybrid Model										
TEST#	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F1	MAE	RMS	CoD
1	89.7	54.8	53.0	89.0	52.9	55.4	65.9	65.9	0.385	0.502	0.620
2	93.1	52.5	57.6	90.5	55.4	56.3	67.8	67.8	0.342	0.456	0.686
3	92.6	51.3	62.2	92.3	56.4	57.2	68.7	68.8	0.308	0.420	0.734
4	92.9	49.5	62.6	92.4	56.1	56.2	68.4	68.4	0.303	0.413	0.743
5	88.4	52.2	60.1	92.7	53.0	56.5	66.9	67.0	0.321	0.430	0.721
6	92.1	51.6	58.5	89.5	55.3	56.4	67.4	67.5	0.333	0.448	0.698
7	92.4	48.4	61.4	92.8	54.0	55.5	67.5	67.5	0.322	0.428	0.724
8	90.8	50.0	58.7	90.9	52.6	55.9	66.5	66.6	0.321	0.431	0.721

Table B.7: Test of best attribute configuration for the Hybrid system on the Raw dataset

B.1.3 Layer test

Here an extended result from the exhaustive search is presented. The tables show how the evaluation parameters change when adding more complexity to the network.

Mixed System

Asphalt Model										
HL	Mem	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD
[5]	2.14	84.1	56.6	79.6	63.8	74.9	75.0	0.300	0.378	0.360
[10]	2.85	84.8	68.8	84.6	69.2	79.5	79.5	0.269	0.352	0.446
[15]	3.55	84.6	69.3	84.6	68.0	79.0	79.0	0.261	0.344	0.471
[20]	4.33	85.0	68.3	84.4	69.3	79.4	79.5	0.257	0.341	0.479
[25]	5.05	83.4	72.1	85.9	68.9	79.9	80.0	0.250	0.332	0.505
[30]	5.78	83.2	68.5	84.2	66.9	78.3	78.3	0.259	0.344	0.470
[35]	6.50	84.6	74.9	87.2	70.7	81.3	81.3	0.239	0.322	0.536
[40]	7.22	86.2	76.0	87.9	73.2	82.8	82.8	0.228	0.311	0.567
[45]	7.95	86.1	76.4	88.0	73.2	82.9	82.9	0.227	0.309	0.572
[50]	8.67	89.0	57.1	81.2	71.2	78.6	78.7	0.224	0.315	0.548
[15,15]	4.54	94.4	87.4	93.8	88.5	92.0	92.1	0.124	0.217	0.790
[20,10]	5.15	95.4	85.8	93.1	90.2	92.2	92.2	0.101	0.222	0.780
[20,20]	6.01	94.9	87.5	93.9	89.4	92.4	92.4	0.117	0.211	0.801
[25,10]	6.05	95.4	87.1	93.7	90.4	92.6	92.6	0.097	0.205	0.812
[25,25]	7.61	95.0	87.1	93.7	89.6	92.3	92.4	0.111	0.203	0.816
[35,15]	8.53	94.3	86.9	93.6	88.4	91.8	91.9	0.082	0.202	0.817
[35,30]	10.70	95.4	87.1	93.7	90.5	92.6	92.7	0.103	0.198	0.824
[40,35]	12.79	95.4	87.2	93.7	90.3	92.6	92.7	0.090	0.192	0.836
[45,25]	12.38	95.5	86.9	93.6	90.5	92.6	92.6	0.097	0.194	0.831
[50,10]	10.54	95.3	88.0	94.1	90.4	92.9	92.9	0.089	0.193	0.833

Table B.8: Results of one and two layers for asphalt model on the Raw dataset. Best configuration marked in red

Grass Model

HL	Mem	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD
[5]	2.14	84.0	48.5	77.3	59.2	72.4	72.5	0.282	0.366	0.392
[10]	2.85	87.2	50.0	78.5	65.3	75.2	75.2	0.258	0.346	0.456
[15]	3.55	87.1	51.0	78.8	65.6	75.4	75.4	0.254	0.343	0.465
[20]	4.33	87.8	51.6	79.2	67.1	76.1	76.2	0.251	0.342	0.470
[25]	5.05	89.1	55.8	80.8	71.0	78.3	78.3	0.238	0.323	0.528
[30]	5.78	88.9	54.0	80.1	70.0	77.5	77.5	0.243	0.331	0.504
[35]	6.50	88.7	56.4	81.0	70.5	78.2	78.3	0.231	0.319	0.539
[40]	7.22	90.3	56.1	81.1	73.5	79.2	79.3	0.224	0.311	0.560
[45]	7.94	90.0	56.5	81.2	72.8	79.1	79.1	0.218	0.308	0.569
[50]	8.67	89.0	57.1	81.3	71.2	78.6	78.7	0.224	0.315	0.548
[10,10]	3.35	96.5	83.7	92.5	91.9	92.3	92.4	0.110	0.207	0.806
[15,15]	4.54	96.6	82.9	92.2	92.1	92.1	92.1	0.095	0.207	0.806
[20,10]	5.15	96.7	83.0	92.2	92.2	92.2	92.2	0.093	0.205	0.810
[30, 25]	8.80	96.6	92.1	92.4	92.1	92.2	92.3	0.073	0.191	0.834
[35,35]	11.39	96.4	83.7	92.5	91.8	92.2	92.2	0.084	0.193	0.831
[40,30]	11.98	96.5	83.5	92.4	91.8	92.2	92.2	0.082	0.188	0.934
[40,40]	13.59	96.1	84.2	92.7	91.1	92.2	92.2	0.093	0.187	0.842
[45,20]	11.47	96.5	83.2	92.3	92.0	92.1	92.2	0.075	0.187	0.842
[50,15]	11.52	96.7	82.8	92.1	92.2	92.1	92.1	0.091	0.193	0.832
[50,20]	12.57	96.7	82.6	92.0	92.4	92.0	92.1	0.078	0.191	0.834

Table B.9: Results of one and two layers for grass model on the Raw dataset. Best configuration marked in red

Hybrid

Hybrid Model

HL	Mem	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F1	MAE	RMS	CoD
[5]	2.14	95.0	47.4	53.0	86.9	52.1	53.2	65.2	65.2	0.372	0.484	0.647
[10]	2.85	91.7	47.4	59.3	90.8	52.6	54.5	66.2	66.2	0.336	0.447	0.699
[15]	3.55	94.2	47.5	57.7	90.1	53.7	54.1	66.6	66.6	0.344	0.454	0.689
[20]	4.33	94.4	51.2	61.7	92.0	57.2	57.2	69.2	69.2	0.325	0.432	0.718
[25]	5.05	90.4	56.8	61.3	92.2	57.3	59.6	69.5	69.6	0.314	0.422	0.732
[30]	5.78	95.4	52.9	58.8	91.4	56.9	57.2	69.1	69.1	0.318	0.429	0.724
[35]	6.50	92.0	53.5	65.2	93.2	58.5	59.3	70.3	70.3	0.304	0.413	0.743
[40]	7.22	91.1	55.6	63.7	93.2	58.0	59.9	70.2	70.2	0.294	0.402	0.756
[45]	7.95	92.7	56.2	64.6	93.3	59.8	60.5	71.2	71.2	0.293	0.402	0.756
[50]	8.67	90.1	58.0	61.1	92.9	57.1	60.3	69.8	69.8	0.294	0.406	0.752
[35,30]	10.70	94.1	83.0	88.6	94.9	84.6	86.2	88.4	88.5	0.138	0.264	0.897
[35,35]	11.40	96.3	82.3	87.8	95.3	85.7	85.4	88.7	88.7	0.151	0.277	0.887
[40,10]	8.74	94.4	85.8	88.6	96.9	85.1	87.1	89.5	89.5	0.107	0.252	0.906
[45,25]	12.38	95.4	83.9	87.3	95.4	84.6	86.7	88.8	88.8	0.128	0.261	0.900
[45,40]	15.07	96.7	82.6	88.6	95.0	86.8	85.9	89.1	89.1	0.147	0.268	0.894
[50,15]	11.52	94.1	86.3	87.8	96.8	84.2	87.6	89.3	89.3	0.115	0.252	0.906
[50,20]	12.57	97.1	83.3	88.7	95.3	87.0	86.6	89.6	89.6	0.137	0.266	0.896
[50,25]	13.57	93.9	85.3	89.1	96.9	84.8	87.1	89.3	89.4	0.128	0.256	0.903
[50,35]	15.56	96.8	85.3	87.6	95.7	85.8	87.9	89.7	89.8	0.134	0.265	0.896
[50,50]	18.55	96.4	82.9	87.7	95.2	85.5	86.1	88.9	88.9	0.145	0.272	0.891

Table B.10: Results of one and two layers for Hybrid model on the Raw dataset. Best configuration marked in red

B.2 Features Dataset

The tables is presented in the order they were created; test of attributes and layers construction. The rows that are marked in yellow are the models that we considered to be the best for each test.

B.2.1 Attribute test

Attribute Test, Features Dataset

		STD			VAR			MAX			MIN		
TEST#	HL	Lag	Del	BMA	Lag	Del	BMA	Lag	Del	BMA	Lag	Del	BMA
1	[14]	1-2	1-2	3	1-2	1-2	3	1-2	1-2	3	1-2	1-2	3
2	[24]	1-4	1-4	5	1-4	1-4	5	1-4	1-4	5	1-4	1-4	5
3	[15]	1-5	0	5	1-5	0	5	1-5	0	5	1-5	0	5
4	[15]	0	1-5	5	0	1-5	5	0	1-5	5	0	1-5	5
5	[30]	1-5	1-5	5	1-5	1-5	5	1-5	1-5	5	1-5	1-5	5
6	[15]	1-5	0	10	1-5	0	10	1-5	0	10	1-5	0	10
7	[15]	0	1-5	10	0	1-5	10	0	1-5	10	0	1-5	10
8	[30]	1-5	1-5	10	1-5	1-5	10	1-5	1-5	10	1-5	1-5	10
9	[52]	1-10	1-10	10	1-10	1-10	10	1-10	1-10	10	1-10	1-10	10

Table B.11: The test for finding the best configurations for attributes on features. Lag represent a lag attribute, Del represent a delta attribute, and BMA represents a backward moving average attribute

Mixed System

Asphalt Model

TEST#	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD
1	94.9	85.5	94.1	87.4	92.1	92.1	0.146	0.239	0.748
2	96.3	91.9	96.6	91.0	94.8	95.0	0.113	0.188	0.844
3	97.7	88.3	95.3	94.2	94.8	95.0	0.129	0.207	0.812
4	95.6	93.9	97.4	89.9	94.9	95.1	0.105	0.185	0.849
5	93.9	92.4	96.8	86.2	93.2	93.4	0.092	0.174	0.867
6	97.3	97.7	99.0	93.8	97.0	97.4	0.096	0.168	0.876
7	95.4	97.7	99.0	89.8	95.7	96.1	0.078	0.156	0.893
8	95.2	96.5	98.5	89.3	95.2	95.6	0.075	0.151	0.899
9	96.3	95.9	98.3	91.5	95.7	96.2	0.069	0.126	0.930

Table B.12: Test of best attribute configuration for the asphalt model, Features dataset z-axis

Grass Model

TEST#	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD
1	99.3	89.8	90.3	99.3	94.4	94.5	0.046	0.134	0.910
2	99.4	90.4	90.9	99.4	94.7	94.8	0.033	0.098	0.952
3	98.7	91.9	92.1	98.7	95.0	95.2	0.023	0.093	0.957
4	99.4	90.5	91.0	99.4	94.7	94.9	0.030	0.101	0.950
5	99.3	91.2	91.6	99.3	94.9	95.2	0.033	0.088	0.962
6	99.6	93.0	93.2	99.6	95.8	96.2	0.018	0.078	0.970
7	99.9	92.1	92.4	99.9	95.5	95.9	0.027	0.072	0.974
8	99.6	91.0	91.4	99.6	94.8	95.2	0.021	0.067	0.978
9	99.1	92.1	92.3	99.1	95.0	95.5	0.019	0.050	0.988

Table B.13: Test of best attribute configuration for the grass model, Features dataset z-axis

Hybrid System

Hybrid Model, y-axis

TEST#	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F1	MAE	RMS	CoD
1	97.2	74.3	88.5	96.8	91.0	631	86.1	86.2	0.187	0.294	0.873
. 2	99.8	80.0	90.6	97.3	95.2	69.6	89.8	90.0	0.163	0.256	0.904
3	99.9	76.1	89.2	96.2	94.6	66.0	88.0	88.2	0.151	0.243	0.913
4	100.0	79.4	90.3	94.5	95.7	72.1	89.6	89.8	0.169	0.262	0.899
5	98.9	74.8	89.5	95.3	93.2	66.0	87.1	87.3	0.168	0.263	0.898
6	100.0	85.1	90.3	94.2	95.4	81.9	91.7	92.1	0.152	0.243	0.914
7	100.0	82.4	94.2	97.2	97.1	74.0	91.3	91.7	0.124	0.199	0.942
8	99.8	77.7	93.2	95.2	96.3	70.3	89.1	89.5	0.141	0.226	0.925
9	9.99	61.3	91.1	88.3	94.0	61.3	82.0	82.4	0.132	0.205	0.938

Table B.14: Test of best attribute configuration for the Hybrid system, Features dataset, y-axis

Hybrid Model, z-axis

TEST#	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F1	MAE	RMS	CoD
1	99.6	77.8	92.2	87.5	91.8	97.5	91.2	91.3	0.169	0.280	0.880
2	99.4	84.5	94.8	91.1	94.3	97.9	93.6	93.8	0.125	0.226	0.922
3	99.4	85.4	94.6	91.5	94.2	97.9	93.8	94.0	0.107	0.203	0.937
4	99.4	83.9	93.3	90.7	92.9	97.5	93.0	93.2	0.137	0.238	0.913
5	99.3	86.9	95.3	92.8	94.5	97.6	94.4	94.6	0.111	0.226	0.921
6	99.1	92.9	93.9	95.5	93.6	98.8	95.4	95.8	0.118	0.211	0.933
7	99.1	89.6	95.4	94.9	94.7	96.6	94.9	95.3	0.094	0.194	0.942
8	99.1	91.8	95.3	95.8	95.0	96.9	95.4	95.9	0.095	0.178	0.951
9	98.9	89.9	94.1	95.2	93.9	95.7	94.5	94.9	0.070	0.138	0.971

Table B.15: Test of best attribute configuration for the Hybrid system, Features dataset, z-axis

B.2.2 Layer test

Mixed System

Asphalt Model										
HL	Mem	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD
[5]	1.86	97.6	97.5	99.0	94.4	97.2	97.6	0.123	0.190	0.840
[10]	2.41	97.9	97.4	98.9	95.2	97.4	97.8	0.116	0.181	0.855
[15]	2.96	97.3	97.7	99.0	93.8	97.0	97.4	0.096	0.168	0.876
[20]	3.58	97.8	92.1	96.8	94.6	95.7	96.1	0.112	0.185	0.850
[25]	4.15	97.3	97.2	98.8	93.8	96.9	97.3	0.102	0.164	0.881
[30]	4.71	97.5	96.5	98.5	94.1	96.8	97.2	0.103	0.167	0.877
[35]	5.28	95.8	97.4	98.9	90.6	95.9	96.3	0.094	0.155	0.894
[40]	5.85	97.6	96.7	98.6	94.4	96.9	97.3	0.082	0.145	0.907
[45]	6.41	97.0	96.2	98.4	93.0	96.3	96.8	0.091	0.152	0.898
[50]	6.98	97.5	96.0	98.3	94.2	96.7	97.1	0.087	0.150	0.901
[10,5]	2.72	97.8	97.5	99.0	94.9	97.3	97.7	0.035	0.131	0.924
[15,5]	3.36	97.5	98.2	99.2	94.3	97.3	97.7	0.034	0.127	0.929
[15,10]	3.66	97.9	97.9	99.1	95.2	97.5	97.9	0.032	0.122	0.935
[25,15]	5.63	97.9	97.5	99.0	95.0	97.4	97.8	0.019	0.095	0.960
[20,20]	5.26	97.4	98.0	99.2	94.0	97.2	97.6	0.022	0.103	0.953
[30,5]	5.30	98.0	97.9	99.1	95.3	97.5	98.0	0.022	0.102	0.954
[30,30]	8.34	97.9	98.0	99.2	95.0	97.5	97.9	0.026	0.110	0.947
[35,15]	7.31	97.5	98.5	99.4	94.2	97.4	97.8	0.021	0.102	0.954
[35,30]	9.48	97.5	98.3	99.3	94.3	97.3	97.8	0.023	0.104	0.952
[40,25]	9.81	97.9	98.5	99.4	95.2	97.7	98.1	0.023	0.106	0.950

Table B.16: Results of using one hidden and two layers for the asphalt model, Features dataset z-axis

Grass Model

HL	Mem	P_0	P_1	R_0	R_1	A_T	F1	MAE	RMS	CoD
[5]	1.86	100.0	93.1	93.3	100.0	96.1	96.5	0.021	0.097	0.953
[10]	2.41	99.8	91.7	92.0	99.8	95.2	95.6	0.020	0.083	0.966
[15]	2.96	100.0	92.4	92.7	100.0	95.7	96.1	0.019	0.078	0.970
[20]	3.58	100.0	92.2	92.5	100.0	95.6	96.0	0.023	0.085	0.964
[25]	4.15	100.0	92.6	92.8	100.0	95.8	96.2	0.026	0.089	0.961
[30]	4.71	100.0	92.1	92.4	100.0	95.6	96.0	0.029	0.077	0.971
[35]	5.28	100.0	91.8	92.1	100.0	95.4	95.8	0.026	0.080	0.969
[40]	5.85	100.0	93.0	93.2	100.0	96.0	96.4	0.030	0.068	0.977
[45]	6.41	100.0	92.1	92.3	100.0	95.5	96.0	0.032	0.076	0.971
[50]	6.98	99.9	92.0	92.2	99.9	95.4	95.8	0.028	0.070	0.975
[25,5]	4.65	99.7	93.6	93.7	99.7	96.1	96.5	0.009	0.065	0.979
[25,15]	5.63	99.6	93.8	93.8	99.6	96.2	96.6	0.002	0.027	0.996
[25,25]	6.70	99.6	93.8	93.8	99.6	96.2	96.6	0.007	0.056	0.985
[30,20]	7.13	99.7	93.5	93.6	99.7	96.1	96.5	0.003	0.033	0.995
[35,25]	8.77	99.6	93.6	93.7	99.6	96.1	96.5	0.004	0.040	0.992
[35,30]	9.48	99.3	94.2	94.3	99.3	96.3	96.7	0.003	0.033	0.995
[40,5]	6.59	99.6	94.2	94.3	99.6	96.4	96.9	0.004	0.043	0.991
[40,30]	10.61	99.4	93.8	93.8	99.4	96.1	96.5	0.006	0.045	0.990
[45,30]	11.74	99.7	93.9	93.9	99.7	96.3	96.7	0.006	0.051	0.987
[50,50]	16.86	99.3	93.5	93.6	99.3	95.9	96.3	0.004	0.040	0.992

Table B.17: Results of using one and two hidden layers for the grass model, Features dataset z-axis

Hybrid System

Hybrid Model												
HL	Mem	P_0	P_1	P_2	R_0	R_1	R_2	A_T	F1	MAE	RMS	CoD
[5]	2.57	98.7	89.6	94.8	91.4	93.3	97.7	94.0	94.4	0.089	0.201	0.939
[10]	3.50	98.7	91.7	94.9	92.9	93.0	98.5	94.7	95.1	0.133	0.213	0.931
[15]	4.44	98.7	94.7	95.2	95.5	93.5	98.7	95.7	96.1	0.100	0.188	0.946
[20]	5.46	98.7	90.9	95.4	92.9	93.4	98.0	94.6	95.0	0.099	0.184	0.948
[25]	6.41	98.6	93.9	96.5	94.6	95.0	98.8	95.9	96.3	0.095	0.181	0.950
[30]	7.37	98.6	88.8	96.1	93.2	94.2	96.1	94.2	94.6	0.092	0.174	0.954
[35]	8.33	99.0	90.2	95.6	92.7	94.6	97.3	94.6	95.0	0.116	0.187	0.947
[40]	9.29	98.7	94.4	95.7	96.0	93.6	98.5	95.8	96.2	0.093	0.172	0.955
[45]	10.24	98.6	92.8	96.5	94.6	94.8	98.0	95.6	96.0	0.095	0.178	0.952
[50]	11.20	99.2	89.3	95.9	93.4	94.6	96.3	94.5	94.9	0.098	0.170	0.956
[15,10]	3.65	99.3	97.3	94.9	97.8	95.7	98.7	97.0	97.4	0.026	0.107	0.982
[15,15]	3.94	98.9	97.3	95.1	98.5	94.4	98.7	96.9	97.3	0.032	0.145	0.967
[25,25]	6.70	99.0	97.3	96.1	98.5	95.7	98.6	97.3	97.7	0.039	0.145	0.968
[30,5]	5.29	98.9	96.9	95.7	98.3	95.5	98.2	97.0	97.4	0.033	0.142	0.968
[30,15]	6.46	98.9	94.1	95.4	98.9	94.1	98.9	97.1	97.5	0.030	0.136	0.971
[35,30]	9.47	99.3	9.6.4	96.1	98.0	96.9	97.4	97.1	97.5	0.021	0.089	0.987
[40,30]	10.61	99.3	96.6	96.4	98.4	96.4	98.2	97.0	97.4	0.020	0.102	0.983
[40,35]	11.41	99.4	96.8	96.1	98.7	96.4	97.4	97.3	97.7	0.016	0.081	0.989
[45,15]	8.99	98.9	96.9	97.0	98.9	96.0	97.9	97.4	97.8	0.021	0.113	0.980
[45,30]	11.74	98.9	96.8	96.8	98.6	96.0	98.1	97.3	97.7	0.018	0.104	0.983

Table B.18: Results of using one and two hidden layers for the Hybrid system, Features dataset z-axis

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-12-09

EXAMENSARBETE Surface Detection with Multilayer Perceptron models on the Edge
STUDENT August Zenk och Johan Ekman
HANDLEDARE Flavius Gruian (LTH)
EXAMINATOR Jacek Malec (LTH)

Klassificering av underlag med hjälp av maskininlärning på mikroprocessor

POPULÄRVETENSKAPLIG SAMMANFATTNING **August Zenk och Johan Ekman**

En stor utmaning inom nätverkskommunikation är hur man ska hantera den ständigt ökande mängden data som produceras. I detta examensarbete utvärderades den potentiella lösningen, Edge Computing, genom implementation av flera maskininlärningsbaserade system exekverade på en mikroprocessor.

Idag sker det allra flesta maskininlärningslösningarna på Moln-serverar. Data samlas in via sensorer som är integrerade i olika system och skickas sedan över ett nätverk till servern där den analyseras. Utifrån den informationen kan det implementerade maskininlärningsprogrammet ta ett beslut som sedan kan skickas tillbaka till systemet där det exekveras. I den här kommunikationskedjan skickas väldigt mycket data och belastar kommunikationslänken. Vid väldigt hög belastning kan krockar och förseningar på den begränsade kommunikationslänken uppstå. I takt med att mer elektronik kopplar upp sig till internet växer problemen med överbelastning. Ett alternativt koncept för att komma runt detta problemet är Edge Computing. Detta flyttar analys och beslutstagande från Moln-servern närmare systemet som samlar in data. Då man maskininlärningen sker innan data skickas över internet kommer mängden att minska drastiskt och problemen med överbelastning kommer minska.

Det finns flera fördelar med att implementera en analytisk modell i detta steg. Man minskar trafiken över nätverket, vilket leder till att dess kapacitet kan användas till ett högre antal användare samt att minska kostnaden för strömmad data. Genom att kringgå bearbetning av data i Molnet och göra den direkt på insamlingssystemet minskar även tiden mellan insamling av data och taget beslut. Detta leder till en snabbare och effektivare lösning.

Nackdelarna med ett system där datan bearbetas i Edge, är att beräkningskapaciteten är sämre och minneskapaciteten begränsad i jämförelse till lösningar i Molnet. I många fall är Edge-systemen utplacerade på platser med begränsad energitillförsel och måste därför drivas så energisnålt som möjligt.

Vi har under detta examensarbete implementerat och utvärderat fyra olika maskininlärningslösningar på ett Edge-system. Datan bearbetades på en mikroprocessor med begränsat minne. Maskininlärningslösningarna bestod av neurala nätverk av typen Multilayer Perceptrons. Modellerna tränades för att identifiera underlag baserade på de vibrationer som skapades när en radiostyrd bil körde runt. Vibrationerna uppmättes med en accelerometer-sensor. Utvärderingen av lösningarna baserades på deras träffsäkerhet vid identifiering av underlag, deras storlek i minnet, samt på deras energiförbrukning.

Trots begränsningarna visade det sig att en hög träffsäkerthet kunde uppnås med modeller som var både minnes- och energieffektiva.