UNIVERSITY OF TARTU
Institute of Technology
Robotics and Computer Engineering

Shan Wu

Road Surface Recognition Based on DeepSense Neural Network using Accelerometer Data

Master's Thesis (30 ECTS)

Supervisor: Amnir Hadachi, Ph.D.

Tartu 2019

Road Surface Recognition Based on DeepSense Neural Network using Time Series Accelerometer Data

Abstract:

Nowadays, Smart devices plays a big role in our lives, especially in our daily activities. Therefore, Smartphones can be considered as one of the most interesting sensor for depicting our activities and our surroundings. Furthermore, the computation power of smartphones has increased a lot recently as most of them have multiple sensors like accelerometers and gyroscopes. Besides, They are capable of processing more tasks than we ever imagined. Because of their advantages of convenience and low-cost, the portable computation platforms has been adopted in the development of autonomous vehicles.

The most critical issue of the intelligent system assisted vehicles is that the safety problem. The recognition of the road surface is one of the components to ensure the safety drive. Most of the solutions use sensor fusion to recognize road surfaces such as combining cameras and LiDARs, which is costly for equipment and they usually need installations to re-equip existing cars, but these methods provide overall excellent results.

This thesis proposes a method for recognizing the road surface based on using accelerometer data collected from smartphone. The process uses time series data collected from a smartphone's accelerometer, followed by a massive time series feature extraction and selection. After that, the features are fed into trained DeepSense variant neural network framework to get the recognition of the road surfaces. The proposed method provides three classes recognition for smooth, bumpy and rough roads.

Moreover, in this thesis we conducted a thorough evaluation and analysis of the proposed method by comparing it with conventional machine learning methods like SVM, random forest, fully connected neural network and convolutional neural network. The accuracy of the method in this thesis overmatch the compared examples. The road surface type will be classified into three categories which will indicate smoothness of the road surface.

Keywords: Road Quality, Machine Learning, SVM, Random Forest, Neural Network, Mobile Phones, Accelerometer Data

CERCS: P170 Computer science, numerical analysis, systems, control

Tüübituletus neljandat järku loogikavalemitele

Lühikokkuvõte:

Tänapäeval omavad nutiseadmed meie elus suurt rolli, eriti igapäevastes tegemistes. Sellepärast võib kaaluda nutitelefoni kui üht kõige huvitavamat andurit kujutamaks meie tegevusi ja meie ümbrust. Lisaks sellele on nutitelefonide arvutusjõudlus hüppeliselt kasvanud, mida kinnitavad nendes sisalduvad erinevad andurid nagu kiirendusmõõturid ja güroskoobid ning võimekus sooritada rohkem ülesandeid kui kunagi varem. Nende mugavuse ja madala hinna tõttu on nutitelefone hakatud kasutama kui kaasaskantavaid arvutusplatvorme autonoomsete sõidukite arenduses.

Intelligentsete sõidukite süsteemide kriitiliseimaks probleemiks on turvalisus. Teekatte tuvastus on üks turvalise liiklemise põhikomponentidest. Enamik praeguseid lahendusi teekatte tuvastamiseks kasutavad erinevate sensorite nagu kaamerate ja LiDARite kokkusulatamist. See on küll efektiivne meetod, kuid tegemist on kallite anduritega ning mille kasutamine vajab auto enda modifitseerimist.

Lõputöö pakub välja meetodi teekatte tuvastamiseks kasutades nutitelefonis oleva kiirendusmõõturi andmeid. See protsess kasutab ajaliselt jätjestatud kiirendusmõõturi andmeid, millele järgneb masiivne ajaliselt järjestatud tunnuste eraldamine ja valimine. Peale seda suunatakse eraldatud tunnused DeepSense närvivõrgu raamistikku, et teekate tuvastada. Meetod klassifitseerib kolme erinevat teekatte tüüpi: sile, munakivitee ja kruusatee.

Põhjalik pakutud metoodika uurimine ja analüüs viiakse läbi kasutades üldlevinud masinõppe meetodeid nagu tugivektor-masinad, otsustusmets, täielikult ühendatud närvivõrgud ja konvulutioonteisendus närvivõrgud. Metoodikal põhinevad katsed näitavad, et pakutud lähenemine võimaldab tuvastada teekatte siledust väljapakutud kolme kategooriasse.

Võtmesõnad:

Maanteede kvaliteet, masinõpe, SVM, juhuslik mets, närvivõrk, mobiiltelefonid, kiirendusmõõturi andmed

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Acknowledgement

I would like to show my greatest gratitude to my supervisor, Dr. Amnir Hadachi of the Institute of Computer Science at the University of Tartu, who has provided me valuable guidance within this master thesis work. I am so grateful that his office is always open for me to do research on my thesis, and allowed this thesis to be my own work. He gave me the necessary right directions, so my progress went smoothly.

I shall also thank all the other advisors during my study period who assist me whenever I faced troublesome issues. Mr. Artjom Lind from ITS Lab, Institute of Computer Science at the University of Tartu, also gave me essential help regarding the processes of big data and machine learning. I really appreciate his help during this period.

Last but not least, I must express my enormous gratitude to my parents and my girlfriend for providing me great support and encouraging me throughout many years of education. This thesis could not be accomplished without them. Thank you very much.

Contents

Abstract **2**

Lühikokkuvõte **3**

Acknowledgement **4**

Abbreviation and Acronyms **10**

1 Introduction **11**
1.1 General View 11
1.2 Objectives 12
1.3 Limitations 13
1.4 Contribution 13
1.5 Road Map 14

2 State-of-the-art **15**
2.1 Introduction 15
2.2 Road Surface profiling 15
2.2.1 3D reconstructions 16
2.2.2 Vision-based methods 17
2.2.3 Vibration-based methods 18
2.3 Recognition Approaches 18
2.3.1 Threshold based 18
2.3.2 Machine learning 19
2.3.3 Dynamic time wrapping 20

3 Methodology **22**
3.1 Introduction 22
3.2 Dataset 22
3.2.1 Data Collection 22
3.2.2 Coordinate Systems 23
3.2.3 Time Series 25
3.2.4 Data Filtering 26

3.2.5 Data Segmentation . . . 27
3.3 Features . . . 28
3.3.1 Feature Extraction based on Scalable Hypothesis tests (FRESH) 28
3.3.2 Feature Mappings . . . 28
3.3.3 Feature Selection . . . 30
3.4 Principal Component Analysis . . . 32
3.5 Convolutional Neural Network Framework . . . 33
3.5.1 Introduction . . . 33
3.5.2 Inputs . . . 34
3.5.3 Structure . . . 34
3.5.4 Individual Convolutional Subnet . . . 35
3.5.5 Merge Convolutional Subnet . . . 36
3.5.6 Recurrent Layers . . . 36
3.5.7 Output Layer . . . 37
3.5.8 Configuration . . . 37
3.6 Summary . . . 39

4 Results and Discussion **41**
4.1 Experimental Strategy . . . 41
4.1.1 Data Acquisition . . . 41
4.1.2 Data Preprocessing . . . 43
4.1.3 DeepSense framework . . . 44
4.1.4 Other methods . . . 46
4.2 Evaluation Results . . . 47
4.3 Summary . . . 49

5 Conclusions **50**
5.1 Conclusions . . . 50
5.2 Future Perspectives . . . 51

References **52**

Appendices **56**

A DeepSense variant network configuration **56**

B Feature mappings **58**

Non-exclusive License **62**

List of Figures

1 The process of road surfaces recognition. 16
2 An example of the data collected by the motion sensor from a smartphone. It illustrate the accelerometer values for 3 axis from world coordinate system. When you put the smartphone vertically in a car, the Y-axis represents the axis perpendicular to the earth ground which can infer the smoothness of the road surfaces. 24
3 The plot on the left is the coordinate system on a smartphone while the right one is world coordinate system. 25
4 The figure shows the data contains some records may influence the later training process. 26
5 The figure shows the data after filtering. 27
6 The figure shows the structure of the DeepSense vairant network. It has 6 convolutional layers, 2 flatten & concatenate layers, 2 recurrent layers, and 1 output layer. 35
7 Three types of road surfaces tested. The upper one is smooth road surface while the bottom left one is bumpy road surface and the bottom right one is rough road surface. [39] 42
8 The phone holder used while testing. [40] 43
9 The training accuracy and loss during the testing. The upper left is training accuracy. The upper right is training loss. The bottom left is validation accuracy and the bottom right is validaiton loss. 45
10 The overall performance on three categories for proposed methodology and other methods. 48

List of Tables

1 Annotation table. 41
2 The performance for 10-folds cross-validation. 48
3 DeepSense variant network configuration 56
4 Table of feature mappings. 61

Abbreviation and Acronyms

CSV
: Comma-Separated Values

tsfresh
: Time Series Feature Extraction based on Scalable Hypothesis Tests

GPS
: Global Positioning System

HAR
: Human Activity Recognition

FDR
: False Discovery Rate

CNN
: Convolutional Neural Network

GRU
: Gated Recurrent Unit

LTSM
: Long Short-Term Memory

1 Introduction

1.1 General View

Smartphones have substantial popularity among consumers. Around 1.56 billion smartphones were sold last year worldwide [1]. The functionality of smartphones is updated a lot since there are more sensors integrated into one device. Their computational power also has improved. Hence they can process tougher tasks like image processing and machine learning [2]. The applications on the phones make it possible for developers to create useful tools using full advantage of the components of a smartphone. The accelerometer, gyroscopes and GPS sensor inside the smartphones can provide accurate locations, position, and movement status of the device in real time. With these powerful sensors, one can conduct complex research like human activity recognition (HAR) using mobile sensors [3].

Smart vehicles is another increasingly popular topic recently since the experts got the breakthrough achievements in deep learning thanks to the increase of onboard computational power. The safety issues are always on the highest priority in smart vehicles [4]. Road surface recognition ensures the vehicles to know what type of road they are driving on, therefore they can make corresponding measures taking effect. A good road surface recognition mechanism will help all potential users of smart vehicles to ensure a safe trip.

Combining the mobility of smartphones and traditional cars could save much time and cost in road surface recognition problem. Smartphones which mostly using Android system are affordable and straightforward to develop an application to make it like a tool. Compared to iOS devices, Android devices have less sensor accuracy but provide more freedom to utilize them. They also support writing the file to the external storage directly.

This process of collecting such data necessitate that the phone should be installed in a fix position inside the car. The implemented convolutional neural network will take those data and predict the possibilities of each road type. We select the category with the highest possibility to be the prediction.

1.2 Objectives

The aim of this thesis is to develop an algorithm which is based on smartphones' sensors for recognizing road surfaces. The existing methods require either expensive sensors or re-equipment for a regular car to achieve high accuracy. The proposed algorithm will reduce much labors and costs.

The proposed methodology contains three components, which are data collection, feature extraction and selection, and neural network framework.

We use an Android phone to collect accelerometer data. When the car is driving, there will be vibrations in the car especially when the road surface has different types of smoothness. A smartphone can record these vibrations by using its built-in accelerometer.

The data is later used for time series feature extraction in order that the neural network and classifiers can find patterns from the data. Numerous features are extracted and sifted in such manner to keep useful ones for the final step.

A DeepSense variant neural network will take these features and learn the differences between different road surfaces signatures. The results proved to be better than conventional classifiers. The solution can be used in various scenes for both end users and authorities to inspect the road quality or be used in the smart vehicles for safe driving.

The objective of this thesis is as follows:

- Reviewing the state-of-art techniques and summarizing the advantages and disadvantages of both speed performance and accuracy.
- Based on the literature review, a methodology is proposed capable of performing road surface recognition with high accuracy.
- Proposed solution for the road surface recognition should uses the minimum computation power and get better results.
- The proposed methodology should be useful in the real environment to assist the intelligent transportation domains.

Finally, the overall testing is conducted to have a comprehensive understanding of the new methodology and evaluate it.

1.3 Limitations

In the proposed methodology, a Nexus 5 and a compact SUV car are used to conduct the experiment. The phone is an old model, the accelerometer and gyroscopes inside it may not have the best accuracy, but the data collected by it is still useable and valuable. The frequency of the sensors also affects the precision of the data. Different sensor frequency may produce totally different features. In this thesis, due to the lack of time, tests for different frequencies are not conducted.

Different cars will also perform differently in vibrations, mainly because of their suspensions, which can lead to different features and results. Due to the lack of fund, only one car is used during the experiment. Speed is another aspect that may influence the collected data. When the car is passing the same surface, different speed will cause the differences in vibration frequency. The speed is fixed around 30 km/s during the experiments.

The proposed methodology can be integrated into a smartphone based on the paper of DeepSense original framework, but in this thesis, all experiments are performed on a desktop computer. Running on a smartphone is not tested.

The solution can run instantaneously if the mobile collector can produce the sensor data in a real-time, but it is not implemented in this thesis which can be a part of future work.

1.4 Contribution

The proposed methodology gives a new perspective to perform road surface recognition. The basic smoothness recognition can be recognized with only accelerometer data and a light-weighted neural network model. The neural network model can be adapted for multiple sensors. As a consequence, the community can combine our methodology with other sensors to have a better understanding of the road surface.

At the moment of writing this thesis and to our knowledge, there is no clear full implementation of DeepSense framework yet. However, the full capacity DeepSense variant neural network framework is implemented in the thesis. This model can handle multiple sensors' data originally. However, only one sensor's data is used in the thesis.

1.5 Road Map

The thesis is composed as follows:

Chapter 2 introduces the state-of-art methodologies used in road surface recognition. The nature of data, techniques and the performances are discussed within this chapter.

Chapter 3 describes the methodology proposed in this thesis including the details of data collection, time series feature extraction and selection, and DeepSense variant neural network framework.

Chapter 4 will reveal the evaluation results of the proposed methodology, and comparison with other conventional techniques.

Chapter 5 shows the conclusion and discussion for the proposed methodology and the direction of future study are talked.

2 State-of-the-art

2.1 Introduction

Road surface recognition is a critical issue for both road maintenance authorities and smart vehicles. Every year, a considerable amount of labors and budget are spent in road maintenance [5]. One of the aspects of the road surface quality is the anomalies alone with the road surface. The typical way of profiling a road surface is to use a car equipped with particular devices to monitor the road surface changes. These devices can be either visual based, vibration-based or sensor fused.

On the other hand, road surface recognition is widely used in self-driving cars because it is crucial for a smart vehicle to sense the environment. Self-driving cars will behave differently on various environments including lanes, traffic signs, other vehicles, objects, and pedestrians and so on. The road surface is one of the environmental factors that a self-driving vehicle needs.

2.2 Road Surface profiling

Sattar et al. [6] reviewed the general steps to recognize the road surfaces. The methodologies mainly diverge at the part of data collection and the main process. The general steps are shown in ***Figure 1***. 3D reconstructions, vision-based methods, and vibration-based methods are three different ways to collect road surface data, and each of them has pros and cons. After the data are collected, the preprocessing can be applied. It usually contains noise filtering, normalization, standardization, and feature extraction. At the main process part, several techniques can be selected such as SVM, Dynamic Time Wrapping (DTW), threshold-based methods, and so on. After the main process, post-processing can be done which depends on the needs of each research.

There are several ways to sense the road surface which can be classified into 3D projection-based, camera-based or vibration-based methods. The most accurate way to profile the road surface is to use laser scanners and stereo cameras which can create accurate surface models. However, the vibration based methods also produce sufficient information for road surface profiling [9].

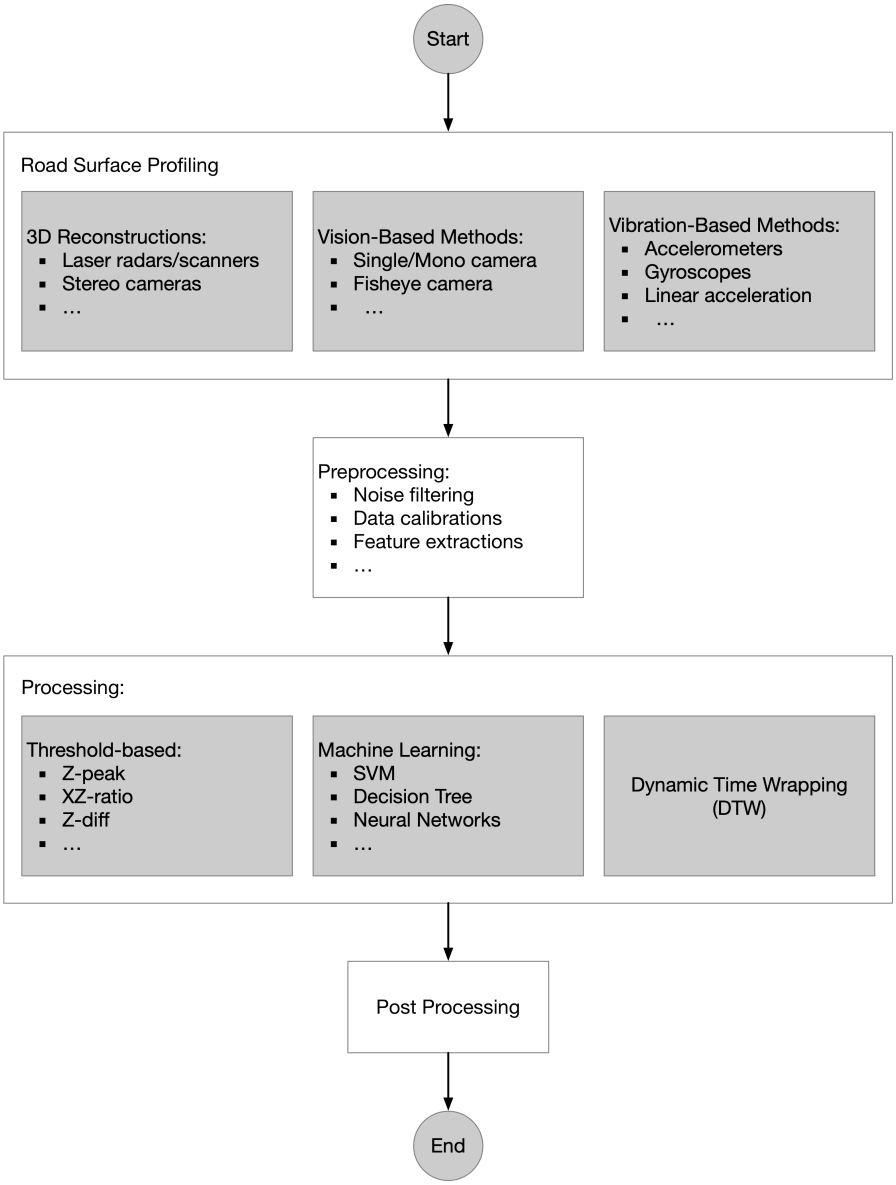

Figure 1. The process of road surfaces recognition.

2.2.1 3D reconstructions

Yang et al. [7] and Aki et al. [8] uses laser scanners which can create high-frequency laser pulses and use the reflection of the laser pulses to create a 3D reconstruction of the environment which is point clouds. Then the objects and obstacles, as well as anomalies,

will be extracted from the reconstructed model. This method produces accurate road surface profiles, but the laser sensors are quite expensive to afford.

Other scholars found an alternative way to construct a 3D environment using stereo cameras because it is cheaper and also produces an accurate 3D reconstruction. Haq et al. [10] and Staniek el al. [11] both use two digital cameras capturing images for the road surfaces. The cameras are installed outside of the car facing the roads. Then the algorithm estimates the disparity between stereo images. The key points are extracted and matched for images. After that, the mathematical transformation ensures the algorithm to determine the anomalies such as potholes and cracks.

This type of methods give comprehensive details about the anomalies, but again, it requires the installation of the devices and cost relatively more than other solutions.

2.2.2 Vision-based methods

Vision-based methods also use cameras, but only mono camera system is used. This type of methods uses 2D static images and image processing techniques to analyze the road surfaces.

Koch et al. [15] uses a high-speed wide-angle fisheye camera which is installed on the rear of the car and tilted downside to the road. The process of removing distortion is necessary because the correct shapes and edges are crucial for road surface objects detection. The method uses two stages to recognize the anomalies on the onboard computer in the car. First, the real-time algorithm detects the area in the frames that might have defects and in the second stage, apply detect recognition algorithm on those frames are selected.

Quintana et al. [16] produced a method to inspect the road surfaces on the highways. First, the hard shoulder of the road is detected using HT. After that, two procedures run in parallel to detect cracks. The size and orientation of the cracks are learned and determine the type of cracks.

The 2D vision-based methods need a good skill of digital image processing because the evidence of the anomalies is not apparent. Features or areas of interests must be extracted from the photos. Besides, the algorithm using one camera also affected by the conditions of the environment. The anomalies in different weather conditions will reflect the light unfavorably, and as a consequence, the camera might not recognize them.

2.2.3 Vibration-based methods

With vibration-based methods, the data collected usually is in the form of acceleration. The collected data can be derived from vehicles, professional equipment or mobile sensors. Recent years, more studies start to use smartphones as the collector which can produce necessarily acceleration data Khang et al. [14].

Harikrishnan et al. [12] and Singh et al. [13] all come up with a method to utilize the accelerometer inside the smartphone. The phone is fixed while collecting the data. While driving on degraded road surfaces, there will be more intense vibrations in the car. The car will vibrate less on the smooth road respectively.

The limitation of accelerometers is the lack of details for the road surfaces. The 3D reconstruction methods can provide a detailed view of anomalies such as shapes and depth. Different cars with multiple configurations may have different vibrations, thus the collector may need to calibrate with the cars first before using.

2.3 Recognition Approaches

In this process, useful information is derived from collected data which can be point clouds, images or vibrations. However, instead of processing on the raw collected data, filters usually are applied to the collected data. This step is not always necessary, but it can filter certain noises that may influence the main process to get accurate results.

The main process takes either filtered data or raw data and can be classified into three types. Threshold-based methods usually set up a threshold for vibrations because extremely large changes in vibration probably caused by road surface anomalies, thus this type of methods often used for vibration-based data.

2.3.1 Threshold based

Threshold-based methods analyze the magnitude of the accelerometer and make decisions by a predefined threshold. The threshold can be fixed or dynamic.

Harikrishnan et al. [12] proposed an X-Z filter to classify the road surfaces. After the sensor data has been filtered, the abnormal events are detected in the accelerometer data using a Gaussian Model. Then the X-Z filter is used to classify the events. It uses the pairs of the data from X-axis and Z-axis respectively and computes the X-Z ratio over a

small time window derived from X-axis points. The ratio is compared to the predefined threshold T_x to complete the classification.

Mednis et al. [17] uses four different threshold-based methods, Z-THRESH, Z-DIFF, Z-DEV, and G-ZERO, to classify an event in road surfaces. These four methods are common in signal processing, and they are fast to process. Fast processing is a quite important criterion because it means convenience and less cost in devices. Z-THRESH set a pair of upper and lower bounds to identify events. Z-DIFF are measured by consecutive measurements with their difference values. Events are detected if the differences are greater than a threshold. Z-DEV computes the standard deviation among the measurements within a time window and set a threshold for the maximum deviation. The last one, G-ZERO is one of the empirical techniques because it depends on the problems to solved and the observations in the accelerometer data.

This type of methods are efficient in processing, but the techniques themselves are simple. Thus they will misclassify many events if the preprocessing can not filter out the noises. The results also affected by the vehicle types and road surfaces and many efforts needed in parameter tuning. When testing in a new environment, the old threshold may not suitable for current road type.

2.3.2 Machine learning

Machine learning is a kind of algorithms and statistical models that computers use to accomplish specific tasks effectively. The algorithm does not need to be explicit instructions. Instead, the algorithms can learn the patterns or features inside the data and infer the results based on what they learned. This technique is efficient and accurate if the classifiers are well-tuned, but also has disadvantages like enormous data consumption.

Support vector machine

Support vector machine (SVM) is one of the supervised learning models. It classifies the data points by a gap which is trained as wide as possible to two or more categories.

Bhoraskar et al. [18] uses SVM in their road condition estimation system. The data are collected by the accelerometer and labeled by an unsupervised learning method K-means clustering algorithm. Feature selection is the most important part of their algorithm. They found the deviation of the Z-axis accelerometer data dominant the performance because when vehicles pass different road surfaces, the vertical vibrations

change the most. The SVM can give overall good results and is balanced between the accuracy and the speed.

Decision tree

The decision tree is a tree-like model that holds three types of nodes: decision nodes, chance nodes, and end nodes. Each internal node is an evaluation of one attribute, and the leaf nodes are the label of the classes. A path from the root to the leaf is the classification rules for this data.

Allouch et al. [19] uses C4.5 Decision tree classifier to classify road segments. The tree selects the node has the most information gain to split. Feature extraction and feature selection are applied to the data before being used for training. In total, Three techniques are tested by the authors with and without feature selection. C4.5 Decision tree yielded the highest accuracy. Even without feature selection, C4.5 Decision tree can work accurately.

Naive Bayes

Naive Bayes is a kind of classifiers uses the probabilities added by different features. Each feature should be independent other than all other features. The classifier uses each feature to contribute to the probability of the classes.

Hoffmann et al. [20] proposed a methodology to monitor the road surface quality. Five different features are extracted before any classification: speed, inclination, mean acceleration, the variance of acceleration, and standard deviation. 10-folds cross-validation is applied to measure the performance of the algorithm, and the results are satisfying with an average of 78% accuracy.

2.3.3 Dynamic time wrapping

Dynamic time wrapping (DTW) is a technique for time series data to find the similarities between two temporal series data. The process can find the best matches in two series, and the matches must be monotonically increasing, which means there should not be cross lines for paired points.

Singh et al. [13] uses DTW in their road surface monitoring system. They increased the classification accuracy of detecting road surface conditions using DTW and motion sensor data. The DTW algorithm has the ability to process the time deformations and

various speeds from different time series data. It can be used in portable devices, and the training process can be as fast as SVM. However, the system is sensitive to the noises. The detection rate decreases when there are too much noise vibrations.

DTW algorithm is generally not reliable compared to machine learning methods. The technique behind is to compare the time series data with reference data. The reference data can not represent all road surface conditions. Thus the DTW based systems are affected by different vehicles types, road conditions, and noises.

3 Methodology

3.1 Introduction

In this section, the proposed methodology will be introduced in three components. Firstly, a feature extraction process transforms time series data into a feature matrix. Secondly, a feature selection process will use a hypothesis test to select relevant features only which can reduce the dimension of the feature matrix. In the end, the selected features will be fed into a neural network to learn the patterns of road surfaces and predict new road surfaces as well. This neural network framework is originally used for mobile sensor data. In this thesis, a variant of this neural network is created for road surface recognition. All the processes are executed and evaluated in a virtual environment of Python3 which is a popular scientific research programming language and easy for implementation.

3.2 Dataset

3.2.1 Data Collection

The time series data used in this thesis is collected using an app installed on the Android smartphone. This phone has motion sensors like accelerometer and gyroscopes as well as GPS sensor. All data is collected from a Compact SUV vehicle on different road surfaces in Tartu, Estonia. When collecting the data, the phone is fixed vertically inside the car by phone support and the driving speed is around 40 km/h for urban roads, 30 km/h for off-roads and 80 km/h for highways. The frequency of sampling the motion sensor is around 10 Hz.

After the data is collected, the app will generate a CSV formatted file in the device's external storage. The following information is recorded into the file:

- Timestamp: it is written in milliseconds, starts from January 1, 1970, 00:00:00.0 UTC.
- Raw accelerometer data for X, Y, Z respectively: it is not used in the methodology but used in development stage.
- Transformed accelerometer data for X, Y, Z respectively: the raw accelerometer data is transformed into world coordinate system. This data is used for the

methodology.

- GPS coordinates for longitude and latitude: The position coordinates are recorded every 1 second for development purpose.
- Road surface type label: labels are set before recording and written along with the recorded information.

In every second, there will be 10 data points for accelerometer data. GPS coordinates are recorded at first, and the 11^{th} data point and the 11^{th} data point is also the start point for the next second and so on. For example, one plot of the collected sensor data is shown in ***Figure 2***. In the figure, the vibration details are revealed from 3 lines that indicated the linear acceleration from 3 axes of the smartphone. However, the three axes in the plot are not the actual three axes in our world which means we need to transform this data in order to refer the world coordinates system. The benefit from transformed sensor data is that the Z-axis is vertical to the ground so that Z-axis values can represent the smoothness of the road surface.

3.2.2 Coordinate Systems

The first problem faced because of using smartphones' motion sensors is how to handle the coordinate system. During the driving on the same road surface, vibration patterns will vary because of the position of the smartphones inside the car, which leads to completely different accelerometer values. In order to let the accelerometer values always reflect the same earth reference, we should perform a transformation of the coordinate system for the collected data. Usually, the phone has its own coordinate system which is shown in ***Figure 3*** where X-axis is the direction along the lateral axis, Y-axis is the direction along the longitudinal axis, and Z-axis is the direction along the vertical to the phone's surface. In the world's coordinate system, X-axis is the direction along the latitudinal axis, Y-axis is the direction along the longitudinal axis, and Z-axis is the direction vertical to the ground. Since different angles the phone laid in the car will give dissimilar values along with three axes, it is crucial to use a world coordinate system for all collections.

In order to get the accelerometer values in world coordinate system, the rotation matrix first need to be extracted. With the help of Android sensor framework, we

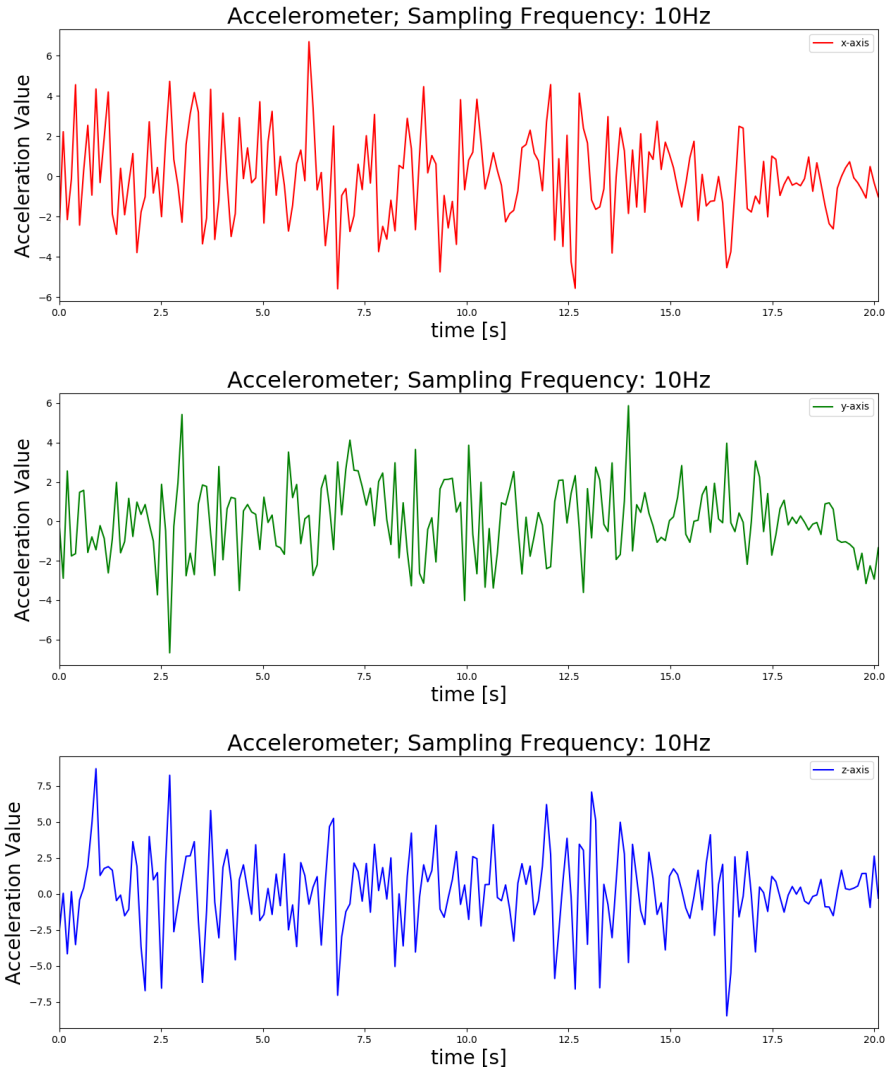

Figure 2. An example of the data collected by the motion sensor from a smartphone. It illustrate the accelerometer values for 3 axis from world coordinate system. When you put the smartphone vertically in a car, the Y-axis represents the axis perpendicular to the earth ground which can infer the smoothness of the road surfaces.

can transform the values in smartphones. As shown in the ***Listing 1***, The method $getRotationMatrix$ takes the gravity values and geomagnetic field strength values and output the rotation matrix R along with the inclination matrix I. Then the rotation matrix is inverted and multiplied with the accelerometer values in device's coordinate system

```
float[] R = new float[16], I = new float[16];
float[] worldAcceleration = new float[16];
float[] inv = new float[16];

SensorManager.getRotationMatrix(R, I, gravityValues, magneticValues);
android.opengl.Matrix.invertM(inv, 0, R, 0);
android.opengl.Matrix.multiplyMV(worldAcceleration, 0, inv, 0,
    deviceAcceleration, 0);
```

Listing 1. Transform to world coordinate system

using the method $invertM$ and $multiplyMV$ which will produce the acceleration values in the world coordinate system.

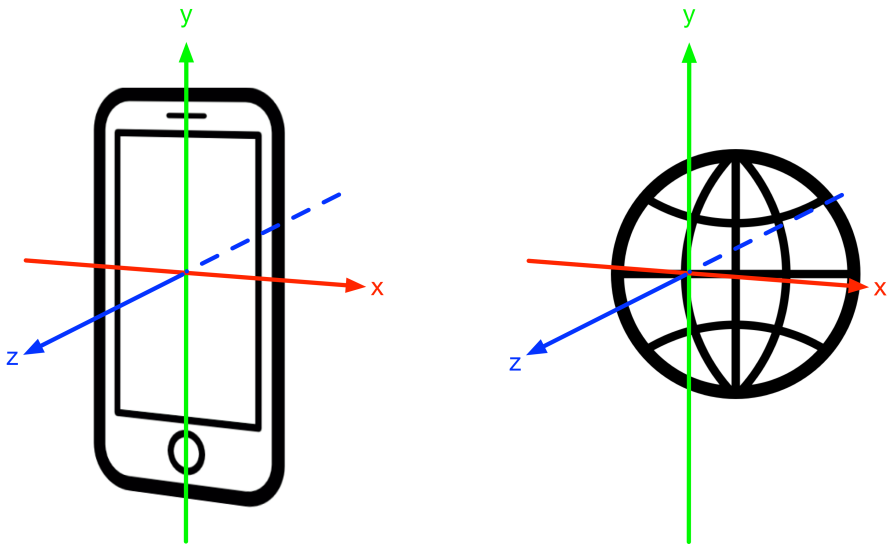

Figure 3. The plot on the left is the coordinate system on a smartphone while the right one is world coordinate system.

3.2.3 Time Series

Theorem 1. *A time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. [21]*

A time series data is a kind of discrete data in our case. This sampling process

captures the current states of the phone from the motion sensor j and repeats every interval of delta t. After the sampling, the collection is a sequence which has the length of n discrete points.

$$C^j = \{c_1, c_2, ..., c_n\}^j$$

Here, we have $n * m$ matrix which has the columns: timestamp, X-raw, Y-raw, Z-raw, X-axis, Y-axis, Z-axis, longitude, latitude and label and n rows. Each collection should have the length which is a multiple of tens. The collections are used as raw data in the desktop computer for further process.

3.2.4 Data Filtering

It is notable that there must be some noises or abnormal values that may occur during the collecting session. For example, while driving on a bumpy road, there are extremely small values recorded in motion sensor. It is mostly caused by the interruption of driving like stopping at red traffic lights or still collecting after finished driving. As shown in ***Figure 4***, this sensor data is collected on a bumpy road and it obvious that there are pause patterns during 160-200 seconds and 260-280 seconds.

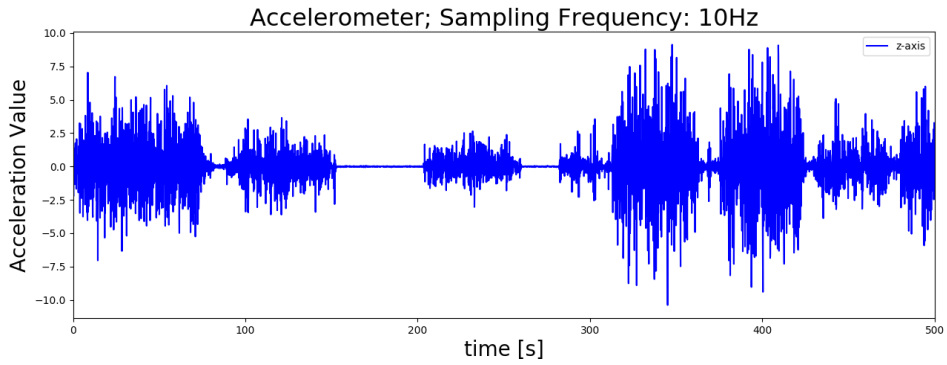

Figure 4. The figure shows the data contains some records may influence the later training process.

Here, we used a method to sift this kind of abnormal data collection by checking their standard deviations of the values. First, we record the sensor data when the smartphone is entirely motionless. There will be some small arbitrary values from the accelerometer

sensor, and we compute the standard deviation from that values using ***Formula 1***. Next, a threshold is set up based on the standard deviation calculated. Finally, we need to tune the threshold a few times in order to find a suitable value and filter out the idle periods. This filtering process will not change the data structure but only remove useless values. The filtered data is shown in ***Figure 5***, which has removed stationing values thoroughly.

$$\sigma = \sqrt{\frac{\sum (x - \overline{x})^2}{n}} \tag{1}$$

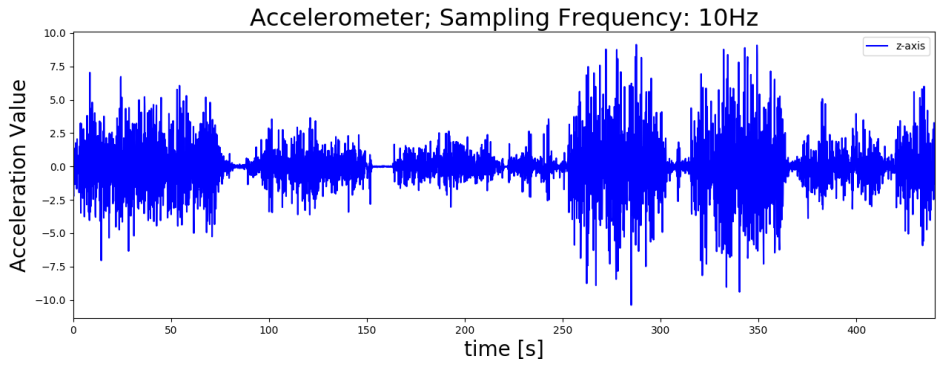

Figure 5. The figure shows the data after filtering.

3.2.5 Data Segmentation

The data collected is separated in files and the shape is $n * m$ where n is row index and 'm' is column index. In order to create formatted features for machine learning purpose, a segmentation process is necessary for all collections. A various selection of segment length can be chosen depends on different types of projects. Yao et al. [22] uses 0.25 seconds as a time window for their human activity recognition (HAR) problem. A human can change activities for short intervals, but driving usually takes longer time. In this thesis, a 5 seconds time window is used for all data collections S derived from sensor j. Thus, each segment s has 51 points including one extra ending point (10 points represent 1 second while the sampling rate is 10 Hz). The collections of data will become new collections of segments. Each segment has a universal index for later process.

$$s^j = \{c_1, c_2, ..., c_{51}\}^j$$

$$S^j = \{s_1^j, s_2^j, ..., S_n^j\}^j$$

After the segmentation process, a new label L for each segment is generated based on the labels among 51 points. Because all the raw data collections can be split by 50 points (5 seconds time window), each segment must have the same label. Thus, the new label L for sensor j can be inferred from the records in segments.

$$L^j = \{l_0^j, l_2^j, ..., l_n^j\}$$

3.3 Features

3.3.1 Feature Extraction based on Scalable Hypothesis tests (FRESH)

This process is introduced by Christ et al. [23] using the full parallelism ability of modern desktop computers. They provide a Python tool named $tsfresh$ to perform this process and give all freedom to customize the process.

The first step of the process finds the features of the time series data using well-established features mappings. Each segment is considered as a time series data. It will be transformed into a feature vector x_m after the first step process. The collection of all feature vectors is a feature matrix X_{nm} where n is the length of segments or number of time series and m is the number of feature mappings.

In the second step of the process, each feature vector will be evaluated individually and independently based on their p values which indicates the importance of this feature contributing to its label or class. The feature selection is based on the scalable hypothesis tests in order to sift the features and keep the useful features only.

3.3.2 Feature Mappings

In this section, the segmented time series data will be transformed into features matrix using feature mappings. A feature mapping $\theta : R^m \rightarrow R$ which characterize a time series and reduce its dimension.

Theorem 2. *In machine learning and pattern recognition, a feature is an individual measurable property or characteristic of a phenomenon being observed. [24]*

Based on the definition of the feature, a feature can represent one of the properties inferred from the time series data. In this work, various features will be created for segmented time series data. For example, a feature mapping could be a maximum function which finds the maximum value in the given time series.

$$\theta_{max}(S^j) = max\{c_1, c_2, ..., c_{51}\}$$

Other mappings could be the mean, standard deviation or number of peaks, and so on. These features describe a unique segmented time series data and can not derive from other time series. Christ et al. [23] collected and extended the feature mappings for different domains such as medicine, mathematics, industrial applications and so on.

In this methodology, 65 feature mappings are used in total in order to successfully recognize road surfaces [25], and the full list of mappings will be given in ***Appendix B***. To use the desired feature mappings, we can use $feature\_extraction$ module from $tsfresh$ package. The $feature\_extraction.ComprehensiveFCParameters$ object is a dictionary of feature calculators which holds the configurations of each feature mapping. Thus, we can provide a $ComprehensiveFCParameters$ dictionary to keep the feature mappings we need. The feature matrix can be extracted by using function $extract\_features(df, column\_id = \text{'id'}, column\_sort = \text{'time'},$ $default\_fc\_parameters = \text{params})$ which takes the segmented time series data with index and timestamp column to identify and sort the data. The feature configuration dictionary also passed to the function call.

After the feature mapping process, segmented time series data will be transformed into a feature matrix X_{nm} where n is the number of segments aka. the index of the segments and m is the number of final features. Note that all three axes sensor data will be calculated individually. Thus there will be more than 65 features in the end.

The extracted feature matrix along with the labels can be used for classifiers for machine learning purpose directly, but there are also some features that do not help the classification at all. For those useless features, it is better to remove them because there are two benefits. One is that the dimension of features will reduce. As a result, the performance got improved, while another is that the accuracy will improve if only use

relevant features.

3.3.3 Feature Selection

In this step, the features are sifted based on their relevances to the labels. First of all, the definition should be established to determine whether a feature is relevant or not.

Theorem 3. *A feature X_ϕ is relevant or meaningful for the prediction of Y if and only if X_ϕ and Y are not statistically independent. [23]*

To find the features that are not statistically independent, there are many methods to achieve that. The one used in the methodology is hypothesis testing, a mean to determine the probability that a given hypothesis is true. For each feature $X_1, ...X_m, ...X_{nm}$ (n is the number of segmented time series data), we apply one test statistic to check if it is relevant.

The general steps of hypothesis testing are that [26]:

- Set up a null hypothesis with the sign H_0 and an alternative hypothesis with the sign H_1.
- Create a test statistic which can evaluate the truth of the null hypothesis.
- Calculate the $p - value$ which is a probability that indicates the null hypothesis is true.
- Compare the $p - value$ with the acceptable significance value α. If $p \leq \alpha$, that means the null hypothesis is ruled out and the alternative hypothesis is accepted.

In the first step, two hypotheses are established:

$$H_0 = \{X_m \; is \; irrelevant \; for \; the \; label \; L\} \tag{2}$$

$$H_1 = \{X_m \; is \; relevant \; for \; the \; label \; L\} \tag{3}$$

We need to find those features associated with hypothesis tests which are rejected when H_0 is false. Each feature is assigned a hypothesis test based on their features and labels are continuous or binary.

Exact Fisher test of independence [23, 27]: This hypothesis can be used when both features and labels are binary. The method tests whether features and labels are independent based on the contingency table formed by X_m and Y.

Kolmogorov-Smirnov test[23, 28]: This hypothesis can be used when features are continuous and labels are binary. The method checks whether two random variables follow the same distribution f. The original hypotheses can be defined as follows:

$$H_0 = \{f_A = f_B\}$$

$$H_1 = \{f_A \neq f_B\}$$

Where f_A means the distribution of feature A and vice versa. When we have binary labels, we can compare the distribution of the same feature f_{X_m} with two labels y respectively. Then, the hypotheses become the follows:

$$H_0 = \{f_{X_m}|Y = y_1 = f_{X_m}|Y = y_2\}$$

$$H_1 = \{f_{X_m}|Y = y_1 \neq f_{X_m}|Y = y_2\}$$

In the last step, the non-parametric Benjamini-Yekutieli procedure was adopted by Christ el al. [23, 29]. The $p-values$ are sifted based on the intersection of the equation 4 and the $p-values$. Here, n_ϕ is number of all tested hypothesis and q is the false discovery rate (FDR) level which is set to 0.05 by default.

$$r_\phi = \frac{\phi q}{n_\phi \sum_{\mu=1}^{\phi} \frac{1}{\mu}} \tag{4}$$

Based on the hypothesis H_0, all $p - values$ that below the intersection are rejected, and they are relevant to the labels. Hence all the features corresponding to rejected $p - values$ are selected for further process.

In the Python environment, it is convenient to perform such a process since Christ et al. [23] provided the tool for feature selection. As shown in the algorithm block 2, because the recognition of the methodology is based on the multiple classes, we use a binary hypothesis test here. For each class, all the features are tested and sifted by using one-to-all binary labels. We test features for one class at a time only. After the features of 3 classes are tested, three sets of selected features are merged.

```
selected_features = set()

for l in np.unique(train_labels):
    train_labels_binary = train_labels == l
    train_features_filtered = select_features(train_features,
        train_labels_binary)
    selected_features = selected_features.union(set(
        train_features_filtered.columns))

train_features = train_features[list(selected_features)]
test_features = test_features[list(selected_features)]
```

Listing 2. Feature selection process

3.4 Principal Component Analysis

After the features are selected, there is still much redundant information, and each feature vector has the length of more than 200 which is pretty too much for the following neural network. Some of the features are related to each other such as mean and median values. The hypothesis test process cannot remove that redundant information. Thus the principal component analysis (PCA) is added after the feature selection.

Theorem 4. *Principal component analysis is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. [30]*

PCA is a widely used tool for the feature dimension reduction by finding the maximum variance in the features. The general steps are:

- For each feature in the feature matrix, subtract its mean values.
- Calculate the covariance matrix for the feature matrix using the function.
- Finding the eigenvalues and eigenvectors of the covariance matrix.
- Sort the eigenvalues in descending order, then select the most significant k values and its corresponding eigenvectors.
- Project the original points to the selected eigenvectors. The final size of the features will become $n * k$ from $n * m$.

In Python, it's convenient to perform PCA using the existing package scikit-learn. First, a PCA calculator will be created by command line pca = PCA(n_components=10, svd_solver='full') which defines number of component to keep. Next, we fit the features to the calculator using pca.fit(X). It will calculate the mean for each feature, variance of selected components and so on. Finally, a new feature matrix will be generated by using pca.transform(X). We perform a fit and transform process separately because both training and testing features should be calculated. The PCA will calculate based on the training features and perform the transformation on both training and testing features.

In the methodology, we only keep ten components from those selected features which means the final feature matrix has the shape of $(n\_samples, n\_features)$ where $n\_featues = 10$. The number $n\_features$ is determined by empirical tests collaborate with a convolutional neural network. If $n\_features$ is too small, the neural network can not learn gradually and fluctuate a lot. However, if $n\_features$ is too big, it will give much pressure for the hardware to run the neural network because the dimension of the tensor is too big.

The features and the labels prepared before are both fed into the next component of the methodology.

3.5 Convolutional Neural Network Framework

3.5.1 Introduction

In order to perform a time series classification task, a classifier needs to be used for recognizing road surfaces based on the sifted features. There are plenty of choices for this task such as SVM, Random Forest, and DTW. The SVM is often selected because of its consistent performance between execution time and accuracy, while the convolutional neural network (CNN) is less selected because it requires heavier computational power than conventional methods. CNN is widely used in the visual imagery domains and has less complexity and connectedness comparing with fully connected networks [32]. Yao et al. [22] purposed a unified convolutional neural network framework called DeepSense for solving different mobile sensing and computing tasks. It just needs few steps of customizing for specific tasks and can adopt many kinds of sensors such as motion sensor, microphone, Wi-Fi signals, barometer, and light sensor. In this methodology, a DeepSense variant is created for road surface recognition.

3.5.2 Inputs

The neural network has a unified format for the inputs. The unified format will be introduced first. From the motion sensor we got the raw time series data in 3 dimensions X, Y, and Z which denoted by X_{dn} where d is axis dimensions and n is sampled points. A time window τ is selected to split the data into T non-overlapping segments $X_{d\tau}^{T}$. For each time window, the Discrete Fourier Transform is applied to transform the time series to the frequency domain because frequencies have better patterns to represent the original data. Thus, the output tensor X has the shape $d * 2f * T$ where f is the frequency domain dimension containing the magnitude and phase pair values. For each sensor, there will be an output tensor like X, and the final input of k sensors for the neural network is $X = \{X_1, X_2, ..., X_k\}$.

This inputs without any preprocessing usually contain many noises. Although this DeepSense framework can take the noisy inputs and still learn the patterns and interactions among the sensors, it performs not quite well in our task. The original data we used is from an old Nexus 5 smartphone which is not calibrated and contains many noises while collecting the data which make the recognition of the road surface extremely difficult. In order to use the prepared features matrix X which has the shape of $(n\_samples, n\_features)$, it has to be reshaped. In the aforementioned feature matrix, each sample is made of a 5-seconds time series. Here, the feature matrix is segmented again for a 2-samples window which means a 10-second window is used for the original data. Since our original data length is a multiple of 10-seconds, we can apply a 10-second window for the feature matrix. Thus, the feature matrix is reshaped into a four dimension tensor $n\_samples * d * n\_features * T$ where d is 1 and T is 2.

3.5.3 Structure

As shown in ***Figure 6***, the DeepSense framework mainly has three components which are convolutional layers, recurrent layers, and output layers. In the convolutional layers, there are two subnets called individual convolutional subnet and merge convolutional subnet. Both subnets have a 3-layer structure. For each input X from multiple sensors, there is one individual convolutional subnet. The output of the first subnets will later become the input of the second subnet. There will be only one merge convolutional subnet for all sensors' subnets output.

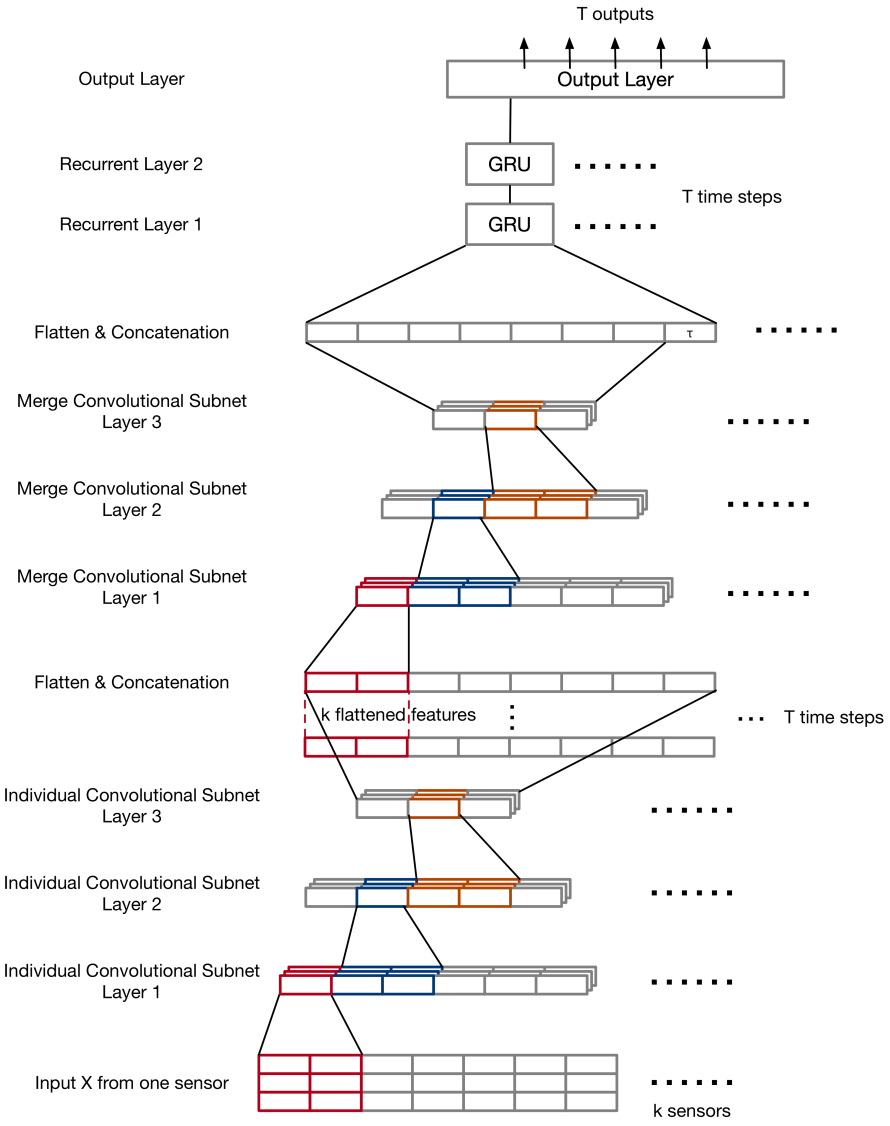

Figure 6. The figure shows the structure of the DeepSense vairant network. It has 6 convolutional layers, 2 flatten & concatenate layers, 2 recurrent layers, and 1 output layer.

3.5.4 Individual Convolutional Subnet

Since the individual convolutional subnets for all sensors data have the same structure, here we focus on one individual convolutional subnet with the input tensor X. Remind that the tensor X has the shape of $n\_samples * d * n\_features * T$. The subnet will take

one slice from the input tensor X which is one timestep from T. The input tensor will be $X_{..t}$ where t denotes the time step and it has the shape of $n\_samples * d * n\_features$. Because we want to focus on the interaction and relation between features, therefore we apply the 2D filters with shape $(d, conv1)$ in the first layer. The following layers also do the same convolution process to the tensor with 1D filters shape $(1, conv2)$ and $(1, conv3)$ respectively. It will find the hidden patterns within the abstract representation of the data.

3.5.5 Merge Convolutional Subnet

The output of the individual convolutional subnet for one timestep is collected and flattened into a 1D vector x. After collecting from all sensors, we concatenate the flattened outputs which are from the same time step into a k-row matrix X. The newly formed matrix then will be fed into merge convolutional subnet. In the first layer, 2D filters with the shape $(k, conv4)$ will be applied to find the patterns among k sensors. The following two layers apply 1D filters to the tensor with shape $(1, conv5)$ and $(1, conv6)$ respectively.

The output from the merge convolutional subnet is flattened after three layers' process. At the end of the flattened vector, the time window τ is added. This vector is just from one time step, and then we have to do the same process from beginning for all time steps and collect the outputs in an array. The array is later concatenated into a matrix with shape $(timesteps, gru\_features)$. This matrix will be fed into recurrent layers to learn the patterns among the time steps. In each aforementioned convolutional layer, the convolution process uses 64 filters and ReLU as the activation function. Furthermore, batch normalization layer is added between the convolutional layers to reduce covariance shift [31].

3.5.6 Recurrent Layers

The recurrent neural network is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence [33]. It can learn the patterns along with the time sequences which is powerful and meaningful for our time series data. Gated Recurrent Unit (GRU) is used in the methodology, but there is another alternative Long Short-Term Memory (LSTM). The performance of both are almost

the same, but only the GRU is used because it is simpler in structure and faster than LSTM [34].

The DeepSense framework uses a stacked GRU module which is a 2-layers structure in the methodology. According to Yao et al. [22], stacked GRU layers will increase the performance. The layers use ten units and three units respectively. We choose three units in the second GRU layer because our road surface recognition task has three classes. This three features generated from GRU module is later used for predicting the labels. Drop out layer is also added between two GRU layers with a drop rate of 0.5 to avoid the overfitting issue. The input tensor X for GRU module is explained above with shape $(timesteps, gru\_features)$. The last GRU layer produces a sequence of results instead of only produces one vector. Therefore, the output X of the GRU module has the shape $(timesteps, gru\_units2)$ where gru_units2 is three as mentioned above. The result from GRU module is then used as input for the output layer.

3.5.7 Output Layer

The output layer can differ based on the purpose of the task. Our recognition task is a classification task. Thus the final output should have a fixed length for prediction. Here, we use the averaging technique to the input tensor X based on the equation 5. A softmax layer is used after the averaging layer to predict the possibilities of each category. Other techniques such as weighted averaging over time steps can be used, but it is not implemented and can be further researches if possible.

$$\chi = \left(\sum_{t=1}^{T} x_t / T \right) \tag{5}$$

3.5.8 Configuration

Some modifications need to be done to adopt this neural network framework. Here we only have one kind of time series data derived from the motion sensor. Hence there is one individual convolutional subnet for one sensor and one merge convolutional subnet which is connected to the previous one.

The CNN is built based on Tensorflow library which is a free and open-source library for neural network and deep learning [35]. Now, it is even more convenient

to build a CNN from scratch using a tool called Keras. It is an open-source neural network library written in Python, and it is capable of running on top of Tensorflow as well as other libraries like Microsoft Cognitive Toolkit and Theano [36]. Keras is adopted by Tensorflow core officially and it is suggested as the official interface for deep learning [37].

To build this network, an input layer should be defined as a placeholder first. Then we define two subnets separately. Each of them is a Keras Sequential Model and the command lines used is intuitive like showed in ***Listing 3***. Each subnet has three convolutional layers, three batch normalization layers, 3 activation layers and 1 reshape layer. After defining two subnets, the GRU module is created. It is also a Sequential Model with 2 GRU layers and one dropout layer.

```
model = keras.models.Sequential([
        keras.layers.Conv2D(filters=64,
            kernel_size=[3,3],
            strides=[1,1],
            padding='valid',
            input_shape=[3,10]),
        keras.layers.BatchNormalization(scale=False),
        keras.layers.ReLU()
    ], name='Sample_Net')
```

Listing 3. Keras Sequential Model

The above three main components are built as an independent model. Then, the framework is created to combine those three models together. As shown in ***Algorithm 1***, the input layer is followed by a loop for time steps. In each time step, there is a slice layer to take one time step as the input for the individual convolutional subnet. After got the result from the first subnet, the result is expanded by one last dimension for the next subnet because it is flattened before. Then the expanded tensor is fed into merge convolutional subnet for high-level patterns. After getting the result from the second subnet, it is concatenated with a length one vector for time step size, and it is collected in an array. Next, we start again from beginning to process the second time step and store the result in the same array too. The results for all time steps are concatenated in time step dimension to form a matrix for the GRU module. The result of the GRU module is fed into final output layers with an average layer and a softmax layer to get the prediction

for each category.

Algorithm 1: DeepSense Neural Network Structure

Input: Input tensor
Result: Probability tensor

foreach *timestep* t **do**
    // Take one slice of timestep from input tensor X
    $X_{..t} \leftarrow$ slice_tensor(X);
    // Process in the first subnet and got flattened tensor
    $X_{out} \leftarrow$ individual_convolutional_subnet($X_{..t}$);
    // Expand one last dimension for the next subnet
    $X_{out} \leftarrow$ expand_dimension(X_{out});
    // Process in the second subnet and got flattened tensor
    $X_{out} \leftarrow$ merge_convolutional_subnet(X_{out});
    // Concatenate output tensor with timestep τ
    $X_{out} \leftarrow$ concatenate(X_{out}, τ);
// Concatenate all outputs for timesteps
$X_{out} \leftarrow$ concatenate($list\{X_{out}\}$);
// Process in GRU module
$X_{out} \leftarrow$ stacked_gru(X_{out});
// Process in output layers
$X_{out} \leftarrow$ average(X_{out});

The summary of the configuration and the details of each layer is written in ***Appendix A***.

3.6 Summary

In this chapter, the methodology is introduced in every aspects and detail. The data is collected from an app from an Android Nexus 5 phone and written in CSV format files. The files need to be transferred to the desktop computer manually since there is no server-side deployed. The vibration data from the files are read and split into segments. The segments are then used for feature extraction and feature selection using $tsfresh$. During the process, over 100 time series features are extracted and sifted based on hypothesis tests. The feature matrix is created after the process and passed into a DeepSense neural network framework. A reshape process of the feature matrix needs to be done instead of using it directly. The reshaped tensors are used by CNN to

predict the categories. The road surface recognition is based on the prediction from the methodology. The experiment details and the testing environments will be introduced in the next chapter.

4 Results and Discussion

4.1 Experimental Strategy

The Experiment and testing was done in April at the University of Tartu, Tartu City, Estonia. We selected several roads in Tartu to collect the motion sensor data. These roads are classified into three main categories which are smooth, bumpy and rough which are showed in ***Figure 7***.

- Smooth road type is the road surfaces with almost none anomalies such as freeways and some expressways that have well constructed and new road surfaces.
- Bumpy road type is the road surfaces with plenty of anomalies such as stone roads that usually makes the vehicles bumpy while driving.
- Rough road type is the road surfaces with no asphalt on the top such as off-road and some paths in the unpopulated areas.

From now on we will adopt the following annotation for our categories:

Table 1. Annotation table.

Annotation	**Name**
Cat.1	Category one, smooth road
Cat.2	Category two, bumpy road
Cat.3	Category three, rough road

It is noteworthy that the bumpy and rough road types are close to each other in terms of vibration patterns. Both of them are quite rugged, and the acceleration values are similar. It is a difficult task to separate those two categories and recognize them accurately. After our methodology, we got an overall 84.81% accuracy for three categories on 10th-fold cross-validation. It is also compared to other conventional methods which have an around 63%-76% accuracy.

4.1.1 Data Acquisition

The test for the proposed methodology can be divided into two stages. The first stage is collecting all the necessary sensor data. An app developed in the ITS lab is preinstalled

Figure 7. Three types of road surfaces tested. The upper one is smooth road surface while the bottom left one is bumpy road surface and the bottom right one is rough road surface. [39]

in one Nexus 5 smartphone and this app will collect the vibration from accelerometer and gyroscope sensors. A compact SUV vehicle is used for profiling the road surfaces. Here, we only used one smartphone with one car only due to the financial limitations. However, we did many collection sessions in order to have a larger volume of data.

Before starting the collection session, the smartphone should be fixed with a constant angle in the car by using arbitrary methods. As showed in ***Figure 8***, a phone holder is used in our methodology because it is easy to deploy and fixing the phone posture well. The angle is nearly 90 degrees perpendicular to the ground reference, and it can be inaccurate, but it must be fixed during the experiment. The application will transform the acceleration values with any posture into world coordinate system reference, thus technically, the angle fixing the phone does not influence the collection, but we still suggest to fix the posture of the phone while collecting for better accuracy.

Figure 8. The phone holder used while testing. [40]

The collection session takes place at Tartu city roads and some off-road in Tartu county, where has aforementioned three types of road surfaces. While driving, the speed is controlled around 40 km/h for urban roads, 30 km/h for off-roads and 90 km/h for highways. We repeated the experiment several times with slightly different speeds and lanes to get more data. The length of each collection is selectable from 10 seconds, 20 seconds, until 500 seconds. Hence, every time before start collecting, we have to specify the data length we desired. In total, we collected 98 files from the mobile app during the experiment and the length of the files are various but multiple of tens.

4.1.2 Data Preprocessing

All the files are stored in the external storage of the smartphone, and we need to move those files manually to the hard drive of the desktop computer. The computer holds 2 CPUs which are Intel(R) Xeon(R) Silver 4108 CPU @ 1.80GHz. Each CPU has eight cores, and the cores support hyper-threading with two threading. Thus, it has 32 logical cores in total. There are 46.8G ram and 4 GeForce RTX 2080 Ti GPUs also. We used a powerful computer because, in this way, we can speed up our execution and spend less time waiting for the results. The fact is that the processing is much faster than another MacBook Pro 15" 2016 and this allows me to put more time on coding.

The whole processing part on the computer using programming language Python3 and executed using iPython which is a powerful interactive Python shell. With the strong-capability computer and the convenient iPython shell, the preprocessing part can be done within one minute for all files. During the process, all the files are read, segmented, transformed and reassembled into 2 data sets for CNN. One is training data, and the other is training labels which contain around 400 10-seconds samples.

4.1.3 DeepSense framework

In this methodology, only one GPU is used since we do not have much data and the neural network framework is not complicated. Here, we performed the 10-fold cross-validation to get a comprehensive evaluation, which means in each round, we split one portion out from ten and use that portion for validation test. Scikit-learn library provides a useful tool to randomly split the data set into ten splits and return the ten folds' index by using $StratifiedKFold$. Another feature this function brings is that the volume of each category in a fold is the same. Here we try to avoid adding data bias to the neural network which may influence the decision of each category.

During the cross-validation, the DeepSense model is re-created completely for each round. All folds use the same training configuration, and the scores of each fold are saved and evaluated later. For each fold, we perform 1000 epochs. This value is determined by the volume of the dataset. The training status is plotted and showed in ***Figure 9*** As the training epochs increases, the training accuracy increased while validation accuracy also increased. However, the validation accuracy and loss started fluctuating while training accuracy almost reaches the summit. To get the best validation results, we continue training for few more epochs. In the training accuracy figure, the accuracy no longer got improved around 1000 epochs while the validation results were also good. Thus, we use this epoch value for cross-validation.

The other training configuration can be found in ***Appendix A*** for more details. There are also several callbacks used during the training process. These callbacks are useful while using Keras models.

- Checkpoint callback: for saving the best model while training, the callback will run after each epoch to compare the specified evaluation criteria with the best one.

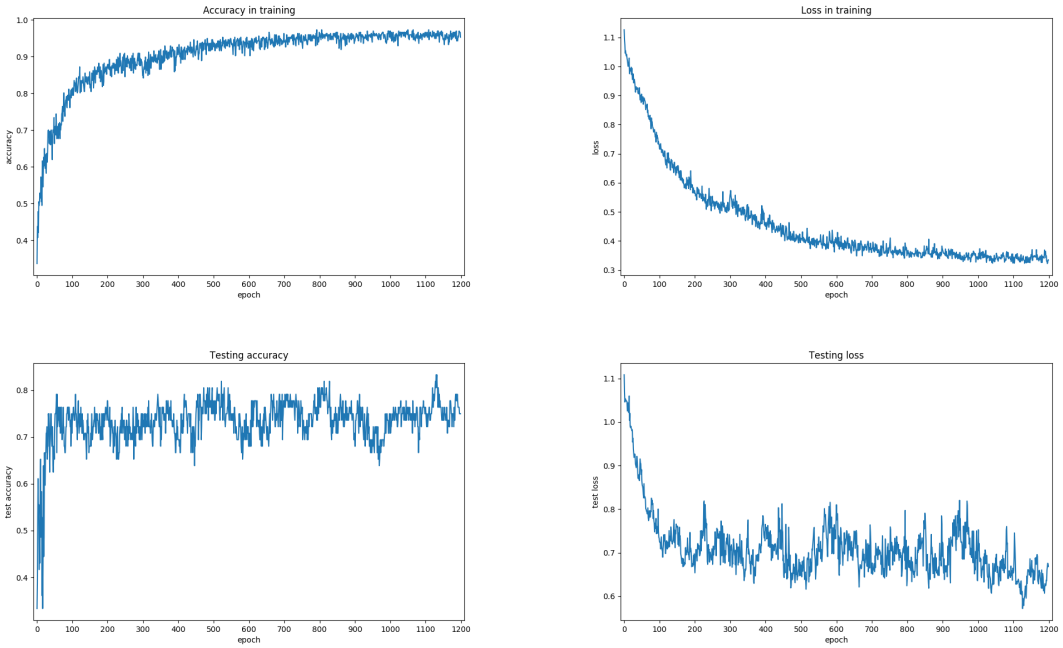

Figure 9. The training accuracy and loss during the testing. The upper left is training accuracy. The upper right is training loss. The bottom left is validation accuracy and the bottom right is validaiton loss.

If the score is better than the best one, this model will be saved and become the current best one.

- Early-stop callback: Stop training if the specified criteria are not getting improved for certain epochs. It also runs after each epoch.
- Plateau callback: Reduce learning rate when the specified criteria are not getting improved for certain epochs, and it runs after each epoch.
- CSV logger: Save the training log into a specified file.
- TensorBoard: Automatically run TensorBoard for the training model for better visualizations. It needs extra hard drive space to store the necessary files to run the TensorBoard.

These callbacks are quite useful while training and experimenting but these also

slow down the system and the whole process. Most of the time we only use checkpoint callback. However, we use other callbacks while developing and optimizing. In an actual usage environment, these callbacks can be disabled. Thus, it will be faster for execution.

The execution time for each sample is around 0.3 milliseconds and less than 0.9 seconds for the whole epoch. As a result, each fold needs around 10 minutes to execute. All the scores are collected after cross-validation, and an evaluation is performed based on the scores. The criteria are average accuracy, the standard deviation of the accuracy, and accuracies for each category.

4.1.4 Other methods

Several conventional classifiers are implemented for comparison as well such as normal CNN, SVM, and RandomForest.

Sequential CNN model & NN

The tested CNN is a simple convolutional neural network built from scratch which has six layers stacked together sequentially. The first three layers are convolutional layers which have $[d * 3]$ kernels and 64 filters, where d is the first dimension of the tensor mentioned in the last chapter. All the next three layers are fully connected layers which have 128, 32, and three neurons respectively. A softmax layer is added to predict the possibility of each categories using the output of the last layer.

The NN is also a 6-layer stacked sequential neural network, and all its layers are fully connected layers with 64, 128, 256, 128, 32, 3 neurons respectively. A softmax layer is used after these six layers to get the final probabilities of all categories.

The data set used is the same data for the proposed methodology, but split into the training set and testing set. When starting each evaluation, the data set will be split randomly each time and restart the experiment thoroughly. All the scores are recorded, and we use the averaged scores as their performance.

SVM & RandomForest classifiers

These two methods are easier to implement and usually faster than other neural networks methods. Scikit-learn also provides built-in classifiers for programmers to deploy the desired classifiers with custom configurations. The library gives full freedom for customizing the classifiers.

We use $sklearn.svm.SVC$ with RBF kernel and penalty parameter C equals to 1 for SVM classifier, and it also supports multi-class classification. RBF kernel is the default settings, and it can handle non-linear separations. However, this may sacrifice some speed performance than the linear kernel.

$sklearn.ensemble.RandomForestClassifier$ is used for RandomForest classifier. Here we use 100 estimators which means the number of trees used in the forest and leave all the other settings as default. The nodes will expand until there is pure end leaves or leaves contain less than two samples.

Because both of the classifiers take the 2D feature matrix to learn, we use the feature matrix before transformation to CNN tensors as input. There are 466 features for each sample, around 600 samples for training and around 200 for evaluation. Although the number of samples is different from CNN's dataset, they came from the same feature selection process, and the later one is transformed for higher dimensions. The categories counts in both training and testing set are equal. We did 10-rounds tests for Sequential CNN, NN, SVM, and RandomForest individually and used the averaged performance as the main criteria for evaluation.

4.2 Evaluation Results

As shown in ***Table 2*** are the result of 10-folds cross-validation for predicting the road surface type based on the motion sensor data. The overall accuracy on all three categories is 84.81% and its standard deviation is +/- 4.14% which means the proposed methodology can run with above 80% accuracy steady. The performance on each category is 94.36%, 73.14%, and 86.92% respectively, which is mainly because the smooth road surface can be distinguished well comparing with the other two road types. Smooth road type has fewer anomalies that make fewer bounces, thus has small sensor values and small deviation. The bumpy and rough road types are much more difficult to separate because both road surfaces contain many anomalies and we only have the training data from motion sensor which provides the similar vibration patterns on both road types. However, the proposed methodology still provides quite good results on those two road types with the standard deviation at +/- 11.64% and +/- 9.82% for bumpy and rough roads. The classification accuracy metric used here is based on the ***Equation 6***.

Table 2. The performance for 10-folds cross-validation.

	Overall accuracy	**Accuracy on Cat.1**	**Accuracy on Cat.2**	**Accuracy on Cat.3**
DeepSense variant network	84.81%	94.36%	73.14%	86.92%
Standard deviation	+/- 4.14%	+/- 5.05%	+/- 11.64%	+/- 9.82%

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

For comparisons with other methodologies, we used four different networks and classifiers to perform the same task. The tests used the same raw motion sensor data but different input shapes because it was formed according to the method adopted. We performed the tests for ten rounds, and the results are averaged. The overall results are shown in ***Figure 10***, and the evaluation criteria are accuracy and accuracies based on each category. The results prove that the proposed method surpasses all other tested methodologies. The CNN and NN are a little worse than our propose method but still good performance at 72.92% and 76.39% accuracy, while two classifiers only have 69.44% and 63.89% accuracy for SVM and RandomForest.

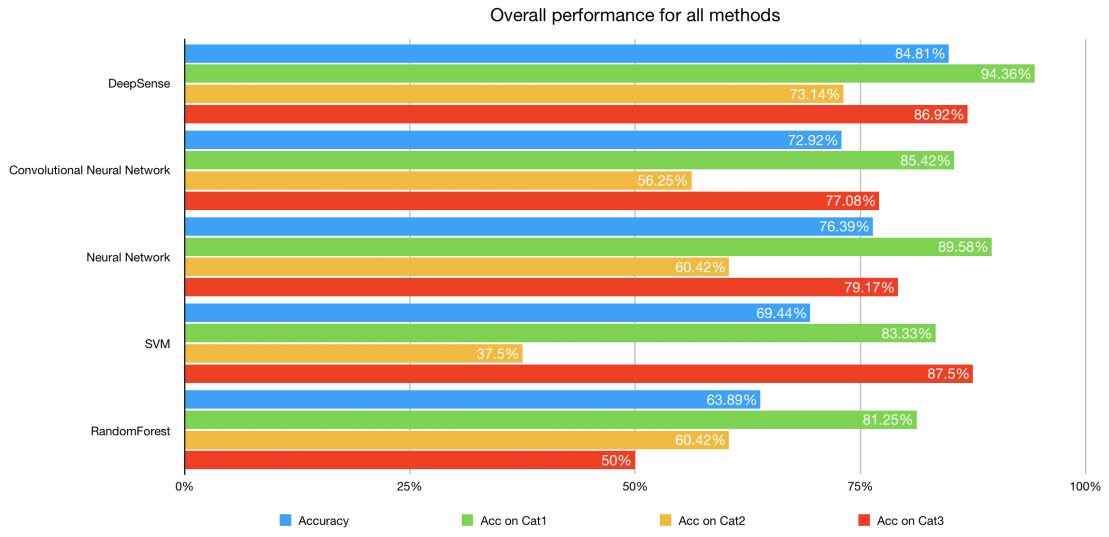

Figure 10. The overall performance on three categories for proposed methodology and other methods.

From the performance with each category, we can see that all of them got the best results in smooth road type because of the aforementioned reason that the smooth one is the easiest to distinguish. All the other methods suffered to separate bumpy and rough road types. Although, SVM had an even better result for the rough road at 87.5% accuracy. However, it sacrificed the accuracy on the bumpy road and got the poorest result on it.

4.3 Summary

In this chapter, we discussed the implementation details of the proposed methodology. All the specifications are introduced, and all the rest details are in ***Appendix A***. The performance of the proposed methodology is revealed and exceeds all other conventional methods for comparison.

In addition, the compared methods usually need an expensive process for preprocessing and feature engineering, but in our methodology, the raw data is almost unfiltered, and it executes massive feature extraction once. The compared methods also need to tune the parameters for the classifiers while the DeepSense CNN framework is built for learning from the time series data automatically and only need to customize the final output layers for the specific task.

5 Conclusions

In this chapter, we will draw a conclusion for this thesis and discuss the limitations during the research works. Finally, we will discuss at the end the future work and perspectives that can be carried out based on this thesis work.

5.1 Conclusions

In the field of Intelligent Transportations Systems, road surface recognition is an important requirements for autonomous vehicles and self-driving vehicles. The solution to this task provides the vehicles a well-understanding of the road conditions which is crucial for the safety issues. It is like part of human's eyes to the vehicles for sensing the road surfaces. The proposed methodology provides a way to recognize the road surface type by using smartphone's motion sensors.

During the research for related works, the data source mainly comes from vibration sensors, smartphone sensors, cameras, and LiDARs. These data sources can be fused for a preciser recognition, but sacrificing some of the speed performance. Moreover, lots of time needs to be spent on data engineering in order to make a clean, well-constructed feature matrix for recognition. We want to make the task simpler and better, and the proposed method is made for these purposes.

In this thesis, Feature Extraction based on Scalable Hypothesis tests (FRESH) is used for time series feature extraction and selection. The corresponding tool $tsfresh$ provides a framework to perform massive feature mappings. As a consequence, we no longer need to spend too much time on producing a feature matrix. The hypothesis tests also help us to examine whether a feature is related to the category in our task. In the second stage, a DeepSense neural network framework is used for learning information in the time series data. We built the network with the specifications tested by ourselves and customized the output layers for road surface recognition task.

The results of the proposed methodology meet our satisfaction and surpass conventional classifiers. For performing our experiment, no complex installation onboard the vehicle was needed. The only device was used is a smartphone with a holder to collect the data and a desktop PC for running the algorithm. Although it has the defect that the data need to be transferred manually, it is still convenient to run all processes. The output

is showed in three category labels indicates the roughness of road surfaces. It can also be used for continuous recognition which can help autonomous vehicles.

5.2 Future Perspectives

In future works, there are still many things that can be done. Now, there are still limitations for the proposed methodology such as the richness of the data, the filtering for the data, and the full integration on a mobile platform.

The data is collected in only one city for this thesis. In the future, we may collect more data and more road types to enrich our dataset in order that CNN can learn comprehensively for the road surface data. The vast of the data also solve some problem in machine learning like overfitting problem. The more data we can get, the less overfitting we may have.

Some filtering also can be done for the raw sensor data to remove some noise vibration such as the vehicle's vibration. Since Fourier transform and low-pass filters are well-known techniques for digital signal processing, we can also study the different type of vehicles to have a model for vehicles' vibrations.

Last but not least, we can integrate the methodology on a mobile platform thoroughly. Since the computation ability of the mobile devices is increasing fast and the light-weight characteristic of the CNN network used in this thesis, it is possible to move the methodology to a mobile device. Thus in the future, this solution can be used on regular vehicles directly and simultaneously.

References

[1] Number of smartphones sold to end users worldwide from 2007 to 2018 https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/ (checked 17.04.2019)

[2] TensorFlow Lite https://www.tensorflow.org/lite/ (checked 19.05.2019)

[3] Ronao, Charissa Ann, and Sung-Bae Cho. "Human activity recognition with smartphone sensors using deep learning neural networks." Expert systems with applications 59 (2016): 235-244.

[4] Kalra, Nidhi, and Susan M. Paddock. "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?." Transportation Research Part A: Policy and Practice 94 (2016): 182-193.

[5] Fwa, T. F., W. T. Chan, and C. Y. Tan. "Genetic-algorithm programming of road maintenance and rehabilitation." Journal of Transportation Engineering 122.3 (1996): 246-253.

[6] Sattar, Shahram, Songnian Li, and Michael Chapman. "Road surface monitoring using smartphone sensors: a review." Sensors 18.11 (2018): 3845.

[7] Yang, Bisheng, et al. "Computing multiple aggregation levels and contextual features for road facilities recognition using mobile laser scanning data." ISPRS Journal of Photogrammetry and Remote Sensing 126 (2017): 180-194.

[8] Aki, Masahiko, et al. "Road surface recognition using laser radar for automatic platooning." IEEE Transactions on Intelligent Transportation Systems 17.10 (2016): 2800-2810.

[9] Kim, T.; Ryu, S.K. Review and analysis of pothole detection methods. J. Emerg. Trends Comput. Inf. Sci. 2014, 5, 603–608.

[10] Haq, M. Uzair Ul, et al. "Stereo Based 3D Reconstruction of Potholes by a Hybrid, Dense Matching Scheme." IEEE Sensors Journal (2019).

[11] Staniek, Marcin. "STEREO VISION METHOD APPLICATION TO ROAD INSPECTION." Baltic Journal of Road & Bridge Engineering 12.1 (2017).

[12] Harikrishnan, P. M., and Varun P. Gopi. "Vehicle vibration signal processing for road surface monitoring." IEEE Sensors Journal 17.16 (2017): 5192-5197.

[13] Singh, Gurdit, et al. "Smart patrolling: An efficient road surface monitoring using smartphone sensors and crowdsourcing." Pervasive and Mobile Computing 40 (2017): 71-88.

[14] Khang Nguyen, Van & Renault, Éric & Ha, Viet Hai. (2019). Road Anomaly Detection Using Smartphone: A Brief Analysis: 4th International Conference, MSPN 2018, Paris, France, June 18-20, 2018, Revised Selected Papers. 10.1007/978-3-030-03101-5-8.

[15] Koch, Christian, and Ioannis Brilakis. "Pothole detection in asphalt pavement images." Advanced Engineering Informatics 25.3 (2011): 507-515.

[16] Quintana, Marcos, Juan Torres, and José Manuel Menéndez. "A simplified computer vision system for road surface inspection and maintenance." IEEE Transactions on Intelligent Transportation Systems 17.3 (2016): 608-619.

[17] Mednis, Artis, et al. "Real time pothole detection using android smartphones with accelerometers." 2011 International conference on distributed computing in sensor systems and workshops (DCOSS). IEEE, 2011.

[18] Bhoraskar, Ravi, et al. "Wolverine: Traffic and road condition estimation using smartphone sensors." 2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012). IEEE, 2012.

[19] Allouch, Azza, et al. "Roadsense: Smartphone application to estimate road conditions using accelerometer and gyroscope." IEEE Sensors Journal 17.13 (2017): 4231-4238.

[20] Hoffmann, Marius, Michael Mock, and Michael May. "Road-quality classification and bump detection with bicycle-mounted smartphones." Proceedings of the 3rd International Conference on Ubiquitous Data Mining-Volume 1088. CEUR-WS. org, 2013.

[21] Brillinger, David R. Time series: data analysis and theory. Vol. 36. Siam, 1981.

[22] Yao, Shuochao, et al. "Deepsense: A unified deep learning framework for time-series mobile sensing data processing." Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2017.

[23] Christ, Maximilian, Andreas W. Kempa-Liehr, and Michael Feindt. "Distributed and parallel time series feature extraction for industrial big data applications." arXiv preprint arXiv:1610.07717 (2016).

[24] Bishop, Christopher M. Pattern recognition and machine learning. springer, 2006.

[25] Overview on extracted features, tsfresh-Docs `https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html` (checked 03.05.2019)

[26] Lumley, Thomas. "Kendall's advanced theory of statistics. Volume 2A: classical inference and the linear model. Alan Stuart, Keith Ord and Steven Arnold, Arnold, London, 1998. §20.2

[27] R. A. Fisher, On the Interpretation of X2 from Contingency Tables, and the Calculation of P, Journal of the Royal Statistical Society 85 (1) (1922-01) 87.

[28] F. J. Massey, The Kolmogorov-Smirnov Test for Goodness of Fit, Journal of the American Statistical Association 46 (253) (1951-03) 68.

[29] Y. Benjamini, Y. Hochberg, Controlling the False Discovery Rate: A Prac- tical and Powerful Approach to Multiple Testing, Journal of the Royal Statistical Society. Series B (Methodological) (1995) 289–300.

[30] Wold, Svante, Kim Esbensen, and Paul Geladi. "Principal component analysis." Chemometrics and intelligent laboratory systems 2.1-3 (1987): 37-52.

[31] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

[32] Zhang, Wei, et al. "Shift-invariant pattern recognition neural network and its optical architecture." Proceedings of annual conference of the Japan Society of Applied Physics. 1988.

[33] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[34] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555, 2014.

[35] TensorFlow - Homepage
`https://www.tensorflow.org/` (checked 03.05.2019)

[36] Keras Documentation
`https://keras.io/` (checked 03.05.2019)

[37] Get Started with TensorFlow - TensorFlow Core
`https://www.tensorflow.org/tutorials` (checked 03.05.2019)

[38] B.D. Fulcher and N.S. Jones. hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. Cell Systems 5, 527 (2017).

[39] Lille Road, Riia Road, and Lemmatsi County from Google Maps, 07 May 2019.

[40] Universal In Car Mobile Phone Holder `https://www.argos.co.uk/product/5469275` (checked 07.05.2019)

Appendices

A DeepSense variant network configuration

Here is a full table of the configuration used during the implementation and testing.

Name	**Value**	**Name**	**Value**
Conv1 kernel	[d,5]	Window size	2
Conv1 stride	[d,1]	Window width	51
Conv2 kernel	[1,3]	Window shape	[1,1]
Conv2 stride	[1,1]	Window duration	5
Conv3 kernel	[1,3]	K size[1]	1
Conv3 stride	[1,1]	N featuers	10
Conv4 kernel	[K size,5]	Learning rate	0.001
Conv4 stride	[K size,5]	Decay	0.0
Conv5 kernel	[1,3]	Optimizer	RMSprop
Conv5 stride	[1,1]	Loss	Categorical crossentropy
Conv6 kernel	[1,3]	Metrics	Accuracy
Conv6 stride	[1,1]	d[2]	1
GRU dropout	0.5	Checkpoint callback[3]	1
GRU units1	10	Monitor metric	val_acc[4]
GRU units2	3	Shuffle[5]	True
Conv filters	64	Batch size	128
ICS input shape	[d,N features,1]	Epoch	1000
MCS input shape	[1,256,1][6]	Plateau patience[7]	50
GRU input shape	[2,31745][6]	Plateau fraction	0.8

Table 3. DeepSense variant network configuration

The convolutional layers one to three are from individual convolutional subnet while the layers from four to six are from merge convolutional subnet. N features are created

by the PCA process which reduces the dimension from over 300 to 10. The output of the model will be a series of vectors for two timesteps which will be used to generate the probabilities for each category.

[1]K size is the number of sensors. We use only acceleration values from the smartphone's motion sensor. Thus it is set to one.

[2]d is the number of the axis from the input data.

[3]It is a callback used during training in order to save the best model. One means it will check monitored metric before saving in every epoch.

[4]Monitor metric used for callbacks. If monitored metric improved, activate callbacks.

[5]Shuffle the training data before every epoch during the training process.

[6]The shapes of merge convolutional subnet and stacked GRU module are calculated based on the configuration of the layers.

[7]Plateau callback is used when the monitored metric no longer got improved for a patience period, and it decreases the learning rate by the fraction.

B Feature mappings

Here is a full table of feature mappings. It is also showed in the $tsfresh$ official documentation [25].

Mapping	**Description**
Absolute energy	Returns the absolute energy of the time series which is the sum over the squared values
Absolute sum of changes	Returns the sum over the absolute value of consecutive changes in the time series.
Aggregation over autocorrelation	Return the value of an aggregation f_{agg} over the autocorrelation $R(l)$ for different lags.
AR coefficient	This feature calculator fits the unconditional maximum likelihood of an autoregressive AR(k) process.
Autocorrelation	Calculates the autocorrelation of the specified lag.
Aggregation linear trend	Calculates a linear least-squares regression for values of the time series that were aggregated over chunks versus the sequence from 0 up to the number of chunks minus one.
Approximate entropy	Returns approximate entropy.
Augmented Dickey-Fuller test	This is a hypothesis test which checks whether a unit root is present in a time series sample.
Binned entropy	First bins the values of x into max_bins equidistant bins.
C3	Discrimination power of measures for nonlinearity in a time series.
Change quantiles	First fixes a corridor given by the quantiles ql and qh of the distribution of x. Then calculates the average, absolute value of consecutive changes of the series x inside this corridor.
CID CE	This function calculator is an estimate for a time series complexity.
Count above mean	Returns the number of values in x that are higher than the mean of x.

Count below mean	Returns the number of values in x that are lower than the mean of x.
CWT coefficients	Calculates a Continuous wavelet transform for the Ricker wavelet.
Energy ratio by chunks	Calculates the sum of squares of chunk i out of N chunks expressed as a ratio with the sum of squares over the whole series.
FFT aggregated	Returns the spectral centroid (mean), variance, skew, and kurtosis of the absolute fourier transform spectrum.
FFT coefficient	Calculates the fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast fourier transformation algorithm.
First location of maximum	Returns the first location of the maximum value of x.
First location of minimum	Returns the first location of the minimal value of x.
Friedrich coefficients	Coefficients of polynomial $h(x)$, which has been fitted to the deterministic dynamics of Langevin model.
Has duplicate	Checks if any value in x occurs more than once.
Has duplicate max	Checks if the maximum value of x is observed more than once.
Has duplicate min	Checks if the minimal value of x is observed more than once.
Index mass quantile	calculate the relative index i where $q\%$ of the mass of the time series x lie left of i.
Kurtosis	Returns the kurtosis of x.
Large standard deviation	Boolean variable denoting if the standard deviation of x is higher than r times the range = difference between max and min of x.
Last location of maximum	Returns the relative last location of the maximum value of x.
Last location of minimum	Returns the last location of the minimal value of x.
Length	Returns the length of x.

Linear trend	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one.
Linear trend timewise	The same as above. This feature uses the index of the time series to fit the model.
Longest strike above mean	Returns the length of the longest consecutive subsequence in x that is bigger than the mean of x.
Longest strike below mean	Returns the length of the longest consecutive subsequence in x that is smaller than the mean of x.
Max langevin fixed point	Largest fixed point of dynamics: argmax_x $\{h(x)=0\}$ estimated from polynomial $h(x)$.
Maximum	Calculates the highest value of the time series x.
Mean	Returns the mean of x.
Mean abs change	Returns the mean over the absolute differences between subsequent time series values.
Mean change	Returns the mean over the differences between subsequent time series values.
Mean second derivative central	Returns the mean value of a central approximation of the second derivative.
Median	Returns the median of x.
Minimum	Calculates the lowest value of the time series x.
Number crossing m	Calculates the number of crossings of x on m.
Number cwt peaks	This feature calculator searches for different peaks in x.
Number peaks	Calculates the number of peaks of at least support n in the time series x.
Partial autocorrelation	Calculates the value of the partial autocorrelation function at the given lag.
Percentage of reoccurring datapoints to all datapoints	Returns the percentage of unique values, that are present in the time series more than once.
Percentage of reoccurring values to all values	Returns the ratio of unique values, that are present in the time series more than once.

Quantile	Calculates the q quantile of x.
Range count	Count observed values within the interval [min, max).
Ratio beyond r sigma	Ratio of values that are more than $r * std(x)$ (so r sigma) away from the mean of x.
Ratio value number to time series length	Returns a factor which is 1 if all values in the time series occur only once, and below one if this is not the case.
Sample entropy	Calculate and return sample entropy of x.
Set property	This method returns a decorator that sets the property key of the function to value.
Skewness	Returns the sample skewness of x.
Spkt welch density	This feature calculator estimates the cross power spectral density of the time series x at different frequencies.
Standard deviation	Returns the standard deviation of x.
Sum of reoccurring data points	Returns the sum of all data points, that are present in the time series more than once.
Sum of reoccurring values	Returns the sum of all values, that are present in the time series more than once.
Sum values	Calculates the sum over the time series values.
Symmetry looking	Boolean variable denoting if the distribution of x looks symmetric.
Time reversal asymmetry statistic	Calculate the value of highly comparative feature.
Value count	Count occurrences of value in time series x.
Variance	Returns the variance of x.
Variance larger than standard deviation	Boolean variable denoting if the variance of x is greater than its standard deviation.

Table 4. Table of feature mappings.

Non-exclusive licence to reproduce thesis and make thesis public

I, **Shan Wu**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Road Surface Recognition Based on DeepSense Neural Network using Time Series Accelerometer Data

   supervised by Dr. Amnir Hadachi

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Shan Wu
20.05.2019