POSTURE DETECTION OF SMARTPHONE USERS USING DEEP LEARNING

BY

TAN SONG LIM

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

JANUARY 2018

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: ____

____

____

Academic Session: _____________

I ____

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in

Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

	Verified by,
_________________________	_________________________
(Author’s signature)	(Supervisor’s signature)
Address:	
_________________________	
_________________________	_________________________
_________________________	Dr.Ng Hui Fuang
Date: ____________________	**Date**: ____________________

POSTURE DETECTION OF SMARTPHONE USERS USING DEEP LEARNING

BY

TAN SONG LIM

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

JANUARY 2018

DECLARATION OF ORIGINALITY

I declare that this report entitled “**POSTURE DETECTION OF SMARTPHONE USERS USING DEEP LEARNING**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : ___________________________

Name : ___________________________

Date : ___________________________

ACKNOWLEDGEMENTS

I am grateful to have those with whom provide me assistance and courage to carry out my final year project. I would sincerely like to thank Dr. Ng Hui Fuang, the supervisor of my final year project for given me the chance to be involved in a deep learning related project. His willingness to sacrifice his time in order to guide me and provide valuable suggestion is deeply appreciated.

I would like to thank Jacqueling Lee Fang An and Dr. Ooi Boon Yaik for sponsoring an Android application which is able to collect raw sensors data from the smartphone built in sensors such as accelerometer, magnetometer and gyroscope. A million thanks for their help in the project.

Last but not least, I would like to express my great appreciation to my family for their love, unconditional support and continuous encouragement throughout my final year project.

ABSTRACT

Emerging of new technology is unavoidable in this modern era. Deep learning technology arises and gets into the eyes of many researchers, achieving wonderful result in different fields which includes computer vision, speech recognition, sentiment analysis and so on. However, there seems to be lack of developers who utilize deep learning in predicting the hand posture of user and phone placement of smartphone. This project involves applying deep learning technology in predicting the hand posture of a user while interacting with smartphone and its placement while it is idle. The final product of this project will be an Android application which is able to do hand posture and phone placement detection correctly. A long short-term memory (LSTM) network will be built from scratches and deployed into Android platform. Different optimization such as different activation function, optimization algorithms, updater, loss functions and other hyperparameter will greatly affect how the model going to perform. In order to train and test the neural network, an Android application will be used to collect random set of raw sensors data produced by accelerometer, magnetometer and gyroscope of a smartphone. The network will later be deployed in smartphone and produced an application named ‘Scarecrow’. The result produced by Scarecrow is absolutely great. Among the 6 test cases (right hand, left hand, both hand, put on table, put in pocket, single hand), the result produced by Scarecrow are 74.44%, 76.66%, 84.44%, 100% , 95.56% and 100%.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
API	Application Programming Interface
BPTT	Truncated Backpropagation through Time
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL4J	DeepLearning4j
ELU	Exponential Linear Unit
GMM	Gaussian Mixture Model
GPU	Graphical Processing Unit
HMM	Hidden Markov Model
LSTM	Long short-term Memory
ND4J	N-Dimensional Arrays for Java
NFC	Near-field Communication
ReLU	Rectified Linear Unit
RFID	Radio Frequency Identification
RNN	Recurrent Neural Network
SD	Standard Deviation
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine

Chapter 1: Introduction

1.1 Motivation and Problem Statement

In the matter of hand posture and phone placement recognition, there is a number of researchers who applied algorithm and other methods as the solution. For instance, Goel, Wobbrock and Patel, (2012) had developed an application to do hand posture, grip and pressure detection with smartphone. With complex algorithm and careful design, the application produced a good result in term of accuracy. Even though the result is good, the complexity of algorithm filters out much of the people who can understand them. Not everyone possessed with the ability in understanding how the system works, its behavior and weaknesses. Besides, when an algorithm becomes complex, its vulnerability becomes an issue. Despite the good result, there remains space for improvement.

Besides, some research had used external devices connected to smartphone in order to detect posture. For example, Tanaka et al. (2015) had created a mobile application which monitors and detects head posture called "Nekoze". The application requires user to wear sensing glasses which send out sensor data related to head angle to smartphone for interpretation. The limitations are obvious. The application requires the users to wear an external equipment in order for the application to work. This research gives inspiration on developing a posture detection application based on the sensor data from smartphone's built-in sensors in the absence of external tracking devices.

Over the last few years, several good and useful techniques arise and in term of pattern recognition, deep learning definitely works great. In the current trend, application which practice deep learning approach to detect hand posture and phone placement doesn't seems to appear. When it comes to pattern recognition and huge volume of data, deep learning definitely outperform other approach by a significant amount. The advantages allow the detection and identification of hand posture and phone placement to have better result and easier to implement as it replaces complex algorithm with the process of training, tuning and testing neural networks.

1.2 Project Scope

The scope of the project includes building, training, testing and optimizing neural network which would predict hand posture and phone placement based solely on accelerometer, magnetometer and gyroscope data from smartphone built-in sensors. A LSTM network will be built from scratch and go through the process of training, tuning and testing before deployment.

1.3 Project Objectives

The objectives of the project are:

a. To identify hand posture and phone placement based on the sensors data from smartphone
b. To utilize deep learning in learning hand posture and phone placement
c. To create an Android application that predict user’s hand posture and phone placement correctly

1.4 Contribution

By the end of this project, an Android application based on deep learning which is able to recognize users’ hand posture and phone placement will be produced. This application is able to do recognition based solely on smartphone built-in sensors without any additional devices.

1.5 Background

Smartphone has becoming trendy and common. It is one of the information technology gadgets that everyone tends to have. With its mobility and small size, users able to carry it from one place to another easily. Nowadays, users prefer to use smartphone to get their things done compared with sitting in front of a desk facing a computer. Due to enormous number of smartphone users, interaction between the users with their smartphone has catch the eyes of many researchers. The way how the users interact with the phone may change the way how efficiently a phone works.

Among all the factors that affects how a phone functions, user's hand posture is one of the most important factor that have a gigantic impact on the phone. The changes of user's hand posture manipulates the way a mobile device works (Goel, Wobbrock and Patel, 2012). When a user interacting with a smartphone, several hand posture can be generated such as left hand, right hand, thumb, both hand and so on. These posture actually affects much of the performance of a smartphone. For instance, users normally use their dominant hand in holding and interacting with their smartphone in the case of one hand. If they switch their hand, the interaction between the user and the phone may not be that efficient.

Besides, phone placement detection is also a rather hot topic among the developers. For instance, place it in pocket, out of pocket, in a table and so on. Different developers had tried out different method to detect the position of the phone. For instance, RFID based approach. In a project developed by Wahl and Amft in the year 2014, the system developed continuously utilized near-field communication (NFC) technology equipped in every smartphone to read the data from a radio-frequency identification (RFID) tag.

As mentioned earlier, deep learning technology is currently a hot topic. Deep learning is emerging and it is inspired from the structure of human brain. It is an artificial intelligence mostly used in pattern recognition with huge amount of data sets which are impossible for human to process (Sun and Vasarhelyi, 2017). In discussing deep learning, the term neural network should be known beforehand. The term neural network is commonly referred to artificial neural network (ANN). Deeplearning4j (2017) stated that neural networks are defined as a set of algorithm which designed to recognize pattern and help on clustering and classifying. According to Nielsen (2015), a neural network has three different parts: input layer, hidden layer(s) and output layer. In a neural network, the leftmost layer is known as input layer while the rightmost layer is known as output layer (Refer to Figure

1.1). The rest of the layers in between the input layer and output layer are known as hidden layers. Each layer contains a number of nodes called neurons. For each neurons, it is assigned with a weight (coefficient). When a neuron receives an input (X_n), it will multiply it with the weight (W_n) and sum up with other dot product from other neurons. This can be summarized as summation of dot product (X_nW_n). This either increase or decrease the significance value of the inputs for the network to learn. Next, the sum is pass to a node's activation function to do classification leading to output.

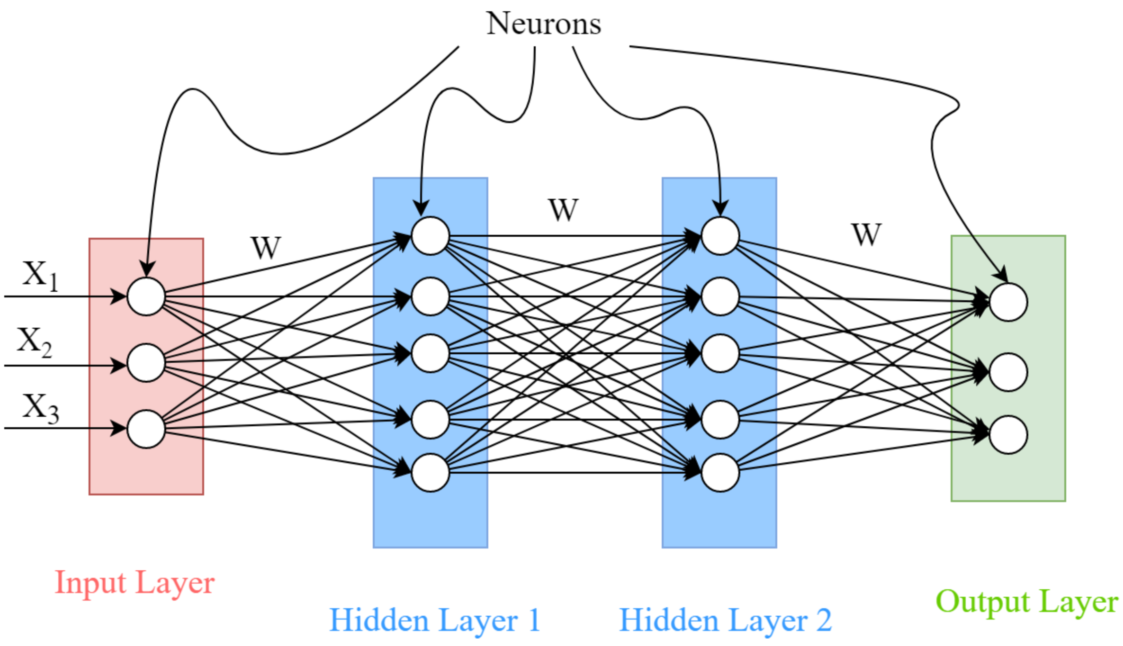


Figure 1.1: Simple Neural Network

In the field of deep learning, there are several architecture which are famous and practical in use. They are convolutional neural network (CNN), recurrent neural network (RNN) and others. For different cases, different architectures of network can be applied. For instance, CNN is well known in processing image and video frame. It has been proven that it performs well in task related to images/frames. For RNN, it is used to handle sequential data such as speech, sensors data, stock market price and so on.

In a nutshell, deep learning is the current trend right now and it definitely has a great advantage in solving different types of problems when vast volume of high quality data and resources such as graphical processing unit (GPU) are available.

1.6 Report Organization

This report consists total of 5 chapters. In the first chapter, motivations for working on this project, scopes of the project and project objectives are clearly defined. This chapter also introduces some background regarding the technology involved in the project. In the chapter 2, several papers regarding detection of users' hand posture and phone placement are studied, reviewed and compared. In the same chapter, information and knowledge regarding the technology involved in this project is explained.

For the next chapter, it explains the design of the long short-term memory (LSTM) network that are going to be built and deployed. Besides, it also contain the description regarding the methodologies and tools that had been used throughout this project. In chapter 4, the result of the final product are being shown and discussion had been done on the result.

The last chapter concludes the project and states some possible future improvements on the application.

Chapter 2: Literature Review

2.1 Hand Posture & Phone Placement System Review

2.1.1 Introduction

In the past few years, hand posture and phone placement detection and identification had become a relatively hot topic among the researchers. According to Wobbrock, Myers and Aung (2007), hand posture includes one hands, both hands, index fingers and thumbs will affect the performance of the phone crucially. Besides, the interaction with a smartphone is limited to touching on the front screen and some physical button along the sides of a phone (Zhang et al., 2015). Limitations in the different types of hand posture that a smartphone able to detect currently limits the ways how a user interacts with smartphone.

With regards to phone placement recognition, a variety of phone placement adopted by smartphone users such as carry it in bag, pocket, upper body and hands becomes a challenge (Incel, 2015). Furthermore, by running an application which process phone placement recognition on the smartphone, it consumes much of the resources and energy. According to Yang, Munguia-Tapia and Gibbs (2013), "an efficient approach is needed in sensing applications which are resource intensive and power consuming in terms of sensing, computation and communication".

Among all the solutions proposed by researchers regarding hand posture and phone placement, most of the solutions preferred machine learning. Compared to deep learning, machine learning actually focused more on feature engineering and less on feature learning. Although most of the solutions are machine learning related, the difference between the algorithms used by different researchers can be seen. Besides that, each solution has its own unique strengths and weaknesses in detecting different types of hand postures and phone placement.

2.1.2 Hand Posture Recognition

2.1.2.1 GripSense

This application was developed by Goel, Wobbrock and Patel in the year 2012. GripSense is a system that utilizes touchscreen, built-in sensors and built-in actuators of smartphone in making inference regarding hand posture, grip and pressure applied to the phone. It doesn’t require any additional sensors. For grip detection, even though no external devices needed, the user needs to interact with the smartphone in order for it to make inference about the detection. From the built-in sensor data, GripSense is able to interpret hand posture that the user performed such as thumb, index finger, the hand that hold the phone or whether the phone is lying on table. In order for the sensing mechanism to works, the researchers of GripSense put their effort in measuring tap size, arc of swiping motion and device’s rotation. With the combination of these three techniques, the accuracy of the application in sensing hand posture is relatively high. Besides hand posture, the application is also able to detect pressure and grip precisely.

For the measurement of device’s rotation, several cases had been discussed by the researchers regarding hand posture used. For instance, touches at the top of smartphone rotates the phone more compared with touches at the bottom due to range limitation of thumb. With regards to touch size, the researchers discussed that when the user operates the phone with one hand, the tap size will be bigger when the touch is far away from the thumb. In contrast, touches on the same side as the hand gives smaller tap size. For the last technique, it is only applicable when the user makes a swipe on the screen. According to Goel, Wobbrock and Patel (2012), index finger doesn’t really create an arc when swiping but for thumb, there will be a consistent and obvious arc either to the right or left depending on which thumb is being used. Flowcharts of core algorithms of three techniques are shown in Figure 2.1.

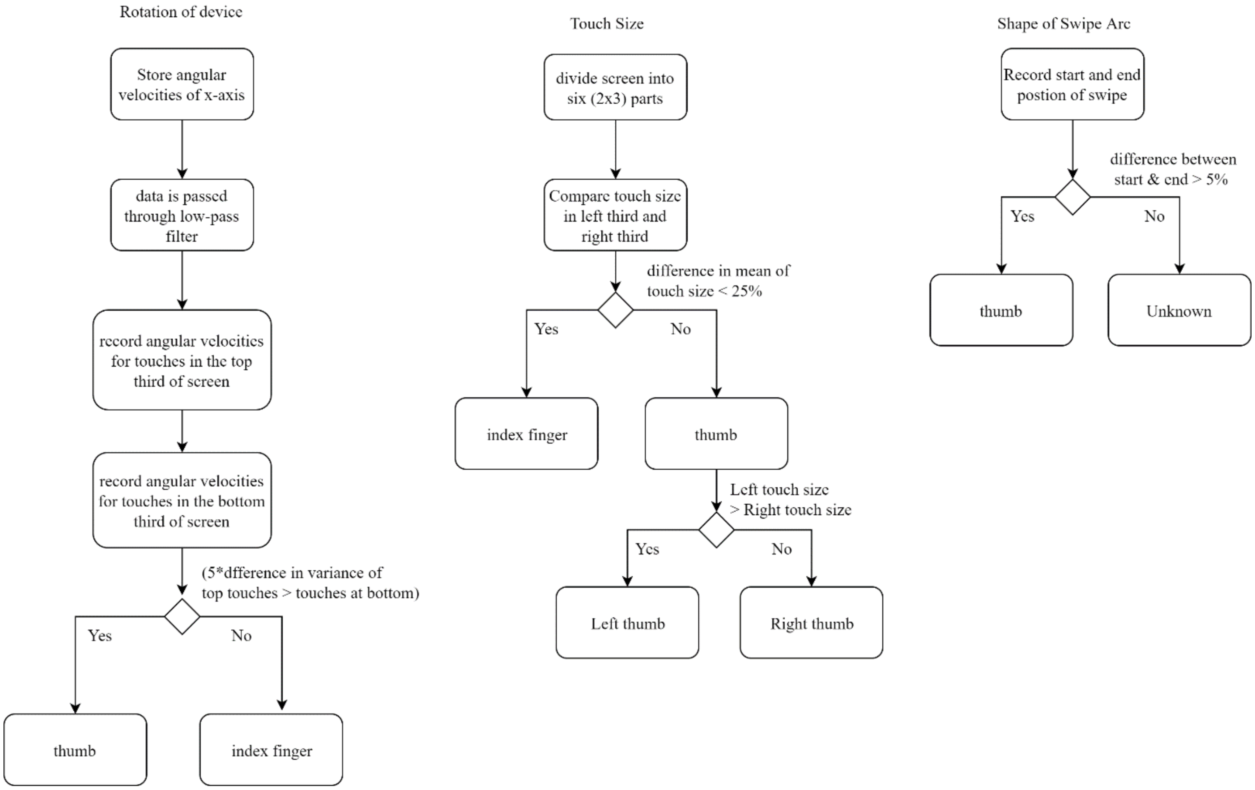

Rotation of device
Store angular velocities of x-axis
data is passed through low-pass filter
record angular velocities for touches in the top third of screen
record angular velocities for touches in the bottom third of screen
(5*dfference in variance of top touches > touches at bottom)
Yes
No
thumb
index finger
Touch Size
divide screen into six (2x3) parts
Compare touch size in left third and right third
difference in mean of touch size < 25%
Yes
No
index finger
thumb
Left touch size > Right touch size
Yes
No
Left thumb
Right thumb
Shape of Swipe Arc
Record start and end postion of swipe
difference between start & end > 5%
Yes
No
thumb
Unknown

Figure 2.1: Flowchart of Rotation of device, Touch Size & Shape of Swipe Arc

The combination of three techniques and majority voting in the final decision creates an algorithm which able to detect hand posture efficiently and effectively in GripSense and produces a good result for posture detection in term of accuracy.

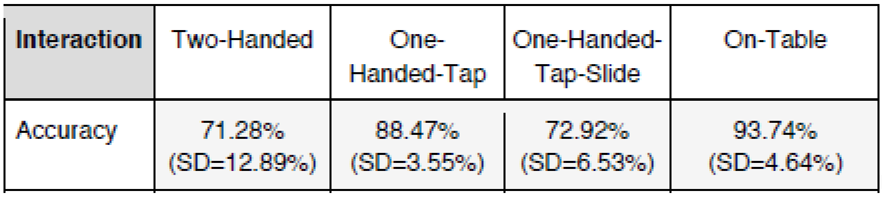
2.1.2.2 BeyondTouch

BeyondTouch is an application developed by Zhang et al. in the year 2015 with the motive to increase input languages by utilizing built-in sensors. The application extends a smartphone input variety with additional inputs and these additional inputs are categorized into two different groups, on-case interaction which includes phone and hands interaction, and indirect interaction which includes touching the surface the phone placed at. In the design of BeyondTouch, three interaction scenarios had been focused and they are one-handed, two-handed and on-table.

In the one-handed interaction scenario, it is not easy for a thumb to point or move across the entire front screen when the user is using the phone with one hand especially the current trendy smartphones are having large screen. For the two-handed interaction, when a user is holding the phone in landscape, the area that both thumbs can reach is relatively small. While for the on-table scenario, the researchers tried to capture any possible interactions around the phone without any additional sensors.

For all of these case scenarios, the researchers had used a combination of rule-based and machine learning approach to implement. Rule-based approach will be applied first to do segmentation and machine learning approach will do the classification job later. More data sets are collected for one-handed and two-handed interactions because these cases are dependent on how the user holds and interacts with the phone. The result in term of accuracy of BeyondTouch is above 70% for all three interactions scenarios and is shown in Table 2.1 below.

Table 2.1: Accuracy of each interaction techniques of BeyondTouch



Interaction	Two-Handed	One-Handed-Tap	One-Handed-Tap-Slide	On-Table
Accuracy	71.28% (SD=12.89%)	88.47% (SD=3.55%)	72.92% (SD=6.53%)	93.74% (SD=4.64%)

2.1.2.3 ContextType

ContextType is a system which leverages details about user’s hand posture in reducing screen text entry error. It was developed by Goel et al. in the year 2013. Some of the researchers who develop this system were from the team which developed GripSense. ContextType has different touch models for different hand posture. For instance, left thumb, right thumb, two thumbs and index finger. It switches between these models based on the inference about hand posture of the user. Just like their previous work GripSense, ContextType doesn’t require any additional sensor to do the sensing.

According to the research team, four different touch pattern models were created to change the underlying keyboard layout based on the user's typing behavior. The visual keyboard layout remains the same. The four models are for left thumb, right thumb, both thumb and index finger.

Besides that, the research team created a language model based on the works of other researchers. The language model works effectively in reducing error rates. Overall, ContextType is able to detect four types of hand postures. With the hand posture information combined with a language model, the error rate of participants was reduced by 20.6%.

2.1.3 Phone Placement Recognition

2.1.3.1 Discovery Framework

Discovery is a framework made by Miluzzo et al. in the year 2010. It is designed to detect phone sensing context based on mobile sensors with improved accuracy. According to the research team, phone sensing context refers to how the phone is carried in relation to the event being sensed. In the preliminary work of this framework, the research team put their focus in identifying whether the phone is in the pocket or out of the pocket. The framework design is separated into 3 level of inference which is shown in Figure 2.2.

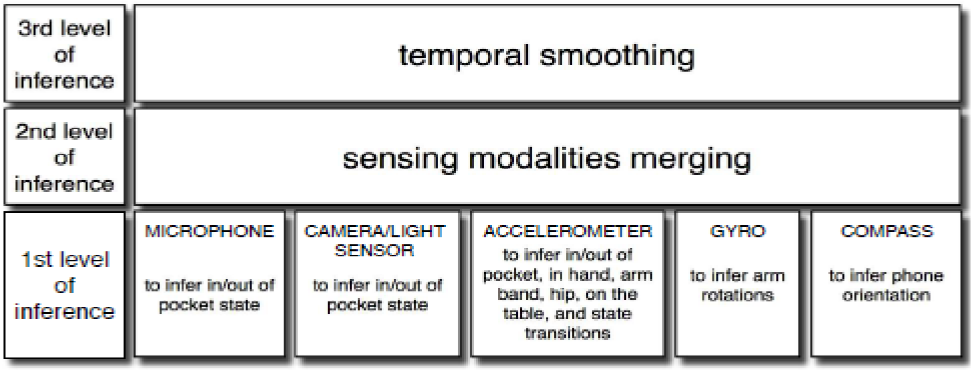

3rd level of inference
temporal smoothing
2nd level of inference
sensing modalities merging
1st level of inference
MICROPHONE
to infer in/out of pocket state
CAMERA/LIGHT SENSOR
to infer in/out of pocket state
ACCELEROMETER
to infer in/out of pocket, in hand, arm band, hip, on the table, and state transitions
GYRO
to infer arm rotations
COMPASS
to infer phone orientation

Figure 2.2: Discovery Inference Steps

The first level of inference is known as uni-sensor inference. In this layer, sensor data from a single smartphone built-in sensor gives hints about the context that the phone is sensing and it is not conclusive. The second level of inference is called multi-sensor inference. In this level, the input is the output of first level. At this phase, combination of data from different built-in sensor produces a more convincing result and gets closer in sensing the actual context. The third level of inference is known as temporal smoothing. Techniques such as temporal smoothing and Hidden Markov Model (HMM) are being used to process the output of second level.

Initially, the system includes Gaussian Mixture Model (GMM) and Support Vector Machine (SVM) in their inference models. GMM and SVM were included in the inference models with the motive to evaluate which technique is better and to see the adoption of more than one learning techniques. In the feature selection implementation, Discovery relies on a 23-dimensional feature vector and this vector is extracted from an audio clip.

Discovery framework achieves an accuracy of 80% during the evaluation with practicing single sensing modality which is the microphone. With more and more sensing models, the potential of Discovery framework can be exploited.

2.1.3.2 In-Pocket Detection

In-Pocket detection is an efficient algorithm created by Yang, Munguia-Tapia and Gibbs in the year 2013. Due to resource intensive context aware application, an efficient algorithm in the process of sensing is very important. In-Pocket Detection focused mainly in developing a more accurate, robust and energy-efficient recognition algorithm which detects phone placement context automatically.

In-Pocket Detection categorized phone placement into three types: “in pocket” (inPocket), “in bag” (inBag) and “out of pocket or bag” (outOfPocket). By recognizing one of these placement types, the accuracy of recognizing other context can be improved. The project utilizes the combination of embedded proximity and light sensors in sensing the phone placement surrounding. Next, the data from these sensors will be processed before data stream synchronization. After the synchronization of data stream based on a fixed timer, feature extraction will be performed. Figure 2.3 below shows the overall system design for In-Pocket Detection and Figure 2.4 shows the pseudo code of core algorithm:

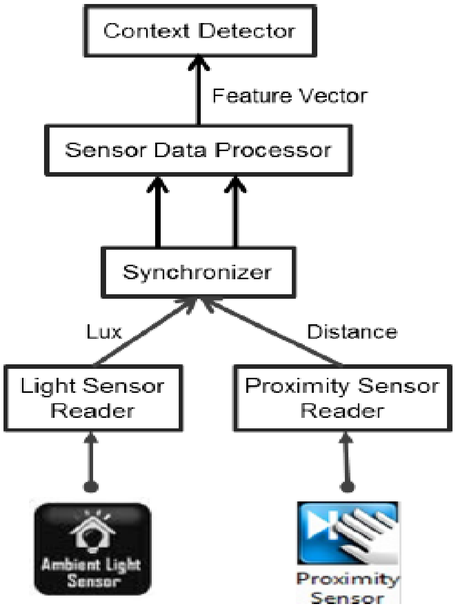
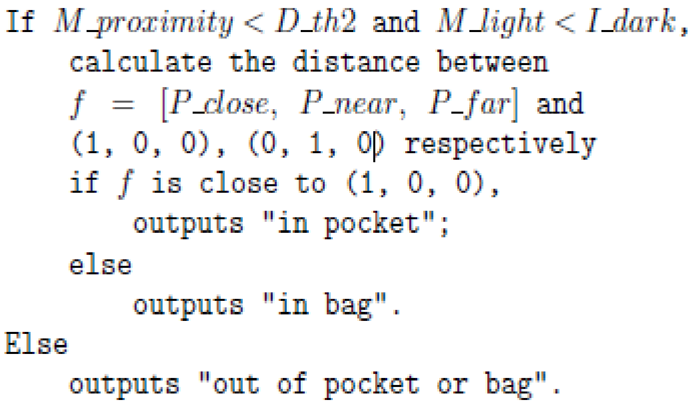


Figure 2.3: System Diagram of In-Pocket Detection



```
If $M\_proximity < D\_th2$ and $M\_light < I\_dark$,
    calculate the distance between
    $f = [P\_close, P\_near, P\_far]$ and
    (1, 0, 0), (0, 1, 0) respectively
    if $f$ is close to (1, 0, 0),
        outputs "in pocket";
    else
        outputs "in bag".
Else
    outputs "out of pocket or bag".
```

Figure 2.4: Pseudo code of Core Algorithm

Overall, In-Pocket Detection achieved an accuracy of 98% in their demo prototype. The system also successfully reduces the power consumption of sensor reading to less than 6mW. For the CPU resources, the processing power can be negligible.

2.1.3.3 ARService

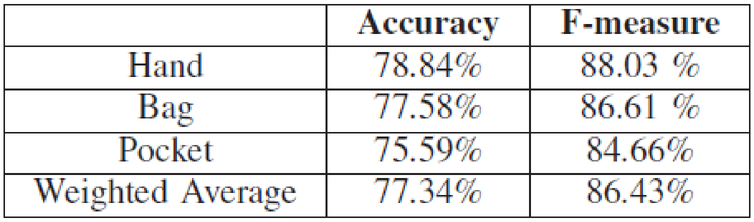
ARService is an Android application developed by Coskun, Incel and Ozgovde in the year 2015 to predict phone placement based solely on accelerometer data without any additional data from other sensors. The three main phone placement aimed to be detected are in a backpack, in the hand and in a pocket.

According to Coskun, Incel and Ozgovde (2015), “the effect of the location information on classifier performance is dependent on the activities and the locations involved.” Besides, the developers also mentioned resource optimization is a very important issue in running sensing application in obtaining sensor data.

The system will only utilize accelerometer as mentioned earlier. The data collected will be processed using Random Forest classifier from Weka machine learning toolkit. In details, features will be extracted from raw data and this feature set will be mapped to a set of activities by using a classification algorithm. The project team not only put their effort in position recognition, but also activity recognition. The feature sets used for both were different. However, random forest classifier was used in both cases in the classification stage. Classification used can be differentiated into two types: classification with angular acceleration features and classification with basic acceleration features. For classification with angular acceleration features, the researchers placed their concern on finding the orientation of smartphone using only information provided by accelerometer such as roll and pitch values. For classification with basic acceleration features, common features in activity recognition like mean and standard deviation are investigated to check which feature provides the highest accuracy among all.

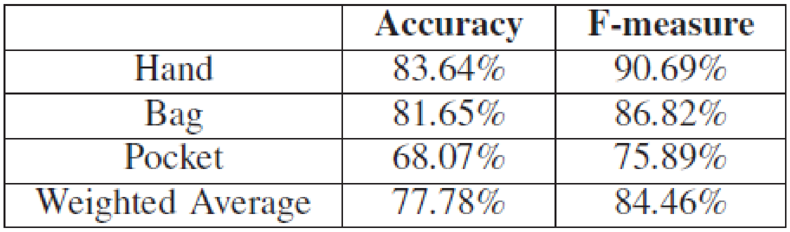
With only acceleration data, the result reached 77.34% in term of accuracy. By adding features such as mean, variance and standard deviation used in activity recognition, the accuracy increased to 85.04%. Overall, by just using acceleration data, the result wasn’t that bad. Without any additional data from other sensors, resource optimization is achieved in this project.

Table 2.2: Phone Position Recognition with Basic Acceleration Features



	Accuracy	F-measure
Hand	78.84%	88.03 %
Bag	77.58%	86.61 %
Pocket	75.59%	84.66%
Weighted Average	77.34%	86.43%

Table 2.3: Phone Position Recognition with Angular Acceleration Features



	Accuracy	F-measure
Hand	83.64%	90.69%
Bag	81.65%	86.82%
Pocket	68.07%	75.89%
Weighted Average	77.78%	84.46%

2.1.4 Strength of Solutions

2.1.4.1 Strength of Solutions: Hand Posture Recognition

System	Strengths
GripSense	• Achieved relatively high accuracy in hand posture, grip and pressure detection. For hand posture the accuracy is 84.3% on average while for pressure, the application differentiate 3 different levels with a success rate of 95.1%. • Used smartphone built-in sensors to do hand posture recognition, it doesn't require any additional devices to assist in the process of sensing context.
BeyondTouch	• The results of one-handed, two-handed and on table recognition is good in term of accuracy, range from over 70% to over 80%. • Used only smartphone built-in sensors to detect hand posture and extend the input language of smartphone. It doesn't require any external devices in sensing process. • Do personalization according to users. It requires only a small set of particular user training data to works on and personalize that user's model. This increases the accuracy.
ContextType	• Improved users typing experience by combining posture recognition with a language model to classify user pressed key and reduce the error rate. The application successfully reduced 20.6% of total text entry error rate.

2.1.4.2 Strength of Solutions: Phone Placement Recognition

System	Strengths
Discovery Framework	• By using only one sensing modality, the application achieve an average accuracy of 80% in placement recognition. The system has potential in growing, and when more sensors data are added on top of it, its performance might grow to a higher level. • The application requires no additional sensor in the process of processing phone placement recognition and thus, result in low cost.
In-Pocket Detection	• Achieved an accuracy of 98% with a demo prototype in recognizing two types of phone placement: “in pocket” and “out of pocket”. It performed better compared with ARService due to sensor data fusion over proximity and light. • The application is a pure light-weight software solution. It doesn’t need additional hardware which will increase the size and costs of smartphones. • The application is energy and resource efficient due to an efficient algorithm. Average of power consumption while doing the sensing is less than 6mW and CPU processing power is negligible.
ARService	• Successfully identified phone placement with an accuracy of 77.34% on three types of positions: in backpack, in the pocket and in the hand. The accuracy can be further increases to 85% when basic features such as mean, variance and standard deviation are added into angular feature calculated. • Optimize resource in term of computing and energy by using only the sensor data from accelerometer to do phone placement recognition.

2.1.5 Weaknesses of Solutions

2.1.5.1 Weaknesses of Solutions: Hand Posture Recognition

System	Weaknesses
GripSense	• For grip recognition, the user had to be interacting with the device so that the application can make inferences. Because GripSense used built-in vibration motor to detect pressure applied, the motor is triggered only when the user interact with the screen. • Pressure detection algorithm of the system works fine in limited phone. For different phone, the algorithm had to be adjusted in order to works perfectly. • In all sort of recognition performed by the application, many built-in sensors were used to do sensing. Power consumption of the smartphone is an issue.
BeyondTouch	• Energy consumption is an issue. Utilization of many built-in sensors in computing and sensing consumes much power and computing resources. • The application unable to bear with extra user’s hand movement. For example, the user shake the device heavily while performing a hand posture. • For two-handed interaction, when a user makes fast inputs (the difference between two inputs is less than 0.5 second), the classification result might be affected.
ContextType	• In processing hand posture recognition, a number of sensors are used in sensing the context. Energy optimization becomes an issue.

2.1.5.2 Weaknesses of Solutions: Phone Placement Recognition

System	Weaknesses
Discovery Framework	• The application practiced single sensing modality approach, the only sensor used is the accelerometer, resulting in lower accuracy when compared with others. • Simple heuristics which derived from a small data set was used in the application to determine the classification rules can be inaccurate..
In-Pocket Detection	• The application only able to detect two phone placement, either “in pocket” or “out of pocket”.
ARService	• Just like Discovery Framework, the application used single sensing modality approach and accelerometer is the only sensor being used.

2.1.6 Summary

For hand posture recognition, GripSense, BeyondTouch and ContextType produced satisfying result. All of these three system actually achieved quite high accuracy in the detection which range from 70% to 85%. With different approaches, previous researchers bounded to the same objective in developing their project as they tried hard to eliminate the needs of external hardware and utilized fully in the built-in sensors of a smartphone in recognizing hand postures of the users. Energy optimization is still an issue due to enormous number of sensors processing sensing jobs and collecting raw sensors data in order to determine hand postures.

With regards to phone placement recognition, the result is pretty good in term of accuracy for all three systems. Besides, the energy optimization had been achieved by all these system because these systems either practiced single sensing modality approach or implemented an efficient algorithm. As these systems main focus is in energy efficient, their limitations are also obvious. Single sensing modality approach provides insufficient information for a system to make excellent guess. Besides, In-Pocket Detection algorithm has limits phone placement to only two even though it is very efficient in saving energy. However, the limitations in these three systems are small and will not causes much issue to the overall performance of the system. Their potential can be exploited by including more sensors in predicting the phone placement.

In a nutshell, the six systems that had been reviewed have relatively high performance with few limitations. Despite prior research and system developed by previous researchers, there remains upgrade for a better system especially when new technology arises.

2.2 Introduction to Deep Learning

2.2.1 Introduction

Deep learning is very different from conventional machine learning techniques. Machine learning requires wide knowledge and engineering skill to build and design an extractor which will process the raw data before applying a classifier. Because of this, traditional machine learning is more towards feature engineering and focus less on feature learning. For deep learning, it is classified as high feature learning and low feature engineering. Deep learning is considered as representation learning, as it can takes raw data as inputs and find out the elements needed for classification. The function needed for the classifier to work is learn by the network through the training process. Layers by layers, in deep learning, higher layers in the model indicates critical attributes of the inputs which are important and essential in differentiating and crush those unrelated attributes. With deep learning, large amount of effort in planning a careful design can be avoided. Basically, deep learning is made up of multiple layers of nodes, and by going through the process of learning, it is able to transform raw data into desired outputs.

In the middle of 19th century, the concept of neural network was already been introduced. However, people preferred to use machine learning methods to solve a variety kinds of problem. This is due to several reasons. First, network based on deep learning is so hard to train at that time in the absence of backpropagation. Besides, the result produced by deep learning wasn’t fascinating enough for machine learner to change their interest. But now, with more computation power (introduction of GPU), more data and better algorithms and models, deep learning starts to get popular and definitely works great in variety of fields.

2.2.2 Backpropagation

Backpropagation is a technique which computes the gradient of the cost function from the output back to the inputs with respect to the weights (W_n) of multiple layers of nodes. With backpropagation technique, how the changes in weights and biases of multiple layers of nodes affects the overall performance of a neural network can be interpreted in detail. Backpropagation actually involves chain rule and that’s it. It uses chain rule to compute the gradient of the output with respect to neurons in the network. In the 19th century, neural network and backpropagation were ignored as most of the developers think that the best way to solve computer vision and speech recognition related problem is by having careful design and engineering skills to build a feature extractor and it is unavoidable. Later, deep learning and backpropagation technique begins to gain its popularity as graphics processing units (GPUs) are introduced to the world.

2.2.3 Activation Functions

Activation function is an important aspect in deep learning. It consists of non-linearity which will transform the weighted input (X_nW_n) of a node into output signal. Basically activation function decide whether a neuron should be activated or not. Without activation function, neural network collapse to linear functions and it can’t solve complex tasks. Some examples of activation functions are sigmoid, tanh, ReLU and so on. Different activation functions have different strengths and weaknesses.

2.2.4 Convolutional Neural Network

Convolutional neural network, often referred as CNN, is designed to process data in the form of n-dimensional array. For instance, image. In the design of CNN, it is a structure which build up from a series of stages. In the first few stages, a convolutional neural network is made up of two different layers: convolutional layers and pooling layers. Convolutional layers basically learns a set of filters and these filters are applied across different spatial locations locally and in parallel. The size of the filter is a hyperparameter. Convolutional layers is responsible to find the local conjunctions of features based on previous layer while pooling layers will find the similarity between features and merge those which are similar into one. CNN performs especially well in object detection, segmentation and recognition from image. Currently, there are quite a number of pre-trained model available for CNN such as AlexNet, VGG, ResNet, ZFNet, GoogLeNet,

LeNet and so on. These pre-trained models avoid the process of building a neural network starting from scratches. With these pre-trained models, the process of training can be speed up and it does improve the performance in overall.

2.2.5 Recurrent Neural Network

Recurrent neural network, often referred as RNN, is designed to deal with time series data and sequential inputs. It is a network with loops which allows information to stay. Figure below shows different design of RNN:

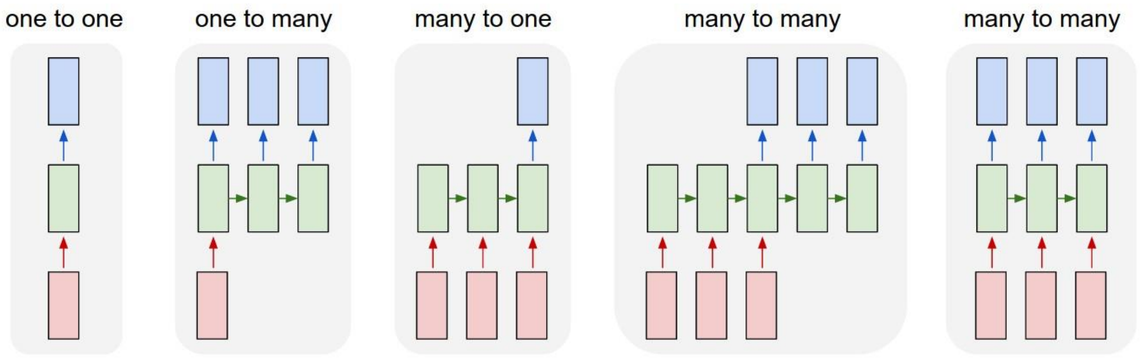


Figure 2.5: Different architecture of RNN

Without backpropagation, difficulty of training a RNN will increase a lot. RNN has proved itself to be a very powerful dynamic system. However, training process of a RNN is troublesome and problematic as the backpropagated gradients will either grow or shrink at each time steps. This leads to gradient vanishing or gradient explodes problem after several time steps. Some techniques are introduced to solve these situations. Gradient clipping can be implemented to solve gradient explodes issue while long short-term memory (LSTM) is an effective nodes designed to overcome gradient vanishing problem.

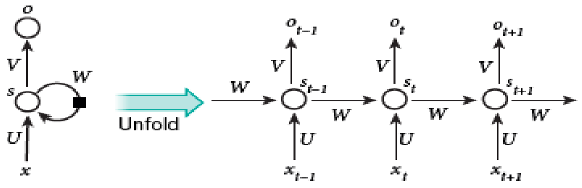


Figure 2.6: Unfolding of RNN in time computation

When a RNN unfolds at a certain time step, it can be seen as a feedforward network and the layers lies in the network are sharing the same weights (Refer to Figure 2.6). According to DeepLearning4j (2017), RNN has a feedback loop, which allows the output of last step n-1 to be fed back to the same neural network and make an influence in the result of step n and this continues for each following steps. RNN's main objective is to learn long term dependencies, however, it can only learn dependencies which are not too long. The introduction of LSTM perfectly solve what RNN couldn't do as LSTM able to learn long term dependencies well. Basically, LSTM is a special neuron which consists of 4 inputs and 1 output. It has 3 gates controlling the value in the memory and they are input gate, output gate and forget gate (Refer to Figure 2.7).

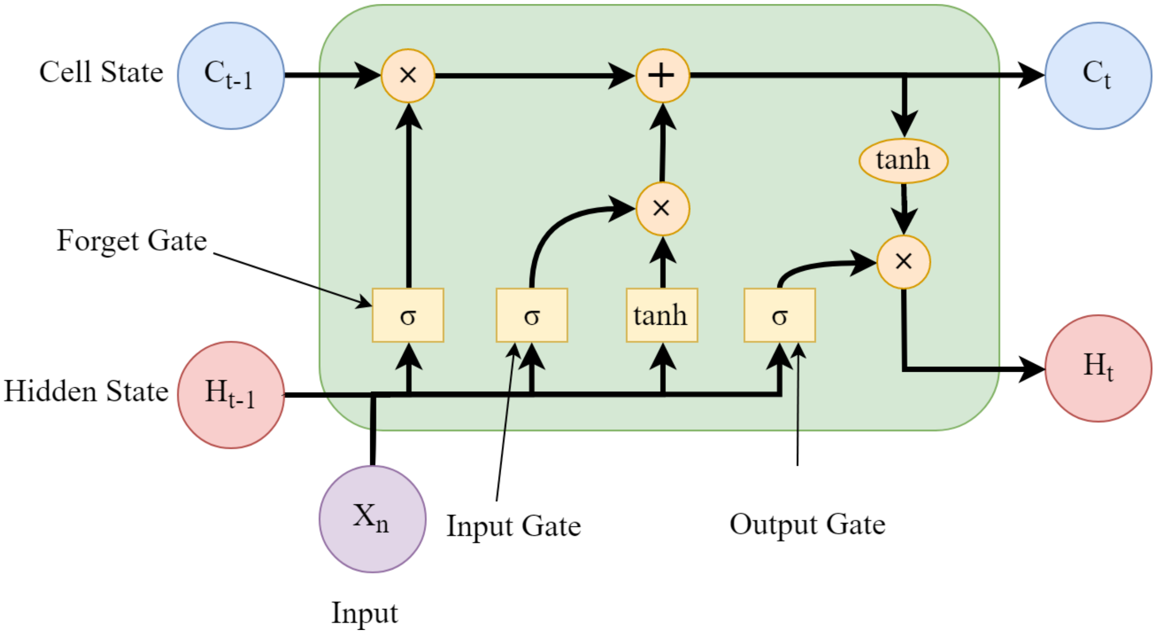


Figure 2.7: Structure of LSTM

Input gate is controlling the inputs passing into the memory. While the output gate decides whether the value in memory would affect the operation in the next step and for forget gate, it controls the value update in the memory and clears the memory if needed. The cell sate is like the memory for the LSTM. It either keep or discard the information stored in itself based on the signal from the forget gate. While for the hidden state, it is the output of the cell state.

Chapter 3: System Design

3.1 Hand Posture and Phone Placement

Hand posture is relatively important in the interaction between smartphone and human. The way how a smartphone user interacts with the phone can greatly enhance the user’s experience. Among various type of hand posture and phone placement, several types had been identified as the focus of this project (Refer to figures below).

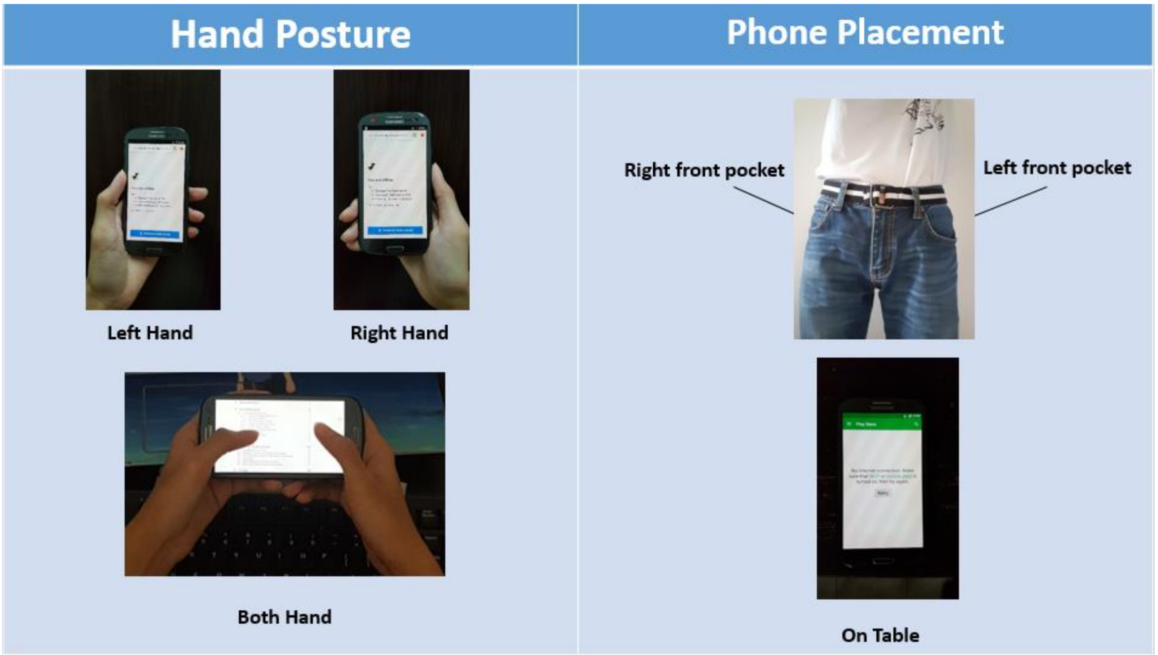


Figure 3.1: Hand Posture and Phone Placement

3.2 Train and Validation Data

For the training process of neural network, huge amount of high quality data is needed. An Android based application named Watcher is used to collect sensors data generated by the smartphone sensors. Watcher application is derived from an application sponsored by Jacqueling Lee Fang An with some modification such as setting file name, converting data collected into comma-separated values (.csv) format, different sampling rate and so on. The application practices multiple sensors approach which would collect a bunch of accurate and good quality sensor data. Sensors data collected are labeled with the action requested by the application to the users.

3.2.1 Multiple Sensors Approach

In the sensor data collection, three built in smartphone sensors had been utilized to collect the data and they are accelerometer, magnetometer and gyroscope. Accelerometer handles changes in orientation and measures the forces of acceleration which may be caused by earth's gravity. For magnetometer, it acts like a compass and detects Earth's magnetic fields. It would produces a simple orientation (X, Y, Z) about the smartphone. With these two sensors, orientation of the phone can be calculated. However, result outputted by magnetometer and accelerometer can be inaccurate as it contains a lot of noise. Among all the sensors, gyroscope is considered the most accurate sensors. Besides, it has relatively small response time compared to others. The sensors data collected from accelerometer and magnetometer are combined into 3 inputs while gyroscope contributes another 3 inputs resulting in a total of 6 inputs and these inputs will later be inserted to the network for train and validation.

3.3 Data Preprocessing

3.3.1 Training and Testing Stage

As mentioned earlier, sensor data collected are saved in comma-separated values (.csv) format. The data has 6 inputs (accel[X], accel[Y], accel[Z], azimuth, pitch, roll) and 6 outputs labeling the actions. These data will be used in training and testing phase of neural network before the deployment. As these data are raw sequence data, they can't be inputted directly to the neural network. Extraction of input sequence and output sequence pairs must be done before training or predicting can be done.

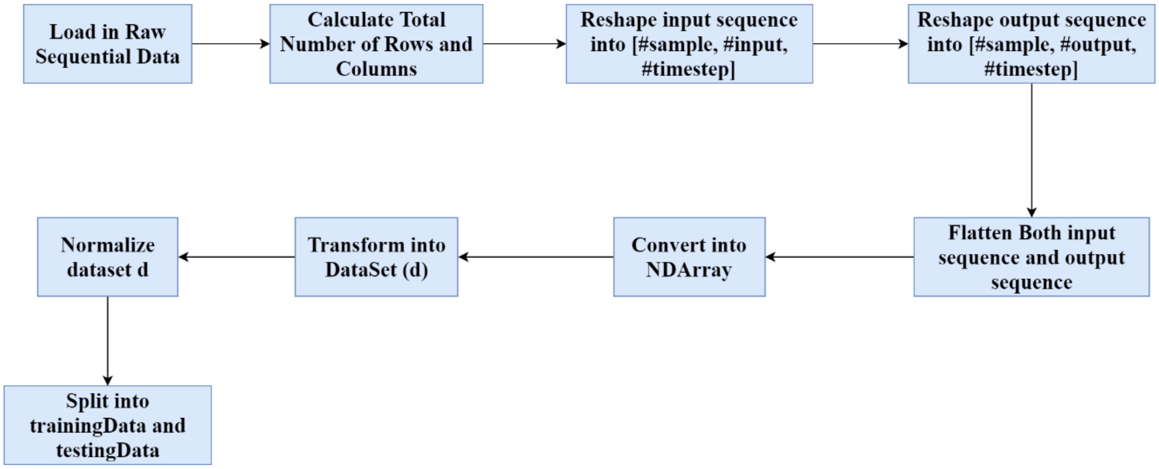

Load in Raw Sequential Data
Calculate Total Number of Rows and Columns
Reshape input sequence into [#sample, #input, #timestep]
Reshape output sequence into [#sample, #output, #timestep]
Flatten Both input sequence and output sequence
Convert into NDArray
Transform into DataSet (d)
Normalize dataset d
Split into trainingData and testingData

Figure 3.2: Data Preprocessing in Training and Testing Stages

By referring to the figure above, raw sequential data collected is first loaded. Next, total number of rows and columns of the data are calculated. This step is a must as different dataset may have different number of rows since the data collection process is random and control by timer. Each dataset contains roughly of 2800 rows of records. These sequential data are then separated into two: input sequence and output sequence. Both the sequences were reshaped into 3D array in the form of [#sample, #input/#output, #timestep] because the input to the LSTM library of DL4J has that particular form. Afterwards, both input sequence and output sequence are flattened and converted into NDArray object. According to ND4J (2017), NDArray are in essence n-dimensional arrays and are stored in the memory as a single contiguous block of memory which makes it different from typical Java arrays. Next, both converted input sequences and output sequences will be transformed into a dataset object. The dataset later undergoes normalization process and splits into training data and testing data in the ratio of 0.7 (70% for training and 30% for testing).

3.3.2 Deployment in Android

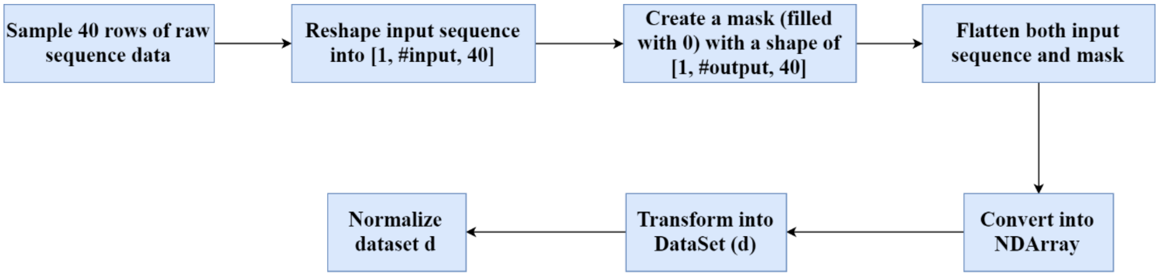

Sample 40 rows of raw sequence data
Reshape input sequence into [1, #input, 40]
Create a mask (filled with 0) with a shape of [1, #output, 40]
Flatten both input sequence and mask
Convert into NDArray
Transform into DataSet (d)
Normalize dataset d

Figure 3.3: Data Preprocessing in Android Application

The data preprocessing task in the Android deployment is almost the same as the one in the training and testing stage. Unlike the previous one, the raw sequence data is straight obtained from the smartphone sensors in a dynamic environment. These data are later reshaped and flattened together with a masking array. The masking array is an array consists of few columns of zeros so that the total number of columns are identical through all time (to fit the dimension). Next, both the input sequences and masking array are converted into NDArray and merged as a dataset object. The dataset is normalized using the normalizer that had been saved during the training process. After all these preprocess, the data is ready to be inputted into the neural network for prediction.

3.4 Design of Neural Network

3.4.1 Architecture of Network

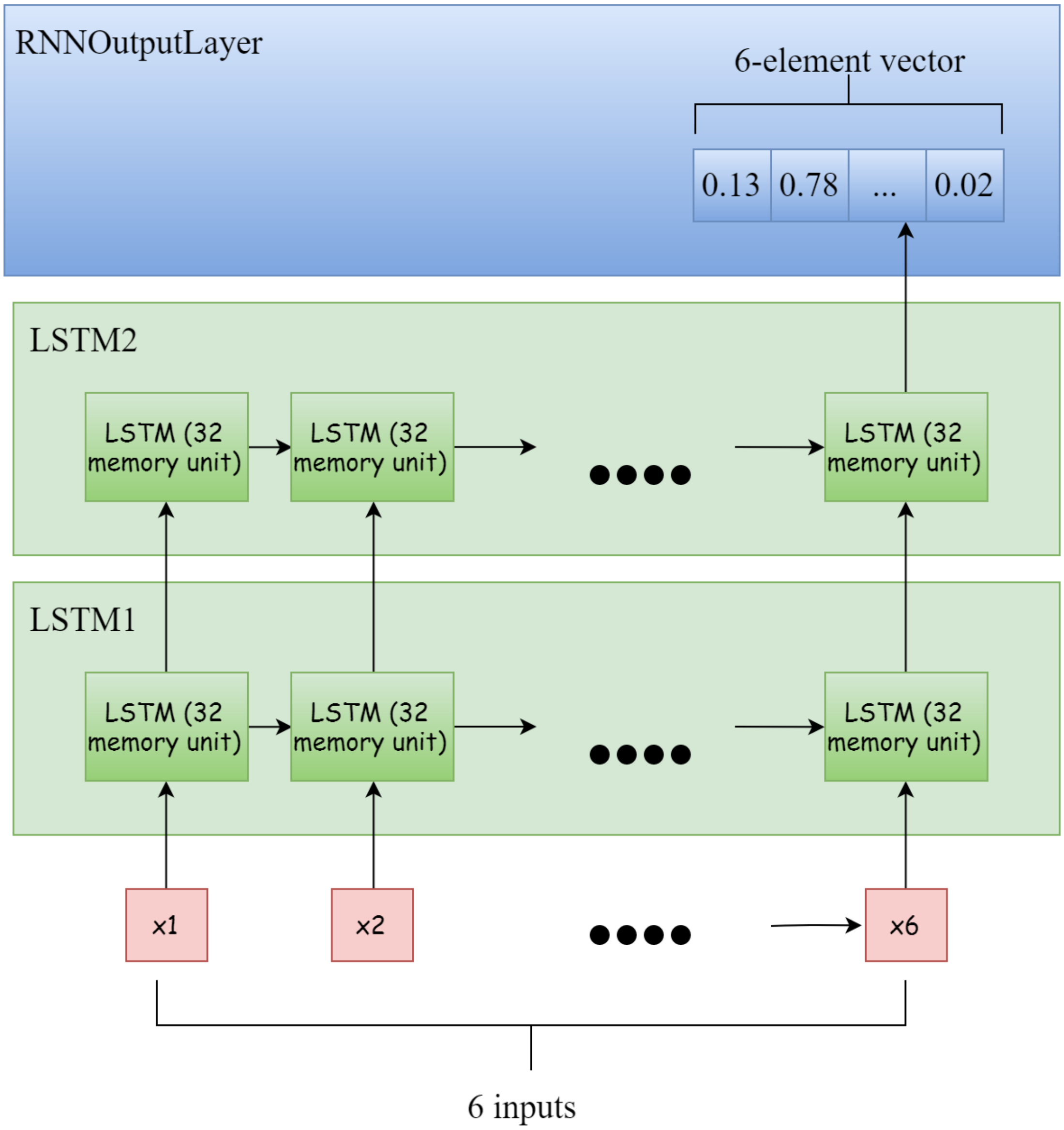


Figure 3.4: Architecture of Neural Network

In this project, a recurrent neural network with 2 layers of long short-term memory (LSTM) is built up from scratch and deploy after training. LSTM is a special node of RNN, which is able to learn long-term dependencies and eliminates some weaknesses of RNN. LSTM cells were introduced by Hochreiter and Schmidhuber in the year of 1997. Currently, this special cells are widely used in solving different types of complex problems and worked tremendously well. The network consists two layers of LSTM layers and for each LSTM layer, there are 32 memory unit. As the neural network will be deployed to Android platform later, it would be better if the overall size is keep small and enough to learn useful feature. Less computing resources and power needed when processing prediction if the neural network is small. Commonly, a recurrent neural network is constructed of 2 to 3 layers as going beyond 3 layers are proven to have no benefits empirically. Basically, the network takes in an input sequence constructed of 6 columns (sensor data) and produces an output vector containing the probability of 6 classes. A weights array had been added in the output layer because the dataset collected is imbalance (having more left hand and right hand data and the data collection process is random). Thus, the output vector which contains the probability of 6 classes will be averaged.

3.4.2 Cross Validation

In deep learning, hyperparameters are attributes related to the network that have to be set beforehand rather than learn. The process of deciding the value of hyperparameters can be difficult as it is very problem-dependent. In order to find the optimal value of hyperparameters for this particular scenario, cross validation technique had been applied. Cross validation is a technique which splits a dataset into several folds (k) with equal size. Among k number of folds, a single fold will be used as the validation set to evaluate the performance while the rest k – 1 folds will be utilized as the training set with different hyperparameter settings. The process is repeated k times and each fold will be the validation set exactly once. Lastly, the model with least validation error will be taken as the best model. This had been implemented using KFoldIterator class from DL4J library. Besides, random search is utilized together with this technique.

3.4.3 Hyperparameter

For different cases, value of hyperparameters can be different to suit each case. The setting of hyperparameter is time-consuming and every combination have to be tried out in order to find the best. In this project, several hyperparameter had been fine tune in order to find the optimal value which suits the case.

Size of LSTM layer

Empirically, deeper and larger network learns and performs better. However, the layer size of LSTM layers are kept small in this project because small neural network will not consume much computation power and battery when it is deployed in Android. The value is set as the following.

$* layer\_size = 32$

Input Sequence Length

This hyperparameter had been searched through cross validation technique in order to get an optimal value. The value is set as the following.

$* input\_sequence\_length = 40$

Learning Rate

Learning rate is considered one of the most important hyperparameter in deep learning. According to DL4J (2017), learning rate decides the size of a step to take towards smaller error rate as the networks weights get updated. When a learning rate is too high, the network may fails to converge or miss out local minimal. Vice versa, the network would takes a very long time to train as the step taken at every epoch is very small. In this project, the learning rate had been set to the following value:

$* learning\_rate = 0.08$

Regularization

Different design of neural network often suffered overfitting and underfitting problem. Regularization is a technique which deals perfectly against overfitting. Overfitting is a term which describe a network which predicts train data well but makes bad prediction on unseen data. Some common types of regularization that had been widely used are l1 and l2. Both l1 and l2 regularization penalizes the weight of a node when it is too large. However, they are quite different (Refer to Table 3.1).

Table 3.1: Difference of L1 and L2 Regularization

L1 Regularization	L2 Regularization
$R(W) = \sum_k \sum_l W_{k,l}$	$R(W) = \sum_k \sum_l W_{k,l}^2$
Computational inefficient (non-sparse case)	Computational efficient (analytic gradients)
Sparse output	Non-sparse output (Prefer diffuse weight vector)
Built-in feature selection	No feature selection

In this project, l2 regularization had been used with a regularization factor set to the following value:

$$*regularization\_factor = 0.002$$

Weight Initialization

In a neural network, each node are equipped with weights. Weights of each node shouldn't be too big or too small. If not, it would in term leads to problem such as difficulty in updating weights. The way how it is initialized will affect the performance of the network. Thus, weight randomization is very important at the initialization process before training. If there are two neurons holding the same weight value, they will not be able to diverge and learn different useful features. Xavier initialization is widely used currently and it is a work based on the paper written by Xavier Glorot and Yoshua Bengio. In this project, Xavier initialization will be utilized in initializing the weights.

Gradient Clipping

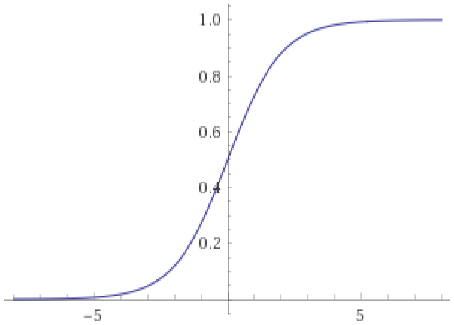
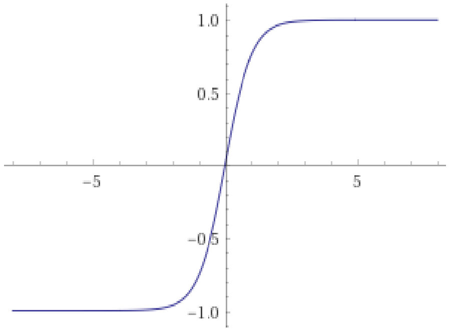
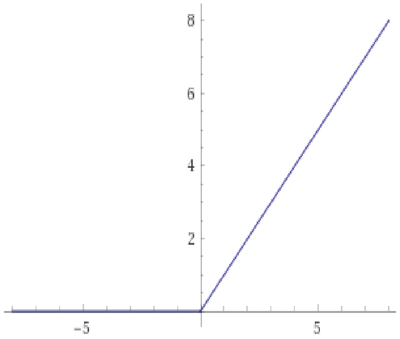
Gradient clipping technique had been utilized in the project because when the gradients propagated back (in training), there seems to be gradient exploding problem. Gradient exploding problem happens when the gradients grow exponentially large, thus causing difficulty in learning. It is very common in recurrent neural network. In this project, the gradient is clipped on a per-element basis. A hyperparameter was introduced and it is gradient threshold. For example, when gradients have a value larger than or smaller than the threshold, they will be truncated. Gradient threshold had been initialized to the following value in this project.

* $gradient\_threshold = 0.05$

Activation Function

Activation functions in neuron contains non-linearity which would decides whether to activate that particular neuron or not. Its non-linearity ensures that useful features can be learn throughout the training process. Currently, there are a variety types of activation functions available and some common activation details are listed below.

Table 3.2: Common Activation Functions Details

	Sigmoid	Tanh	ReLU
Equation	$1 / (1 + e^{-x})$	$(e^{x} – e^{-x}) / (e^{x} + e^{-x})$	max(0,x)
Graph			
Strengths	• Input squashed into [0,1]	• Input squashed into [-1,1] • Zero-centered	• Efficient computation • Converge faster • Does not saturated in the positive region
Weaknesses	• Not zero-centered • Computation of exp() is expensive • Gradients killed in saturated region	• Gradients killed in saturated region	• Not zero-centered • Saturated in the negative region • Dead ReLU problem

In the LSTM layers, tanh had been used as it is very common to use a combination of LSTM neuron together with tanh activation function even though ReLU converge faster. In the output layer, softmax activation function had been chosen as the choice because it is commonly applied in classification problem. Softmax is used to generate class probability on the number of classes being categorized and outputted a vector of size equals to total number of classes.

Truncated Backpropagation through Time Length

Truncated Backpropagation through Time (BPTT) is a very important hyperparameter in recurrent neural network especially dealing with long sequences data. According to DL4J (2017), BPTT is created in order to reduce training time and complexity of computing each parameter update in recurrent neural network. It trains the network faster by doing more parameter update. For example, given an input sequence of 10, 1 forward pass and 1 backward pass will be performed in normal backpropagation. For BPTT, it splits the passes into smaller set of operation. For instance, if the BPTT length is initialized to 5 in this case, total of 2 forward pass and backward pass will be done (Refer to Figure 3.4).

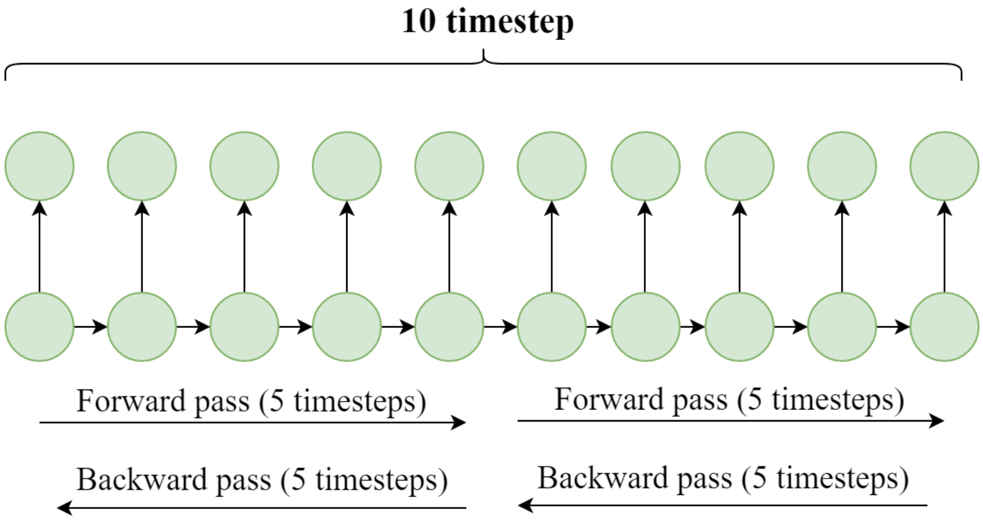

10 timestep
Forward pass (5 timesteps)
Forward pass (5 timesteps)
Backward pass (5 timesteps)
Backward pass (5 timesteps)

Figure 3.5: Truncated Backpropagation through Time

In this project, BPTT is initialized to the following value.

$* \ tbptt\_length = 4$

Optimizing Algorithm & Updater

In DL4J, optimization algorithm decides how the updates are performed given gradients. While for updater, it basically means training mechanisms such as Adam, adagrad, RMSProp, momentum and so on. In this project, stochastic gradient descent and momentum have been chosen as the optimization algorithm and updater for the network. For the momentum updater, it did introduced a hyperparameter called momentum rate. This hyperparameter had been tune using cross validation and the value is defined as below.

$* \ momentum\_rate = 0.85$

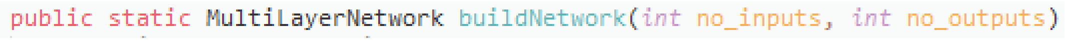
Total Number of Epochs

One epoch basically means one full pass of the training set through the network. In this project, the data set size is quite small, thus no mini batching had been done. All training data are passed into the network as a single batch. In the training process, the model which performs the best is saved and will be loaded in the next training. The total number of epochs is defined as below.

$* \ no\_epochs = 1024$

3.4.4 Functions and Methods

buildNetwork()



```
public static MultiLayerNetwork buildNetwork(int no_inputs, int no_outputs)
```

- ➢ Builds the architecture of the recurrent neural network and returns the model created.

- **Java**: public static MultiLayerNetwork **buildNetwork**(int no_inputs, int no_outputs)
- Parameter(s):
  - no_inputs – total number of inputs for the neural network.
  - no_outputs – total number of outputs for the neural network.
- Return:
  - A MultiLayerNetwork object.

<u>saveModel()</u>

```
private static void saveModel(MultiLayerNetwork net, String filename, boolean updater)
```

- Save the MultiLayerNetwork object into to a .zip file. If updater is true, the state of network's updater will be saved for further training. Vice versa, updater's state will not be saved.
- **Java**: private static void **saveModel**(MultiLayerNetwork net, String filename, boolean updater)
- Parameter(s):
  - net – MultiLayerNetwork object to be saved.
  - filename – path where the model is saved.
  - updater – decide whether to save network's updater state or not.
- Return:
  - None

<u>loadModel()</u>

```
private static MultiLayerNetwork loadModel(String filename, boolean updater)
```

- Load the model (MultiLayerNetwork) object given the file path.
- **Java**: private static MultiLayerNetwork **loadModel**(String filename, boolean updater)
- Parameter(s):

    - filename – path where the model is located.
    - updater – decide whether to load network’s updater state or not.
- Return:
    - A MultiLayerNetwork object.

saveNormalizer()

```
private static void saveNormalizer(SplitTestAndTrain s ,String filename)
```

- Save DataNormalization object for further use.
- **Java**: private static void **saveNormalizer**(SplitTestAndTrain s ,String filename)
- Parameter(s):
    - s – splits dataset into training set and testing set for normalizer.
    - filename – path where the normalizer is saved.
- Return:
    - None

loadNormalizer()

```
private static void loadNormalizer(String filename)
```

- Load DataNormalization object given the path.
- **Java**: private static void **loadNormalizer**(String filename)
- Parameter(s):
    - filename – path where the normalizer is located.
- Return:
    - None

UIStatistic()

```
private static void UIStatistic(MultiLayerNetwork net)
```

- Binds a neural network to UIServer object for visualization.
- **Java**: private static void **UIStatistic**(MultiLayerNetwork net)
- Parameter(s):
  - net – MultiLayerNetwork object to be binded to UIServer object.
- Return:
  - None

readFile()

```
private static double[][] readFile(String filename, int[] t)
```

- Reads data file into 2d array.
- **Java**: private static double[][] **readFile**(String filename, int[] t)
- Parameter(s):
  - filename – path where the data file is located.
  - t – a 1d-array which store total number of rows and columns of data file.
- Return:
  - A 2d-array consists of the data file.

getRowCol()

```
private static int[] getRowCol(String filename)
```

- Compute the total number of rows and columns of a dataset.
- **Java**: private static int[] **getRowCol**(String filename)
- Parameter(s):
  - filename – path where the data file is located.
- Return:
  - A 1d-array with type int consists of the total number of rows and columns.

preProcessData()

```
private static DataSet preProcessData(double data[][], int sequence_length, int[] t)
```

- Transform the 2d-array data into 3d-array matching the format of DL4J recurrent neural network input and output.
- **Java**: private static DataSet **preProcessData**(double data[][], int sequence_length, int[] t)
- Parameter(s):
    - data – 2d-array of data.
    - sequence_length – input sequence length.
    - t – a 1d-array which store total number of rows and columns of data file.
- Return:
    - A DataSet object.

3.5 Android Application

3.5.1 Architecture of Application

In the android application, same sensor fusion technique as the data collection application, Watcher had been implemented. The flow of the application is shown below.

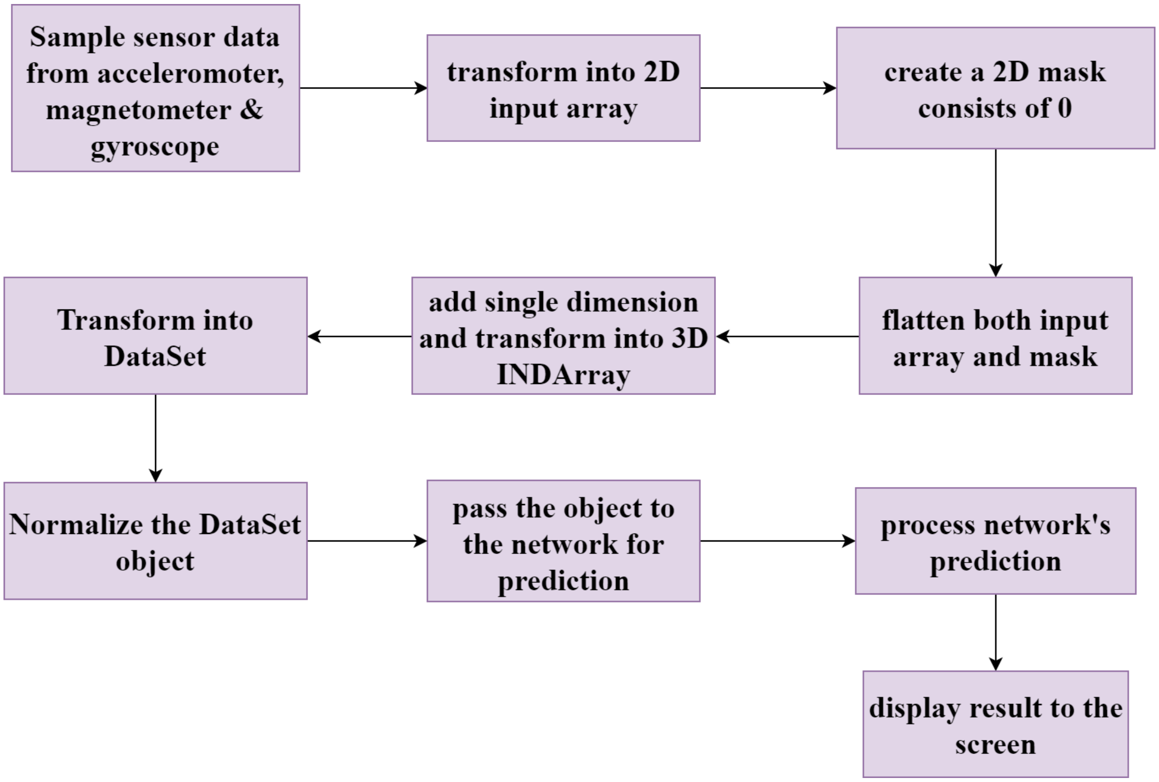

Sample sensor data from accleromoter, magnetometer & gyroscope
transform into 2D input array
create a 2D mask consists of 0
flatten both input array and mask
add single dimension and transform into 3D INDArray
Transform into DataSet
Normalize the DataSet object
pass the object to the network for prediction
process network's prediction
display result to the screen

Figure 3.6: Workflow of the Application

The sensor data will be generated by the smartphone accelerometer, magnetometer and gyroscope. These data will be sampled and transform into a 2D array in the form of [#input, #sequence_length]. The input data will consists of 6 inputs and have the structure just like the data set for the training and testing process of neural network. Unlike training, the output array is replaced by a mask. The mask consists rows of zeros is created in order to fit the dimension and its shape matches the output shape in the training data set. The mask is created to pad the DataSet object because in DL4J library, the saved normalizer doesn’t allow DataSet object with shape other than the one in training set to be normalized. Next, both the input array and the mask are flattened and added a single dimension to form 3D array in the form of [1, #input/#output, #sequence_length]. Then, they were converted into

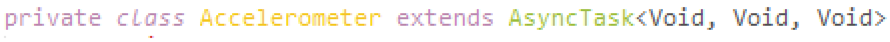
INDArray and transformed into a DataSet object. The object will later get normalized before throwing it into the neural network. After normalization process, the object is inputted to the network to do prediction. The network would return the result with a shape of [1, #output, #sequence_length]. The result contains probability of 6 classes for each row of sequence. The result will go through processing such as majority voting before displayed on the screen. For the right front pocket and left front pocket classes, these two classes had been combined and outputted as a single label because it doesn't seems significant to differentiate which pocket the smartphone is in. Thus, left front pocket and right front pocket are both classified as pocket.

Majority Vote Algorithm

This algorithm basically means finding the majority of a group of elements. In this application, a set of data is predicted by the network for n times. For each time, the neural network gives the probability of 6 classes for each row of sequence. Rows of probability for each class will be sum up and the class with the largest score will be interpreted as the predicted label. Since each prediction produces a result, we will be having *n* predicted labels if the data is predicted by the network for *n* times. Here is where the voting algorithm takes place. The algorithm will find the element with the highest occurrence and display it as the final predicted label. With this, the accuracy of the application can be improved.

3.5.2 Functions and Methods

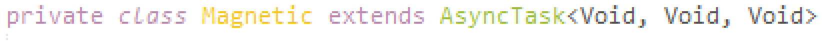
Private class Accelerometer



```
private class Accelerometer extends AsyncTask<Void, Void, Void>
```

- A class which is used to initiate smartphone's accelerometer process and extends AsyncTask class so that these work will be done in the background.

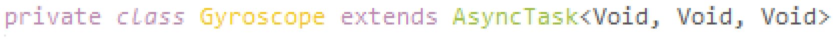
Private class Magnetic



```
private class Magnetic extends AsyncTask<Void, Void, Void>
```

- ➢ A class which is used to initiate smartphone’s magnetometer process and extends AsyncTask class so that these work will be done in the background.

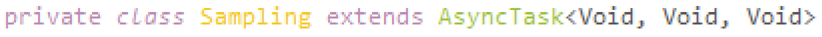
Private class Gyroscope



```
private class Gyroscope extends AsyncTask<Void, Void, Void>
```

- ➢ A class which is used to initiate smartphone’s gyroscope process and extends AsyncTask class so that these work will be done in the background.

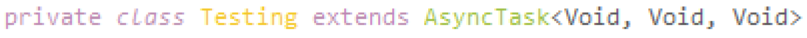
Private class Sampling



```
private class Sampling extends AsyncTask<Void, Void, Void>
```

- ➢ A class which is used to do sampling and preprocess the sensor data. It will also interpret the prediction produced by the network and displayed it on the screen. It extends the AsyncTask class so that these work will be done in the background.

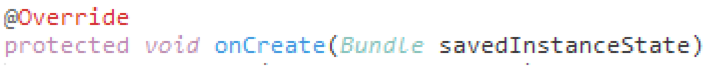
Private class Testing



```
private class Testing extends AsyncTask<Void, Void, Void>
```

- ➢ A class which is used to do testing and evaluate the performance of the application.

onCreate()



```
@Override
protected void onCreate(Bundle savedInstanceState)
```

- ➢ Android Activity’s function. It is called when the activity is first created.
- ➢ **Java**: protected void **onCreate**(Bundle savedInstanceState)
- ➢ Parameter(s):
  - ❖ savedInstanceState – application state.
- ➢ Return:
  - ❖ None.

onClick()

```
@Override
public void onClick(View view)
```

- Override function from OnClickListener interface. It is used to detect click on button event.
- **Java**: public void **onClick**(View view)
- Parameter(s):
    - view – basic building block for user interface components.
- Return:
    - None.

preProcess()

```
private void preProcess()
```

- Do preprocessing on the sensor data before inputted into the neural network.
- **Java**: private void **preProcess**()
- Parameter(s):
    - None
- Return:
    - None.

predictOutput()

```
private void predictOutput()
```

- Do prediction and process the prediction result before displayed the final result on the screen.
- **Java**: private void **predictOutput**()
- Parameter(s):
    - None
- Return:
    - None.

registerAccelerometer()

```
private void registerAccelerometer()
```

- Register accelerometer listener on the sensorManager object.

- **Java**: private void **registerAccelerometer**()
- Parameter(s):
    - None
- Return:
    - None.

registerMagnetometer()

```
private void registerMagnetometer()
```

- Register magnetometer listener on the sensorManager object.
- **Java**: private void **registerMagnetometer**()
- Parameter(s):
    - None
- Return:
    - None.

registerGyroscope()

```
private void registerGyroscope()
```

- Register gyroscope listener on the sensorManager object.
- **Java**: private void **registerGyroscope**()
- Parameter(s):
    - None
- Return:
    - None.

calculateAccMagOrientation()

```
private void calculateAccMagOrientation()
```

- Calculate the absolute orientation of the smartphone from accelerometer and magnetometer.
- **Java**: private void **calculateAccMagOrientation**()
- Parameter(s):
    - None
- Return:

- None.

getRotationVectorFromGyro()

```
private void getRotationVectorFromGyro(float[] gyroValues, float[] deltaRotationVector, float timeFactor)
```

- Creates a rotation vector from the gyroscope data.
- **Java**: private void **getRotationVectorFromGyro**(float[] gyroValues, float[] deltaRotationVector, float timeFactor)
- Parameter(s):
  - gyroValues – gyroscope based rotation matrix.
  - deltaRotationVector – empty vector to store the processed gyroscope based rotation matrix.
  - timeFactor – time interval.
- Return:
  - None.

gyroFunction()

```
private void gyroFunction(SensorEvent event)
```

- Do additional processing on the gyroscope data.
- **Java**: private void **gyroFunction**(SensorEvent event)
- Parameter(s):
  - event – holds details related to sensor (sensor's data, accuracy, timestamp, sensor's type).
- Return:
  - None.

getRotationMatrixFromOrientation()

```
private float[] getRotationMatrixFromOrientation(float[] o)
```

- Convert orientation angles into rotation matrix.
- **Java**: private float[] **getRotationMatrixFromOrientation**(float[] o)
- Parameter(s):
  - o – orientation angles vector to be converted into rotation matrix.
- Return:

- Rotation matrix.

matrixMultiplication()

```
private float[] matrixMultiplication(float[] A, float[] B)
```

- Do multiplication of two matrices.
- **Java**: private float[] **matrixMultiplication**(float[] A, float[] B)
- Parameter(s):
  - A – matrix 1.
  - B – matrix 2.
- Return:
  - Result of two matrices multiplication.

3.6 General Workflow

For this project, we want to create an Android based application which is able to do hand posture and phone placement recognition with deep learning technology. As mentioned in the introduction, it seems that there is no researchers who had utilized deep learning in this particular field. A recurrent neural network will be built from scratch, which includes the process of tuning, training, optimizing and testing.

The project is separated into few stages. Firstly, a recurrent neural network is built and undergoes the tuning process. Next, the network is trained, optimized and tested. Lastly, the network is transformed to Android application and deploy in a dynamic environment. Evaluation will be made accordingly to test the final product and the result will be discussed.

3.7 Technology Involved

3.7.1 Hardware

For hardware selection, a laptop will be used. The laptop specification are as below:

- Model: ASUS X550D
- Operating System: Windows 10 Professional
- Processor: AMD A-10-5750M 2.50 GHZ Quad Core
- Graphic Card: AMD Radeon 8650G + HD 8670M Dual Graphic
- Memory: 8GB RAM

No graphical processing unit (GPU) will be used due to lack of computing resources. The whole process regarding the network will take place using the CPU.

3.7.2 Software

Watcher

An Android application developed from the work of Jacqueling Lee Fang An and is used to collect sensors data.

Android Studio 3.0.1

An IDE which provides tools related to Android platform for building Android based application on different type of Android devices.

Java

The java programming language is used to develop the neural network and also the Android application.

IntelliJ IDEA Community Edition 2017.1.4 x64

An IDE for java programming.

DL4J

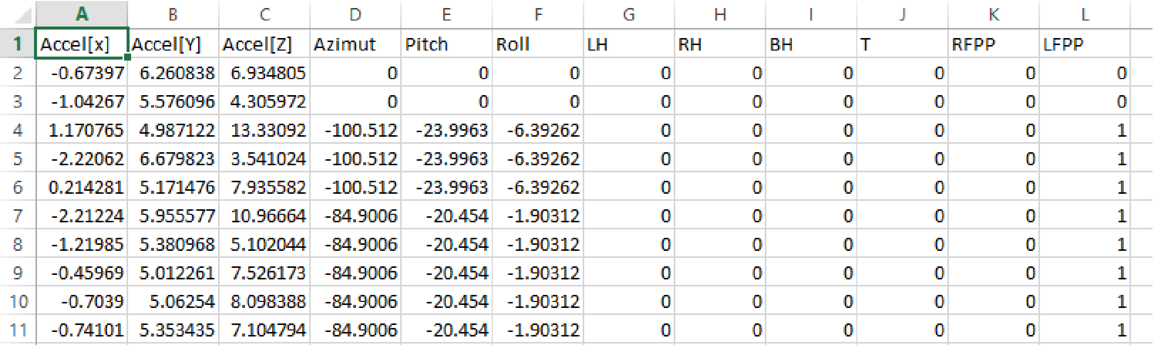
It is an open source deep learning library for java virtual machine (JVM).In this project, it is used in coding related to deep learning.

ND4J

It is an open source scientific computing library for the java virtual machine (JVM). It is meant to be used in production environment. In this project, it is utilized in handling data.

3.8 Data Sets

In order to train the neural network built, a lot of sensors data are collected from the smartphone accelerometer, magnetometer and gyroscope. These data collected will be saved in a comma-separated values (.csv) format. Total of 20 datasets were collected to train and validate the network. Basically, the dataset consists of 6 columns of inputs and 6 columns of labels for the action. Three of the input are generated by accelerometer and magnetometer while the other 3 are generated by gyroscope. For the labels, they are mentioned previously in the section 3.1. Example of the data set is shown below.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Accel[x]	Accel[Y]	Accel[Z]	Azimut	Pitch	Roll	LH	RH	BH	T	RFPP	LFPP
2	-0.67397	6.260838	6.934805	0	0	0	0	0	0	0	0	0
3	-1.04267	5.576096	4.305972	0	0	0	0	0	0	0	0	0
4	1.170765	4.987122	13.33092	-100.512	-23.9963	-6.39262	0	0	0	0	0	1
5	-2.22062	6.679823	3.541024	-100.512	-23.9963	-6.39262	0	0	0	0	0	1
6	0.214281	5.171476	7.935582	-100.512	-23.9963	-6.39262	0	0	0	0	0	1
7	-2.21224	5.955577	10.96664	-84.9006	-20.454	-1.90312	0	0	0	0	0	1
8	-1.21985	5.380968	5.102044	-84.9006	-20.454	-1.90312	0	0	0	0	0	1
9	-0.45969	5.012261	7.526173	-84.9006	-20.454	-1.90312	0	0	0	0	0	1
10	-0.7039	5.06254	8.098388	-84.9006	-20.454	-1.90312	0	0	0	0	0	1
11	-0.74101	5.353435	7.104794	-84.9006	-20.454	-1.90312	0	0	0	0	0	1

Figure 3.7: Data Set

3.9 Implementation issue and Challenge

In this project, several difficulties and challenges had been faced. Firstly, the lack of computing resources. Without GPU, tuning and training of a neural network can eat up much of the time. For instance, training the neural network with a single data set takes around 3 hours to complete in CPU. Besides, quality and quantity of data sets are a problem too. Even though we have the application to collect data from smartphone users, not everyone willing to sacrifice their precious time to do the data collection.

Chapter 4: Implementation and Result

4.1 Data Collection

In this project, an Android application called 'Watcher' had been used to collect the dataset. Total of 20 datasets were collected from 10 participants (10 males) who are volunteered to sacrifice their 10 minutes to complete two session (each session required 5 minutes to complete). A session basically outputted 1 set of data. Before starting the collecting process, users are required to enter a unique file name so that later the data file can be identified. Next, they will be redirect to another simple interface which consists of two buttons. One is for starting the data collection process while the other one to stop it halfway. Once the user presses the start button, the application will prompt random action for the user to complete which includes left hand, right hand, both hand, put on table, put in left front pocket and put in right front pocket. For each action, users are given 7 seconds to switch and maintain the posture. Users are told to be relax while performing different actions. Sensor data collected will then be recorded in the file specified. These data will later be used to train the neural network. Below are some interface of 'Watcher':

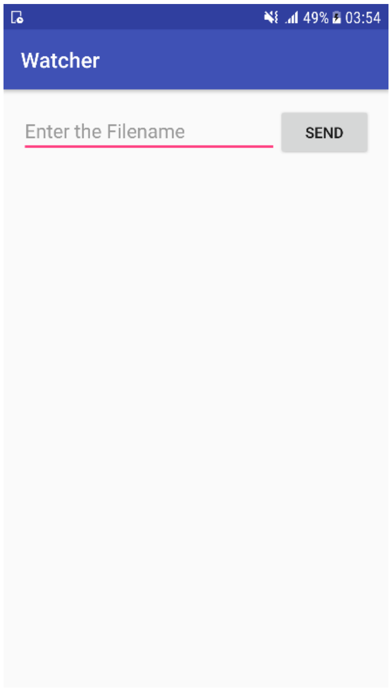


Figure 4.1: Watcher Interface

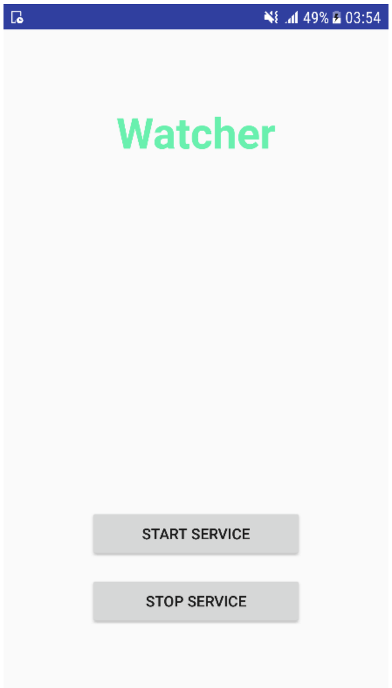


Figure 4.2: Watcher Interface (1)

4.2 Evaluation of Neural network

Total of 20 datasets collected from the data collection stage will be used to train and validate the neural network built. In training the neural network, several graphs had been observed to monitor any unusual behavior of the network. If there is any strange phenomena found, the network is adjusted right away. For each dataset, the network is trained for 1024 epochs. Each time in training, the best model will be searched and saved. In DL4J library, it provides a training UI so that the network training process can be well monitored. The figures shown in this section will only include graphs of training a particular dataset.

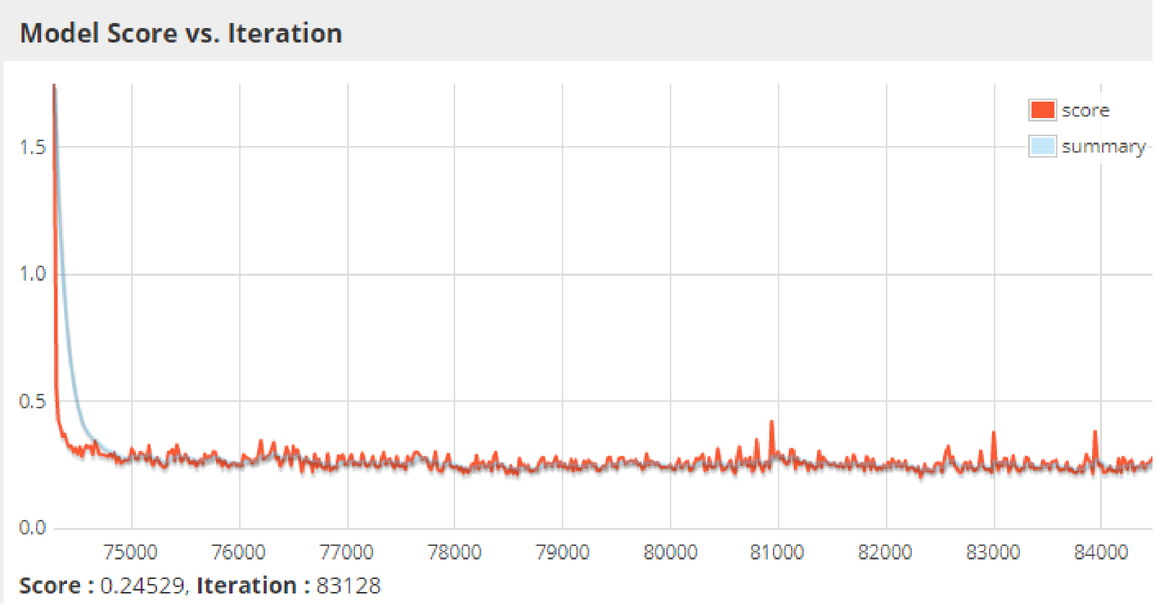


Figure 4.3: Model Score vs. Iteration Graph

In Figure 4.3, the score over time had been plotted. We can observed that the line goes up and down within a small range. This is caused by truncated backpropagation through time. As we initialized our input sequence length as 40 and truncated backpropagation through time length as 4, the graph will show a zig-zag every 10 iterations. If the stutter is large, it can be a problem. For example, it can be issue related to learning rate, normalization, mini batch and so on.

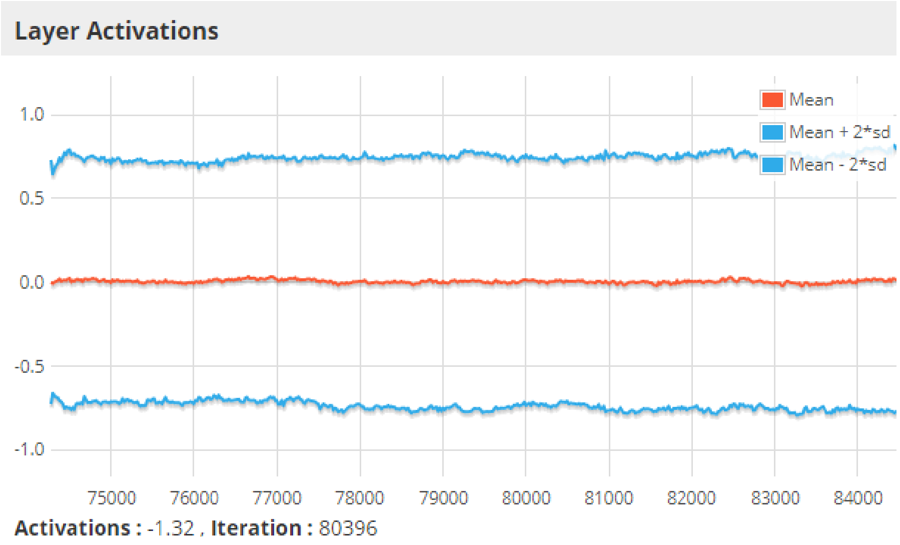


Figure 4.4: Layer Activations Graph

With layer activation graph, we are able to recognize gradient explodes or gradient vanishing problems. From Figure 4.4, we can see that the chart is stable over time which means the architecture of the model is good enough to suit the case and no problem mentioned previously had occurred. If the chart stutter too much, either gradient vanishing or gradient explodes had happened.

In training, dataset is split into train set and validation set in the ratio of 70:30. For each epoch, validation set is used to identify the performance of the network. The model which produces the highest validation accuracy is saved for further use. Besides, we had also calculated the accuracy for each class with the validation set during training. The accuracy of each classes is calculated with the following formula:

$* \ Accuracy = (TP + TN)/(TP + FP + FN + TN)$

- *TP: True Positive*
- *TN: True Negative*
- *FP: False Positive*
- *FN: False Negative*

The result are recorded in the tables below.

Table 4.1: Accuracy of Proposed Prediction Model for each Posture Type

Model	Left Hand (%)	Right Hand (%)	Both Hand (%)	Table (%)	Left front Pocket (%)	Right front Pocket (%)
1	98.7	98.7	99.7	99.4	99.9	99.3
2	97.8	98.4	99.5	99.6	99.3	99.6
3	98.1	99.0	99.7	99.3	99.4	99.4
4	98.6	98.4	99.2	99.4	99.4	99.1
5	98.7	98.0	99.9	99.0	100.0	99.4
6	98.8	97.9	99.4	99.3	99.4	99.1
7	97.6	98.0	99.8	98.7	99.5	99.0
8	98.1	98.5	99.1	99.2	98.7	99.6
9	98.3	98.6	99.6	98.7	99.2	99.8
10	97.5	98.0	99.1	98.6	99.6	99.0
11	98.8	98.3	99.6	98.9	99.7	99.5
12	98.5	97.9	99.6	98.4	99.9	99.3
13	98.6	98.9	99.4	98.9	99.8	100.0
14	98.4	98.7	99.4	99.4	99.1	99.1
15	98.9	98.8	99.5	98.8	99.7	99.9
16	98.8	98.5	99.8	99.6	99.6	99.1
17	98.5	98.7	99.4	99.0	99.5	99.6
18	98.0	98.7	99.2	99.0	99.5	99.5
19	98.9	98.3	99.8	99.0	99.5	99.4
20	98.5	98.2	99.6	99.2	99.7	99.6
Average	98.59 (SD=0.414)	98.425 (SD=0.333)	99.515 (SD=0.233)	99.07 (SD=0.326)	99.52 (SD=0.296)	99.415 (SD=0.289)

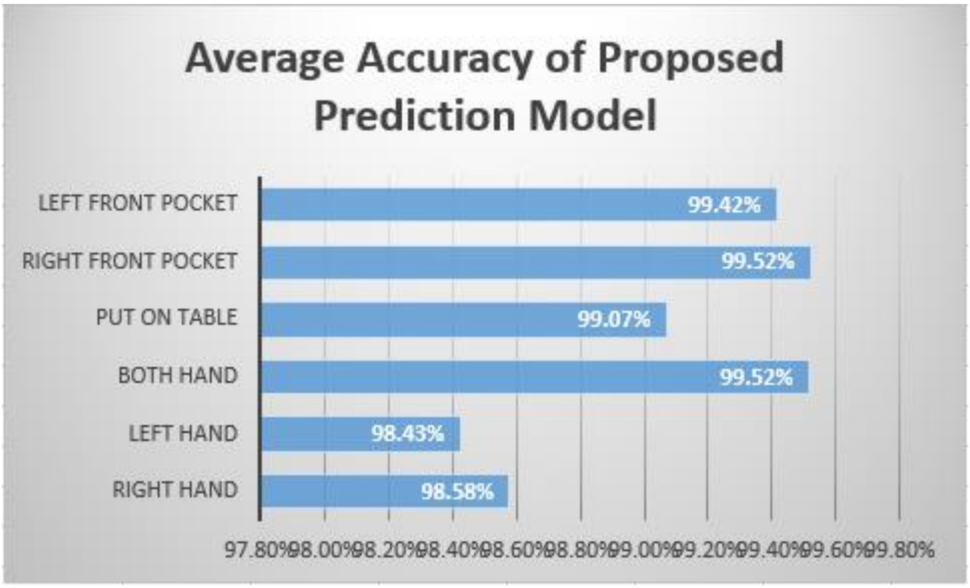


Figure 4.5: Graph of Accuracy of Proposed Prediction Model for each Posture Type

Table 4.1 had displayed the accuracies of each classes for 20 best models which we had saved during training. The result of these tables are averaged and the standard deviation is calculated. As we can observed from the result, the accuracy for each cases are quite stable over 20 times of training (20 datasets). The recall accuracies range from 98% to 100% for six different cases. From Figure 4.5, the accuracy for 'Left Hand' and 'Right Hand' are slightly lower than the rest of the cases. This is because 'Left Hand' and 'Right Hand' are complex and consists much variation. It is not easy for the network to learn a function to differentiate them with little amount of data. For the rest of cases like 'Table', they are relatively simple and the network is able to learn them well even though the amount of datasets supplied to the network wasn't that huge. Given more datasets, the performance of the network can further be improved.

4.3 Evaluation of Android Application

After the neural network was deployed to the Android platform, we would like to know how well it can perform. In a dynamic environment, the network will receives real time sensor data and do prediction based on the data it received.

4.3.1 Participants

3 random users will be selected from the same group of participants who joined in the data collection. The user will need to spend around 6 minutes to complete the whole evaluation process.

4.3.2 Experimental Setup:

Participants will be provided a smartphone with an Android application named ‘Scarecrow’ installed. The ‘Scarecrow’ application has a simple yet user friendly interface. Users will only need to press certain button to initiate the testing.

4.3.3 General Flow

Firstly, users are introduced to the new application and some explanation regarding what they are going to do next. For each users, they are required to complete a total of 6 tests which includes different hand posture and phone placement. Each test takes approximately 1 minute to complete. The ‘Scarecrow’ simple interface allows the users to initiate the process easily. Example of interface is shown below:

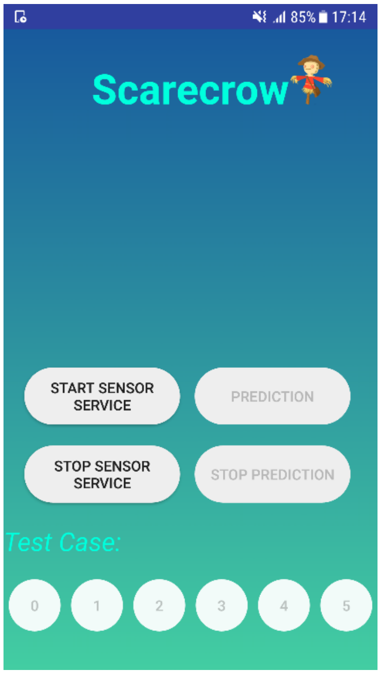


Figure 4.6: Interface of the application 'Scarecrow'

As shown in Figure 4.6, there are total of 6 buttons indicate 6 different test cases:-

- Button '0': Test case for right hand
- Button '1': Test case for left hand
- Button '2': Test case for both hand
- Button '3': Test case for put on table
- Button '4': Test case for put in pocket
- Button '5': Test case for single hand

For each test case, the user is asked to press the button and maintain a particular posture. When a button is pressed, 'Scarecrow' will display a message telling the user what action they should do and give them 3 seconds to get ready. After 3 seconds, the application starts to do prediction and calculate the accuracy. The user is required to keep a particular posture until accuracy is outputted. After 30 seconds, the application will display the accuracy calculated. The accuracy will be recorded. The process will be repeated for the rest of the test cases. For 'Single Hand' test case, the user is asked to hold the phone with their dominant hand which they used commonly. This test case is created to compare with other systems.

4.3.4 Result

The result from participants is recorded in the table below:

Table 4.2: Result of Scarecrow

Test Case	Right Hand (%)	Left Hand (%)	Both Hand (%)	Put on Table (%)	Put in Pocket (%)	Single Hand (%)
User 1	76.67	73.33	83.33	100.00	90.00	100.00
User 2	73.33	83.33	90.00	100.00	96.67	100.00
User 3	73.33	73.33	80.00	100.00	100.00	100.00
Average	74.44 (SD=1.57)	76.66 (SD=4.71)	84.44 (SD=4.16)	100.00 (SD=0.00)	95.56 (SD=4.16)	100.00 (SD=0.00)

The evaluation result performed by 3 users are averaged and a graph is plotted.

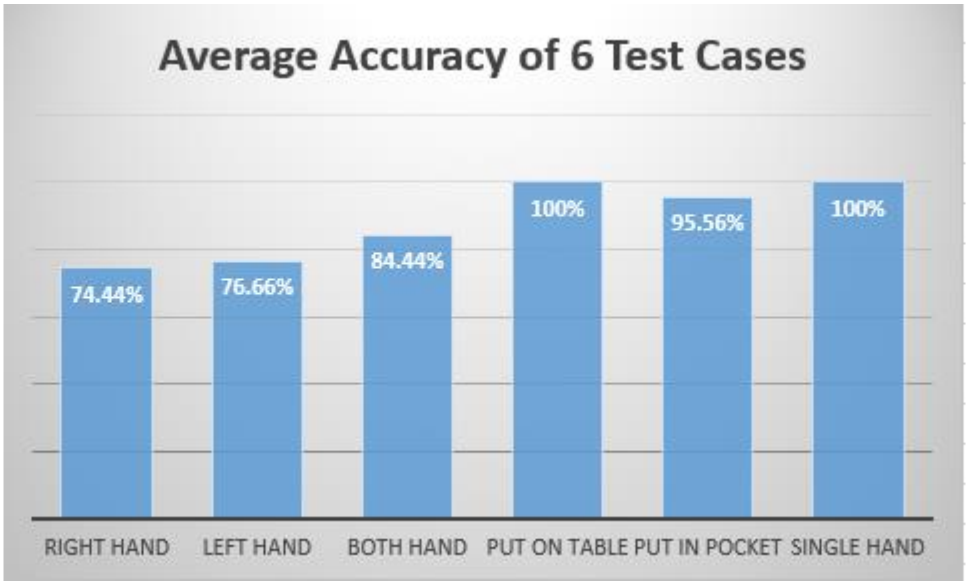


Figure 4.7: Average Accuracy of 6 Test Cases

From the graph above, we can observed that the result for the test cases, ‘Left Hand’ and ‘Right Hand’ weren’t that good compared with other test cases. The result for left hand was in the range of 73% to 83% while for the right hand the accuracy was around 73-76%. Furthermore, the network seems to predict left hand sometimes even though the smartphone is held using right hand. Vice versa, when a user holds the smartphone in right hand, the network sometimes predicts left hand. The network doesn’t seems to learn absolutely well on how to differentiate left hand and right hand.

The main reason why this happened is that the total datasets used to train the network is too little. Besides, compared to cases like table and pocket, left and right hand cases contains much variation and are more complex. In real life, different users may holds a smartphone differently when they are interacting with the smartphone. Thus, it wasn’t easy for the network to learn a particular pattern to separate these two cases given that the data supplied to the network wasn’t that much. Given more high quality datasets, the performance of the network in predicting left hand and right hand can be further be improved.

From what we had noticed from the table, the accuracy for the case ‘put on Table’ and ‘Single Hand’ are 100%. This is because these two cases weren’t really complex compared to left and right hand. The network basically learns them well and predicts perfectly in this situation. Scarecrow also produced a great result in the rest of cases such as ‘Both Hand’ and ‘Put in Pocket’. The result collected were averaged and the averaged result recorded will be used to compare with other systems that we had reviewed.

4.3.5 Comparison with other Systems

Table 4.3: Comparison of Scarecrow with other system

System	Single Hand (%)	Both Hand (%)	Put on Table (%)	Put in Pocket (%)
BeyondTouch	88.47	71.28	93.74	-
In-Pocket Detection	-	-	-	98.00
ARService	83.64	-	-	68.07
Scarecrow	100.00	82.22	100.00	92.22

In this project, we had compare the performance of Scarecrow with other final products from published papers such as BeyondTouch, In-Pocket Detection and ARService. From the table, we can observed that some system doesn't have the ability to perform multiple different types of hand posture or phone placement. When comparing 'Scarecrow' with BeyondTouch, we can noticed that 'Scarecrow' actually outperformed BeyondTouch in every cases available. For single hand and table cases, 'Scarecrow' is able to predict them 100% correctly. While for BeyondTouch, it can only score an accuracy of 88.47% for single hand and 93.74% for table case. For the both hand case, 'Scarecrow' actually performed better and overtaking BeyondTouch by 10.94%.

In the other hand, In-Pocket Detection performed great in detecting whether the smartphone is in the pocket or out of the pocket. It defeated 'Scarecrow' in the pocket case with an accuracy of 98%. However, the postures or placement it can process are far less compared to 'Scarecrow'. For ARService, the system able to score an accuracy of 83.64% in single hand case and 68.07% in pocket cases. Yet, 'Scarecrow' actually produced greater result in these cases.

Chapter 5: Conclusion

5.1 Project Review

Deep learning technology is a very powerful technology and it brings improvement in different fields such as computer vision, sentiment analysis, speech recognition, language translation and so on. Even though deep learning is popular these days, there seems to be lack of researchers who had tried to apply it in detecting smartphone users' hand posture.

In this project, we had utilized deep learning in identifying the hand posture and phone placement of users. By applying deep learning in identifying 6 different types of postures/placement, a great result had been achieved. The result produced by the end product of this project definitely meets the project objective. Besides, it did outperform other systems which practiced traditional methods in detecting hand postures and phone placement.

5.2 Future Work

Scarecrow definitely had its potential to improve. Given more and more high quality datasets, the network able to learn more and more variations of different postures and the performance of the application certainly will reach a higher level.

In this world, there is no one solution to all problems. Thus, different combinations of sensors can be tried for different kinds of hand postures and phone placement detection. In a smartphone, there are a variety kinds of built in sensors available such as proximity sensor, temperature sensor, light sensor and so on. These sensors produces sensor readings too just like the sensors we had used in this project.

Bibliography

Andrej, K. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. *Andrej Karpathy blog* [Online]. Available at: http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Accessed on 2th April 2018.

Christopher, O. (2015). Understanding LSTM Networks. *colah's blog* [Online]. Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed on 11th November 2017.

Coskun, D., Incel, O. D. & Ozgovde, A. (2015). Phone position/placement detection using accelerometer: Impact on activity recognition. *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 1-6.

DeepLearning4J. (2017). Troubleshooting Neural Net Training. *DEEPLEARNING4J* [Online]. Available at https://deeplearning4j.org/troubleshootingneuralnets. Accessed on 2th April 2018.

DeepLearning4J. (2017). Building Neural networks with DeepLearning4J. *DEEPLEARNING4J* [Online]. Available at https://deeplearning4j.org/building-neural-net-with-dl4j. Accessed on 2th April 2018.

DeepLearning4J. (2017). Introduction to deep neural network. *DEEPLEARNING4J* [Online]. Available at https://deeplearning4j.org/neuralnet-overview. Accessed on 16th August 2017.

DeepLearning4J. (2017). Tutorial: Recurrent Networks and LSTMs. *DEEPLEARNING4J* [Online]. Available at https://deeplearning4j.org/recurrentnetwork. Accessed on 6th November 2017.

Goel, M., Jansen, A., Mandel, T., Patel, S. N. & Wobbrock, J. O. (2013). ContextType: using hand posture information to improve mobile touch screen text entry. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*, pp. 2795-2798.

Goel, M., Wobbrock, J. O. & Patel, S. N. (2012). GripSense: Using built-in sensors to detect hand posture and pressure on commodity mobile phones. *Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST '12),* pp. 545-554.

Miluzzo, E., Papandrea, M., Lane, N. D., Lu, H. & Campbell A. T. (2010). Pocket, bag, hand, etc. - automatically detecting phone context through discovery. *First International Workshop on Sensing for App Phones (PhoneSense) at SenSys'10*.

Incel, O. D. (2015) Analysis of movement orientation and rotation-based sensing for phone placement recognition. *Sensors 2015,* pp. 25474-25506.

Nielsen M. A. (2015). Neural Networks and Deep Learning. *Determination Press*.

Sepp, H. & Jurgen, S. (1997) LONG SHORT-TERM MEMORY. *Neural Computation 9(8):1735-1780*.

Sun, T. & Vasarhely, M. A. (2017). Deep learning and the future of auditing: How an evolving technology could transform analysis and improve judgement. *The CPA Journal*.

Tanaka, K., Ishimaru, S., Kise, K., Kunze, K. & Inami, M. (2015). Nekoze! - monitoring and detecting head posture while working with laptop and mobile phone. *2015 9th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth),* pp. 237-240.

Yang, J., Munguia-Tapia, E. & Gibbs, S. (2013). Efficient in-pocket detection with mobile phones. *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication (UbiComp '13 Adjunct)*, pp. 31-34.

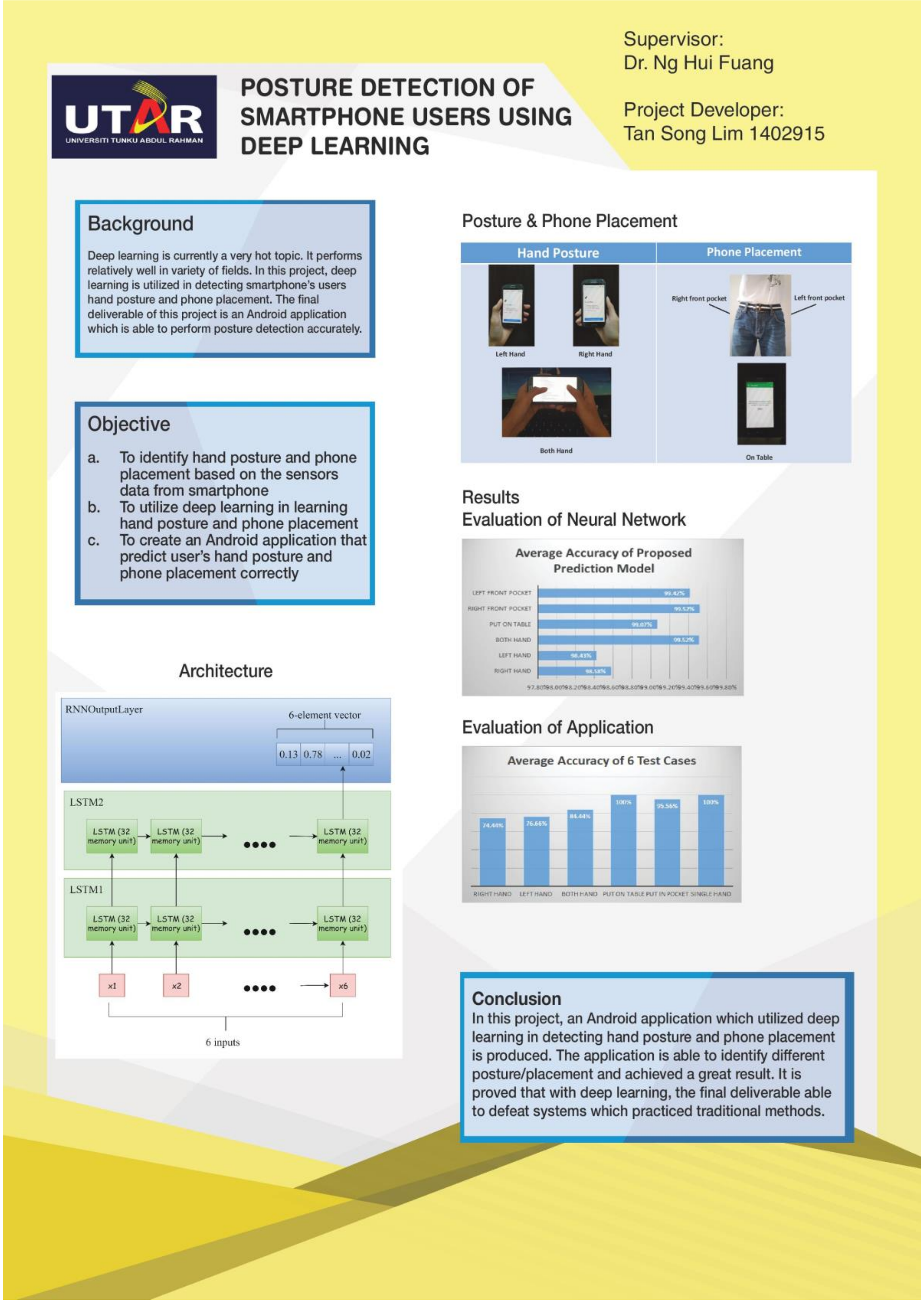
Wahl, F. & Amft, O. (2014). Personalised phone placement recognition in daily life using RFID tagging. *The First Symposium on Activity and Context Modeling and Recognition*, pp. 19-26.

Wobbrock, J. O., Myers, B. A. & Aung, H. H. (2007). The performance of hand posture in front- and back-of-device interaction for mobile computing. *Int. J. Human-Computer Studies 66,* pp. 857-875.

Xavier, G. & Yoshua, B. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010*, vol. 9

Yann, L.C., Yoshua, B. & Geoffrey, H. (2015). Deep learning. *Nature 521*, pp. 436-444

Zhang, C., Guo, A., Zhang, D., Southern, C., Arriaga, R. & Abowd, G. (2015). BeyondTouch: extending the input language with built-in sensors on commodity smartphones. *Proceedings of the 20th International Conference on Intelligent User Interfaces (IUI '15)*, pp. 67-77.



Supervisor:
Dr. Ng Hui Fuang

Project Developer:
Tan Song Lim 1402915

UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

POSTURE DETECTION OF SMARTPHONE USERS USING DEEP LEARNING

Background

Deep learning is currently a very hot topic. It performs relatively well in variety of fields. In this project, deep learning is utilized in detecting smartphone's users hand posture and phone placement. The final deliverable of this project is an Android application which is able to perform posture detection accurately.

Objective

a. To identify hand posture and phone placement based on the sensors data from smartphone
b. To utilize deep learning in learning hand posture and phone placement
c. To create an Android application that predict user's hand posture and phone placement correctly

Architecture

Posture & Phone Placement

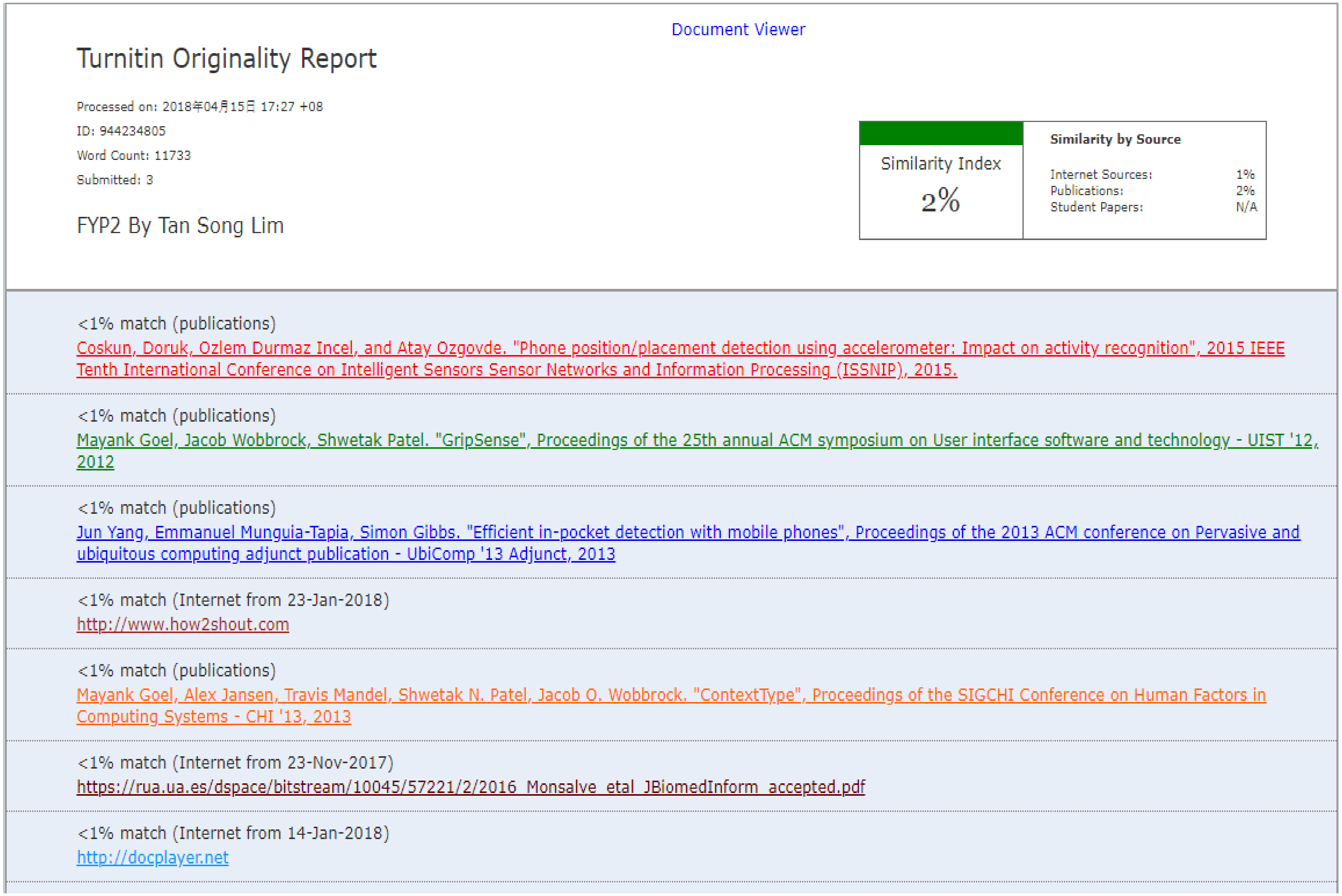
Results

Evaluation of Neural Network

Evaluation of Application

Conclusion

In this project, an Android application which utilized deep learning in detecting hand posture and phone placement is produced. The application is able to identify different posture/placement and achieved a great result. It is proved that with deep learning, the final deliverable able to defeat systems which practiced traditional methods.



Document Viewer

Turnitin Originality Report

Processed on: 2018年04月15日 17:27 +08

ID: 944234805

Word Count: 11733

Submitted: 3

FYP2 By Tan Song Lim

Similarity Index	Similarity by Source	
2%	Internet Sources:	1%
	Publications:	2%
	Student Papers:	N/A

<1% match (publications)
Coskun, Doruk, Ozlem Durmaz Incel, and Atay Ozgovde. "Phone position/placement detection using accelerometer: Impact on activity recognition", 2015 IEEE Tenth International Conference on Intelligent Sensors Sensor Networks and Information Processing (ISSNIP), 2015.

<1% match (publications)
Mayank Goel, Jacob Wobbrock, Shwetak Patel. "GripSense", Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12, 2012

<1% match (publications)
Jun Yang, Emmanuel Munguia-Tapia, Simon Gibbs. "Efficient in-pocket detection with mobile phones", Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication - UbiComp '13 Adjunct, 2013

<1% match (Internet from 23-Jan-2018)
http://www.how2shout.com

<1% match (publications)
Mayank Goel, Alex Jansen, Travis Mandel, Shwetak N. Patel, Jacob O. Wobbrock. "ContextType", Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13, 2013

<1% match (Internet from 23-Nov-2017)
https://rua.ua.es/dspace/bitstream/10045/57221/2/2016_Monsalve_etal_JBiomedInform_accepted.pdf

<1% match (Internet from 14-Jan-2018)
http://docplayer.net

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	
ID Number(s)	
Programme / Course	
Title of Final Year Project	

Similarity	**Supervisor's Comments** **(Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index:_________ % **Similarity by source** Internet Sources: _______________% Publications: _______________ % Student Papers: _______________ %	
Number of individual sources listed of more than 3% similarity: _________	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

____
Signature of Supervisor

Name: ____________________________

Date: _____________________________

____
Signature of Co-Supervisor

Name: ____________________________

Date: ___________________________ _

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (PERAK CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	
Student Name	
Supervisor Name	

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Cover
	Signed Report Status Declaration Form
	Title Page
	Signed form of the Declaration of Originality
	Acknowledgement
	Abstract
	Table of Contents
	List of Figures (if applicable)
	List of Tables (if applicable)
	List of Symbols (if applicable)
	List of Abbreviations (if applicable)
	Chapters / Content
	Bibliography (or References)
	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
	Poster
	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report. ________________________ (Signature of Student) Date:	Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction. ________________________ (Signature of Supervisor) Date: