Classifying Motion Patterns of Bikes using Machine Learning

Filip Larsson

Pontus Hallqvist

LUND UNIVERSITY

Department of Automatic Control

Msc Thesis
TFRT-6192
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2023

Abstract

Electric bikes have become ubiquitous in traffic, and with a growing user base and expensive prices, a demand for bike protection is increasing. Bike protection applications could include detecting and notifying the owner if their bike has been stolen or fallen over. This thesis aims to develop solutions for recognizing and classifying motion patterns of an electric bike to allow for improvements in bike protection applications.

Using accelerometer, gyroscope and magnetometer data as input, machine learning models were developed to perform classification. The data was labeled to six classes of different motions and then normalized, split into time windows and featurized. The different machine learning models built and tested were *k*-nearest neighbors (KNN), Convolutional neural network (CNN), Long short-term memory (LSTM) and a combined CNN-LSTM network. Time windows with different lengths and overlaps were tested and evaluated to achieve the best accuracy possible. Lastly, a filter was applied to the output to correct misclassifications.

To increase the understanding of how decisions were made by the models, Grad-CAM was applied to highlight what parts of the information the model found most crucial. Using the Grad-CAM heatmaps, it was found that the gyroscope data was the most influential for the model's decisions.

The model with the best performance was a CNN-LSTM combination network that uses a time window of 2 seconds and 75% overlap. It performed with an accuracy of 94.65%. When testing the best model with data from other bikes with different mounting positions, the accuracy was 35.23% indicating that different sensor placements or orientations changes the data in a way the current model cannot handle.

Acknowledgements

We would like to thank our supervisors Yiannis Karayiannidis, Gustav Träff, Martin Heyden and Stefan Noll for all their helpful inputs, creative ideas and enthusiasm during the thesis. We would also like to thank Anders Svensson and the eBike team for the opportunity to do our thesis at Bosch, and all the support in the meanwhile. Lastly we would like to thank all people we've met during our studies who have helped to spark our interest for both machine learning as a topic and engineering at large.

Contents

List of Acronyms

- **AI** - **A**rtificiell **I**ntelligence
- **AR** - **A**ctivity **R**ecognition
- **CNN** - **C**onvolutional **N**eural **N**etwork
- **DFT** - **D**iscrete **F**ourier **T**ransform
- **DTW** - **D**ynamic **T**ime **W**arping
- **eBike** - **E**lectric **Bike**
- **Grad-CAM** - **Grad**ient-weighted **C**lass **A**ctivation **M**apping
- **IMU** - **I**nertial **M**easurement **U**nit
- **KNN** - **K** **N**earest **N**eighbours
- **LSTM** - **L**ong-**S**hort **T**erm **M**emory
- **ML** - **M**achine **L**earning
- **MLP** - **M**ulti **L**ayer **P**erceptron
- **MSE** - **M**ean **S**quared **E**rror
- **NN** - **N**eural **N**etwork
- **RNN** - **R**ecurrent **N**eural **N**etwork
- **SVM** - **S**upport **V**ector **M**achine

1

Introduction

1.1 Motivation

Artificial intelligence, or AI, is a growing field of research with applications becoming present in peoples every day life. Automatic vacuum cleaners and speech recognition are two examples. However, giving AI a single definition is difficult, but the mathematician Richard Bellman defines it as "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." and futurist Ray Kurzweil defines it as “The art of creating machines that perform functions that require intelligence when performed by people.”. In short, AI can be broadly defined as a machine’s ability to think and act like a human in a rational manner [Russell, 2010].

Machine learning, ML for short, is a sub-category of AI where an agent learns how to make decisions through training. Learning means that the program improves its performance on future tasks by observing the outcome and using evaluation metrics as feedback. Machine learning is dynamic, as ML-models change with experience and new data [Mitchell and Mitchell, 1997] [Russell, 2010].

ML is a growing field of research due to its broad area of usage, and its major hurdle of requiring a large amount of processing power is becoming less of an issue as the technical development increases the availability of powerful computational resources. Machine learning models have shown to be able to beat world champions in advanced games such as Poker [Brown and Sandholm, 2018], Dota 2 [OpenAI et al., 2019] and Go [Silver et al., 2016], and is a powerful tool in image recognition [Chollet, 2021, Chapter 8], natural language processing [Chollet, 2021, Chapter 11], robotics [Mitsioni et al., 2021][Nguyen-Tuong and Peters, 2011] and activity recognition [Ramasamy Ramamurthy and Roy, 2018].

Activity recognition, or AR, is the process of identifying actions and their outcome for one or more agents from a series of observations. Recognizing and differentiating between actions and behavioral patterns is useful in many different

areas such as security, elder care, exercise and more. AR is a challenging task as different actors or agents can do an action in different ways, and some actions or events can consist of a sequence of smaller actions at once, such as carrying a bike which contains both a lift and a translation. Due to the complexity this brings, it can be to difficult to find the necessary patterns which distinguish the actions using analytical solutions. Therefore machine learning is a common approach to solve AR-problems due to its strengths in pattern recognition.

Electric bikes are often expensive and owners want ways to ensure that their bike will not get stolen or damaged. Theft detection with alarms that trigger after a certain amount of motion is detected is one way to increase the security. However if a bike is rolled away carefully by thieves, an algorithm which only detects the motion level could fail to trigger the alarm. If the algorithm had the capability to not only detect the motion level but also classify which kind of motion just happened, the algorithm could be more accurate. A motion detection algorithm can also be used for alerting a user that their bike has fallen over and not been picked back up, so the user can prevent damages that could happen while the bike is laying down.

1.2 Related work

Tsige Tadesse Alemayoh, Jae Hoon Lee and Shingo Okamoto structured time series data from an accelerometer and gyroscope as well as Discrete Fourier Transforms into a virtual image which were fed into a 2-dimensional CNN network to classify human activities, such as walking, running and jumping. They were able to achieve an accuracy of 99.5% [Alemayoh et al., 2019]. The paper inspired a use of CNNs applied on the AR-problem and how to structure the input data in different ways.

Nishkam Ravi, Nikhil Dandekar, Preetham Mysore and Micheal L. Littman calculated a handful of features from accelerometer data and used several base level classifiers to solve a human activity recognition problem. Multiple machine learning methods where used in parallel and the final decision was made using a plurality vote between the different models. They used four different settings for how the data was collected and used, which showed a large variation in results and indicates how sensitive models can be for different data [Ravi et al., 2005]. The different settings inspired the idea of testing the performance of the final model on different bikes, and the plurality voting inspired the development of an output filter using majority voting.

Alemayoh et al. were able to solve the human Activity Recognition-problem with only a CNN using clever input formatting as opposed to Ravi et al. who solved the human AR-problem using multiple ML-models with a plurality vote as the deciding factor. Both papers achieved an equal level of accuracy with two

different approaches, showing that the AR-problem can be solved in many manners.

Further inspiration was taken from Ben D. Fulcher and Nick S. Jones with their feature based approach for time series classification. They compared thousands of different features for 20 different time series datasets, and the most informative features were selected using greedy forward feature selection. The final features selected was used to provide an understanding of the properties of the dataset [Fulcher and Jones, 2014].

Rohit J. Kate used distance based methods to classify time series problems. The article discusses the differences between using feature based methods and distance based methods. The paper also shows how Dynamic Time Warping, a common distance method, can be used to create new features which in combination with standard machine learning algorithms achieves improved results in 37 of 47 benchmark datasets [Kate, 2016].

Both the paper by Kate, and the paper by Fulcher and Jones inspired different methods for how to create suitable features for time series data as input to machine learning models. Where Fulcher and Jones created many features and tested which combination of features performed best, Kate focused on a single feature and its performance in ML-methods.

When doing research about the activity recognition problem, applications outside human activity recognition were rare and none applied on bikes or other vehicles were found.

1.3 Problem formulation

This thesis aims to find out if it is possible to detect and classify motions on bikes using machine learning applied on data collected by sensors.

Different machine learning methods will be tested and compared to each other, with a focus on different types of neural networks. Focus will also be put on taking decisions that would work in real-time if implemented on a bike. Therefore the challenge of not knowing how long an event will be beforehand and also how to compare events of different lengths will be a theme during the project.

It can be difficult to understand why different models take the decisions they take and what values or information it bases its decisions on. Therefore a part of the thesis will be dedicated to model interpretability. If one could understand what the model finds important in the data, that information could be used to better tune input values or model parameters to increase the performance.

The thesis aims to answer the following questions:

- Is it possible to detect and classify eBike motions with machine learning?
- What type of model performs best for this problem, and what is the performance?
- How can the data be presented to the models and how can events of different lengths be compared?
- Is it possible to understand the model's decision making?
- What can the solution be used for and what future extensions could be added to improve the performance?

1.4 Report outline

Chapter 2 Background explains how the relevant data processing works and then moves on to explain the area of machine learning in detail. Chapter 3 Methodology describes how the data was processed, how the machine learning models were developed and evaluated, and finally how the results were filtered. Next, Chapter 4 Results presents the different model performances with more details in an appendix. Lastly, in Chapter 5 Discussion the models are discussed and compared to other AR solutions. Improvement areas and possible applications with be presented and finally some conclusions will be drawn.

1.5 Limitations

This thesis is limited to finding an optimal solution without regard to any limit in computational power or memory size. Thus, when talking about performance, the thesis refers to a model's ability to correctly classify an event. It will also be limited to exploring a select few machine learning methods; the K-nearest-neighbours classifier and different types of neural networks. The thesis will limit itself to only train the models on data collected from one bike rather than multiple, but event classification accuracy will be tested on a small dataset with data from other bikes with different mounting positions.

2

Background

In this chapter the sensor for data collection is described. Next, data preprocessing is discussed. The final part of the chapter will cover machine learning. Some general concepts will be described followed by a thorough explanation of the algorithms used in this thesis.

2.1 Inertial measurement unit

An Inertial measurement unit (IMU) is a measurement instrument consisting of an accelerometer, a gyroscope and a magnetometer. The accelerometer measures the acceleration, the gyroscope measures the angular velocity and the magnetometer measures the magnetic field. All three quantities are taken in three orthonormal directions, resulting in nine degrees of freedom [Madgwick et al., 2011].

The IMU is constantly affected by gravity which creates a downward offset of about $9.8\,ms^{-2}$. Since it is likely that the IMU is rotated, the offset will be reflected on all three components. This can be hard to compensate for when it is constantly moving, or when the relative mounting position is unknown. However, one possible solution to remove the offset is to high pass filter the sensor output, and if the IMU is allowed to stand still when it is started it might be possible to have the IMU automatically measure and calibrate the offset. The calibration method depends on whether the IMU is accurate enough to measure the same offset every time, otherwise small offsets will remain.

Depending on the placement of the IMU, it might be located nearby the drive unit which creates a magnetic field when the motor is running or the user is pedaling. The induced magnetic field will disturb the magnetometer and is problematic as it drowns out the magnetic field of the earth. However, it is not only an issue as the induced magnetic field allows the possibility to differentiate between motions that are using the pedals and motions that do not.

2.2 Data processing

"Garbage in, garbage out" is a phrase in data science describing that the quality of the output is correlated with the quality of the input. To increase the input data quality, the data is processed before it is fed into any algorithm. In this part, the processing methods, data normalization, time window creation, featurization of the data and discrete Fourier transform are explained.

2.2.1 Data normalization

Whether a value is considered large or not depends on the relation between the value and all other values in the dataset. Due to this, if the values are observed without context, it can be hard to interpret the value. Data normalization is the concept of changing the scale of the data to fit a certain range to make the data more comparable. Two common methods of scaling data are described below [Patro and Sahu, 2015].

The first method is to fit the data, $\mathbf{x}$, based on its distribution using the mean value, μ, and the standard deviation, σ. $\mathbf{x}_{scale}$, denoting the scaled data, is scaled according to

$$\mathbf{x}_{\text{scale}} = \frac{\mathbf{x} - \mu}{\sigma} \tag{2.1}$$

Another common normalization method is to apply a linear transformation to map the data to the range $[0, 1]$ by scaling according to the maximum and minimum values according to

$$\mathbf{x}_{\text{scale}} = \frac{\mathbf{x} - x_{\min}}{x_{\max} - x_{\min}} \tag{2.2}$$

2.2.2 Time windows

When an event starts and when it stops can be difficult to define. Different events can also last for different periods of time. As an example, a person will ride the bike for a longer period of time than it takes for the bike to be knocked over. This is an issue when the goal is to detect and classify a motion as the amount of data for each motion is unknown.

One possible solution is to split the data into equally sized time windows. Time windows work by gathering a selected amount of data points into a single data frame. When the data frame has a specific size, the input size is known and the model can be designed accordingly. Thus, the model can determine which motion has happened in this time window rather than detecting when a motion starts and stops, and what type it is.

It is important to find a time window length that is long enough so sufficient information is included to classify events with confidence. At the same time it has to be short enough to minimize the risk of multiple events happening within the same window.

2.2.3 Data features

Features in data are values that express certain characteristics of the data. Having features is a way to reduce the dimensionality of the data but at the same time highlight what makes each class unique [Wang et al., 2006]. Featurization is therefore a way to have information more densely packed and allows for a model to more easily find patterns.

One important part of feature extraction is the selection of features. For the features to be useful, their values need to represent crucial characteristics about the data while excluding irrelevant information. It is thus necessary for the model developer to have a good understanding of the problem and the data [Wang et al., 2006].

2.2.4 Discrete Fourier transform

The Fourier transform is a way of transforming a time series to the frequency domain. It can highlight information that is otherwise hard to notice in the time domain. A continuous time signal $x(t)$ can be transformed to its frequency representation $X(\Omega)$ as follows

$$X(\Omega) = \int_{-\infty}^{\infty} x(t)e^{-j\Omega t}dt \tag{2.3}$$

where $\Omega = 2\pi f$ with f being the frequency in Hz. This however only works in theory as it is impossible to sample a signal continuously. Instead, the sampling happens at specific time points with a specific sampling rate. The discrete Fourier transform (DFT) is a way to transform the sampled signal $x(n)$ to a frequency representation approximation $X(\omega)$.

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \tag{2.4}$$

In reality, there is a finite amount of sampled data points N so that the expression instead becomes

$$X(\omega) = \sum_{n=0}^{N-1} x(n)e^{-j\omega n} \tag{2.5}$$

where $\omega = \frac{2\pi k}{N}$ with $\frac{k}{N} = f_s$ as the sampling frequency.

The quality of $X(\omega)$ is dependent on the sampling frequency, as a signal's frequency representation cannot include frequencies more than half the sampling frequency ($\frac{f_s}{2}$) according to the Nyquist-Shannon sampling theorem [Proakis J. G., 1996, Chapter 4].

2.3 Machine learning

In this chapter, machine learning as a general concept will be explained. Afterwards, the K-nearest-neighbours classifier and neural networks will be discussed in more detail.

2.3.1 General concepts

The idea of classification with machine learning is to develop a model that can find patterns or solutions in a training dataset and use the information from those patterns to recognize previously unseen data. Successful models are capable of correctly classifying similar but unknown data. This is known as generalization [Goodfellow et al., 2016, Chapter 5]. Since models cannot be fully certain of their classification due to differences between the training and general data, it is common that the models use statistical methods to get probabilistic results instead of deterministic results. Machine learning can be divided into three main categories; unsupervised learning, supervised learning and reinforcement learning.

Unsupervised learning is a type of learning where the model is trained on a dataset without labels to find patterns in the data. Unsupervised learning is used in tasks like clustering, data denoising and generation, and data association [Goodfellow et al., 2016, Chapter 5].

Supervised learning works by feeding a model with input data and corresponding labels or targets. The idea is to get the model to give labels to new input data similar to the data the model is trained on. Supervised learning is used in classification and regression tasks [Goodfellow et al., 2016, Chapter 5].

Reinforcement learning works by having an agent interacting with an environment. By following a simple set of rules, known as a policy, it chooses the actions to take. To make sure the agent has a possibility to try new actions, the policy allows for a choice of actions through probability rather than forcing a single action. The model learns through trial-and-error, and is given feedback by a designed reward-function which will change the behaviour slightly for the next run. There needs to be a balance between the agent trying new actions and learning (exploration) and the agent doing the action which is believed to be optimal (exploitation). Reinforcement learning can be used in use cases where a developer can define the environment for an agent to act in, a policy to follow, a set of possible actions and

a reward-function. Some examples where reinforcement learning is used include solving games and controlling robots [Sewak, 2019, Chapter 1].

Having proper ways of evaluating the performance is essential, thus the concept of cross validation as well as a few common evaluation metrics are explained next. Another important part of machine learning is the model parameters which will be described further down.

Cross-validation. Machine learning algorithms usually need large amounts of data to find reliable patterns and to avoid possible overfitting, which is common problem that will be described in Chapter 2.3.3. Cross-validation is a useful strategy to employ to get a more general understanding of how well the model performs. It works by first splitting the dataset into N subsets. The model is then trained N times with $N-1$ of the subsets and tested on the remaining set with the test set being a different subset every time. Figure 2.1 shows a visualization of how the training and test subsets can change over the iterations. Lastly a mean of the evaluation metrics is used to determine the performance of the model.

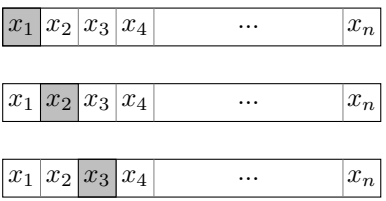


Figure 2.1 Visualization of cross validation. The subset of the data which is used for validation changes over iterations as to get a better understanding of the performance on general data.

Evaluation metrics. To evaluate the performance of ML models, there are a few metrics commonly used.

The first and most basic metric is a simple accuracy score. It is measured by taking the sum of all correct classifications and divide that with the total amount of classifications made, as per

$$\text{Accuracy} = \frac{\text{True positives}}{\text{Number of predictions}} \tag{2.6}$$

The main benefit is that it is a score which is easy for anyone to understand, while the main drawback is that it can give a misleading score depending on how the dataset is balanced in regards to amount of examples per class. An unbalanced dataset can give a high accuracy score by classifying everything as the same class, and an observer will not realize this from only looking at the value.

Another way is a confusion matrix which is a way to display all classifications by placing the predicted classes on a vertical axis and the true classes on the horizontal axis. Each classified sample is then placed in the matrix. For a model without misclassifications, only elements on the main diagonal will be non-zero since the elements on the main diagonal are filled with samples that got correctly classified. Examples of how confusion matrices may look like can be viewed in Appendix A.

Precision, recall and F_1-score are three other common performance measures used in classification problems [Powers, 2020]. Precision is a measure that gives the percentage of how many of the positive predictions that are truly positive and is calculated as

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \tag{2.7}$$

Recall gives a percentage of how many positive samples were correctly predicted as positive and calculated as

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \tag{2.8}$$

The precision and recall scores are used together since they complement each other. The F_1-score is the harmonic mean between recall and precision to give an accuracy score which performs better on an unbalanced dataset than the ordinary accuracy score. The F_1-score is calculated according to

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.9}$$

Model parameters. Machine learning models have parameters which determine the model performance. There are two types of parameters, trainable parameters and hyperparameters.

Trainable parameters are parameters that changes during training and are usually initialized with random values as to minimize any possible bias. One example of trainable parameter are the network weights in neural networks.

Hyperparameters are fixed parameters that are usually initialized by the model developer. These typically affect the structure of the model and how the model will train. Some examples include number of neighbours in KNN, nodes per neural network layer or decision boundary conditions in a Support vector machine (SVM)

[Goodfellow et al., 2016, Chapter 5].

2.3.2 KNN Classifier

Nearest neighbor classification, also known as K-nearest neighbors (KNN), is a simple algorithm for classification and pattern recognition. It is based on the idea that the closest patterns to a target pattern $\hat{\mathbf{x}}$ in the data space contain useful information about the class. To determine the closest patterns, similarity measures are used. In $\mathbb{R}^q$ this commonly is a distance function where the Minkowski metric, also called p-norm, is a reasonable choice

$$||\mathbf{x}_j - \hat{\mathbf{x}}||^p = (\sum_{i=1}^{n} |x_{i,j} - \hat{x}_i|^p)^{1/p} \tag{2.10}$$

where $\mathbf{x}_j$ is the compared pattern, $\hat{\mathbf{x}}$ is the target pattern, i indicates each element and p is a real number where $p \geq 1$. In this thesis $p = 2$ was used which gives the Euclidean distance. KNN then finds the k-nearest patterns and assigns the class label by majority voting [Hastie et al., 2009, Chapter 13][Kramer, 2013, Chapter 2].

When working with KNNs, the selection of k is what determines the model's characteristics. Choosing a small k creates small neighborhoods in regions were classes are scattered while choosing a larger k gives a more generalized classification. These small neighborhoods tend to overfit but a too generalized model ignores smaller clusters of data. Using a large k in an unbalanced dataset can create issues as the most common class will be overrepresented, meaning more neighbours will naturally be of that class. The best k is problem specific and the problem of finding it is known as model selection. Commonly cross-validation is employed to find the best model and parameters [Kramer, 2013, Chapter 2].

KNN like many other machine learning algorithms struggle with high dimensional data. It is thus important that the data presented to the model is descriptive and uniquely represents its class. If an added dimension of data for two classes contains the same information it only increases the complexity without adding any benefits [Kramer, 2013, Chapter 2].

2.3.3 Neural networks

A neural network is a network of mathematical nodes, inspired by biological neural nodes in the brain. Neural networks has shown to be able to classify images, language patterns and time series with a high accuracy among other objects [Chollet, 2021, Chapter 1].

The neural node and the base network. The network consists of differently weighted inputs and a single output. Each node takes K inputs, x_k, and calculates a

weighted sum, s based on weights, w_k, and a bias b. The sum, also called state is the input to an activation function f yielding an output h.

$$\begin{cases} s = \sum_{k=1}^{K} w_k x_k + b \\ h = f(s) \end{cases} \implies h = f(\sum_{k=1}^{K} w_k x_k + b) \tag{2.11}$$

Commonly chosen activation functions, f, are the hyperbolic tangent function, $\tanh(x)$, the rectified linear unit, $ReLU(x) = \max(0,x)$, the logistic sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$, and the softmax function $\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$. A single node is also called a perceptron [Vieira et al., 2020, Chapter 9][Goodfellow et al., 2016]. A visual representation of the perceptron is shown in Figure 2.2.

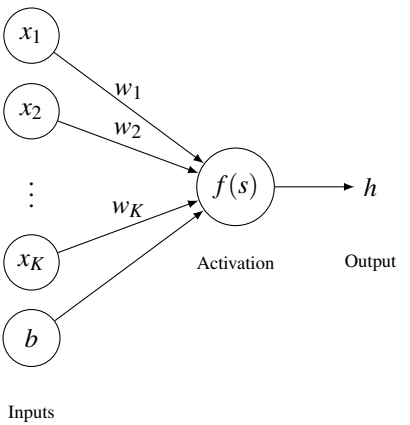


Figure 2.2 Graph representation of a single neural node, the simple perceptron. The inputs x_k are multiplied by their weights w_k and summed together with a bias b and inserted into an activation function $f(s)$ to create the output h. This is a visual representation of (2.11).

To build a network of neural nodes, the nodes exist in layers where the outputs from one layer are the inputs to the next layer. Each node in each layer is connected to all nodes in the next layer, but due to the weights they affect each next node differently. The outputs from the final layer are values between 0 and 1, and each node in the final layer represents a class meaning that the node with the highest value from the final layer will be what the network classifies the inputs as. The first layer is called the input layer, the final layer the output layer and all layers in between are called hidden layers as the internal states are concealed. The network developer is able to choose the amount of layers, which activation function each layer uses and how many nodes are in the hidden layers. The base network is called a multilayer perceptron, MLP, as it consists of several layers of perceptron-nodes. The classical neural network is shown in Figure 2.3

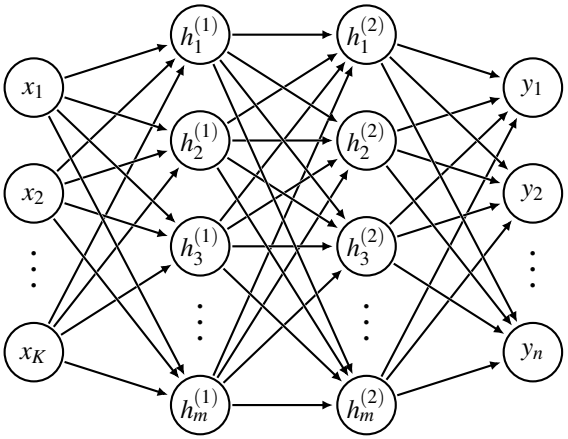


Figure 2.3 Example of a MLP-structure. It has K inputs, 2 hidden layers with m nodes each and n outputs. $h^{(i)}$ denotes that the node belongs to the hidden layer i, x is an input and y is an output.

Training of a network. Training a network means inserting inputs into the network with a known label and give feedback to the network how well its classification for that input went. The weights and biases are changed according to a loss function so that the outputs match the target labels. The final performance after the training is affected by the choice of hyperparameters. Hyperparameters are parameters defining the structure and function of the network. Certain hyperparameters can be tuned during training while others are not modified. Examples of hyperparameters are the size of the network, which activation functions are used, number of training epochs and the learning rate. To evaluate the difference between the output from the network with its target output, loss functions are used. One common loss function used is the mean squared error (MSE),

$$\min_{\mathbf{w}} L(\mathbf{x}, \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.12}$$

where L is the loss, n is the amount of classes, y_i is the output and $\hat{y}_i$ is the target output for each output node i with $\mathbf{x}$ being the inputs to the nodes and $\mathbf{w}$ being the weights. Since the output nodes are perceptrons, the outputs are restricted to the range $[0, 1]$ [Goodfellow et al., 2016, Chapter 5].

To train the network, backpropagation is used. The model adapts the weights in the layers to try and minimize the loss function with respect to the weights using gradient descent. Gradient descent works by taking steps along the negative gradient of a function which is the fastest route towards a minimum.

The gradients are

$$\frac{\partial L}{\partial w_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial w_{ij}}, y_i = f(\sum_{j=1}^{m} w_{ij} x_j + b_i) \implies$$
$$\frac{\partial L}{\partial w_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{w_{ij}} f(\sum_{j=1}^{m} w_{lj} x_j + b_l) = (\sum_{l=1}^{n} \frac{\partial L}{\partial y_l}) f'(\mathbf{x}_m) x_j \tag{2.13}$$

with w_{ij} meaning the weight from node j in the layer with m nodes to node i in the layer with n nodes, f is the chosen activation function, and f' is its derivative. $\mathbf{x}_m$ is the vector of all inputs to the layer with m nodes [Goodfellow et al., 2016, Chapter 6][Hastie et al., 2009, Chapter 11].
Taking the MSE-loss as an example, its gradient is

$$\frac{2}{n} |y_i - \hat{y}_i| f'(\mathbf{x}_m) x_j \tag{2.14}$$

The weights are updated according to

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \gamma^{(t)} \frac{\partial L}{\partial \mathbf{w}^{(t)}} \tag{2.15}$$

with γ denoting the learning rate, also known as the step size, and the index (t) denoting the time point t [Hastie et al., 2009, Chapter 10].

The gradient descent performs best on convex functions as that is the only time that the possibility of reaching a global minimum is guaranteed. Other, non-convex functions may have any number of local minima. This means that there can be several regions where the gradient descent believes to have found a solution, but that is not necessarily an optimal solution. In many cases however, there are functions which have multiple minima with a performance close to the global minimum performance. An example of a non-convex function can be viewed in Figure 2.4.

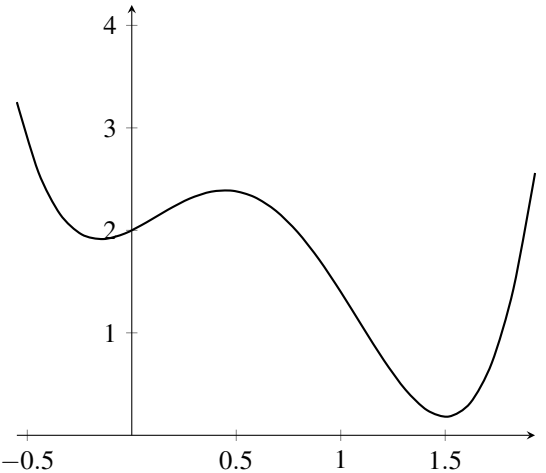


Figure 2.4 Assume the polynomial $f(x) = 2.5x^4 - 6x^3 + 1.9x^2 + x + 2$ with two minima define a set of weights the gradient descent method shall optimize. Depending on the initialization, the training might fit to the left minimum which yields a worse performance than if gradient descent was able to reach the global minimum.

The developer can choose the learning rate, also known as gradient step size, γ. Setting an appropriate value for the learning rate is crucial as too large learning rates will make the training take large steps and risk jumping over a minimum, making the network training volatile. A too small learning rate on the other hand will make the network slow to train and risk not reaching any minima at all or increasing the risk of getting stuck in a local minimum [Deisenroth et al., 2020, Chapter 7].

When all training data has been used once for training using backpropagation, a single epoch has passed. A part of the dataset reserved for validation is passed through the network to test the performance on an unknown dataset. This process is repeated for as many epochs as the developer has defined, but can stop earlier if the loss is minimized and accuracy is maximized earlier. This is called early stopping. To further increase the efficacy of the validation, cross validation can be used. In the very end, after the network is fully trained, the final performance of the network is tested on previously unseen data, separate from the validation data, called a test set [Hastie et al., 2009][Vieira et al., 2020, Chapter 2].

Problems of overfitting and ways to handle it. Achieving high accuracy on training data can many times be misleading due to a common phenomenon known as overfitting. Overfitting means that the network has adapted to finding exact patterns in the training data and has found an optimal set of weights for those patterns, which are often large in magnitude. This can lead to a worse accuracy on new, unknown data. Large weights neglects small differences in the input and thus yield

the same result every time regardless of input, whereas smaller weights retain some variance in them. The larger variance with the smaller weights will give the model a greater chance of not getting stuck in the same path every time and thus have less bias towards a particular solution [Goodfellow et al., 2016].

Regularization is the concept of limiting the magnitude of the network weights to avoid overfitting. Regularization works by having a regularization parameter, λ, multiplied by the weights added to the loss function. Two common regularization methods are the lasso and the ridge regression methods. Lasso utilizes the L_1-norm of the weights

$$\min_{\mathbf{w}} L_{R_1}(\mathbf{x}, \mathbf{w}) = \min_{\mathbf{w}} L(\mathbf{x}, \mathbf{w}) + \lambda ||\mathbf{w}||_1 \tag{2.16}$$

where as the ridge regression on the other hand uses the L_2-norm instead:

$$\min_{\mathbf{w}} L_{R_2}(\mathbf{x}, \mathbf{w}) = \min_{\mathbf{w}} L(\mathbf{x}, \mathbf{w}) + \lambda ||\mathbf{w}||_2^2 \tag{2.17}$$

To minimize the regularization loss, the network has to minimize the chosen loss function together with the weights at the same time. This will limit the size of the weights while at the same time still yielding a high-performing solution, as long as the regularization parameter λ is appropriately chosen [Deisenroth et al., 2020, Chapter 8] [Hastie et al., 2009, Chapter 11].

Another way to limit overtraining is to have a larger training dataset. With more data, the variance in the data for each class naturally increases and thus the weights adapt accordingly. Increasing the training dataset can either be done by collecting more data or by artificially increasing it during the training, for example by applying a rotation and white noise to the training data.

A third way of decreasing the risk of overfitting is by using a smaller, less complex model. A smaller model has less weights and with less weights it becomes more difficult to adapt precisely to the input. This is however a risky way of handling overtraining as a simpler model have the risk of performing worse overall regardless of overfitting or not.

Recurrent neural networks and Long short-term memory. Predicting time-series and strings of text can be difficult using only inputs because of their dependence of previous inputs. It is therefore useful to utilize a previous state in the prediction.

A recurrent neural network, often called RNN, works by feeding inputs as well as previously calculated state values back into the nodes. The feedback effect comes from having the nodes using both the ordinary input $\mathbf{x}^{(t)}$ at time point t and the node's previous output $\mathbf{h}^{(t-1)}$ to create the output $\mathbf{h}^{(t)}$:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}) \tag{2.18}$$

This means that each output is dependent on all previous inputs, and the network has a simple form of memory. The ability to remember previous inputs makes recurrent networks useful in predicting text and time series, as each data point is dependant on the previous point. An example of how a recurrent neural net can look like is shown in Figure 2.5 [Goodfellow et al., 2016, Chapter 10].

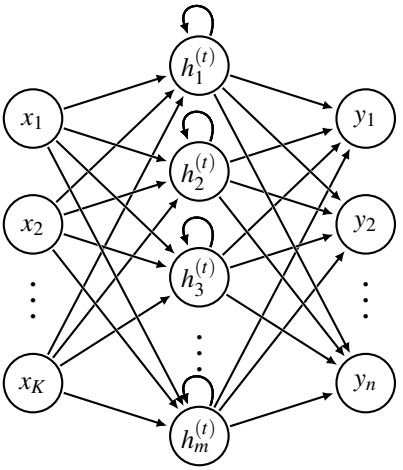


Figure 2.5 Example of how a RNN can look like. It has K inputs, a hidden layer with m nodes and n outputs. $h^{(t)}$ denotes the hidden node at time point t, x is an input and y is an output. The output $h^{(t-1)}$ is used to create the next output $h^{(t)}$ from each recurrent node.

At a point in the future, the next data point is not dependant on the earliest piece of information the network has been shown. In that situation, the network needs to forget the previous state as not to influence the prediction of the next state wrongly. To do this, the importance of the previous state values needs to be lessened. Another problem is that the accumulated gradient can vanish or explode (meaning the gradient tends to zero or infinity) making any new information in the network insignificant. For this problem, the solution is to reset the gradient after a certain time. A gated RNN works by having the ability to reset state values to zero through so called gates. Gates are parts in a node which can nullify certain inputs to the node such as inputs and previous state values.

One of the more common and powerful gated RNNs is called a long short-term memory neural network, LSTM. A node in a LSTM-network is called a cell and also consists of input, forget and output gates. The gates in a cell are supposed to have the ability to dampen the effects of certain parts of the node. The input gate can remove the dependence of the inputs, the forget gate determines how much of the previous state value shall affect the next state value, and the output gate can

make the output from the cell zero. The internal state of the i :th node in a layer at time t $s_i^{(t)}$ is influenced by the previous value of the state $s_i^{(t-1)}$, the forget gate $f_i^{(t)}$, the input gate $g_i^{(t)}$ and the previous output values $\mathbf{h}^{(t-1)}$. The sigmoid activation function is used $\sigma(x) = \frac{1}{1+e^{-x}}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}) \tag{2.19}$$

where

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}) \tag{2.20}$$

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}) \tag{2.21}$$

$$q_i^{(t)} = \sigma(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}) \tag{2.22}$$

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)} \tag{2.23}$$

With $\mathbf{b}$ being biases, $\mathbf{U}$ the input weights and $\mathbf{W}$ the recurrent feedback weights. The indices f, g, o represent the forget gate, external input gate and output gate respectively. The forget gate and the external input gate will have values between zero and one, and determine whether the new state value will mostly be affected by an old state value or the current input to the cell. The output gate is also limited to a value between zero and one, and can effectively temporarily turn off connections between nodes and layers, increasing the flexibility of the network. By having the ability to change the effects of different parts of the node, the LSTM cells are able to be more flexible in regards to the problems it can solve. It has the ability to perform well in situations where the memory length can change and when the importance of the inputs compared to the previous value changes. A model of how the LSTM-cell looks like is shown in Figure 2.6 [Goodfellow et al., 2016].

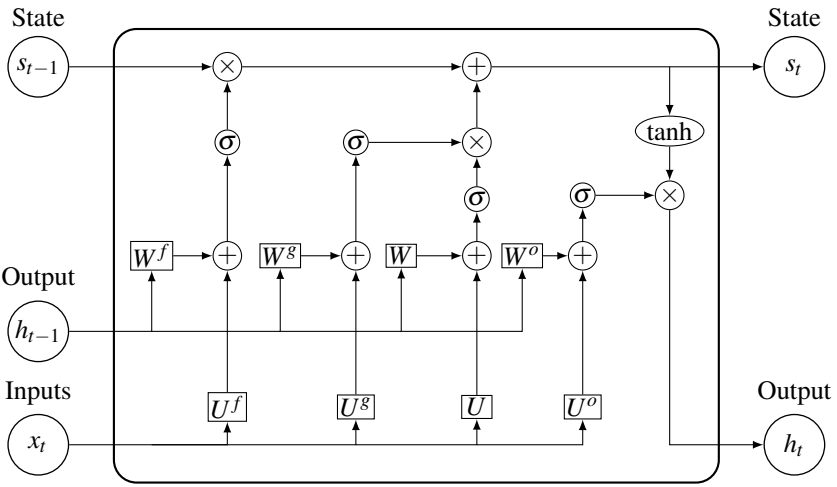


Figure 2.6 Model of the LSTM cell, visualization of equations 2.19-2.23. The biases b are not shown to save space in the image. The state value and output value at time point $t-1$ are used as inputs together with the normal node inputs x at the current time point t.

Convolutional neural networks. A convolutional neural network (CNN) is a network which is powerful at finding patterns in data. It is particularly efficient for equidistant data such as time-series with a fixed sample rate or images consisting of pixels in a grid. Equidistant data can be viewed as arrays or matrices and allows for easy convolutional calculations. CNNs have shown to be able to classify human motions using data from smartphone sensors [Lee et al., 2017], and correctly classify and identify objects in images [Cheeseman et al., 2021].

Like the name suggests, a CNN computes outputs using convolutions between the inputs and smaller sized weight matrices called kernels instead of matrix multiplication between a normal weight matrix and the inputs. Each kernel will be able to detect one feature each, so it is common to use several kernels for an increased performance. Two important properties that are the result of the use of convolution are parameter sharing and sparse connectivity which will be discussed later. The convolutional network can be further enhanced using the pooling technique which is used to generalize the outputs, this concept will also be expanded upon later [Goodfellow et al., 2016, Chapter 9].

The convolution between two continuous, time dependent functions $f(t)$ and $g(t)$ is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.24}$$

which for discrete data such as an input stream $x(t)$ and a weight kernel $w(t)$, with all possible sample times denoted i and the resulting state value $s(t)$, becomes

$$s(t) = (x * w)(t) = \sum_i x(i)w(t-i) \tag{2.25}$$

A two-dimensional CNN has instead the input matrix $X(m,n)$, weight kernel $W(m,n)$ and state $S(m,n)$ for the grid point at m, n. The convolution is now instead

$$S(m,n) = (X * W)(m,n) = \sum_i \sum_j X(i,j)W(m-i,n-j) \tag{2.26}$$

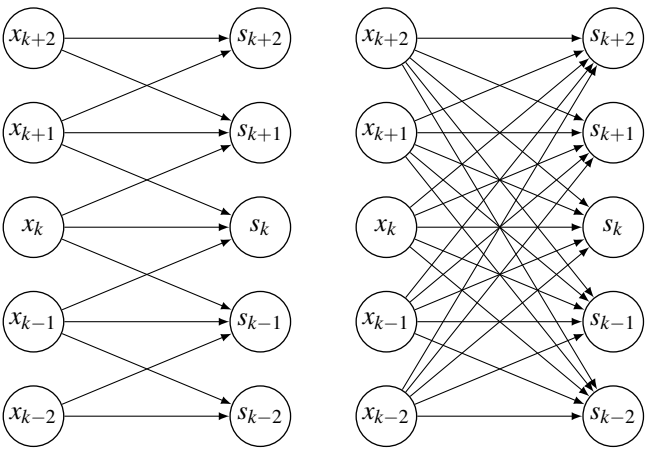
The output from each convolution are features the kernels found with irrelevant parts nullified. Each kernel can find one feature each, so as to extract as much information as possible, it is common to use multiple kernels. As the features are fed forward, each layer will find features in the inputs from the previous layer which are features for all layers except the first. This means that each layer will contain more primitive, but essential, features than the previous. In a two-dimensional CNN detecting facial features this can mean that the first layer detects the shape of an ear but the following layer detects a vertical edge.

A convolutional layer does not necessarily generate a single value in the $[0,1]$-domain, which makes taking the final prediction a difficult task. It is therefore common for CNNs to have a normal MLP-layer as the final decision-making layer. One issue is that each kernel in a convolutional layer generates one channel each, forcing high dimensionality. To get around that, a flattening layer is placed right before the final MLP-layer. The flattening layer works by simply placing all data in a single one-dimensional array which is easy for a MLP to use as an input.

All values in the kernel w will go over all values in the input x when creating outputs regardless of the size of w, meaning the elements in the kernel will be used more than once. This is called parameter sharing. It differs from a normal network where each weight in a weight matrix is only connected to a single input. The parameter sharing allows the kernels to be much smaller than what an equivalent weight matrix would be while retaining the performance.

When the size of the kernel is smaller than the size of the inputs, each calculated state value will only be dependent on a subset of the input since the kernel can only reach parts of the input at the same time. By utilizing sparse connectivity, as it is called, the amount of calculations required in the network is reduced which lowers the memory usage and speeds up the training. Figure 2.7 shows a visualization of sparse connectivity [Goodfellow et al., 2016, Chapter 9].

The convolution finds features in specific locations, and the network learns the feature-location pair. This has the drawback that if any part of the data is translated compared to previously seen examples, the network will miss their patterns. This can be counteracted by using pooling. Pooling means to take a neighbourhood, a pool, of outputs of chosen size from the convolution and calculate a common value for them. The most common type of pooling is the max pooling, which means that the output from the pooling is the maximum value of the chosen points. The way to decide how many steps the pooling-window takes before creating the new pool is called stride. If the pooling has a stride of one, the majority of data points in the new pool will be the same which makes the output likely to be identical to the previous output. The output size will be roughly the same size as the input size. If the stride is larger than one, the output size will be more heavily reduced. For both cases (many outputs being the same and fewer outputs), the amount of detail in the data is reduced. The reduction of detail can be thought of as a form of down sampling. It makes so that the small differences between the same type of data will get eliminated while crucial information remains. Figure 2.8 shows an example of how max pooling works [Goodfellow et al., 2016, Chapter 9].



(a) A sparsely connected network **(b)** A fully connected network

Figure 2.7 On the left, an example of a sparsely connected network where the kernel has size three. Each input only affects three state values and each state is affected by at most three inputs. On the right, a normal fully connected network where all inputs affect all state values.

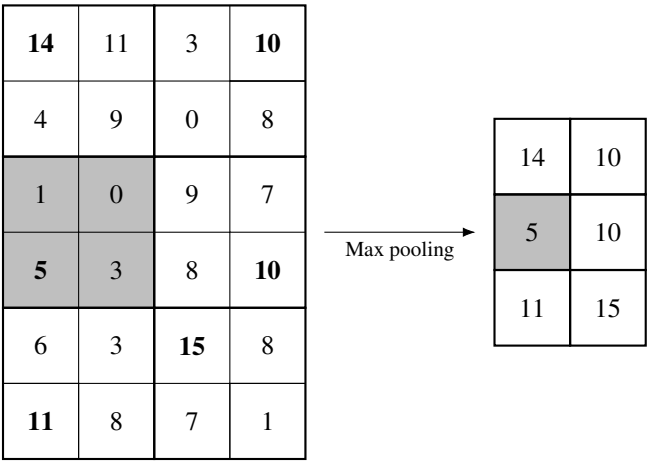


Figure 2.8 Example of a 2×2 max pool with a 2×2-stride acting on a 6×4-grid. The pool moves over the grid taking the maximum value of the looked at part and generates an output of a smaller size than the input. The most important information is retained while the size is reduced. The gray highlighted part of the input shows the neighbourhood generating the gray highlighted output.

Interpretability. Today, many networks are so powerful at classification that they outperform human experts at a fraction of the time. With increasing performances, the network complexity has increased as well. How a network actually classifies data can be hard to understand and as the complexity increases, this task becomes even more difficult. Thus the interest of interpretable neural networks has increased in recent years, where the network not only gives a classification as an output but also an insight into how the decisions are made. Increased interpretability can help understand why a network fails to classify something, notice if parts are redundant and how a non-machine learning approach can be designed to solve the same problem [Mitsioni et al., 2021].

In CNNs, the kernels are arrays, matrices or tensors (dependent on the input dimensions). By extracting and visualizing them, it is possible to interpret the kernels to understand which features they found. For example; A kernel in a one dimensional CNN is an array with two elements equal in amplitude but opposite in sign, that is

$$\mathbf{w} = \begin{bmatrix} -1 & 1 \end{bmatrix} \tag{2.27}$$

The result is that the kernel will take the difference between two adjacent values, see (2.25). If the value the first kernel element affects is smaller than the following value, the output will be positive. If they are identical the output is zero and otherwise the output will be negative.

A second way to visualize features is to take a white noise input and feed it through the node whose feature extraction is of interest. The output from that node is used as input in the same node over and over again until the output of the node is identical to the previous output (or after a certain number of iterations). Anything irrelevant to the node will get suppressed by it, and everything of importance will get amplified. Over time, the output will get optimized to give a maximized output when fed through the node as the relevant information is highlighted more for each pass through [Olah et al., 2017].

Gradient-weighted Class Activation Mapping, known as Grad-CAM, is a recent method for highlighting what parts of an input that was the most important for a CNN performing its classification using a heat map. It works by feeding the inputs as normal through a CNN and getting a classification. The model nullifies all gradients unrelated to the model prediction and the relevant gets maximized. The modified gradients are fed back through the convolutional layers, which has been trained to give large values where the relevant features exists in the original input. The second output gets transformed into a heat map and is placed on the input so the locations of the relevant found patterns are highlighted [Selvaraju et al., 2017].

3

Methodology

This chapter explains all the key elements of the work done. How the data was gathered, labeled and processed is explained followed by the process of building the models and how they were evaluated. The output filtering and how Grad-CAM was applied is described towards the end.

3.1 Data collection and annotation

The dataset consists of sensor streams from an IMU with accelereometer, gyroscope and magnetometer data in the x, y and $z-$directions together with videos of the data collection to help with labeling. The data was labeled using six different classes

- Stationary
- Bike falling over
- Bike lifted
- Bike rolled
- Bike shaken
- Bike picked up from the ground

Several use cases for each class were defined and can be seen in Table B.1 in Appendix B. The different scenarios were meant to increase the variation in the data and reflect the high variance in motions.

The data used was collected on one bike by three different people to mitigate the inherent biases of how motions are carried out. There were a few exceptions where data was collected using other bikes, this data was solely used as testing sets to test the performance on unknown bikes.

The data was all manually annotated with the help of synchronized videos recorded during the data collection. One recording is referred to as a capture and includes all events between a start time and an end time. Figure 3.1 shows one example of how the annotation can look like for an event.

For the time windows, a bike standing still between annotated events in a caption got classified as stationary data. This was mainly done to avoid jumps in the time series and keep it as close to real-time usage as possible. If the bike was moved in the downtime between events such as for correcting the bike's position, that part of the data got an ignore-label. The ignore label was used to make sure that the stationary class does not include other motions. In rare cases it was also to make sure corrupted data was not included. An example of a capture with multiple events with included 'Ignore'-labels can be viewed in Figure 3.2.

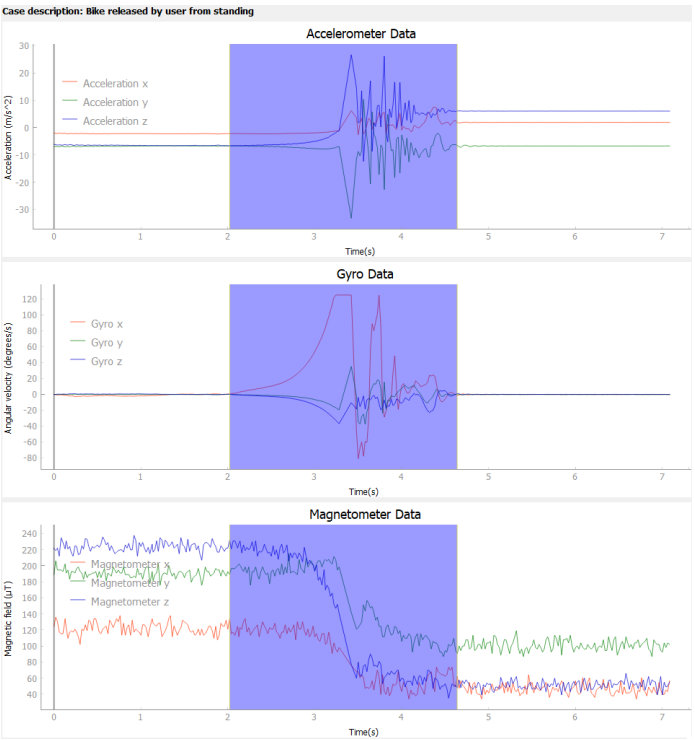


Figure 3.1 Example of an annotation on one split capture of a bike falling over.

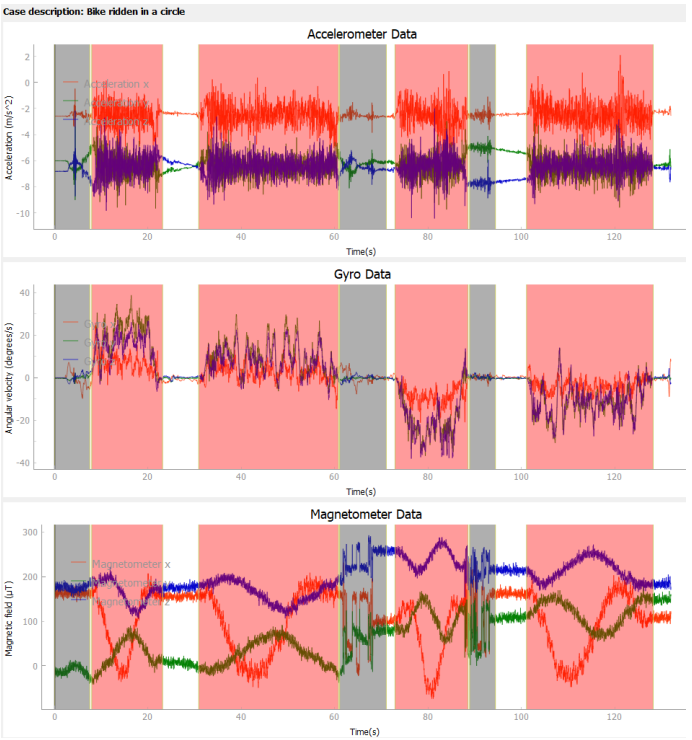


Figure 3.2 Example of a capture with multiple annotated events, highlighted with red. Unwanted parts between the events got the label 'Ignore', highlighted with gray. Unlabeled parts, shown in white, got classified as "Stationary". The data shows multiple rides on the bike.

3.2 Processing

The processing was done using two different methods. In the first method all the data annotated for each event were used. For this method, features were calculated to be used as inputs. In the second method, time windows were used. For the time windows, both features and normalized raw data were used as inputs.

3.2.1 Normalization

A linear transformation of the data was used to normalize the input. The normalization is based on all input values, to guarantee a clear distinction between large and small movements even if they look similar. Each dimension was normalized separately, the values for the $x-$, $y-$ and $z-$directions for each sensor were normalized by themselves. When features were created and used, each feature was normalized individually.

3.2.2 Complete event solution

In the complete event solution method, all events that were annotated were extracted and stored as separate data frames. The frames contained all sample values for each sensor in all orientations for each event and they varied in length depending on the duration of the annotated event. Features were calculated for each frame, normalized and then stored together with the name and class label for easy access. A total of 70 different features were developed, each being one value describing a characteristic of the time series given by the sensors.

This first method was used early on to find out if the classes were separable and to identify features that were descriptive. In a real-time scenario, a processing unit running the model on the bike will need to know when an event begins and ends if this method is to be used. Adding a method to determine starts and stops of events adds a layer of possible errors, and would put a lot of focus on event detection and less on event classification. It would also increase the power consumption in a real case scenario where battery is limited. A decision was therefore made to continue with method using time windows.

3.2.3 Time window creation

Time windows were defined as the second method. A frame moves over the captures, storing a specified part of the data in time windows. The frame moves to the next part of the data and repeats until all data has been put into windows. All created time windows are the same length. The class label assigned to the window was set as the majority of what the time window covers. To find an optimal window length, frames between 0.5 seconds long to 4 seconds were tested. The motivation is that a too short window would lose information about the event. By classifying the window by the most common label of the included data, shorter events risk representing only a minority of the labeled data if the window length is too long and thus miss the event.

Since the windows were labeled by what the majority of the window includes, there can exist cases where an event starts in the second half of the window and the event will stop during the first half of the next window which makes the model completely miss the event despite it existing in two windows. To decrease the risk of missing an event by having the windows poorly placed over the time series, overlaps between the windows were introduced. An example of what that can look like is shown in Figure 3.3. In this thesis, overlaps between 0% and 75% were used.

The features used were calculated on the data in the windows, similar to how the features were calculated for the complete events.

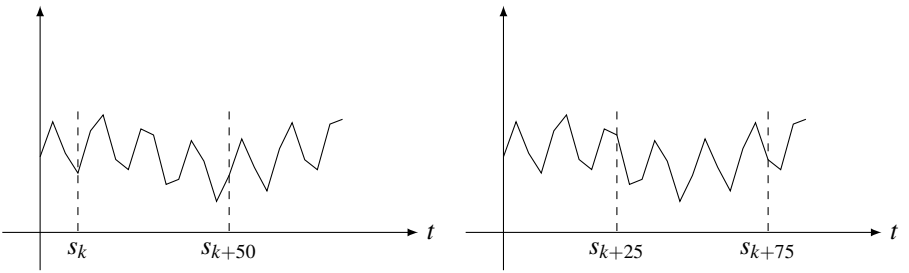


Figure 3.3 Example of how two time windows can be generated. The first window starts by including sample k and every sample up to sample $k+50$. The second window has the same length, but starts at sample $k+25$ meaning there is a 50% overlap between the windows. This generation repeats over the entire capture.

3.2.4 Training and test set

The complete event frames were shuffled and split into a training set and test set where the training consisted of 80 percent of the events and test set 20 percent. For the time windows the distribution was different. Since some of the networks tested contained LSTM layers, it was reasonable to feed the network with frames in a time continuous order. Thus the recorded captures, rather than the windows, were shuffled and split into a $80/20$ percent training and test split.

3.3 Model creation and evaluation

Once all the data was annotated, structured and stored the model creation and evaluation began.

Early tests on feature based KNN showed that there were no large difference between sizes of 1 and 2 seconds and overlaps between 25 and 75 percent. Shorter and longer windows performed worse. To reduce the number of input variations, it was decided to start time windows with an initial value of 1.5 seconds and having a 50 percent overlap during the process of finding an optimal model type. Rigorous evaluation of the different window sizes and overlaps was postponed to after an optimal model type was found.

The KNN-model was the first type of model used on the processed data. Each event was represented with a feature vector and evaluated with 5-fold cross validation. Different combinations of features were tested and for each combination, models with values of k between 1 and 20 were used. Having a total feature count of 70 made it impractical if not impossible to try every combination. However, since most of the features were calculated separately for each sensor and direction which

lowers the total number of feature combinations to try.

The best k found for the features was used to build a model with all training data and evaluated on a test set. After finding the best feature combination, the same algorithm was applied on the time windows data. KNN was not tested using raw normalized input data because each data frame with raw normalized data can be seen as a very high dimensional point as every value in the data will be taken into account, and KNN performs poorly on higher dimensional data due to the curse of dimensionality [Kouiroukidis and Evangelidis, 2011]. Furthermore, using (2.10) on windows of the same class but where the events are not placed similarly within the window, will give poor results. The equation calculates the distance between values from each sample, and thus the distance will be large between windows that captured the beginning end ending of the same event.

The neural networks were built using the Keras library in Python [Chollet et al., 2015], and all networks were tested using different number of layers, nodes/kernels per layer and levels of regularization to find the optimal structure for that network type. The output layer was always a perceptron layer with an equal amount of nodes as the amount of classes.

When training the neural networks the data was given in two ways, both as normalized raw sensor data and as features. When using raw normalized data, the input was two dimensional with size $(\text{Number of samples per window}) \times (\text{Number of sensor streams})$ for all networks. For the 1-dimensional CNN, the kernels were 1-dimensional arrays that only moved in the time-dimension. The different sensor input streams, which could be seen as a second dimension in the data, got treated as different channels by the model.

For the features, two different input structures were used. For the LSTM network the input was a vector of size $1 \times (\text{Number of total features})$. For the 2-dimensional CNN, the features were placed as if they were an image, where each feature value represents a pixel. The features were placed as an array of size $(\text{Number of feature types}) \times 3$. Each row represented a feature type and the columns were the x, y and z value of that feature type.

To find the optimal hyperparameters for each model 5-fold cross-validation was performed on the training data. The model of each type that performed best was then trained with the complete training set and evaluated on the test set. The models tested were CNN, LSTM and CNN followed by LSTM.

The combination network with both CNN and LSTM parts had the inputs first going into convolutional layers, and their output acted as inputs to the LSTM layers. The idea was to get both the pattern recognition from the convolutional layers

in combination with the memory from the recurrent LSTM layers. As each network type was tested with two different inputs, including the two KNNs, it gave eight total models to compare.

Once an optimal model type and parameters were found the model was trained on different combinations of window sizes and overlaps. Since different window sizes means a different sized input in the cases where normalized raw data was used, the input layer naturally varied in size. However, all the hyperparameters remained the same.

Lastly, to see the model's performance in a different setting it was tested on a dataset collected on bikes that were not used in the training.

3.3.1 Grad-CAM

To understand how the models take their decisions, Grad-CAM was used to visualize what information in the inputs was found to be the most crucial. Grad-CAM creates a heatmap which is placed on the input time windows to highlight parts of the data. A more intense red color means the model finds the data point more important for the decision, and a more white color indicates that the information is taken less into account.

Grad-CAM was only applied on models with CNN-layers as it is developed for CNNs [Selvaraju et al., 2017]. It was also only applied on the first CNN-layer as to not have any information distorted by kernels or lost due to pooling.

3.4 Output filtering

With a non-perfect classification, there will be an amount of windows which will get wrongly classified. If the windows are classified in order as they happen, such as in a real life scenario, one can try to reduce the amount of wrong classifications. By using a filtering method which takes into account the neighbouring windows' classes and the confidence of the model, the final output classification can for some outliers be corrected. If the model classifies a long window sequence as the same class, classifies one window as another class, and then continues classifying the following windows as the first class it is not unlikely that the window with an unique class was misclassified.

A filter inspired by the paper by Ravi et al. as described in Chapter 1.2 was developed for this thesis with the aim to reduce the misclassifications by looking at the model confidence and a majority vote of neighbouring windows. The filter walks through a string of time windows classified by the model one at the time. If the classification confidence is above a set confidence level of 90%, the output will

be kept else it will look at the given classifications of windows in a neighbourhood around it. The window focused on will be re-classified as the majority label of the neighbours. It looks at two windows behind and two windows ahead for a total length of five windows. A filter length longer than five could risk filtering out short events and smaller than five could risk not filtering wrongly classified windows on transitions between events. After the filter has gone through all windows in a selected time frame, all windows are re-classified together as not to affect windows that have yet to be filtered. Figure 3.4 is showing an example of how the filter works.

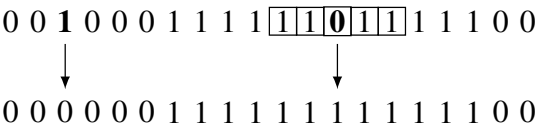


Figure 3.4 An example of how the filter works. There is a string of class 0 followed by a run of class 1 and in the end a return to class 0. The bolded 1 and 0 represent classified windows with low confidence and the boxes represent which windows the filter is looking at in that moment. Due to the neighbourhoods around the low confidence windows, they get changed once the filter has run a lap through.

4

Results

This chapter will focus on giving an overview of the results, and present the best performing models in more detail. For the performance of every model type, see Appendix A. First, the KNN result for the full event solution will be shown. Secondly, an overview of the models' performance will be presented together with more details about the best performing model. Afterwards, a test of what window size and overlap that gives the best performance is listed. Lastly, the result of adding the output filter and examples of the Grad-CAM are shown.

4.1 Model performance

4.1.1 Performance on complete events

When testing different k values for the KNN model it was found that for the full events data, meaning when the start and stop of an event is known and the complete event was included without splits, $k = 4$ gave the best average accuracy of 0.9584% from the cross-validation. Using that k value for a complete model with all training data gave the result presented in Table 4.1 and Figure 4.1.

The performance from the KNN on the full events had the best performance of all variations. Considering the complete information about each event is available, it is not surprising that it outperforms the versions where only limited information is available in each window. However, as mentioned in Chapter 3.2.2, using full events adds problems regarding how a solution would be implemented in a real-time scenario. It would have to include methods to determine when an event starts and stops. With the events being of different lengths, raw data streams cannot be used as inputs as KNN requires data of equal dimensions and the input would be limited to calculated features. Scaling or padding the data can be applied to increase the length of short inputs, however those methods were quickly discarded since it includes a time dependency which could heavily influence the model's decisions based on the time length. Even though the method was not extended further, the test fulfilled its purpose to show that the classes were separable.

Table 4.1 Performance table for the KNN model on features calculated from the data of full events

Class	Precision	Recall	F1-score	Accuracy
Stationary (0)	1.0	1.0	1.0	
Fallen over (1)	0.9796	1.0	0.9897	
Lifted (2)	0.9787	0.9787	0.9787	
Rolling (3)	0.9333	0.9655	0.9491	
Shaken (4)	1.0	1.0	1.0	
Picked up from ground (5)	1.0	0.9375	0.9677	
Global average	0.9819	0.9803	0.9809	0.9816
Weighted average	0.9820	0.9816	0.9816	0.9803

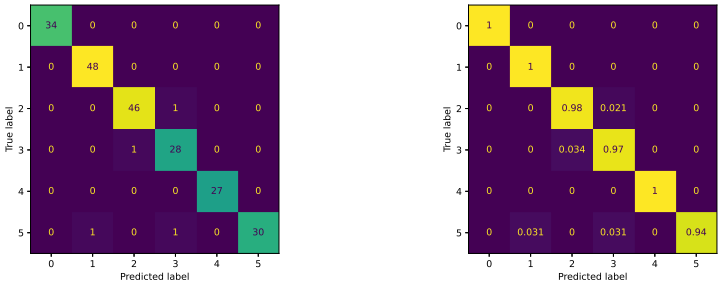


Figure 4.1 Confusion matrices showing the performance for the KNN model on full events. The left matrix shows the amount of classified windows while the right shows the recall percentages.

4.1.2 Performance on time windows

After building and testing all the models with time windows as input and comparing the best achieved performance it was found that the "Lifted" class was more difficult to classify than the rest. In fact the each model's accuracy was correlated to how well it classified "Lifted". In Table 4.2 are the model accuracies and the F1-scores of the "Lifted" class presented.

Table 4.2 The global accuracy and F1-score for all models using time windows as input

Model	Global accuracy	F1-score "Lifted"
KNN (features)	0.9003	0.5762
CNN (features)	0.9210	0.7324
LSTM (features)	0.9116	0.7427
CNN + LSTM (features)	0.9182	0.7338
CNN	0.9299	0.7841
LSTM	0.8843	0.5388
CNN + LSTM	**0.9356**	**0.8302**

The models struggle with the "Lifted" class due to the sensor data for some of the use cases looking similar to either the "Stationary" or "Rolling" class, see Appendix B for a table of the use cases. If the bike is lifted and held in the air for a few seconds and then put down again, a majority of the sensor data, when the bike is held in air, would give the same output as if it stood still on the ground. Using the same reasoning, if the bike is lifted and carried away, the model might interpret the motion as the bike rolling as a large portion of the data is similar to the bike rolling. The sensors experience similar movement, speed and noise-wise, as if it would be rolled slowly.

As seen in Table 4.2, the best performing model is the CNN followed by LSTM network when using normalized raw data as input. The performance is shown in Table 4.3 and Figure 4.2.

Table 4.3 Performance table for CNN-LSTM hybrid network on normalized raw data

Class	Precision	Recall	F1-Score	Accuracy
Stationary (0)	0.8867	0.9302	0.9079	
Fallen over (1)	0.9115	0.9004	0.9059	
Lifted (2)	0.8943	0.7746	0.8302	
Rolling (3)	0.9535	0.9700	0.9617	
Shaken (4)	0.9724	0.9543	0.9633	
Picked up from ground (5)	0.8898	0.9051	0.8974	
Global average	0.9202	0.9063	0.9124	0.9356
Weighted average	0.9357	0.9356	0.9352	0.9063

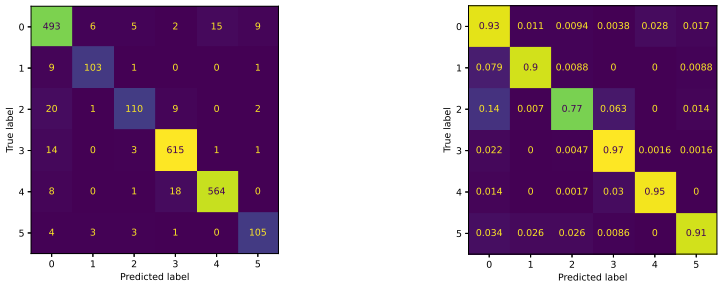


Figure 4.2 Confusion matrices showing the performance for the CNN-LSTM network on normalized raw data. The left matrix shows the amount of classified windows while the right shows the recall percentages.

In general, using normalized raw data over features for the time windows results in a better performance. If this is because features scale down the complexity substantially or if the features chosen in this thesis are not descriptive enough is difficult to say without further research. The feature combination used was derived from experimenting with KNN models, and thus there is no guarantee that they are optimal features for a neural network. Furthermore, for the 2-dimensional CNNs, the position of the features is relevant due to the kernel movement. The positions are not necessarily optimal and a better positional combination could exist.

The results from trying to find the optimal window size and overlap combination on the best performing model, CNN-LSTM network, can be seen in Table 4.4.

Table 4.4 Global accuracy for the different window sizes and overlaps on the CNN-LSTM network on normalized raw data.

	0.5s	1s	1.5s	2s	3s	4s
0%	0.8883	0.8930	0.9064	0.9001	0.8671	0.8128
25%	0.8971	0.9137	0.9266	0.9176	0.8814	0.8160
50%	0.8968	0.9142	0.9356	0.9287	0.8908	0.8282
75%	0.8975	0.9098	0.9340	**0.9396**	0.8963	0.8425

It shows that a slightly better performance was achieved with a 2 second window and 75% overlap. Exactly how well each model performs can vary slightly between each training and therefore one should be careful to confidently say that this would be optimal in a real-time scenario. However, looking at the table, it seems to be preferable to use a window length between 1.5 and 2 seconds. It also seems to be preferable to use more overlap than less. This could be a result of the LSTM layers memory giving a beneficial effect. However, in reality this is likely an effect of having more training data since having more overlap increases the number of win-

dows generated from the time series. Lastly, the performance drop for the 4 second windows was expected. For the "Falling over" and "Pick up from ground" classes, a lot of the events are only about 2 seconds long. Having a 4 second window in those cases includes a lot of samples that does not represent those events and thus a lot of misleading information is included in the classification.

It was also investigated how applying the output filter to the best performing model could improve the performance. The results are shown in Figure 4.3 and Table 4.5.

Table 4.5 Performance table for the final CNN-LSTM hybrid network on normalized raw data with 2 second time windows and 75% overlap after applying an output filter

Class	Precision	Recall	F1-Score	Accuracy
Stationary (0)	0.9311	0.9102	0.9205	
Fallen over (1)	0.9074	0.9545	0.9304	
Lifted (2)	0.8842	0.8077	0.8442	
Rolling (3)	0.9535	0.9872	0.9701	
Shaken (4)	0.9713	0.9658	0.9685	
Picked up from ground (5)	0.9497	0.9379	0.9438	
Global average	0.9329	0.9272	0.9296	0.9465
Weighted average	0.9461	0.9465	0.9460	0.9272

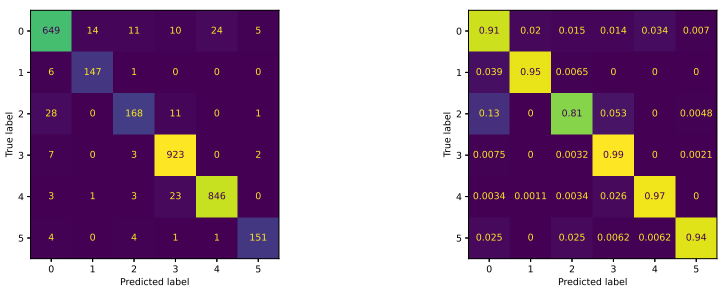


Figure 4.3 Confusion matrices showing the final performance CNN-LSTM network after applying the output filter. The left matrix shows the amount of classified windows while the right shows the recall percentages.

4.1.3 Performance on other bikes

Lastly, results from the tests on data from other bikes that the model has not been trained on can be seen in Table 4.6 and Figure 4.4.

Table 4.6 Performance table for the data on other bikes tested on the final model

Class	Precision	Recall	F1-Score	Accuracy
Stationary (0)	0.8457	0.2338	0.3698	
Fallen over (1)	0.1194	0.1101	0.1146	
Lifted (2)	0.1601	0.2995	0.2087	
Rolling (3)	0.4213	0.7931	0.5503	
Shaken (4)	0.9855	0.0842	0.1551	
Picked up from ground (5)	0.0994	0.5852	0.1699	
Global average	0.4386	0.3510	0.2608	0.3523
Weighted average	0.6525	0.3523	0.3314	0.3510

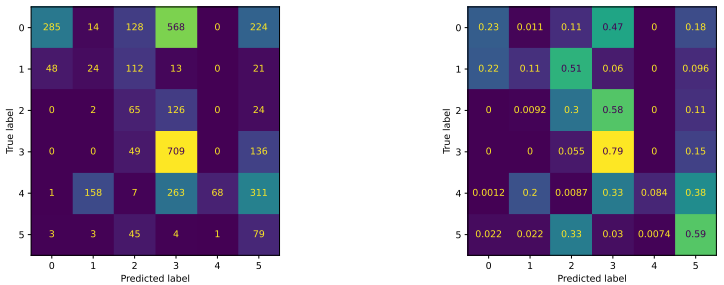


Figure 4.4 Confusion matrices showing performance on data from other bikes tested on the final model. The left matrix shows the amount of classified windows while the right shows the recall percentages.

The performance is significantly worse. Considering the bikes are of different sizes, shapes and weights, it is possible that the sensors measure other accelerations and angle velocities. Naturally this could affect the model's ability to classify other bikes correctly when it has not been trained on them. However, the extreme difference in performance is likely a result of the IMU having a different placement. Suppose the sensor is rotated in any way, then the output would look completely different. A theoretical example is if the IMU is positioned such that the x axis is directly in line with the front of the bike and y being straight up in the air for the original bike trained on. If the new bike's IMU is positioned such that the y axis is in front and the x axis is in the air, a motion moving forward on the new bike would for the model look like it is going up in the air. Thus it is a reasonable result considering that the

rotational position of the IMU is unknown. Ideas of how to potentially solve this problem is discussed in Chapter 5.2.1

4.2 Grad-CAM

Grad-CAM plots were created to highlight the information found to be most important for decision making in the models. It was only applied on the CNN-models and only the first layer is visualized. The plots show accelerometer data on top and gyroscope data on the bottom with the same heatmap placed on both. A data point with a more red part in the heatmap is what the model found more important.

In Figure 4.5 it appears as if the model thinks that the point where the gyro reaches a peak value and then quickly declines is the point of most importance. It is understandable that the reasoning could be that the angle of the bike was changing rapidly until it suddenly stopped which could be a bike falling until it hit the ground.

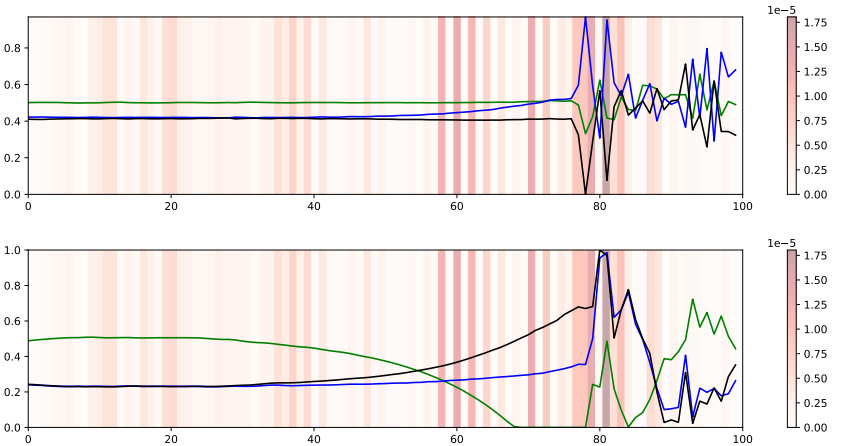


Figure 4.5 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike falling, which it correctly predicted with 99% confidence.

In Figure 4.6, the motion was "Lifted" but it got classified as "Rolling". The window showing the class "Rolling" in Figure 4.7 is not dissimilar from the "Lifted" window in Figure 4.6 so it is understandable why the model predicted wrong. Why the samples around sample 30 and sample 50 in the wrongly predicted window shown in Figure 4.6 are the most important for the model prediction is hard to see as there are no clear changes in the data there.

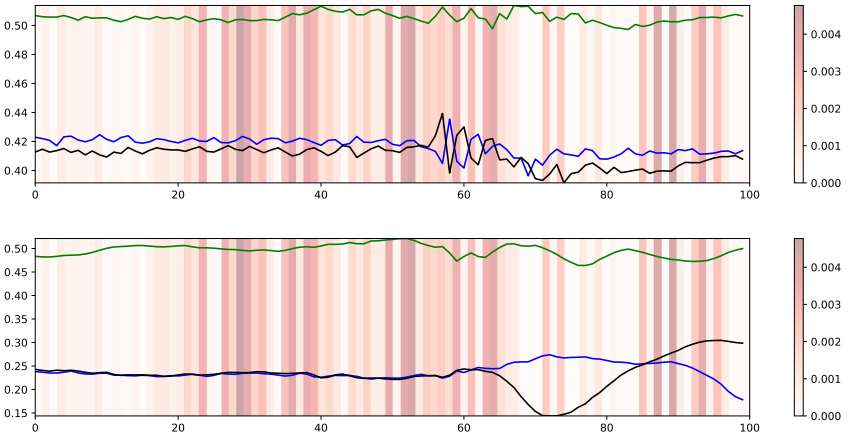


Figure 4.6 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike getting lifted, however it predicted it to be rolling with 52% confidence. The model gave the probability 38% of the window showing a lift, the correct class.

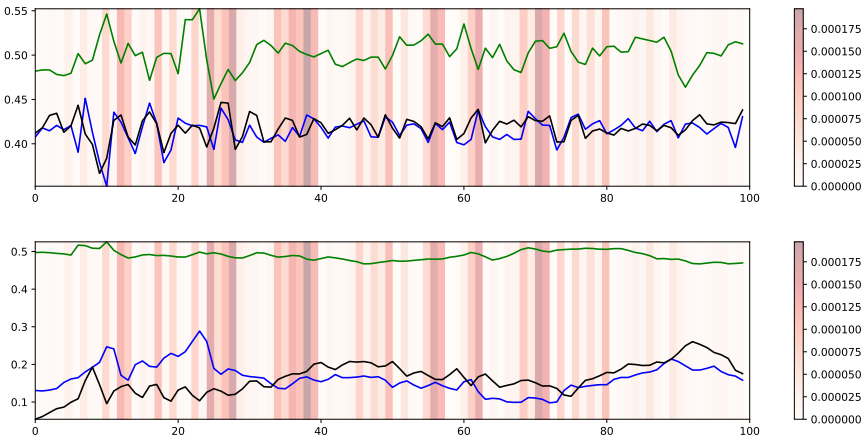


Figure 4.7 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike rolling, which it correctly predicted with 98% confidence.

Figure 4.8 shows a correctly predicted window of the class "Shaken". In the span of samples $30 - 40$ seems to be where the model finds the points of most importance, and the single most important point happens where all accelerometer data axes has the roughly same value which is a pattern that can be seen to repeat.

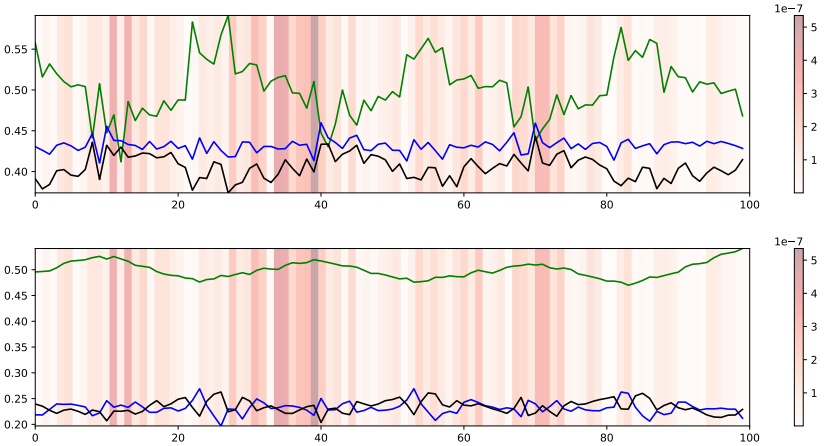


Figure 4.8 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike shaking, which it correctly predicted with 99% confidence.

Figure 4.9 shows the beginning of the bike falling over, but the model classified the time window as stationary. It is understandable why the model classifies the bike as stationary since a large part of the window has sensor values which seems to only change within noise levels. The gyroscope starts changing from around sample 45, where the Grad-CAM says the most important information is but the accelerometer does not seem to change until later which is likely why the classification differs from the ground truth.

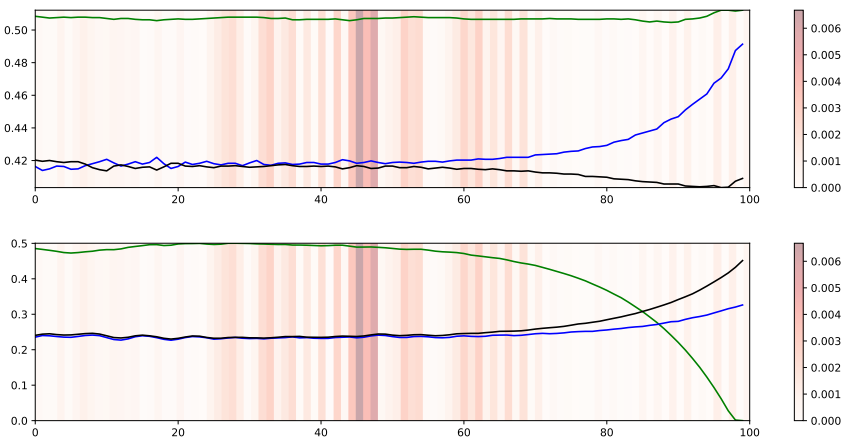


Figure 4.9 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike falling, which it predicted as stationary with 76% confidence.

5

Discussion and conclusions

In this chapter, it will be discussed how the event classification could be better than the window classification by looking at groups of windows. The performance will then be compared to performances from other papers on ML solutions to the AR problem. Thirdly the interpretation of the Grad-CAM will be discussed more thoroughly. Lastly, future improvements and application areas will be mentioned followed by some final conclusions.

5.1 Model evaluation

The KNN model in the full event solution reached an accuracy of 98.16%. Without context this performance is close to perfect and the few misclassifications are spread between 3 classes. However, as mentioned previously, this solution would be difficult to implement without a highly accurate event detection method already implemented.

The model in the time window solution achieved an accuracy of 94.65%. This accuracy is relatively high as well and most classes achieves high precision and recall scores. The "Lifted" class is the class which is the most difficult to classify. Apart from the class "Lifted" the most common misclassifications is between "Stationary" and each respective class. The reason for this could be that at the beginning and end of each event, unless the window is perfectly placed, data representing that event and stationary data will be included in the same window. The windows were labeled as the majority of what the data points in the windows were annotated as. However, if the annotated data is close to an equal split between the classes in the window, it is not surprising if the model has a difficult time deciding between the classes. Figure 4.9 illustrates an example of one of these occurrences. The window is labeled as "Falling over" and looking at the right side of the plot one can see the beginning of the fall which seems to start right before sample 50. It takes about 20 more samples until all axes of the sensors are displaying a change in output. Thus a large portion of the output looks stationary in this window and it is not surprising

that the model predicts it that way. Errors like these are the most common of the misclassifications and thus the accuracy does not perfectly reflect over the model's ability classify each event.

5.1.1 Performance interpretation

The final model is able to accurately classify the windows of most classes correctly. However, the goal of the thesis was to detect occurrences of events and what that event was rather than correctly classifying each single window. Consider the following two real examples of the model's output.

Output

0 5 5 5 5 5 2 5 2 5 0

Ground truth

0 5 5 5 5 5 5 5 5 5 0

Figure 5.1 An example from the model's filtered output and the ground truth. The numbers represent the class of each window

Looking at the output in Figure 5.1, without considering the ground truth, it is more likely that the two "2"s have been wrongly classified than the seven "5"s. One can confidently say from looking at the output that the bike has been picked up from the ground. In this case the ground truth confirms this.

Output

0 0 2 2 2 0 1 1 1

Ground truth

0 2 2 2 2 0 1 1 1

Figure 5.2 An example from the model's filtered output and the ground truth. The numbers represent the class of each window

In Figure 5.2 it seems as if the bike has been lifted in the air, and later fallen over. Once again the ground truth confirms this even though the model wrongly classified the first window in the event of lifting the bike. These examples are two of many in the output that experience similar window errors. The applied output filter was meant to correct misclassified windows like in these examples, but clearly failed to do so. This indicates that a better post-processing method could be one way to improve the performance further. Developing a better output filter less reliant on model confidence would be one way to do it.

To better understand the model's ability to classify events instead of windows a

different evaluation method could be used. If all windows in each event for the ground truth are grouped together and then compared to the majority of the windows in the same grouping from the output, an accuracy for each event would be achieved. In the second of the examples above it would result in the first "0" being compared, then the four "2"s would be compared to the majority of "0" and the three "2"s, and so on. This method would in the two examples above give 100% accuracy, and correctly so, for event classification. This evaluation was never implemented due to time constraints.

5.1.2 Model comparisons

The models developed for this thesis are able to achieve an accuracy of 98.16% and 94.65% for the full event and time window solutions respectively. This performance does not reach the level of accuracy found in the literature solving the human activity recognition problem. Most papers had versions of time windows as input and thus the most suitable model to compare is the 1-dimensional CNN followed by LSTM network reaching 94.65%. A few examples include Filip Malawski and Bogdan Kwolek who achieved a 98.18% accuracy classifying footwork motions in fencing [Malawski and Kwolek, 2016], Tsige Tadesse Alemayoh, Jae Hoon Lee and Shingo Okamoto who reached an accuracy of 99.5% classifying different categories of human movement [Alemayoh et al., 2019], and lastly Nishkam Ravi, Nikhil Dandekar, Preetham Mysore and Micheal L. Littman with an accuracy of 99.57% classifying daily activities [Ravi et al., 2005]. These papers, including this thesis, had similar data settings in regards that the training and test data was collected using the same subjects. The solutions reaching these performances were using Dynamic time warping, 2-dimensional CNN or plurality voting between several base level classifiers such as KNN and SVM. The multitude of different methods reaching high performances indicates that AR problems can be solved in several ways.

When comparing model performances between data settings, it becomes evident that the models all have a problem with classification when the test data and the training data come from different subjects. The model in this thesis dropped from 94.65% accuracy to 35.23% when trained on the data from one bike and tested on data from other bikes. The same phenomenon was found in two of the papers referred to above. Malawski et al. had similar problems as the performance dropped from 98.18% to 56.75%. They however achieved a better performance of 70.71% using an SVM classifier instead of Dynamic time Warping on data from different subjects. Although, it is still far lower than the performance achieved on the same subjects. Ravi et al. had their performance drop from 99.57% accuracy to 65.33%. This is a problem that is of high importance in classification problems considering most, if not all, application areas involves more subjects than those available to train on.

5.1.3 Model interpretability

The Grad-CAM figures 4.5-4.9 and A.7-A.12 are to a varying level understandable. The important values are more evident in some of the examples, such as Figure 4.5 and Figure 4.8. In both figures, the heatmap highlights the values when the gyroscope is at its peak. In Figure 4.8 the important values also seem to align with change of directions in the accelerometer values. Some Grad-CAM heatmaps looks to be striped such as figures 4.5 and 4.7 where some points are more highlighted followed by others with less which could indicate some repeating hidden pattern. In examples like Figure 4.6 and Figure 4.7, what values the model found important are more scattered and does not necessarily seem to coincide with any maximal or minimal value, nor any change of direction.

The Grad-CAM heatmap is dependent on the kernels, absolute values and value changes on all six input sensor streams. However, the heatmap output on each sample is only 1-dimensional. A combination of values from the different input stream can both amplify and nullify the heatmap output. Thus it can be difficult to confidently say why certain samples are of more importance and find the correlation between the heatmap and sensor streams.

When considering all Grad-CAM examples, it seems as the important values from the heatmap coincide with the maximal values of the gyroscope in a lot of the cases which indicates that the gyroscope data has a large impact on the model's decisions. Another conclusion that can be drawn is that the heatmap is less interpretable for motions where the data is interchangeable within the event, such as "Stationary" and "Rolling". The data in these motions looks very similar regardless of how the windows are placed in the event and it is difficult to understand why certain samples are more important. For motions like "Falling over" and "Shaken", there are clear parts of the data where something different has happened. In those examples it is more obvious that a rapid change in acceleration and angular velocity means that the bike is for example falling, and that model correlates these increased values with the "Falling over" class.

Grad-CAM does increase the model interpretability, however, not to the degree that anyone could look at an arbitrary time window and understand the decision. The interpretability could be improved further by using other methods in addition such as plotting the trained kernels or finding an LSTM-based method of visualizing the decision making. The paper "Increasing the Interpretability of Recurrent Neural Networks Using Hidden Markov Models" proposed a usage of hidden Markov models for visualization of RNNs in their paper from 2016 [Krakovna and Doshi-Velez, 2016]. Interpretability of the KNN classifier could be made by plotting a 2-dimensional projection of the data points with markings to differentiate between classes.

With improved interpretability the final performance of the models could be increased by tuning the models to the dataset by using better suited features or structuring the models differently. Considering that the gyroscope values seems to be of high importance, developing better features for the gyroscope data could be a way to increase the performance. An understanding of the decisions made for classifications can also help inspire solutions to the problems without the use of ML.

5.2 Future work

In this part, different ways the thesis could be expanded upon are described followed by some possible use cases.

5.2.1 Possible extensions

More motion patterns to detect. Currently the models are rather easy to expand to include more motions. Classes that could be added are for example parking the bike and crashing with the bike. Certain current motion patterns could be separated for more detail. "Rolling" could be separated into "Rolling" and "Riding the bike", and "Lifted" could be separated into "Lifted" and "Carrying".

KNN does not contain an inner structure in need of a change as the amount of labels it can classify depend only on the amount of labels present in the training data. There could exist a need to tune the k-value, however testing would be needed to verify this.

Neural networks can classify as many classes as there are output nodes, so one apparent change required is to include more nodes in the final layer. Adding more classes increases the complexity of the problem so there might exist a need to change the inner structure. The current networks could however be powerful enough to handle additional classes without needing a change of the inner structure.

Test other machine learning methods. To evaluate which machine learning method is best suited for the problem of activity recognition on bikes more generally, there is a need to study different methods as well. Some other methods which could be evaluated are support vector machine, decision trees, logistic regression and random forests. All these methods have shown previously to solve the activity recognition problem with accuracies in level with or higher than achieved in this thesis [Minarno et al., 2020].

Split network. Different motions can vary wildly in duration so having one single network to detect all motions is not necessarily the best option. Since the network needs to be general enough to catch both long and short motions, there might exist

compromises in parameters such as window length and the final network weights.

The longer motions are often time invariant, meaning that it is possible to classify the motion regardless of when the input to the model starts. Some examples include the "Rolling" class and the "Stationary" class. Shorter motions on the other hand are often event based, meaning different things happen during different parts of the motion and missing parts can lead to a wrong classification. Motions like the bike falling over and the bike getting picked up are some examples.

A model consisting of three networks does not have to compromise its parameters to the same degree as each network can specialize into detecting different things. The first network will only detect if the motion is time invariant or event based and then depending on the class, feed the input into one of two following networks. The first of these networks should be trained on only time invariant motions and will be able to classify which type of time invariant motion it is. The second will similarly be fed the raw input but instead classify the event based motions. Each network will then be specialized to each particular type and the model performance could be higher. Figure 5.3 shows a flowchart of the model idea.

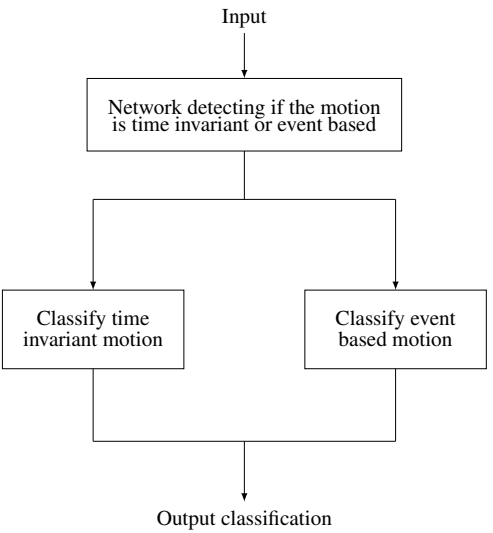


Figure 5.3 Model consisting of three networks. The first detects if the motion is time invariant or event based. Depending on what it detects, the input is fed into one of two networks. One network should be trained to classify time invariant motions only and the other should be trained on event based motions only.

Real-time implementation on the bike. To get the model to run live on the bike, it needs to be efficient considering it has access to fewer resources in terms of processing power and battery usage. It is therefore crucial to convert the model and data processing from Python code to a more memory and computational efficient language like C or C++. The code can either be re-written by hand or use compilers to compile and optimize the code [Chen et al., 2018]. It would also be important to reduce the size of the model as well to decrease complexity, hopefully without a large cost in terms of model accuracy.

With a live implementation on the bike, the model should collect several windows before running the filter as to have the filter working properly. The filter should then run at the same speed as the window creation to have as small delay between data collection and final decision as possible.

Generalization. As mentioned, the model seems to have problems classifying motions on any bike for which it has not been trained on. This is believed to be an effect of the IMU orientation being different on different bikes. One possible way to solve this would be to have a lot of data from a large amount of different bikes with the IMU being placed in various locations. Hopefully the model would then be able to learn that the data for the same motion can look different between bikes and it would be able to find new patterns in the data that would still distinguish each class. There is a risk that the problem would be too complex if the orientation could be anywhere in space. However, increasing the amount of data is often a way to improve any machine learning algorithm.

Another solution that was discussed during the process of this thesis was to rotate the data to a fixed orientation. In theory this would probably be very effective, however it is difficult to implement without knowing the original position of the sensors. If two vectors of the sensors were known one could calculate the third, but the only one that could be known is the gravitational vector. With calibrated sensors positioned in fixed orientations a better result could be achieved. The possibility to try this requires more time and resources then was available during this thesis.

Gravity will induce a DC offset in the accelerometer data which will vary between the x, y, and z-components depending on the rotation of the sensor. This means that if the data is high pass filtered such that zero-frequency data is removed, the model should be more robust against differently rotated sensors. This would be a third possible way to increase the performance on general data.

5.2.2 Possible applications

Improved eBike alarm. One use case for a motion detecting algorithm on eBikes is an improved theft detection alarm. A current common alarm algorithm works by detecting the motion level of the bike, and if it is higher than a threshold for a

time the alarm triggers. If the bike is moving violently because of a fall the alarm should not trigger, where as the bike getting carefully and slowly rolled away by someone should trigger despite low motion levels. Additional feedback from which type of motion is happening can therefore help to trigger an alarm quicker for certain motions and later for other motions.

Inform user of bike fallen over. Another possible application of interest for bike owners is to inform the owner if their bike has fallen over. Suppose someone places their bike in a public place where others also place their bikes. Most people can probably recognize seeing bikes lying on the ground due to reckless behavior from others or from strong winds. While on the ground the bike is more exposed to suffer damages. Thus informing the owner could help prevent the bike from being damaged.

Applying method on similar topics. The results for this thesis shows that it is possible to perform activity recognition on bikes. However, nothing indicates that the approach is limited to bikes. Thus it should be possible to use the approach in any activity recognition problem, with respect to features, models and parameters likely being problem specific.

5.3 Conclusion

This thesis shows that machine learning is a suitable tool to solve the problem of classifying eBike motion patterns. The final filtered output accuracy rate of 94.65% is almost at a level where bike owners can trust the output, especially when considering mainly event detection rather than correctly classifying all time windows as discussed in Chapter 5.1.1. The lowest performance of all developed models is 88.43% indicating that all tested machine learning methods are suitable for motion pattern classification, with the most suitable being a combined CNN-LSTM neural network. The performance of the model developed for this thesis is not at the level of other solutions to the AR problem brought up in Chapter 5.1.2.

Testing the model performance on data collected using other bikes with a different IMU placement showed that the models does not generalize to a level where it could be applied to any bike. A solution would be to either train the models using a much larger dataset with data collected on many bikes with different IMU orientations or have a method which makes the data spatially independent. Putting effort into solving the problem on an arbitrary bike would allow for future applications that include but is not limited to improved eBike theft detection and a warning to owners that their bike has been laying on the ground for a while, risking damages.

To handle the problem of events being of different lengths, the data was placed in equally sized time windows with overlaps. The best performing time window

was 2s long and had an overlap of 75%, however a time window combination with a length of 1.5s and 50% overlap performed with almost the same accuracy. The third and fourth best performing time windows had a 1.5s length with 75% overlap and a 2s length with 50% overlap respectively. The small difference in performance indicates that the optimal time window has a length and overlap in those ranges.

A filter was developed to reduce misclassifications. It reclassified windows with low model confidence according to the majority label of its neighbouring time windows, and was able to increase the performance of the best performing model from 93.96% accuracy to 94.65%. The filter was simple and increased the accuracy only slightly, indicating room for improvements. An improved filter should create complete events by grouping classified windows together and use an evaluation based around classifying complete events, to align more with the final goal of event detection.

The use of Grad-CAM increases the interpretability of the developed CNN based models by highlighting what parts of the input is considered most crucial when the model takes its decision. The generated heatmaps seems to indicate that the gyroscope data is most crucial for the models as the most highlighted points often coincide with maximal or changing gyroscope values. The knowledge that the gyroscope data was important can be used to develop better features, and that a non-ML algorithm should utilize the gyroscope data to a large degree. A large portion of the heatmaps are difficult to understand, suggesting the models take complex decisions. With Grad-CAM only applicable on CNN based models, another method for increasing the interpretability of the other ML models is needed for a more complete understanding of the decision making process.

Bibliography

Alemayoh, T. T., J. H. Lee, and S. Okamoto (2019). "Deep learning based real-time daily human activity recognition and its implementation in a smartphone". In: *2019 16th international conference on ubiquitous robots (UR)*. IEEE, pp. 179–182.

Brown, N. and T. Sandholm (2018). "Superhuman ai for heads-up no-limit poker: libratus beats top professionals". *Science* **359**:6374, pp. 418–424.

Cheeseman, Southerland, and Park (2021). "Advanced image recognition: a fully automated, high-accuracy photo-identification matching system for humpback whales". *Mammalian Biology*.

Chen, T., T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy (2018). *Tvm: an automated end-to-end optimizing compiler for deep learning*. DOI: 10.48550/ARXIV.1802.04799. URL: https://arxiv.org/abs/1802.04799.

Chollet, F. et al. (2015). *Keras*. https://keras.io.

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Deisenroth, M. P., A. A. Faisal, and C. S. Ong (2020). *Mathematics for machine learning*. Cambridge University Press. ISBN: 9781108679930. URL: https://mml-book.com.

Fulcher, B. D. and N. S. Jones (2014). "Highly comparative feature-based time-series classification". *IEEE Transactions on Knowledge and Data Engineering* **26**:12, pp. 3026–3037.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Hastie, T., R. Tibshirani, J. H. Friedman, and J. H. Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.

Kate, R. J. (2016). "Using dynamic time warping distances as features for improved time series classification". *Data Mining and Knowledge Discovery* **30**:2, pp. 283–312.

Kouiroukidis, N. and G. Evangelidis (2011). "The effects of dimensionality curse in high dimensional knn search". In: *2011 15th Panhellenic Conference on Informatics*. IEEE, pp. 41–45.

Krakovna, V. and F. Doshi-Velez (2016). "Increasing the interpretability of recurrent neural networks using hidden markov models". *arXiv preprint arXiv:1606.05320*.

Kramer, O. (2013). "K-nearest neighbors". In: *Dimensionality Reduction with Unsupervised Nearest Neighbors*. Springer Berlin Heidelberg. ISBN: 978-3-642-38652-7. DOI: 10.1007/978-3-642-38652-7_2. URL: https://doi.org/10.1007/978-3-642-38652-7_2.

Lee, S.-M., S. M. Yoon, and H. Cho (2017). "Human activity recognition from accelerometer data using convolutional neural network". In: *2017 ieee international conference on big data and smart computing (bigcomp)*. IEEE, pp. 131–134.

Madgwick, S. O. H., A. J. L. Harrison, and R. Vaidyanathan (2011). "Estimation of imu and marg orientation using a gradient descent algorithm". In: *2011 IEEE International Conference on Rehabilitation Robotics*, pp. 1–7. DOI: 10.1109/ICORR.2011.5975346.

Malawski, F. and B. Kwolek (2016). "Classification of basic footwork in fencing using accelerometer". In: *2016 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*. IEEE, pp. 51–55.

Minarno, A. E., W. A. Kusuma, and H. Wibowo (2020). "Performance comparisson activity recognition using logistic regression and support vector machine". In: *2020 3rd International Conference on Intelligent Autonomous Systems (ICoIAS)*. IEEE, pp. 19–24.

Mitchell, T. M. and T. M. Mitchell (1997). *Machine learning*. McGraw-hill New York. ISBN: 0070428077.

Mitsioni, I., J. Mänttäri, Y. Karayiannidis, J. Folkesson, and D. Kragic (2021). "Interpretability in contact-rich manipulation via kinodynamic images". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 10175–10181.

Nguyen-Tuong, D. and J. Peters (2011). "Model learning for robot control: a survey". *Cognitive processing* **12**:4, pp. 319–340.

Olah, C., A. Mordvintsev, and L. Schubert (2017). "Feature visualization". *Distill* **2**:11, e7.

OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang (2019). "Dota 2 with large scale deep reinforcement learning". arXiv: 1912.06680. URL: https://arxiv.org/abs/1912.06680.

Patro, S. and K. K. Sahu (2015). "Normalization: a preprocessing stage". *arXiv preprint arXiv:1503.06462*.

Powers, D. M. (2020). "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation". *arXiv preprint arXiv:2010.16061*.

Proakis J. G., M. D. G. (1996). *Digital Signal Processing: Principles, Algorithms, and Application (Third edition)*. Prentice-Hall International. ISBN: 978-0133737622.

Ramasamy Ramamurthy, S. and N. Roy (2018). "Recent trends in machine learning for human activity recognition—a survey". *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **8**:4, e1254.

Ravi, N., N. Dandekar, P. Mysore, and M. L. Littman (2005). "Activity recognition from accelerometer data". In: *Aaai*. Vol. 5. 2005. Pittsburgh, PA, pp. 1541–1546.

Russell, S. J. (2010). *Artificial intelligence a modern approach*. Pearson Education, Inc.

Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra (2017). "Grad-cam: visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*, pp. 618–626.

Sewak, M. (2019). *Deep reinforcement learning*. Springer.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. (2016). "Mastering the game of go with deep neural networks and tree search". *nature* **529**:7587, pp. 484–489.

Vieira, S., W. H. Lopez Pinaya, R. Garcia-Dias, and A. Mechelli (2020). "Machine learning". In: Mechelli, A. et al. (Eds.). Academic Press. ISBN: 978-0-12-815739-8.

Wang, X., K. Smith, and R. Hyndman (2006). "Characteristic-based clustering for time series data". *Data mining and knowledge Discovery* **13**:3, pp. 335–364.

A

Detailed results

Appendix containing the performance tables, confusion matrices and Grad-CAM figures not presented in the article.

A.1 Model performances

The best k found during the cross-validation of the time windows data was $k = 7$ which gave an average accuracy of 0.8690%. The detailed performance when trained with all training data and tested on the test set is shown in Table A.1 and Figure A.1.

Table A.1 Performance table for the KNN model on features calculated from time window data

Class	Precision	Recall	F1-score	Accuracy
Stationary (0)	0.9116	0.8754	0.8931	
Fallen over (1)	0.8727	0.8421	0.8571	
Lifted (2)	0.6931	0.4930	0.5762	
Rolling (3)	0.8839	0.9842	0.9314	
Shaken (4)	0.9479	0.9543	0.9511	
Picked up from ground (5)	0.9151	0.8362	0.8739	
Global average	0.8707	0.8309	0.8471	0.9003
Weighted average	0.8969	0.9003	0.8965	0.8309

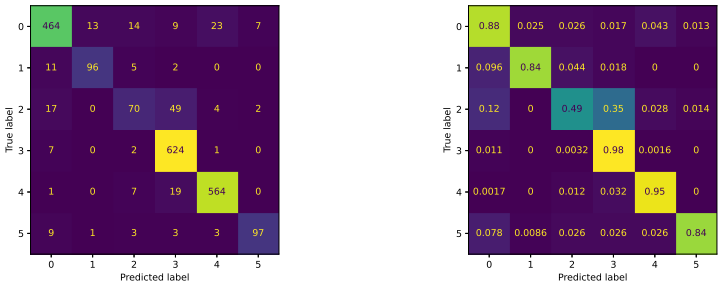


Figure A.1 Confusion matrices showing the performance for the KNN model on time windows. The left matrix shows the amount of classified windows while the right shows the recall percentages.

Table A.2 Performance table for LSTM model on normalized raw data

Class	Precision	Recall	F1-score	Accuracy
Stationary (0)	0.8447	0.9340	0.8871	
Fallen over (1)	0.9326	0.7281	0.8178	
Lifted (2)	0.7662	0.4155	0.5388	
Rolling (3)	0.8699	0.9495	0.9080	
Shaken (4)	0.9560	0.9205	0.9379	
Picked up from ground (5)	0.8596	0.8448	0.8521	
Global average	0.8715	0.7987	0.8236	0.8843
Weighted average	0.8834	0.8843	0.8786	0.7987

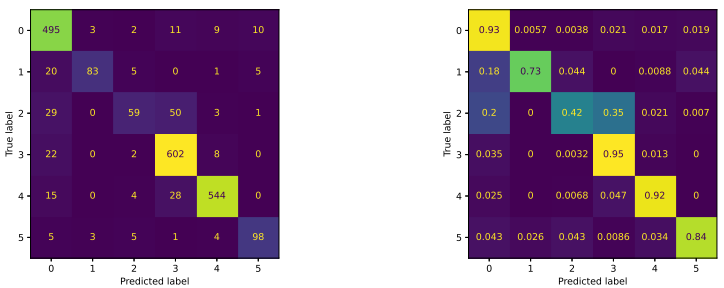


Figure A.2 Confusion matrices showing the performance for the LSTM network on normalizedraw data. The left matrix shows the amount of classified windows while the right shows the recall percentages.

Table A.3 Performance table for LSTM network on features

Class	Precision	Recall	F1-score	Accuracy
Stationary (0)	0.9100	0.8774	0.8934	
Fallen over (1)	0.8667	0.7982	0.8310	
Lifted (2)	0.7769	0.7113	0.7427	
Rolling (3)	0.9331	0.9685	0.9505	
Shaken (4)	0.9435	0.9612	0.9523	
Picked up from ground (5)	0.9009	0.8621	0.8812	
Global average	0.8862	0.8631	0.8739	0.9116
Weighted average	0.9103	0.9116	0.9105	0.8631

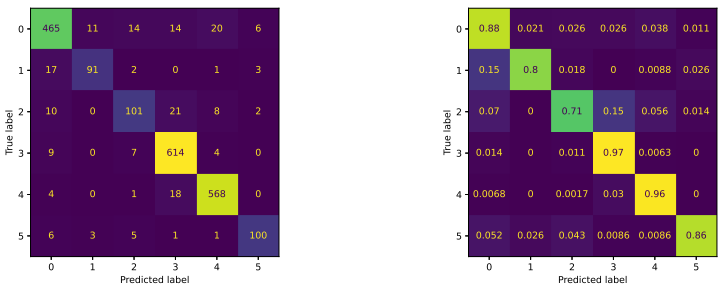


Figure A.3 Confusion matrices showing the performance for the LSTM network on features. The left matrix shows the amount of classified windows while the right shows the recall percentages.

Table A.4 Performance table for CNN model on normalized raw data

Class	Precision	Recall	F1-Score	Accuracy
Stationary (0)	0.9056	0.9226	0.9140	
Fallen over (1)	0.9266	0.8860	0.9058	
Lifted (2)	0.8015	0.7676	0.7841	
Rolling (3)	0.9517	0.9637	0.9577	
Shaken (4)	0.9693	0.9611	0.9652	
Picked up from ground (5)	0.8772	0.8621	0.8696	
Global average	0.9053	0.8938	0.8994	0.9299
Weighted average	0.9297	0.9299	0.9297	0.8938

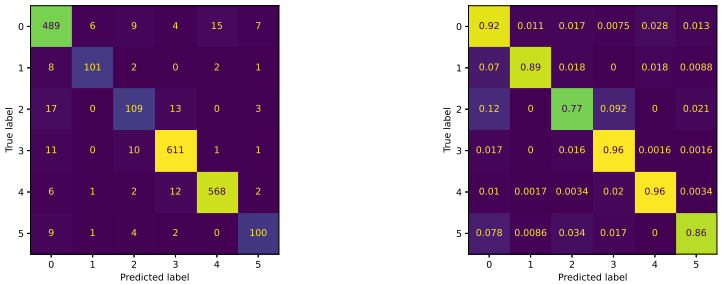


Figure A.4 Confusion matrices showing the performance for the CNN network on normalized raw data. The left matrix shows the amount of classified windows while the right shows the recall percentages.

Table A.5 Performance table for CNN model on features

Class	Precision	Recall	F1-score	Accuracy
Stationary (0)	0.9109	0.8868	0.8978	
Fallen over (1)	0.8803	0.9035	0.8917	
Lifted (2)	0.7324	0.7324	0.7324	
Rolling (3)	0.9381	0.9795	0.9584	
Shaken (4)	0.9741	0.9560	0.9650	
Picked up from ground (5)	0.8727	0.8276	0.8496	
Global average	0.8848	0.8810	0.8826	0.9210
Weighted average	0.9209	0.9210	0.9207	0.8810

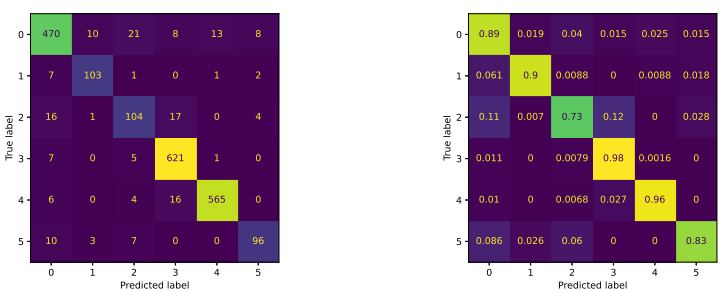


Figure A.5 Confusion matrices showing the performance for the CNN network on features. The left matrix shows the amount of classified windows while the right shows the recall percentages.

Table A.6 Performance table for CNN-LSTM hybrid network on features

Class	Precision	Recall	F1-Score	Accuracy
Stationary (0)	0.9408	0.8396	0.8873	
Fallen over (1)	0.8548	0.9298	0.8907	
Lifted (2)	0.6807	0.7958	0.7338	
Rolling (3)	0.9258	0.9842	0.9541	
Shaken (4)	0.9775	0.9560	0.9666	
Picked up from ground (5)	0.8929	0.8621	0.8772	
Global average	0.8788	0.8946	0.8850	0.9182
Weighted average	0.9219	0.9182	0.9187	0.8946

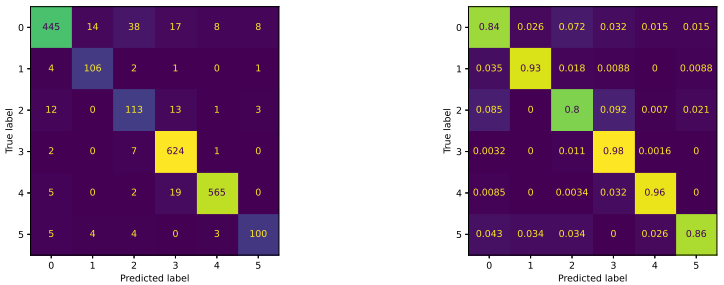


Figure A.6 Confusion matrices showing the performance of the CNN-LSTM network on features. The left matrix shows the amount of classified windows while the right shows the recall percentages.

A.2 Grad-CAM

In this section of the appendix, more examples of created Grad-CAM heatmaps are shown to give a more complete image of how the model interpreted its inputs.

Figure A.7 shows a window of the bike falling like in Figure 4.5. The striped behaviour hinting at a hidden pattern of the data, discussed in Chapter 5.1.3, is clearly visible between samples 20 and 50.

Figure A.8 shows one window of the bike getting lifted. The most important data points occur towards the end, where both the accelerometer and the gyroscope has its maximum values. This is where the motion seems to start, indicating that the window is the first depicting the entire event.

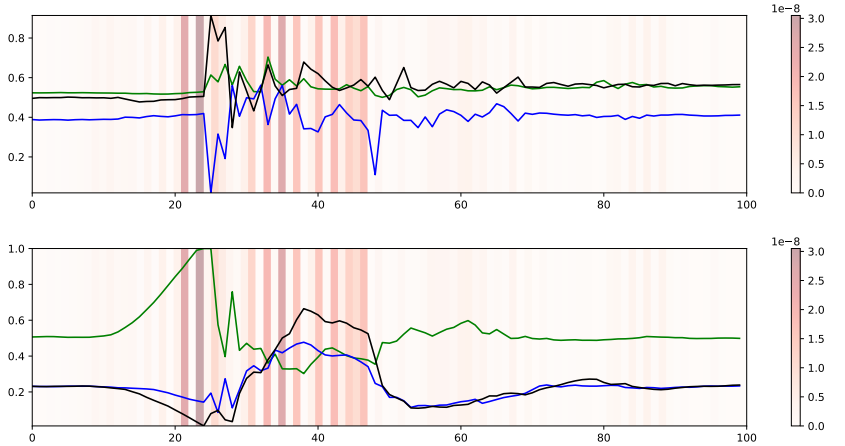


Figure A.7 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike falling, which it correctly predicted with 99% confidence.

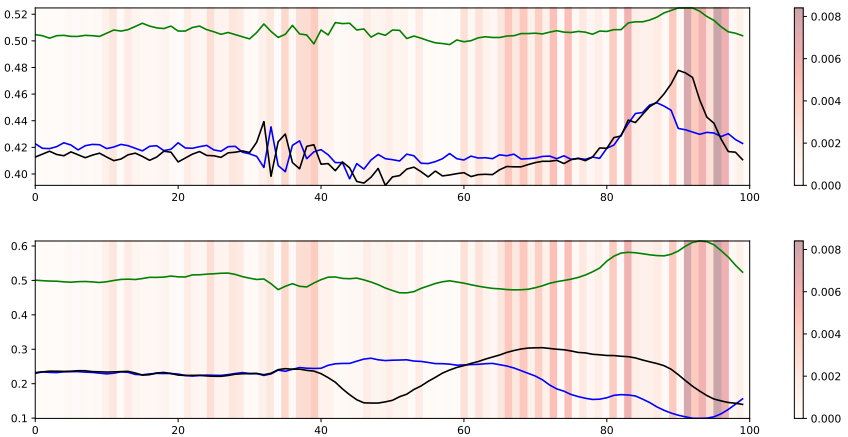


Figure A.8 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike getting lifted, which it correctly predicted with 64% confidence.

Figure A.9 shows one window of the bike getting lifted. The most important data points happen around two thirds into the data, where both the accelerometer and the gyroscope has its maximum values. It depicts the same event as A.8 but slightly later on meaning the windows overlap. Notice that the model seems to find the peak in the data in both windows, and use it to correctly classify the event, despite the peak being positioned differently in the two windows.

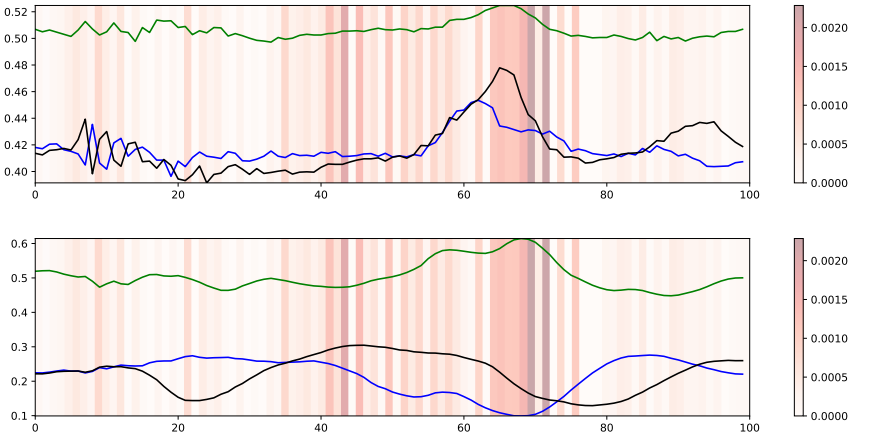


Figure A.9 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike getting lifted, which it correctly predicted with 93% confidence.

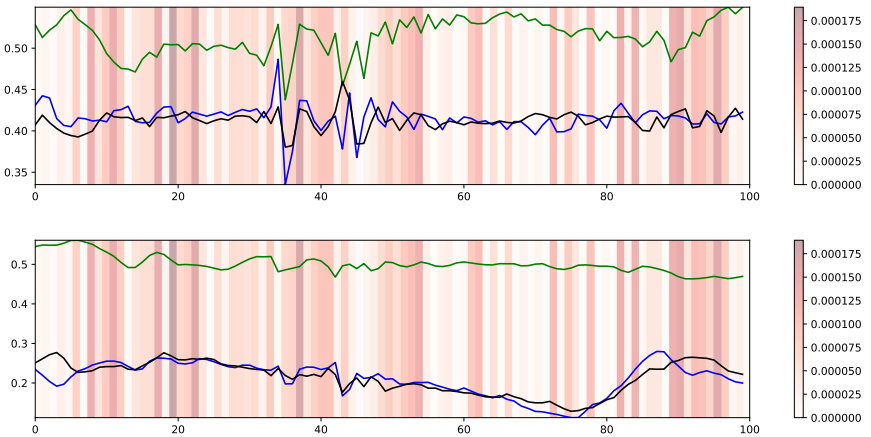


Figure A.10 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike rolling, which it correctly predicted with 99% confidence.

Figure A.10 shows one window of the bike rolling. Here it seems as if many data points are of importance, especially in the first half, but is hard to interpret why. It supports the conclusion that it is more difficult to comprehend the decisions for the interchangeable motions.

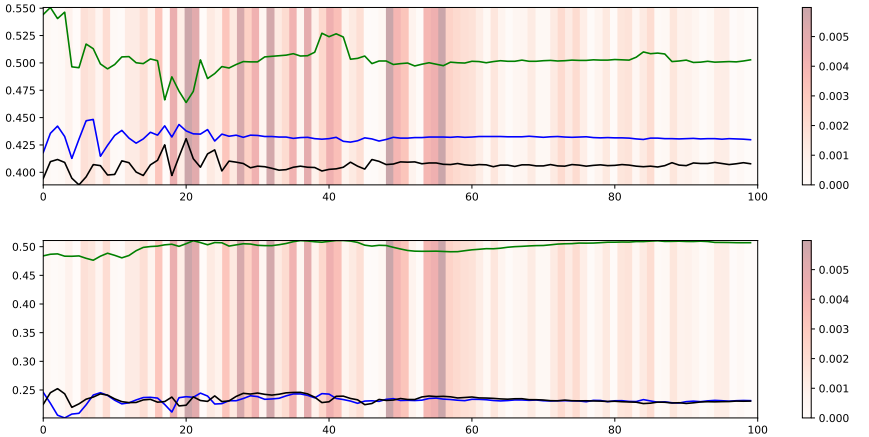


Figure A.11 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike shaking, which it predicted as rolling with 40% confidence.

Figure A.11 shows one window of the bike shaking, but the model was not confident of any motion. The class with the largest confidence, "Rolling", had still only a confidence of 40%. With the conclusion that the model seems to base much of its decision on the gyroscope gives a hint why the model is uncertain. The gyroscope changes only within noise levels.

Figure A.12 shows one window of the bike shaking. There is a lot of motion in the accelerometer in the second half, but the model seems to ignore it. The neighbourhood with the most important points happen where the accelerometer has few changes but the gyroscope has its maximal values.

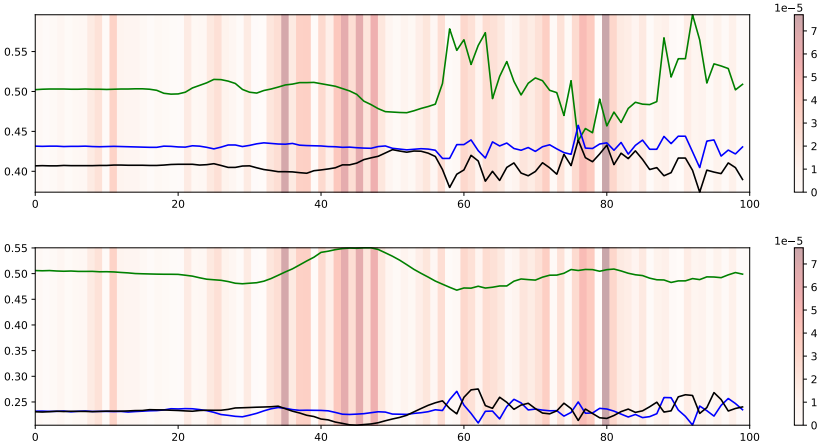


Figure A.12 Accelereometer data on top, gyroscope data on bottom. The data shows one time window of the bike shaking, which it correctly predicted with 97% confidence.

B

List of case definitions

Table B.1 Table showing all the use cases for the data. The stationary cases were only used in full event solution. When the time windows were used all data that was not already annotated as something, were used as the stationary class. All cases except the ones for rolling were performed with both locked and unlocked bikes.

Class	Case
Stationary (0)	Bike stationary on kickstand
	Bike stationary in bike rack
	Bike stationary while held by user
	Bike stationary while lying on ground
	Bike stationary in bike rack with "accidental" contact
Fallen over (1)	Bike released by user from standing
	Bike released by user from low position
	Bike sliding down from wall/tree
	Bike falling from standing up to halfway down
Lifted (2)	Bike carried a distance
	Bike lifted and rotated 180 degrees
	Bike lifted in place for a longer time
	Bike lifted and placed on higher ground
	Bike lifted and dropped instantly
Rolling (3)	Bike rolled by user slowly
	Bike rolled by user quickly
	Bike rolled on uneven ground
	Bike rolled downhill
	Bike rolled uphill
	Bike rolled in a circle
	Bike ridden by user slowly
	Bike ridden by user quickly
	Bike ridden on uneven ground
	Bike ridden downhill
	Bike ridden uphill
	Bike ridden in a circle
Shaken (4)	Bike shaken sideways
	Bike shaken back and forth
	Bike shaken in bike rack
Picked up from ground (5)	Bike picked up from ground

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
March 2023

Document Number
TFRT-6192

Author(s)
Filip Larsson
Pontus Hallqvist

Supervisor
Anders Svensson, Robert Bosch AB, Sweden
Yiannis Karayiannidis, Dept. of Automatic Control, Lund University, Sweden
Pontus Giselsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Classifying Motion Patterns of Bikes using Machine Learning

Abstract

Electric bikes have become ubiquitous in traffic, and with a growing user base and expensive prices, a demand for bike protection is increasing. Bike protection applications could include detecting and notifying the owner if their bike has been stolen or fallen over. This thesis aims to develop solutions for recognizing and classifying motion patterns of an electric bike to allow for improvements in bike protection applications.
Using accelerometer, gyroscope and magnetometer data as input, machine learning models were developed to perform classification. The data was labeled to six classes of different motions and then normalized, split into time windows and featurized. The different machine learning models built and tested were k-nearest neighbors (KNN), Convolutional neural network (CNN), Long short-term memory (LSTM) and a combined CNN-LSTM network. Time windows with different lengths and overlaps were tested and evaluated to achieve the best accuracy possible. Lastly, a filter was applied to the output to correct misclassifications.
To increase the understanding of how decisions were made by the models, Grad-CAM was applied to highlight what parts of the information the model found most crucial. Using the Grad-CAM heatmaps, it was found that the gyroscope data was the most influential for the model's decisions.
The model with the best performance was a CNN-LSTM combination network that uses a time window of 2 seconds and 75% overlap. It performed with an accuracy of 94.65%. When testing the best model with data from other bikes with different mounting positions, the accuracy was 35.23% indicating that different sensor placements or orientations changes the data in a way the current model cannot handle.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-72

Recipient's notes

Security classification