I Know What You Type on Your Phone: Keystroke Inference on Android Device Using Deep Learning

Lei Wang

B.S. Computer Science, The University of Kansas, 2017

Submitted to the graduate degree program in Department of Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Dr. Bo Luo, Chair

Committee members

Dr. Fengjun Li

Dr. Guanghui Wang

Date defended: May 28, 2019

The Thesis Committee for Lei Wang certifies
that this is the approved version of the following thesis :

I Know What You Type on Your Phone: Keystroke Inference on Android Device Using Deep Learning

Dr. Bo Luo, Chair

Date approved: May 28, 2019

Abstract

Given a list of smartphone sensor readings, such as accelerometer, gyroscope and light sensor, is there enough information present to predict a user's input without access to either the raw text or keyboard log? With the increasing usage of smartphones as personal devices to access sensitive information on-the-go has put user privacy at risk. As the technology advances rapidly, smartphones now equip multiple sensors to measure user motion, temperature and brightness to provide constant feedback to applications in order to receive accurate and current weather forecast, GPS information and so on. In the ecosystem of Android, sensor reading can be accessed without user permissions and this makes Android devices vulnerable to various side-channel attacks.

In this thesis, we first create a native Android app to collect approximately 20700 keypresses from 30 volunteers. The text used for the data collection is carefully selected based on the bigram analysis we run on over 1.3 million tweets. We then present two approaches (single key press and bigram) for feature extraction, those features are constructed using accelerometer, gyroscope and light sensor readings. A deep neural network with four hidden layers is proposed as the baseline for this work, which achieves an accuracy of 47% using categorical cross entropy as the accuracy metric. A multi-view model then is proposed in the later work and multiple views are extracted and performance of the combination of each view is compared for analysis.

Acknowledgements

I would like to thank Dr. Luo and Dr. Li for providing me valuable guidance and feedback on this work. Without them, this work won't be possible to complete.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As the technology advances, digital devices especially smart phones are getting smaller and smarter in a way that they can handle more computational heavy tasks, which are impossible decades ago. Among all of the mobile operating systems exist in our market, Android is taking the lead by being used on over 2 Billion monthly active devices [1]. According to a recent report by IDC, Android operating system has been widely adapted by 85% of smartphones all around the world [2] due to its user-friendly interface and agile application development cycle.

With such a convenient device in your pocket, users can freely have the access to their email, online banking services and social media. With a couple of taps on the soft keyboard, users are able to accomplish various tasks normally within seconds. All the tasks can be done within designated mobile applications, such as Uber, Facebook and BankofAmerica banking app, those apps also have fairly secure architectures to protect user privacy using two-factor authentication, fingerprint or even face recognition technology. With all of the secure features those apps are providing us, it's easy to let your guard down and assume all of your online data and personal information are well-protected. However, a wide range of side-channel attacks including keystroke inferences have been exploited using various combination of sensors, among all of the sensors, motion sensors have become popular for keystroke inferences in recent years due to its wide employment in mobile devices, that is every smartphone device equips motion sensors.

With such popularity of smart phone devices and various well-equipped sensors, there has been a wide range of side-channel attacks exploited and mostly are based on motion sensors. The most

commonly researched topics are human activity recognition(HAR), driver profiling and behavioral biometric authentication. Human activity recognition, which is to recognize different human activities such as standing, sitting, running and laying down, is probably the most researched field among them all.The first publication of HAR date back to the late 90s [3], which is the earliest work on HAR to our knowledge. Driver profiling is another popular research area that exploits mobile sensors. The idea of driver profiling is to identify driving behavior, detect accident, and monitor road condition [4].

Behavioral biometric authentication or user authentication is another new research area of sensor-based side channel attacks. Behavioral biometric authentication extracts some behavioral attributes such as touch dynamics, keystroke dynamics, and gait recognition to identify an unique user [5]. In the work of Sun et al. [6], a multi-view deep learning framework is used to identify user based on their biometric information.

In terms of the granularity of sensor data needed to accomplish the classification task, HAR and driver profiling require the least granularity due to their recognition on high-level physical activities, which means HAR and driver profiling can be accomplished with less fine-grained sensor data compared to other side-channel attacks. With the advancements of hardware in our smart devices, it makes behavioral biometric authentication and text inference possible, which definitely would be infeasible decades ago due to the hardware limitations such as low sampling rate, less embedded sensors, and low quality sensor.

There are other innovative side-channel attacks worth mentioning. [7] attempts to infer user's typing on a laptop keyboard using motion sensors from a smart watch. [8] uses smartphone sensor to access the side-channel signals of 3D printer and then hack the IP information. [9] aims to infer ATM pin by using ambient light sensor on wrist-wearables. With the wide employment of various sensors and its more advanced configuration, a wide range of sensor based side-channel attacks become possible in recent years. In the next section, we are going to focus on keystroke inference on mobile devices specifically.

1.2 Related Work

Recently, motion sensors within smartphones start to gather more attentions from researchers and are used to infer a wide range of user information entirely based on the way users interact with their phone screen, that includes tapping, swiping and other gestures [10]. In the motion-based inference attacks, accelerometer and gyroscope are the most favored sensors and they have been used by most of the studies. Accelerometers were first applied to detect hard disk drive failures and gyroscopes were used to correct camera hand wobbles, initially they were not designed to directly assist users. In 2010, iPhone4 was released and it was the first phone that has intergrated gyroscope into its hardware. The use of accelerometers and gyroscopes in keystroke inference has shown promising results in recent years and several pioneering works have been mentioned below.

In 2011, TouchLogger [11] is the first work to describe the new motion-based side channel attack to infer taps on modern smartphones. TouchLogger, an Android application that extracts features from accelerometers and gyroscopes to infer keystrokes on a number-only soft keyboard on a smartphone, which they claim more than 70% of the keys are successfully inferred.

In 2012, ACCessory [12] introduces a motion-based side channel attack that is solely based on accelerometer. The attack has two different modes, area mode inference and character mode inference. In the first mode, the phone screen is divided into eight parts and evaluated separately, which yields the best average individual key area accuracy of 24.5%. The character mode infers a sequence of text based on its per-key inference in the first mode, and this gives the result of 6-character passwords can be inferred in as few as 4.5 trials.

TapLogger [13], a trojan application for the Android platform, which stealthily logs the password of screen lock and the numbers entered during a phone call solely based on accelerometer and orientation sensor. The accelerometer readings are used to detect the taps on the screen and the orientation sensor readings are used to predict the locations of these taps. Considering the best candidate inferred label, the mean accuracy is approximately 0.364 for the first attack, whereas it is approximately 0.887 when the top four labels are considered. The coverage of 0.364 means the key is predicted 0.364% of the time successfully on one single key inference, and the accuracy can

be improved with more around of inferences. Eventually, the average top-1 coverage for a 4-digit pin is 0.4 for the second attack, the average top-2 coverage for a 6-digit pin is 0.6, and the average top-3 coverage for a 4-digit pin is 1.

The research of Aviv et al. [14] focuses on the identification of user PIN by extracting the accelerometer readings in order to learn the user tap-based and gesture-based input to unlock smartphones. in controlled settings, the model can correctly classify 43% of the PIN entered and 73% of the pattern on average within 5 attempts from a test set of 50 PINs and 50 patterns. On the contrary, the model can still classify 20% of the PINS and 40% of the patterns within 5 attempts in uncontrolled settings.

In 2013, Al-Haiqi et al. [15] compares the performance of accelerometer, gyroscope and magnetometer in relation to keystroke inference attacks. They design a benchmark experiment to test the performance of different combinations of sensors based on their inference accuracy. The result shows that gyroscope alone gives the best performance and accuracy improvements can be obtained by fusing accelerometer and magnetometer together.

Narain et al. [16] combines the reading from the stereo-microphone and gyroscopes to infer user taps based on the sounds and vibrations. The model operates on both standard An- droid QWERTY and number keyboards, they claim they are the first to exceed 90% accuracy requiring a single attempt. The author also claim that the stereo-microphone is a more effective side channel as compared to the gyroscope.

In 2015, Song et al. [17] proposes two novel approaches to defend against motion-based text inference attacks, they are reducing sensor data accuracy and random keyboard layout generation. Apart from the defense approaches, when evaluating the implementation on a numerical keypad, an accuracy of 50% is achieved for the top 1 inference meaning the inferred key matches the ground truth 50% of the time and more than 80% accuracy for the top 4 inferences.

Also in 2015, Nguyen [18] attempts to infer keystrokes on trace-based input such as swype keyboard. Only eight complete words are used as the trace input for both training and testing sets. An accuracy of 86.50% and 89.75% for tap input and trace input are claimed respectively.

Finally, Shen et al. [19] utilizes magnetometer and accelerometer to infer keystrokes on both numerical keypad and keyboard. The detection accuracy are 100%, 97% and 82% in hand-hold-input, table-hold-input and hand-hold-walk scenarios respectively.

Year	Work	Inference Type
2011, 2012, 2012, 2013, 2015	[11], [13], [14], [15] ,[17]	Numerical Keyboard
2012	[12]	QWERTY Keyboard
2014, 2015	[16], [19]	QWERTY and Numerical Keyboard
2015	[18]	Swype Keyboard

Table 1.1: Work classified by inference type

In recent years, a majority of the keystroke inferences focus on numerical keyboard of the smart devices and this is because inferring alphabetical keys requires higher quality of sensor data. In order to infer alphabetical keys, it needs to map tiny difference in the patterns of sensor data into discernible features which requires data with high resolution, high frequency and high signal-to-noise ratio [10].

In table 1.1, the above mentioned work is summarized and categorized by their inference type. As you can see, the earlier work mostly focus on numerical keyboard inference. On the other hand, the later work start to focus on QWERTY keyboard as there are more high quality sensors and more variety of sensors available.

In the next section, we are going to introduce several most commonly used techniques in keystroke inference attacks.

1.3 Commonly Used Text Inference Techniques

Before the rise of deep learning, a numerous of studies have depend on hand-crafted features that are extracted from the combination of sensors. The performance of the model heavily relies on the quality of the selected features, which requires expertise and knowledge of the subject.

Cai and Chen [11] extracts two features from each unique pattern that is generated by a key press, the angle between the direction of the upper lobe and the x-axis and the angel between the direction of the lower lobe and the x-axis. The probability density function for Gaussian distribution is used to calculate the probabilities that the signal corresponds to each key.

In the work of Owusu et al. [12] extracts various features, such as the root-mean-sqaure-value, the root-mean-square-error, the maximum value and so on. This work utilizes models including Random Forest(RF), multilayer perceptron artificial neural network(MLP), sequential minimal optimization trained Support Vector Machine(SVM) and C4.5 decision tree. Among all the classifiers, Random Forest yields the most optimal results.

In the research of Xu et al. [13], features are extracted by the measurement of geasture changes, and they are generated by changes of the orientation sensor readings in Roll and Pitch during the key down phase. The extracted features then are fed into a classifier to train a model, both LibSVM and K-means are used in this experiment.

In the work of Aviv et al. [14], statistical features and the inverse Discrete Fourier Transform of the accelerometer reading are extracted, then Logistic Regression and Hidden Markov Model are used as the classifiers.

In the study of Al-Haiqi et al. [15], many statistical time domain features are extracted, such as min, max, mean, median, standard deviation and skewness. A “Bagging” classifier is used and shows the best performance on average, especially bagging ensemble learning with Functional Trees base model.

In the work of Narain et al. [16], a collection of models are exploited including Decision Trees, Naive Bayes, K-Nearest Neighbor (k-NN), Hidden Markov Models, Support Vector Machines, Random Forest and Neural Networks. Among all of those algorithms, Decision Trees (DT), Naive

Bayes (NB), 1-Nearest Neighbor (NN) and 10-Nearest Neighbor (10-NN) perform better and yield higher accuracy. Neural Networks also yield high accuracy but they are excluded in the experiments because of its high resource consumption.

In the study of Nguyen [18], a range of features are extracted including the mean, median, minimum and maximum, skewness, and kurtosis values of each sensor axis. The same set of features are used for both tap and trace input, and it is to investigate the feasibility of inferring both inputs using the same set of features. Additionally, SVM, k-NN, and random forest algorithms are explored as the classification models.

In the work of Shen et al. [19], a selection of features are extracted based on the motion habits of smartphones caused by different input actions. The features consist of the maximum value of the first, second and third fluctuation, the ACCEsum value from the timestamp of start to the timestamp of end and so on. Four classfication models are chose including Random Forest(RF), Support Vector Machine(SVM), Neural Network(NN), and K-Nearest Neighbor(K-NN).

In the work of Hodges and Buckley [20], bigram is used for extracting time series features of two adjacent words and a time series bank is built for Dynamic Time Warping to process the similarity score. The classification of a key press is based on the label of time series that has the highest similarity score to the one being classified.

The table 1.2 below summarizes the included work:

Work	**Model**
[11]	probability density function for Gaussian distribution
[13]	LibSVM, K-means
[14]	Logistic Regression, Hidden Markov Model
[12]	RF, MLP, SVM, Decision Tree
[15]	bagging ensemble learning with Functional Trees base model
[16]	DT, NB, 1-NN, 10-NN
[18]	SVM, K-NN, RF
[19]	RF, SVM, NN, K-NN
[20]	Dynamic Time Warping

Table 1.2: Commonly used models

1.4 Contributions

In this thesis, we explore the applicability and reliability of light sensor and multi-motion sensor behavior for text inference on soft keyboard of Android device. The rationale behind our work is that when a user types on a soft keyboard, the subtle changes in motion of a device can be captured by accelerometer, gyroscope and light sensor, which can be used to classify unique key presses. We propose single key press and bigram as two of our approaches for feature extraction. We then develop a deep neural network as the baseline for our experiment, as well as comparing it with multi-view models with different fusion techniques, Random Forest, and Support Vector Machine. The major contributions of this thesis are below:

First, a native Android application is designed for data collection purpose. The paragraph for users to type are carefully selected based on our bigram distribution analysis, and we have 20700 key presses in total that are collected from 30 volunteers to ensure the robustness and diversity of our data set.

Second, we are trying to solve a difficult multi-class classification problem. There are 26 alphabetic letters and a single space as the output of our model for the single key press approach, and there are 729 different combinations of bigram in total as the output of our model for the bigram approach. In comparison, [11, 13, 14, 15, 17] focus specifically on inferring the keystrokes from a numeric soft keyboard that only contains numbers, which has fewer labels in comparison to our single key press approach.

Lastly, to our knowledge, we consider our work as the first to use deep learning in the research of keystroke inference on QWERTY keyboards. Most of the research on keystroke inference heavily rely on hand-crafted features and subject-related expertise is required to extract meaningful and effective features. In our experiment, we propose a novel deep learning model as the baseline of this work, then a multi-view model with different fusion techniques (early fusion, late fusion) is constructed to compare the performance.

1.5 Thesis Organization

This thesis is organized into the following chapters:

- Chapter 1: Introduction - An introduction to the problem and the existing solutions as well as presents the difference between our approach and previous studies.
- Chapter 2: Preliminaries - Elaborating important concepts used in the following chapters
- Chapter 3: Data Collection - Unfolding our experimental design and data collection processes.
- Chapter 4: Feature Extraction and Analysis - Presenting an analysis of feature extraction using different approaches.
- Chapter 5: Model Configuration - Elaborating the proposed model for multi-class classification, including both deep neural network and multi-view model.
- Chapter 6: Performance Evaluation - Presenting the experimental results and analyses.
- Chapter 7: Conclusion and Future Work - Discussing our conclusion and future work.

Chapter 2

Preliminaries

2.1 Sensors

For our research, we choose three sensors as our sources of data input, they are accelerometer, gyroscope and light sensor. Accelerometer measures the acceleration force in m/s^2 that is applied to a device including the force of gravity and gyroscope measures a device's rate of rotation in rad/s [21], both are measured on all three physical axes x, y and z, see Figure 2.1. The light sensor measures the ambient light level (illumination) in lx [21].

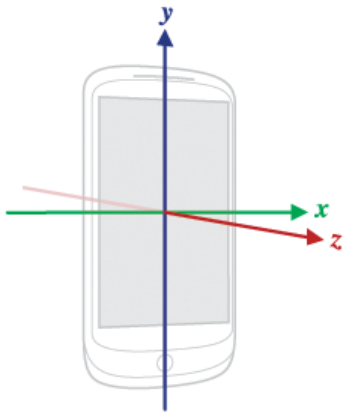


Figure 2.1: Android coordinate system

As you can see the figures below, Figure 2.2 displays the time series of light changes across a certain period of time. The time series data of the light sensor is one-dimensional. Figure 2.3 and Figure 2.4 are the sensor reading from accelerometer and gyroscope respectively, both sensors capture the movement of a device in a three-dimensional plane, meaning any movement of a device is decomposed into x, y and z axis.

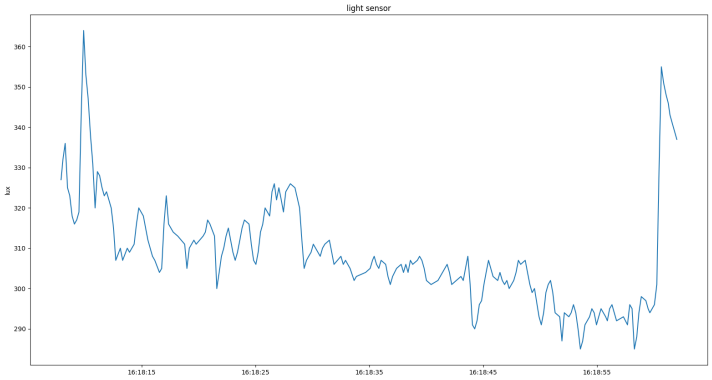


Figure 2.2: Light sensor reading

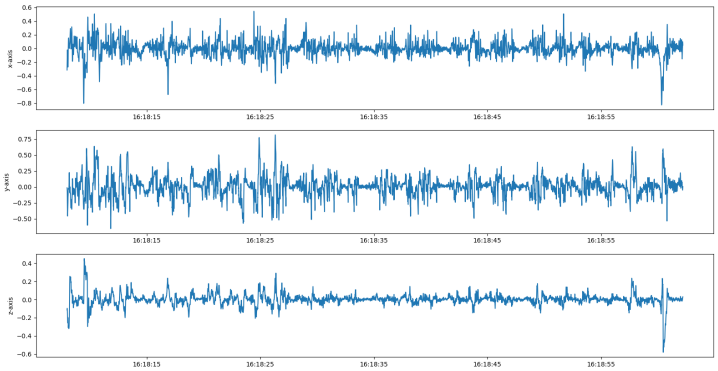


Figure 2.3: Gyroscope reading

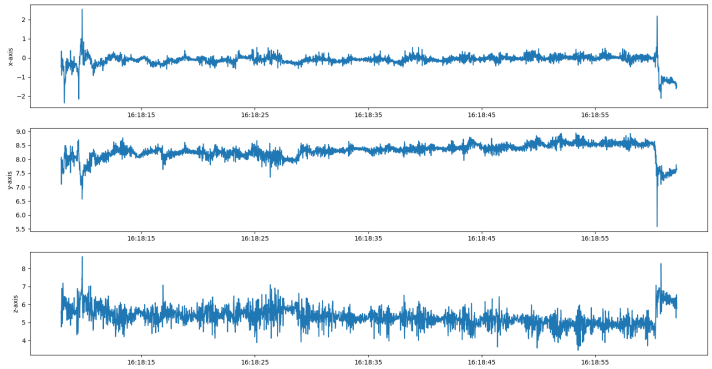


Figure 2.4: Accelerometer reading

2.2 N-grams

An N-gram is an N-character decomposition of a string, and it is widely adopted in the world of text classification and speech recognition. In our experiment, each key press on the screen of a smartphone causes it to slightly tilt, rotate or shake in a way that might not be visible to human naked eyes. However, we assume that the subtle movement of the device can be captured by the motion sensor and light sensor, such that each single key press on the screen can be classified based on its unique time series, which obtained by two of our approaches. The single key press approach is intuitive as we treat each key as an individual class for the model output, in this way, two adjacent key presses are treated as two independent events. In comparison, two adjacent key presses are assumed somehow correlated for the bigram approach, and we hope it can capture some latent properties of two key presses which can be used to improve the accuracy of our model.

> "fly me to the moon and let me play among the stars our freedom of speech is freedom or death we have got to fight the powers that be" [20]

The quote above is the paragraph we used for our experiments, which it contains 24 unique single key presses and 80 unique bigrams.

Figure 2.5 displays the single key press frequency distribution of the above paragraph, note the first bin of the graph represents the space bar , and it covers 23 English letters. This enables us to construct a more comprehensive experiment later on can be exploited to build a model that infers most of the commonly used English words.

Consider there are 729 unique combinations of bigram, including the space bar and 26 English letters, the coverage of bigram in our experiments is quite small due to the limitation of short paragraphs. Figure2.6 shows the frequency distribution of bigrams of the above paragraph.

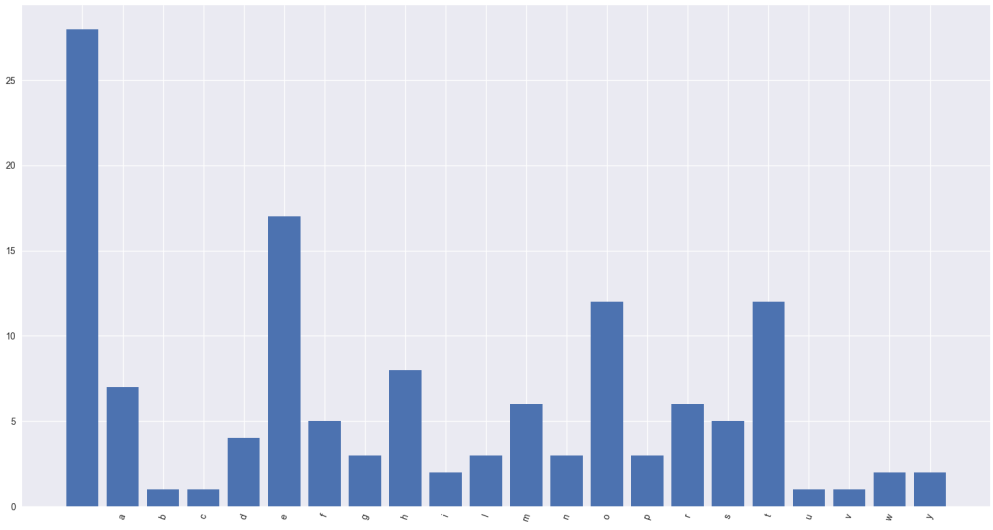


Figure 2.5: single key press frequency

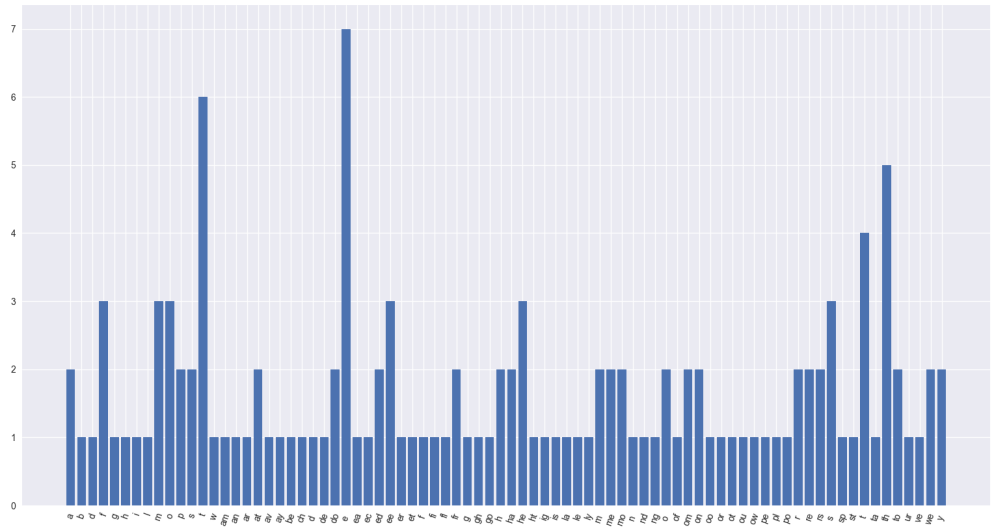


Figure 2.6: bigram frequency

2.3 Deep Learning

Deep learning is a neural network model that contains multiple hidden layers to learn representations of data. In the work of LeCun et al. [22], it has been proven to drastically improve the state-of-the-art in speech recognition, computer vision, and many other fields of machine learning. Deep learning enables the network to discover the intricate structure of large high-dimensional data, each layer learns the representation of the previous layer.

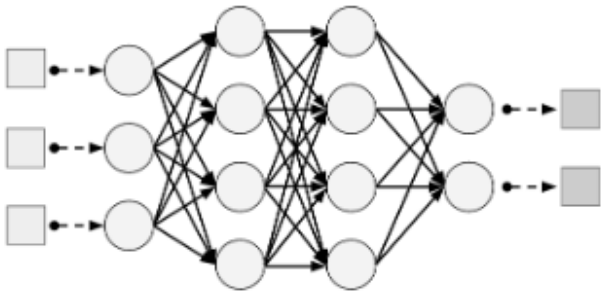


Figure 2.7: Deep Learning Architecture

2.4 Multi-View Learning

In recent years, multi-view learning has caught many researchers attention and it is a promising technique to improve the state-of-the-art. The conventional machine learning usually considers only one view and it results in a sequential architecture. In comparison, multi-view learning considers a range of views and those views should satisfy two principles [23]:

- ***Consensus Principle*** - The consensus principle aims to maximize the agreement on multiple views.
- ***Complementary Principle*** - The Complementary principle states that each view should contain some knowledge that other views do not have.

The challenge of applying multi-view learning is to find suitable views from the data.

Chapter 3

Data Collection

3.1 Introduction

In our work, motion sensor and light sensor data are collected by using our customized native Android application. There is no existing data set that utilizes bigram analysis, thus our data is created from scratch. In the following sections, we are going to walk you through the steps we took to collect our data.

3.2 Experimental Design

In this work, we divide our experiments into three phases and they are described as followings:

- Phase one: This experiment uses the original paragraph from the paper [20]. We are using this data to replicate the experiment from the paper and analyze the accuracy using different methods. This experiment will be repeated twice to average out the user error.
- Phase two: This experiment will use a tweet as its data, the tweet should be short and roughly the same length as the original paragraph (can be longer, tweet has 280 characters now), and its bigram distribution should be complementary from experiment one. This experiment will be repeated twice to average out the user error.
- Phase three: This experiment consists of 30 sets of tweets, 5 tweets in each set. For each set, the tweets will be selected based on their bigram distribution, which is a subset of the union of experiment one and experiment two.

Phase one is inspired by the work of Hodges and Buckley [20] and we design it the same way to compare our results. Additionally, the repetition of the experiment is to generate more data from an individual participant and more statistical analysis can be applied later conveniently.

Phase two is designed to cover more bigrams in our data set due to the limitations of the first phase. By expanding the bigram distribution, we hope the data from phase three can be utilized as the testing data set for our bigram approach.

The data in phase three is generated by our novel tweet analysis, which is introduced later in the chapter. Each tweet is being picked carefully from our 1.4 million tweets and they are used as the final testing set for the bigram approach.

Overall, assign 30 sets to 30 participants. Each participant receives one distinct set. Each participant will type in 5 different tweets into the device one at a time.

3.3 Android Application Design

In our work, a native Android application is designed to record our sensor data while a user is typing on the soft keyboard to mimic the environment of our side-channel attack.

The application includes all experiments, which there are 9 experiments in total, for the above three phases. The application is divided into 10 different screens, a user is asked to input a sample id, which is assigned beforehand and used to protect participants' privacy, on the first screen. The application then is followed by 9 different screens with each being an experiment. Within each experiment screen, two buttons and an input field are designed to facilitate the process of data collection for the user. Additionally, a paragraph is included above the input field in the first 4 experiments, and 5 different tweets that are obtained by our unique bigram distribution analysis, which we will discuss in the tweet analysis section, are provided on a piece of paper for the last 5 experiments. The confirm button is pressed once the user is done with the current one and ready to move to the next experiment, and the restart button restarts the current experiment when the user make a few typos, this way, the data is recorded one experiment at a time and the user is not pressured to type everything correct in the first try. Figure3.1 illustrates the arrangement of the

screens.

The application requires two permission to operate and they are WRITE_EXTERNAL_STORAGE and READ_EXTERNAL_STORAGE. The write permission is to access to the file system of the Android device in order to save the sensor readings locally. The read permission is for reading the file path, it is necessary to know the file structure of the device prior to writing the sensor data. Figure 3.2 displays the required permissions in the AndroidManifest.xml file.

Generally, there are two modes in Android devices, landscape mode and portrait mode. Landscape mode allows the users to hold their phone horizontally for the purpose of gaming, video watching and some other entertainments. However, landscape mode shifts the soft keyboard layout and makes it broader than the keyboard in portrait mode, this will greatly introduce unnecessary noise to our data set thus effect the quality of our models. In figure 3.3, the application sets the screen orientation to portrait mode only.

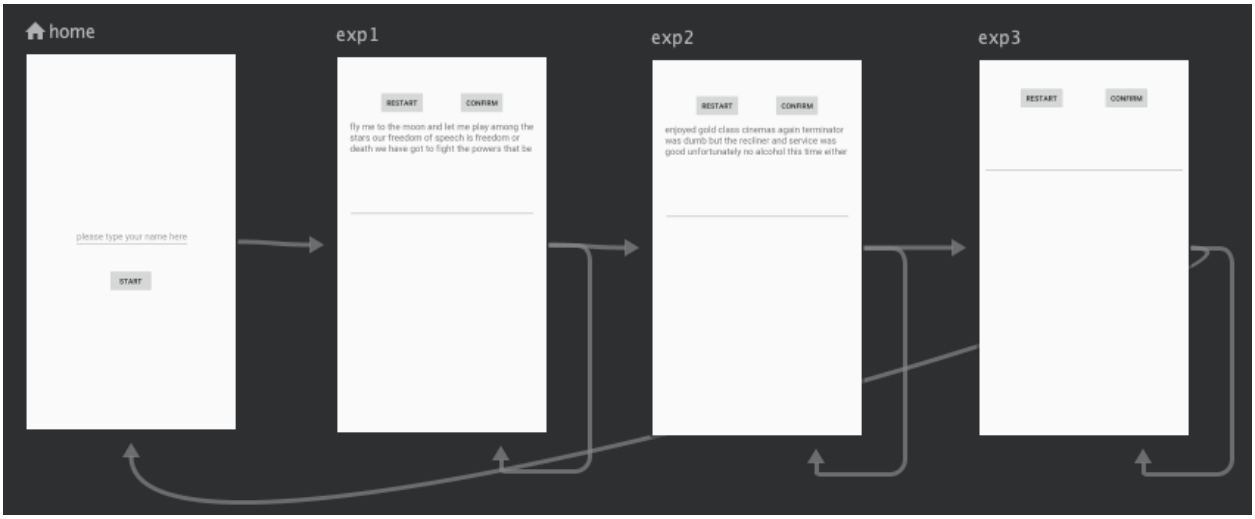


Figure 3.1: Android app design

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Figure 3.2: Android app permission

```
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/AppTheme">
<activity android:name=".MainActivity" android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>
```

Figure 3.3: Portrait mode

3.3.1 Android Sensor Overview

In the ecosystem of Android, motion sensors, environment sensors, and position sensors are the three sensor categories [24]. Table 3.1 displays all the existing sensors that are available in the Android Developers APIs.

Category	Sensor
Motion Sensors	TYPE_ACCELEROMETER TYPE_GRAVITY TYPE_GYROSCOPE
Position Sensors	TYPE_MAGNETIC_FIELD TYPE_ORIENTATION TYPE_PROXIMITY
Environment Sensors	TYPE_AMBIENT_TEMPERATURE TYPE_LIGHT TYPE_PRESSURE TYPE_RELATIVE_HUMIDITY TYPE_TEMPERATURE

Table 3.1: Android sensor overview

3.3.2 Sensor Comparison to Previous Work

In previous work, authors consider different combination of sensors to infer keystrokes in hope to improve the inference accuracy. Considering keystroke inference is still at its early stage, various work choose their sensors for the experiments based on what the previous work have done in the past. In table 3.2, accelerometer and gyroscope are the most widely used sensors. More than half of the sensors included in Android Developer APIs are never exploited and we are the first to use light sensor for text inference attacks.

In the work of Al-Haiqi et al. [15], a comparative study of effectiveness of sensors shows that gyroscope alone has distinguishing improvements in terms of inference accuracy. In comparison, by fusing accelerometer and rotation vector data together increases the inference accuracy. This study proves that gyroscope is the most effective sensor for text inferring purpose in their work.

In conclusion, it is not convincing to conclude that gyroscope is the most effective sensor for text inference attacks. It is also necessary to take the quality of feature extractions into consideration, but [15] does provide us a good insight of what sensors we should focus on.

Work	**Sensor Used**
[11], [18]	Accelerometer, Gyroscope
[12], [14]	Accelerometer
[13]	Accelerometer, Orientation sensor
[15]	Accelerometer, Gyroscope, Magnetometer
[16]	Stereo-microphone, Gyroscopes
[19]	Accelerometer, Magnetometer

Table 3.2: Sensor comparison

3.3.3 Sensor Configuration

The configuration of the sensor is crucial to our work as all the data is sensor-based. The frequency of accelerometer, gyroscope and light sensor are set to SENSOR_DELAY_FASTEST [24] in order to capture the maximum amount of data points during typing. The sensor starts recording at the beginning of each experiment and stops at the end of each experiment, in this

way, the noise can be minimized during typing. Table 3.3 demonstrates the corresponding delay in millisecond of each category.

Delay	Millisecond
SENSOR_DELAY_FASTEST	0
SENSOR_DELAY_GAME	20
SENSOR_DELAY_UI	60
SENSOR_DELAY_NORMAL	200

Table 3.3: Android Sensor Delay

3.3.4 File Architecture

The sensor reading is recorded every millisecond and is saved in a csv file, the key press and its timestamp are recorded as the ground truth and saved in a CSV (Comma-separated values) file. Thus, four csv files are generated, including accelerometer.csv, gyroscope.csv, light.csv and keys.csv, in each experiment. Overall, 36 CSV files are generated by each participant and 1080 CSV files are collected in total. All of the data files are then sent to an email account for further analysis.

3.4 Tweet Analysis

Bigram distribution is the collection of all unique bigrams of a paragraph, and it is used to measure the bigram coverage. 30 different tweets are carefully selected after we run our bigram distribution analysis on nearly 1.4 million tweets. Given the bigram distribution of phase one as Bi(p1), the second paragraph is chosen based on its distinctiveness to the first paragraph, meaning the bigram distribution of phase two as Bi(p2) should be as distinct as possible from Bi(1) and it satisfies the following:

$$\exists\ p2, \quad \ni \text{minimize}\, Bi(p2) = \frac{|Bi(p1) \cap Bi(p2)|}{|Bi(p2)|} \tag{3.1}$$

As for finding the suitable tweets for phase three. Let t be a tweet, and Bi(t) be its bigram distribution, thus t should have the following property:

$$\exists\, t, \quad \ni Bi(t) \subset (Bi(p1) \cap Bi(p2)) \tag{3.2}$$

Since there are 729 unique bigrams, with the length of our paragarph 1 and 2 each being around 130 characters, it's difficult to pick tweets that match the above criteria as well as have semantic meaning. We apply the following steps to narrow down our searches:

1. Strip numbers, special characters, and emojis from the text
2. Convert all letters to lowercase
3. Find all tweets that satisfy property (3.2)
4. Select tweets with proper semantic meaning

After we implement the above process, this boils down to about 30 different tweets and below are 5 of the tweets we selected:

- on our way to see cold play
- we landed on the moon we landed on the moon
- star wars star wars star wars
- no power for the last hours
- on our way out the hotel to go home

As you can see, all the tweet above are relatively short and this ensures the user won't have typing fatigue considering most of our users are not familiar with the Samsung Galaxy S9 soft keyboard.

3.5 Data Collection Process

To keep our data diverse, 30 different participants are recruited across the campus. Each participant is asked to sit still in their seat, and a Samsung Galaxy S9 is given to a participant. Participants are instructed to enter their assigned sample name in the initial screen and they are free to restart each experiment if they make more than 2 typos. During the experiment, only English letters and space should be entered, the experiment needs to be restarted if any other key is pressed, in this way, only letters and space are considered for the future analysis.

Participants are told to not put their phone or rest their hands on the table, otherwise no significant sensor reading can be captured. Participants are also advised to minimize their movements except moving their fingers during typing, hence, noise can be reduced and only relevant movements caused by typing are recorded. A piece of paper contains 5 short tweets is given to the participant beforehand, they are advised to enter one tweet for each experiment.

Due to most of the participants are inexperienced with the new device, they are more likely to make typos. Thus, we leave them in the room until they inform us they are done with the experiment rather than inspecting them closely during typing to reduce their pressure. The average time for each participant to complete the experiment is about 6-10 minutes depends on the familiarity of the device. Participant are asked to stay until we inspect all the sensor reading are correctly recorded, they are asked to repeat the experiment if irrelevant keys are pressed or the data is missing.

3.6 Conclusion

In this chapter, we have elaborated on the details of how we designed the experiments and the Android application, as well as the sensor comparison. The tweet analysis was also carefully revealed and the processes of data collection. In the next chapter, we are going to focus on how we extract our various features for both single key and bigram approaches.

Chapter 4

Feature Extraction and Analysis

4.1 Introduction

A total of 20700 key presses are collected from 30 volunteers, and 350 MB data is generated during our experiments. In following sections, data preprocessing and feature extraction will be explored using our single key and bigram approaches.

4.2 Preprocessing

In the previous application design section, it is mentioned that the motion sensors and light sensor are being recorded with highest frequency, which has no delay noted in [24]. For each experiment, the timestamp of the reading from each sensor is saved in corresponding CSV file, those timestamps are used for aligning sensor readings at each timestamp of key presses. While the key presses are being recorded, racing condition occurs among three sensors causing accelerometer and gyroscope to have 2ms delay between each reading. However, the light sensor behaves unexpected that its frequency is significantly lower than the motion sensors due to inevitable sensor limitations. The comparison of figure 4.1 and figure 4.2 reveals that the motion sensors and light sensor have completely different amount of sample points with their frequency being set to the highest. This can be resolved using our interpolation methods which will be explained in the following sections.

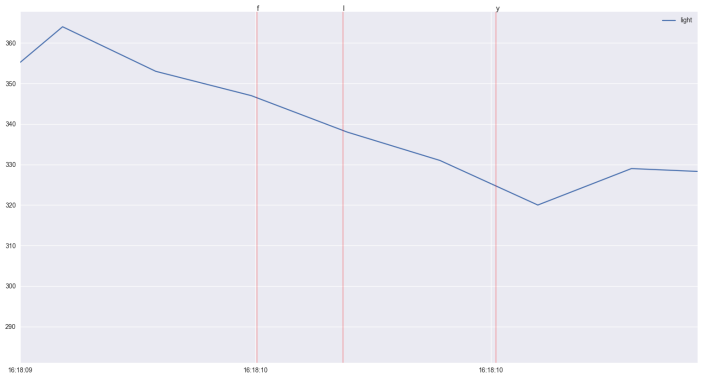


Figure 4.1: Light sensor reading with key presses

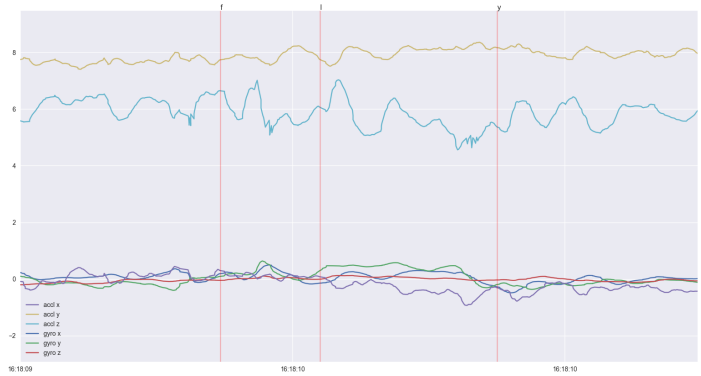


Figure 4.2: Motion sensor reading with key presses

4.2.1 Interpolation and Alignment

After we carefully examine the raw sensor data, the misalignment of sensor readings is observed due to race condition of three sensors are being recorded simultaneously and different sensor sampling rates. In order to align all the sensor readings according to its timestamp, a linear interpolation method is applied to fill the gap of two adjacent readings, which generates a new reading per millisecond.

4.3 Feature Extraction

In this section, various feature extraction parameters are explored and the procedures of extracting our features are elaborated as followings:

***Analysis of* Δt** - The action of the finger tip first touches the soft keyboard to it lifts off the screen is considered as a continuous process and is an important factor to contemplate. Though, key down and key up are failed to be recorded as a result of our application design limitations. To solve this problem, we simply utilize the timestamp of each press as the ground truth and apply a small time window noted as Δt.

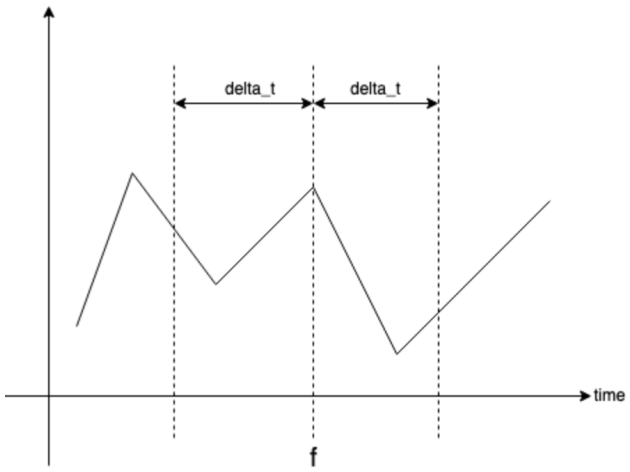


Figure 4.3: Key press f with window Δt

As figure 4.3 demonstrates that we assume the timestamp of key press f is recorded at the middle of the press action, and a Δt can be applied to both the left and right side of the key press f to include all relevant sensor readings. According to the work of Ping et al. [25], a tap is about 100ms long and 75ms is selected as the Δt. In our later work, the deep neural network is experimented with various Δt shown in table 4.1 to find the most suitable one, and we choose 75ms, 200ms and 300ms for our experiments. The performance of each Δt will be discussed in chapter 6.

index	Δt
1	75
2	200
3	300

Table 4.1: Δt parameter

Analysis of Window Sample - Due to the characteristic of Neural Network requires a feature vector with fixed size for its input layer, another parameter window sample, noted as Ω, is used to make uniform feature vectors across all key presses. The Ω for single key press and bigram are different, thus we will discuss them saperately in the following:

Let Ωs and Ωbi be the Ω for single key press and bigram respectively.

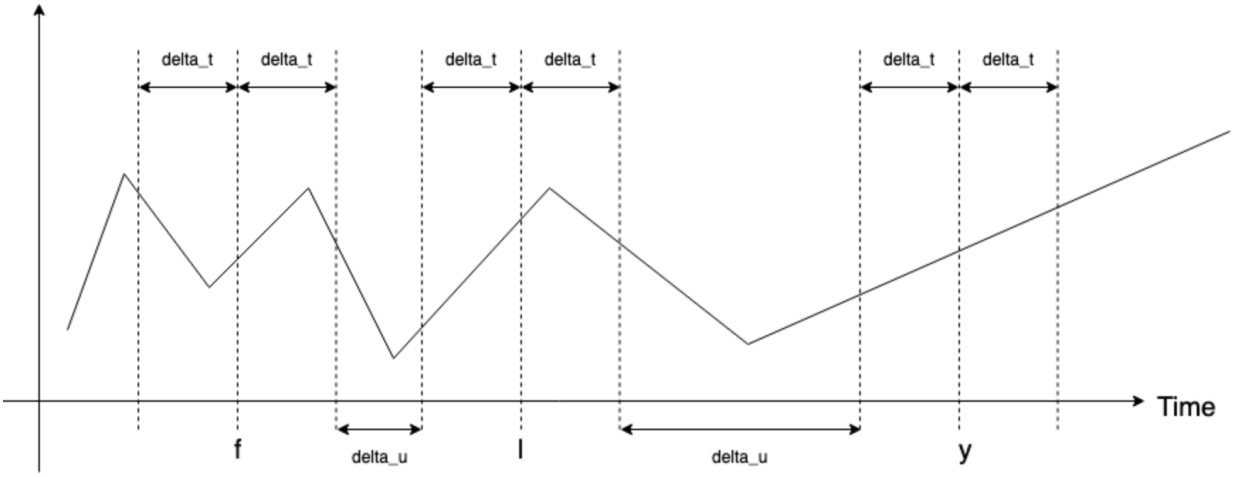


Figure 4.4: "fly" with window Δt

- ***Single Key Press*** - Since each key press is treated as an independent event and the range of key press can be retrieved using Δt for all key presses, a feature vector can be constructed by sampling Ωs many data points evenly within the range of 2*Δt

- ***Bigram*** - Figure 4.4 displays the sample time series sensor data for the word "fly". As you can see, each key press is wrapped with a uniform time window Δt to its left and right. However, due to the pause between two adjacent key presses denoted as Δu, which represents the time between the end of current key up and the start of the next key down, differs from two

adjacent presses. As the result of it, Figure 4.4 demonstrates two different Δu are produced between bigram "fl" and "ly", this results different vector length in the feature vector of the bigram. To create a uniform feature vector for both "fl" and "ly", we sample Ωbi many data points evenly starts from the left Δt of "f" to the right Δt of "l" for bigram "fl", and we also sample Ωbi many data points evenly starts from the left Δt of "l" to the right Δt of "y" for bigram "ly". In this way, we hope capturing latent characteristics of each bigram by including the Δu between adjacent presses will increase the accuracy of the model.

In table 4.2, a list of candidates for Ω is presented for both approaches. Based on the amount of data points are sampled, the length of the feature vector varies from 140 to 420 for single key press and from 420 to 980 for bigram. The performance of various Ω will be explored in chapter 6.

Approach	Ω
Single key press	20, 40, 60
Bigram	60, 100, 140

Table 4.2: Ω parameter

Chapter 5

Model Creation

5.1 Introduction

In this chapter, both the baseline model and multi-view model are introduced. The baseline model is a four hidden layer deep neural network model, it is denoted as the sequential model in the rest of the paper. The multi-view model has two varieties, they are early and late fusion. In the following sections, the analysis of various layers for sequential model is discussed in section 5.2 where we explore the effect of different numbers of neurons, batch normalization layer, dropout layer with various dropout rates and so on. Early fusion and late fusion are discussed in section 5.3 and section 5.4 respectively, and the creation of different views is also elaborated.

5.2 Sequential Model

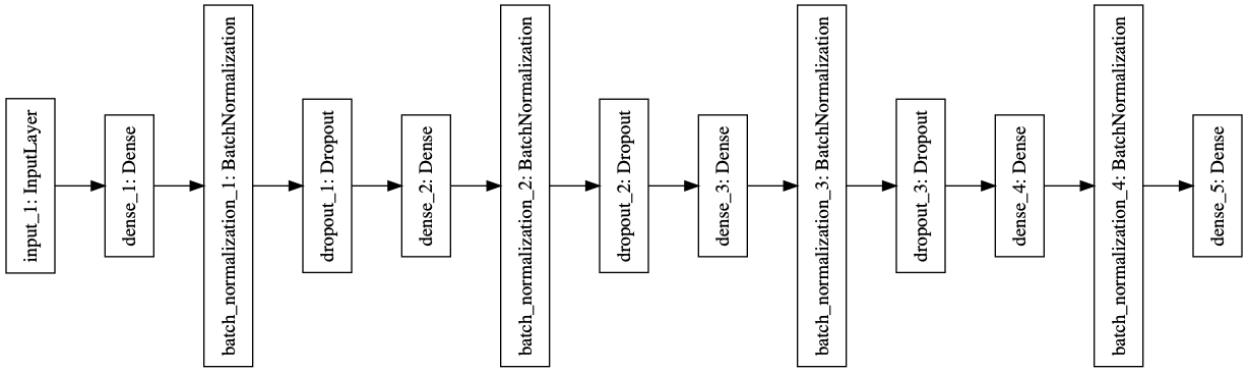


Figure 5.1: Sequential Model Architecture

The sequential model has a linear architecture, as shown in figure 5.1, 4 hidden layers are constructed. We will explore different parameters of our sequential model for single key press and bigram approach separately in the following:

Parameters of single key press approach - The first layer is the input layer, which it takes in a fixed sized feature vector as an input. In our case, a 1 x 280 vector is generated by fusing all three sensor readings and sampling 40 points from 7 different feature dimensions including gyro-x, gyro-y, gyro-z, accl-x, accl-y, accl-z and light.

Parameters of bigram approach - The first layer takes in a 1 x 700 vector, which is generated by fusing all three sensor readings and sampling 100 points from 7 different feature dimensions.

Each of the hidden layer is a dense layer, they contain 280, 480, 480 and 280 neurons respectively. A ReLU (rectifier linear unit) activation function is used for faster training speed and its sparse activation for each hidden layer. Due to it map all the negatives of x to zero as you can see from figure 5.1, this efficiently speeds up the training speed and makes neuron activation more sparse.

$$f(x) = max(0,x) \tag{5.1}$$

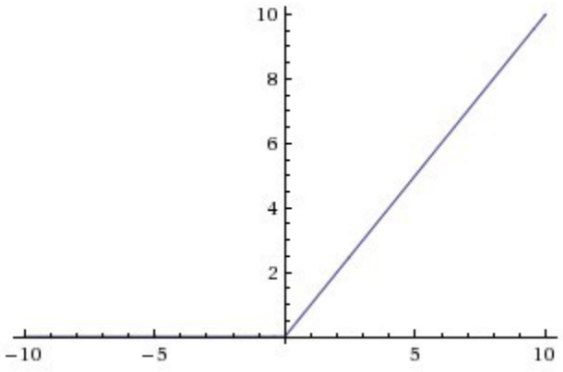


Figure 5.2: ReLU activation function

5.2.1 Batch Normalization

Batch normalization is applied to the output weights of each hidden layer to improve the training process. In the work of Ioffe and Szegedy [26], it is stated that by whitening its input, meaning applying zero means and unit variances, can reduce the internal covariate shift. By applying batch normalization to our inputs of each hidden layer, we are able to experiment with a higher learning rate and reduce overfitting.

5.2.2 Dropout

A dropout layer is applied to the output weights of each first three hidden layers to prevent the network from overfitting, the drop rates of each dropout layer are 0.5, 0.7 and 0.5 respectively. Due to the enormous amount of parameters the network generates, overfitting is an extremely common issue in every aspect of machine learning. In the work of Srivastava et al. [27], dropout is the technique they proposed that is randomly dropout a percentage of units along with their connections from the network, this technique has been widely adopted quickly and proven effective for dealing with overfitting.

5.2.3 Label Binarizer

Considering our task is a multi-class classification problem, label binarizer is used to encode multiple classes into single one hot vectors. Let's assume we have 3 classes and they are a, b and c, label binarizer first maps all the classes to a vector denoted as $\vec{V}_f$ such that $\vec{V}_f$ = [a,b,c]. Secondly, label binarizer transforms the sequence of labels into $\vec{V}_t$, such that $\vec{v}$ = [1,0,0] for a, $\vec{v}$ = [0,1,0] for b and $\vec{v}$ = [0,0,1] for c, $\forall\vec{v} \in \vec{V}_t$. In this way, every possible class can be converted to a uniform vector contains only 0 and 1 for the neural network to process.

5.2.4 Output Layer

The output layer is the last layer of the neural network that it produces an array of probabilities and each probability corresponds to how likely the model predicts the class. In our work, our model has a total of 25 different classes containing 23 letters, a space and 9, which is an unexpected human error.

Due to the characteristics of multi-class classification, a softmax activation function is used to map the input logits to output class probabilites. As equation 5.2 shows, $\vec{x}$ is the input of the layer and it is the weight distribution of the previous layer. The softmax function produces a list of probabilities, each probability fails in the interval between 0 and 1 and all probabilities sum up to 1. This makes perfect sense to use softmax as our activation function because the prediction will be one of the 25 classes and the probabilities will be ranged from 0 to 1.

$$softmax(\vec{x})_i = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)}, x_i, x_j \in \vec{x} \tag{5.2}$$

5.2.5 Parameter Detail

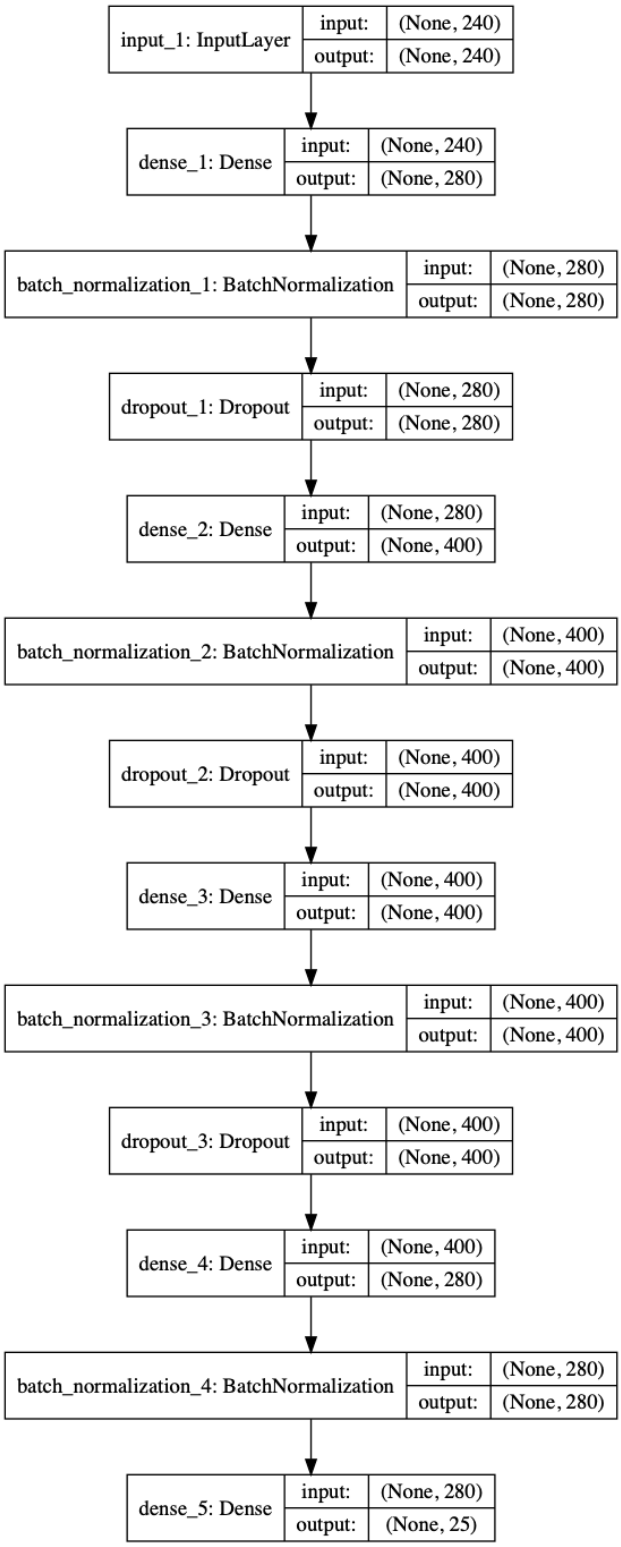


Figure 5.3: Sequential Model Architecture with Parameters

5.3 Early Fusion

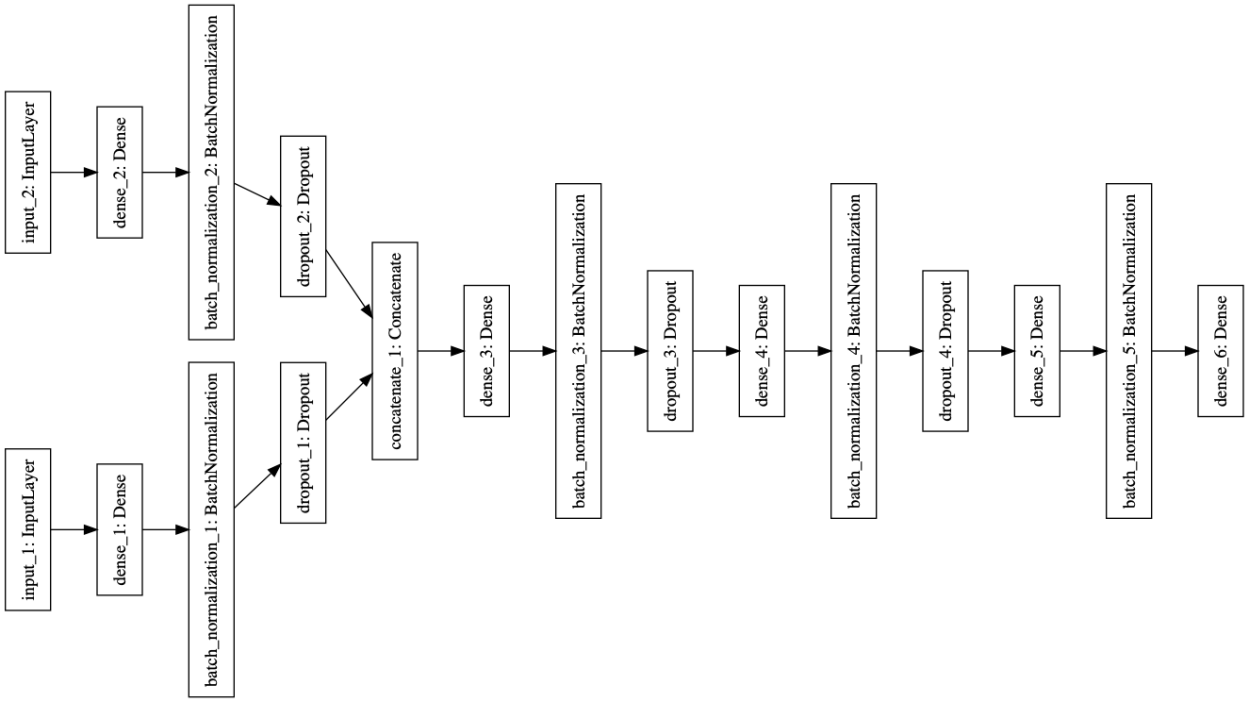


Figure 5.4: Early Fusion Model Architecture

In recent years, multi-view learning has achieved great successes in the field of machine learning as stated in the work of Xu et al. [23]. Comparing to the conventional architecture of neural network, multi-view learning considers a range of diverse views, and these views can be obtained from different sources or feature subsets. For example, in the field of computer vision, multiple views can be constructed from the pixel of an image, the histogram distribution of an image and so on. In our work, we consider each sensor as an individual view, and then we also consider each axis of a sensor as a view for the performance comparison.

Early fusion is a type of multi-view learning technique, the idea is that instead of pooling all the features together to train one single model, each view is trained separately with a model and the learned representations of each view are combined into a single layer. In figure 5.4, two views are generated and trained separately with a single dense layer network, then the representations of each view are concatenated into one single layer.

5.3.1 Parameter Detail

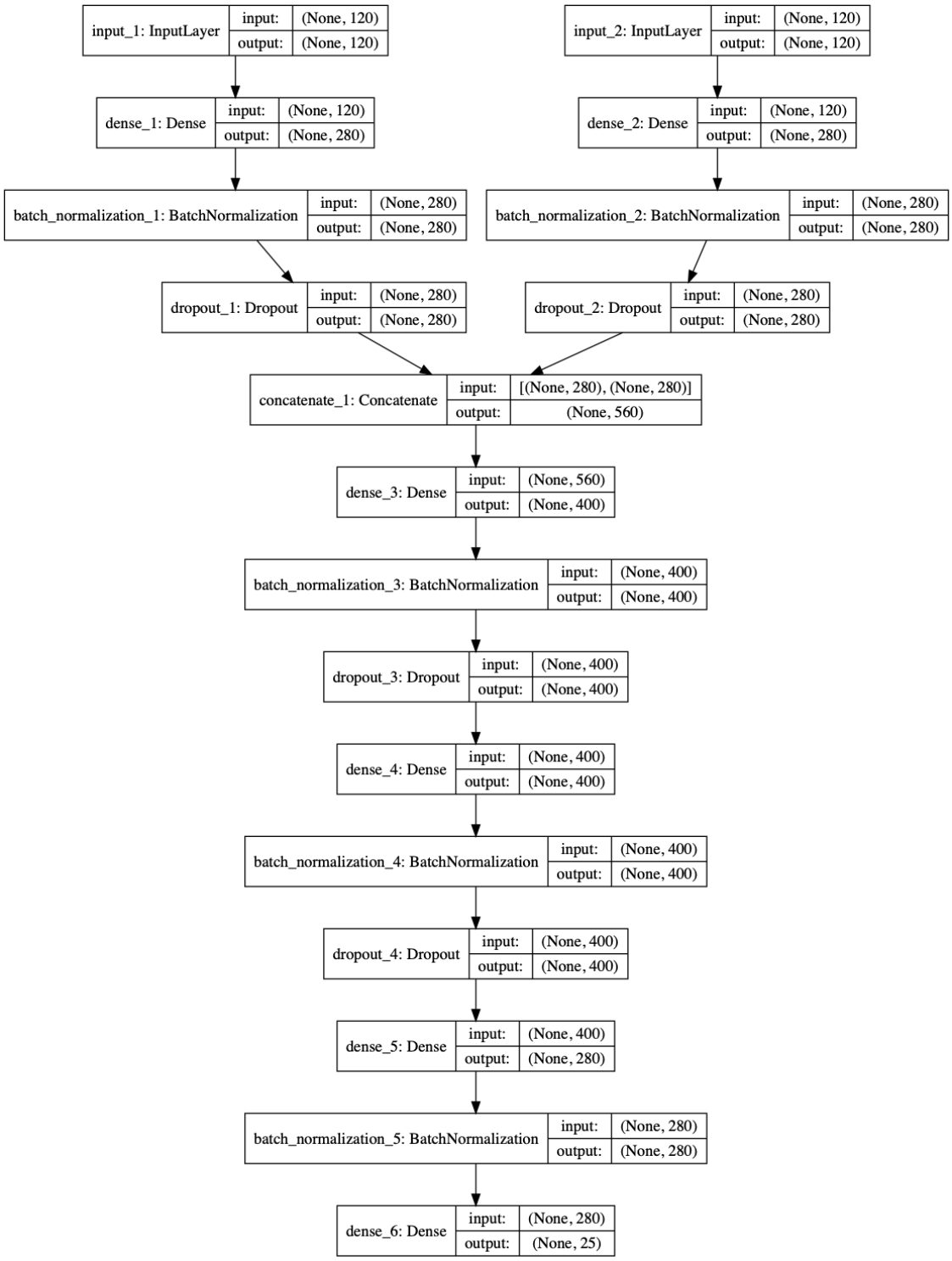


Figure 5.5: Early Fusion Model Architecture with Parameters

5.4 Late Fusion

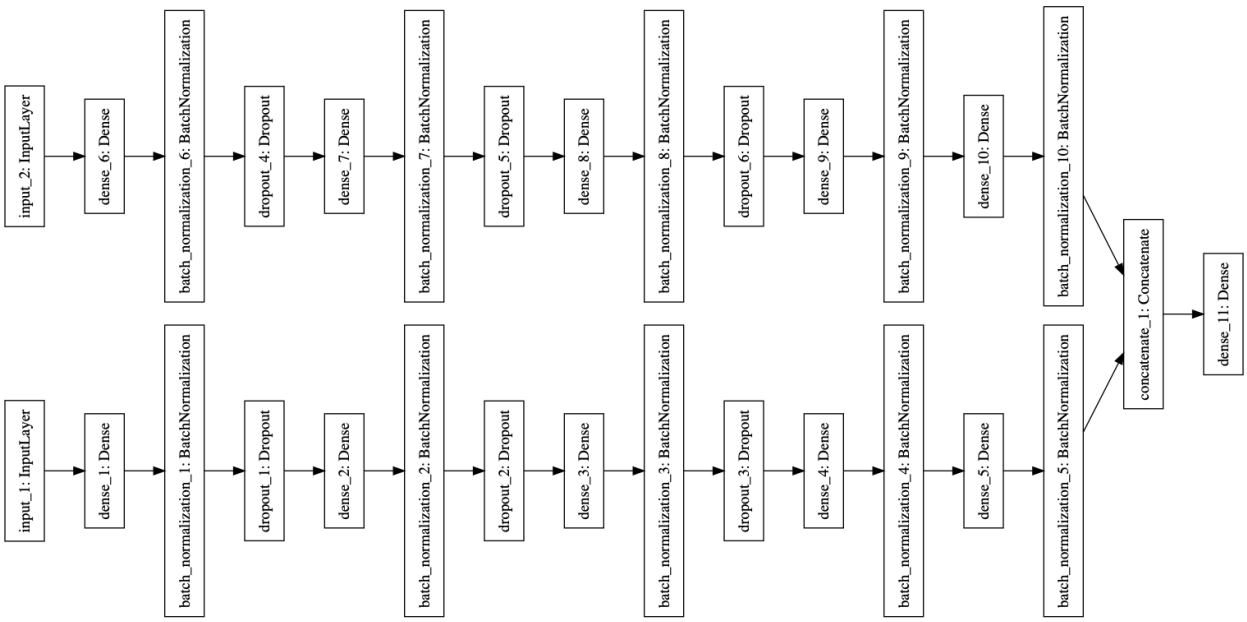


Figure 5.6: Late Fusion Model Architecture

Late fusion is the second architecture we propose for multi-view learning. Instead of merging two models in the early stage of the training, late fusion concatenates the models in the late stage of the training, usually the joined representation of the views is the input of the output layer. In figure 5.6, the model for each view consists of three hidden layers and they have the same architecture.

5.4.1 Parameter Detail

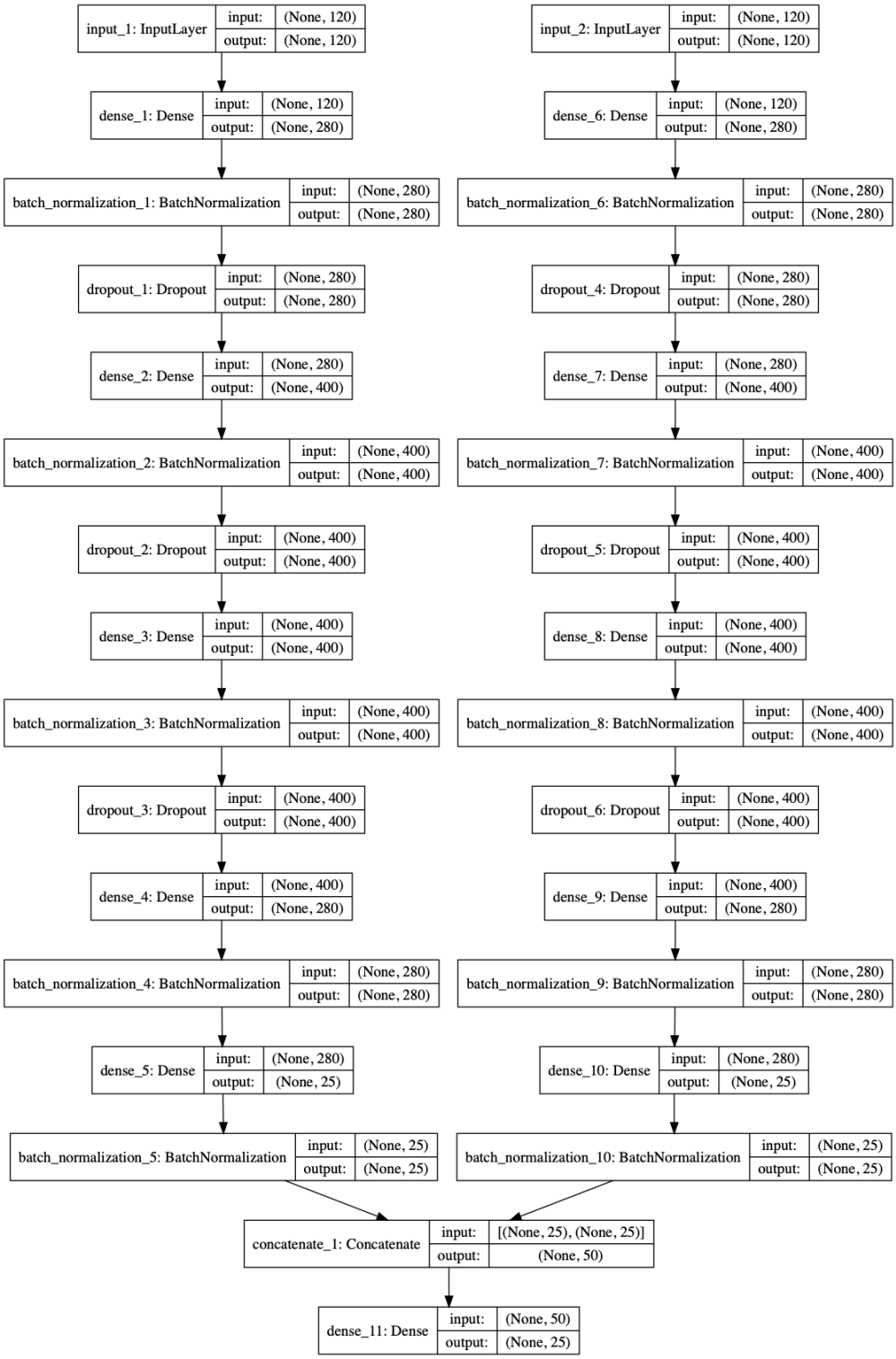


Figure 5.7: Late Fusion Model Architecture with Parameters

5.5 Conclusion

In this chapter, we explored three different deep learning neural network architectures and its parameters, the rationals behind choosing those parameters were discussed. In the next chapter, comprehensive experimental results are presented and evaluated.

Chapter 6

Performance Evaluation

6.1 Introduction

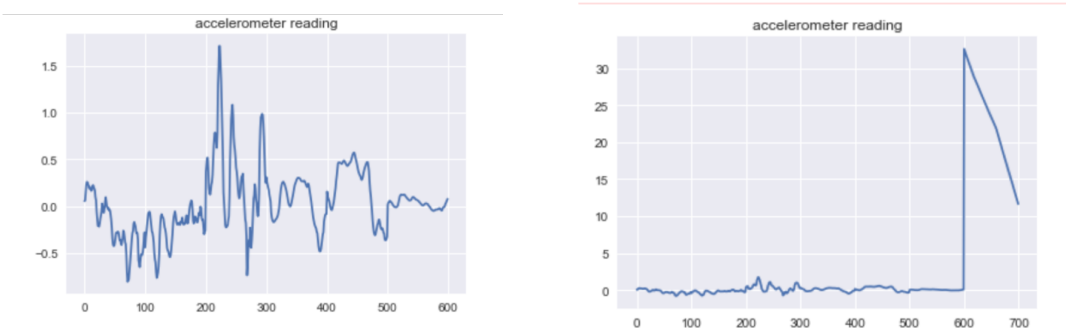
In the previous chapter, the architecture of our models were explored and its parameters were analyzed. In this chapter, we will evaluate the accuracy of our models with various parameters as mentioned earlier. First, we will compare the model performance with different features, that is to analyze the effect of various Δt, Ω and combination of sensor data on both single key and bigram approaches. Second, we will look into different evaluation metrics of the model and analyze their results for both approaches. Lastly, we will compare all the models with different parameters including Random Forest and SVM.

6.2 Feature Analysis

As our work previously stated that the features are constructed with two parameters, and they are Δt and Ω. In this section, we will discuss the effect of incorporating light sensor reading and without it. Next, we will divide the analysis into two parts, one is with single key press and another is with bigram.

6.2.1 Combination of Sensor Data

As mentioned before, the light sensor has unexpected low sampling rate and interpolation was applied to preprocess sensor data. As figure 6.1 displays, the left plot shows the sensor reading without light sensor, and the right one is the reading with light sensor. During our experiments, we discovered that the light sensor reading makes the time series less recognizable, thus we decide to exclude light sensor from our feature vectors and only combine accelerometer and gyroscope sensor reading.



(a) Without light (b) With light

Figure 6.1: light sensor comparison

In figure 6.2, the accuracy is increased by 10% when the feature vector excludes light sensor reading is fed to the network. The green graph represents the feature without light sensor reading and the blue graph represents the feature with light sensors. The green graph is always below the blue graph in loss plot and it is always above the blue graph in accuracy plot, this demonstrates the light sensor does not make any contribution to improve the quality of the features, instead it actually hurts the accuracy of the model. Thus, this proves it is a correct decision to remove light sensor reading from our feature vector.

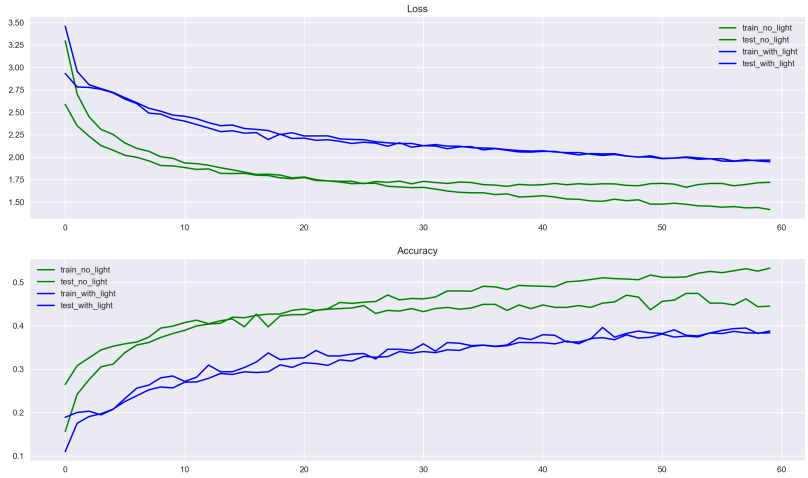


Figure 6.2: Accuracy and loss for features with and without light

6.2.2 Single Key Press

As we have previously stated in table 4.1, a range of Δt is experimented to find the most suitable one. As figure 6.3 shows, the green graph denotes the model with 200ms Δt, clearly the blue graph has highest accuracy and the lowest loss among all three models. Thus, the most suitable Δt is 200ms.

In figure 6.4, the blue graph represents the model with Ω equals 200 and it has the highest accuracy and lowest loss among all three models. Thus, we pick 200 samples/window as Ω.

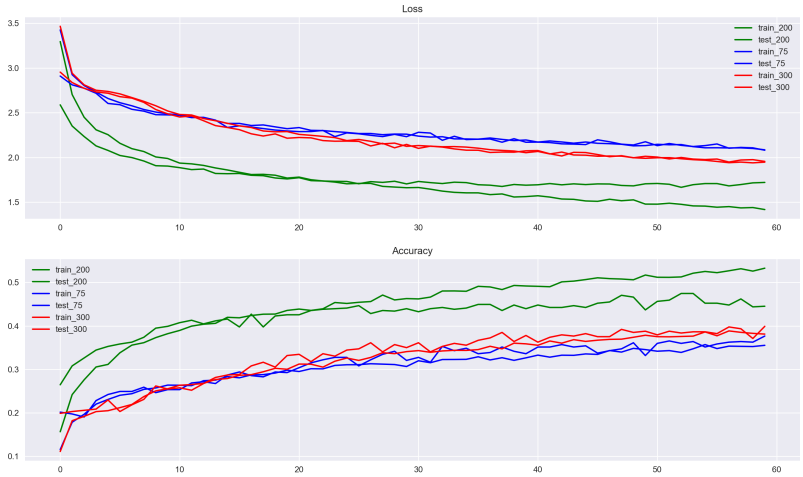


Figure 6.3: Accuracy and Loss during Training with different Δt

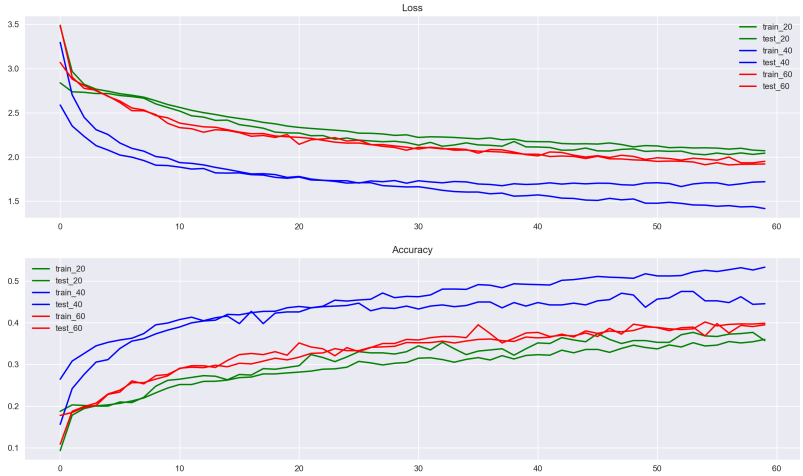


Figure 6.4: Accuracy and Loss during Training with different Ω

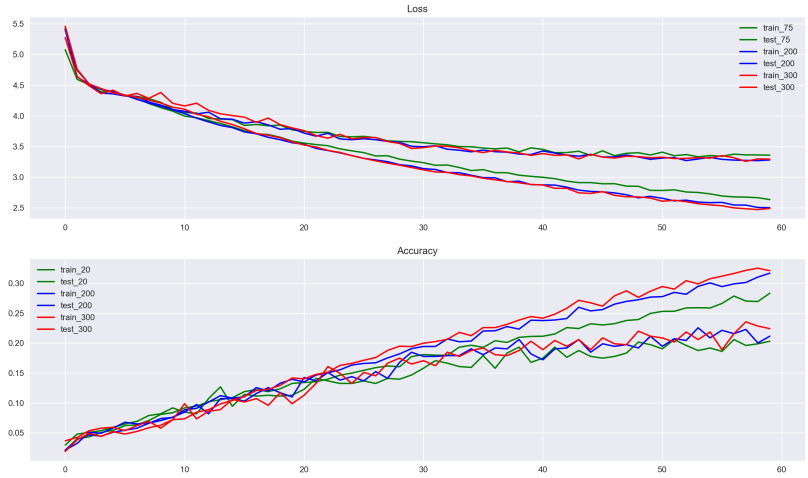


Figure 6.5: Accuracy and Loss during Training with different Δt

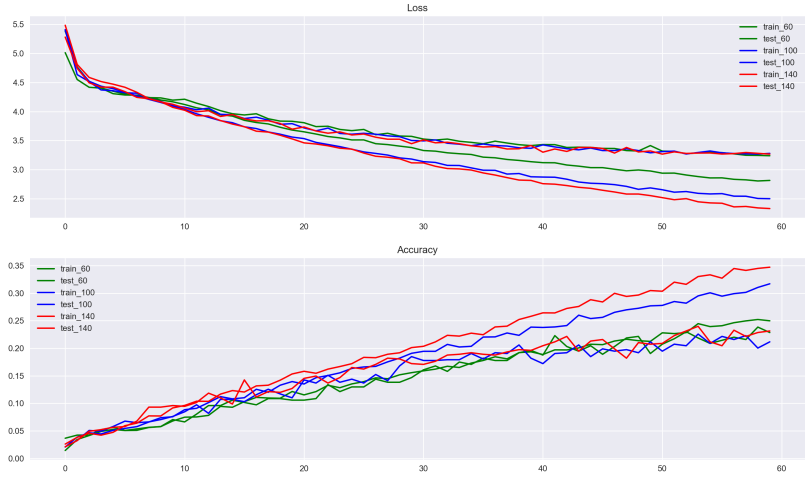


Figure 6.6: Accuracy and Loss during Training with different Ω

6.2.3 Bigram

The bigram utilizes the same set of Δt as the single key approach, however, the Ωs we apply for bigram are larger than the Ωs for single key approach and that is because the bigram approach will always have longer time series data compare to the single key approach. In this section, we

will explore the effect of various Δt and Ω to find the most suitable one for the bigram approach.

As figure 6.5 illustrates, the red and blue graph have similar accuracy and loss, we pick 200ms as Δt for our bigram approach. Figure 6.6, shows the model accuracy and loss of three different Ω values. We can see that the red graph has the lowest loss and highest accuracy among our models, thus 140 is picked as the our Ω for bigram approach.

6.3 Performance Evaluation

In multi-class classification, categorical cross entropy is often used for measuring the performance of a classification model whose output is a probability between 0 and 1. The categorical cross entropy can be represented as the following:

$$-\sum_{c=1}^{M} y_{o,c} * log(P_{o,c}) \tag{6.1}$$

where M is the number of classes, y is the binary indicator (0 or 1) if class c is the ground truth for observation o, and P is the predicted probability of observation o is classified as class c. Because of the natural log function, categorical cross entropy penalizes heavily on those predictions that are confident and wrong. There are two types of metrics used to evaluate our models and they are discussed in the following:

- ***Classification Accuracy*** - Classification accuracy calculates the mean accuracy rate across all predictions.
- ***Top-5 Categorical Accuracy*** - Top-5 categorical accuracy calculates the top-5 categorical accuracy rate, which means it is a success when the target class is within the top-5 predictions.

6.3.1 Accuracy vs Top-5-categorical Accuracy

Top-5 categorical accuracy is included in our evaluation metrics is because semantic analysis can be applied to further increase our model's performance based on top 5 candidates. As we

have previously discovered, single key approach has higher accuracy than bigram approach and the most optimal Δt and Ω are 200 and 40 for single key approach, respectively. We randomly split our data set into a 9:1 ratio which are the sample set and the testing set, then we randomly split the sample set into a 9:1 ratio which are the training set and validation set. We adopt the single key press approach and use the phase 1 data as our data set, the data is split into 6423 and 714 samples for training and validation, respectively and we obtain an accuracy of 0.865 and 0.481 for top-5-categorical accuracy and accuracy, respectively. If we consider the loss and accuracy graphs in figure 6.7, we can observe that the top-5-categorical accuracy always has less loss and higher accuracy than the accuracy metrics across all epochs.

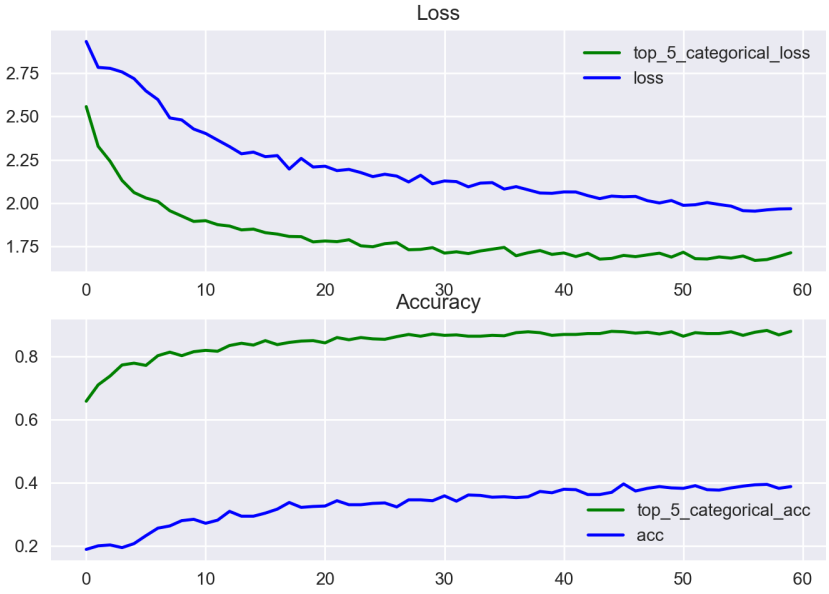


Figure 6.7: Accuracy and Loss comparison

6.3.2 Classification report

Although accuracy is an intuitive measure of calculating the ratio of correct predictions over all predictions and it is useful for the overall performance measurement. However, it does not provide evaluation on individual class. Precision is the ratio of true positive over total predicted positive.

Recall is ratio of true positive over total actual positive. F1 score is the harmonic mean of the precision and recall, where the F1 score is the best at 1 and the worst at 0. Support is the total occurrence of an observation. A macro average computes the metric independently for each class and then take the average hence treating all classes equally, whereas a micro average aggregates the contributions of all classes to compute the average metric hence an equal weight is assigned to each observation. In a multi-class classification setup, micro average is preferable if you suspect there might be class imbalance.

	Precision	Recall	F1-score	Support
	0.67	0.80	0.73	156
a	0.44	0.58	0.50	38
b	0.00	0.00	0.00	8
c	0.00	0.00	0.00	6
d	0.11	0.12	0.11	17
e	0.42	0.69	0.52	89
f	0.33	0.17	0.23	41
g	0.20	0.05	0.08	21
h	0.49	0.59	0.54	51
i	0.00	0.00	0.00	9
l	0.38	0.55	0.44	11
m	0.50	0.26	0.34	46
n	0.62	0.42	0.50	24
o	0.45	0.64	0.53	66
p	0.14	0.07	0.10	14
r	0.38	0.25	0.30	40
s	0.40	0.10	0.15	42
t	0.43	0.57	0.49	70
u	1.00	0.14	0.25	7
v	0.00	0.00	0.00	6
w	0.50	0.31	0.38	13
y	0.62	0.26	0.37	19
micro avg	0.48	0.48	0.48	794
macro avg	0.37	0.30	0.30	794
weighted avg	0.46	0.48	0.45	794

Table 6.1: Classification report

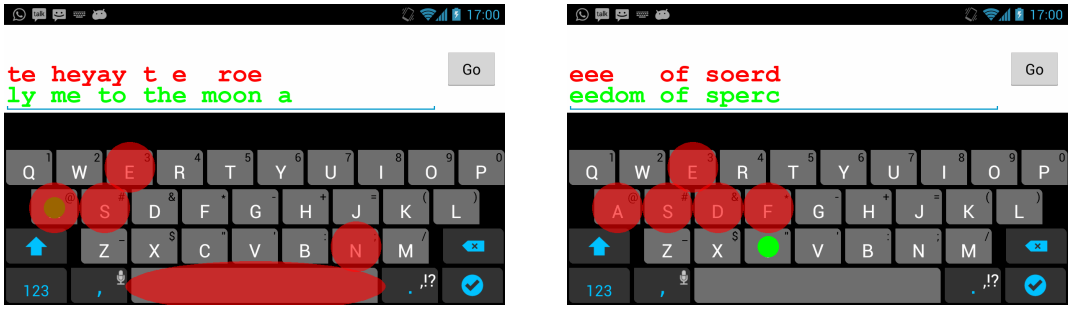
In table 6.1, the precision, recall, f1 score and support are calculated for each class, as well as the micro, macro and weighted average. We can observe that space bar occurred 156 out of 794,

and it has the highest precision, recall and f1 score among all classes. Small classes such as 'b', 'c' and 'i' have close to 0 in precision, recall and f1 score. However, class 'l' and 'u' have relatively high precision, recall and f1 score considering their small supports. Overall, we are able to observe that precision, recall and f1 score can be improved with large support and small support does not guarantee bad performance.

In addition, the micro average precision and macro average precision are 0.48 and 0.37, respectively, which is caused by class imbalance that large classes dominate small classes. It is easy to observe that space bar has the largest support and it performs decently. On the other hand, the majority of small classes perform poorly and they have close to 0 in precision, recall and f1 score. This phenomenon indicates that we have imbalanced classes, and micro average is our preferred classification measure.

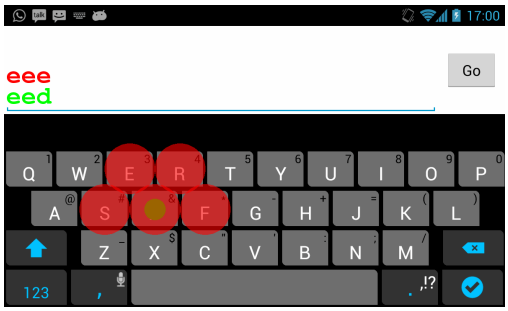
6.3.3 Visualization for Top-5 Categorical Accuracy

In this section, we are going to visualize the results yielded by top-5 categorical accuracy in a spatial manner, that is to map all top 5 candidates onto a keyboard to demonstrate the spatial correlation of the ground truth and the candidates. The figures are listed below:

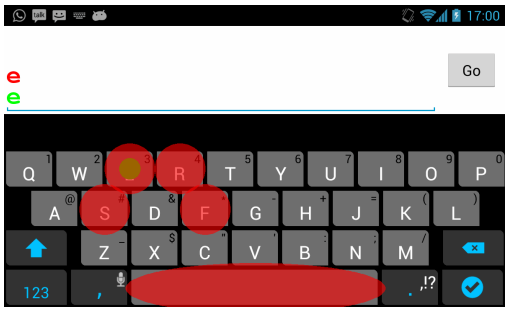


(a) key press a (b) key press c

Figure 6.8: key press a(left) and c(right)

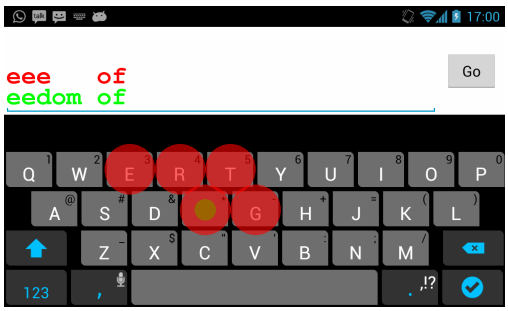


(a) key press d

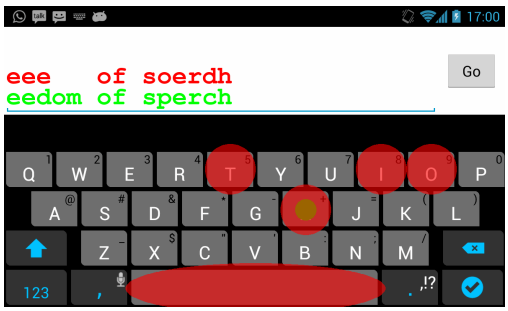


(b) key press e

Figure 6.9: key press d(left) and e(right)

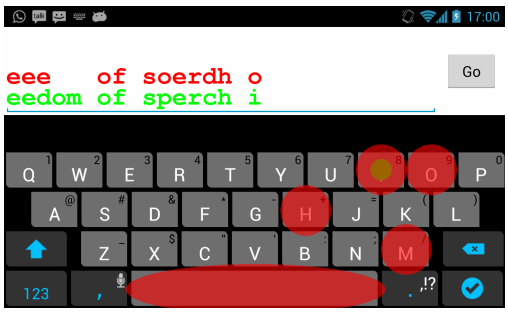


(a) key press f

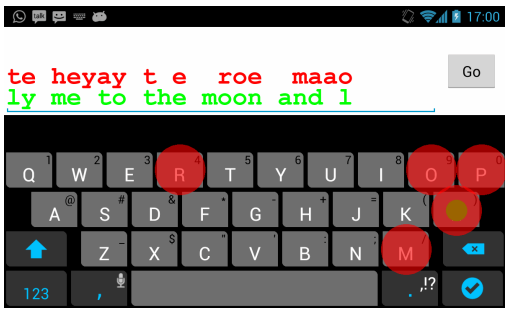


(b) key press h

Figure 6.10: key press f(left) and h(right)

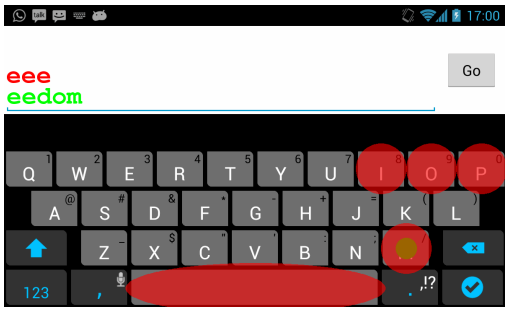


(a) key press i

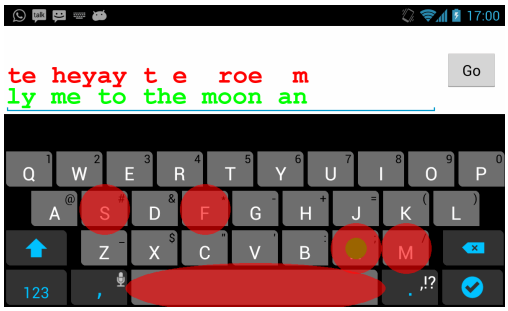


(b) key press l

Figure 6.11: key press i(left) and l(right)

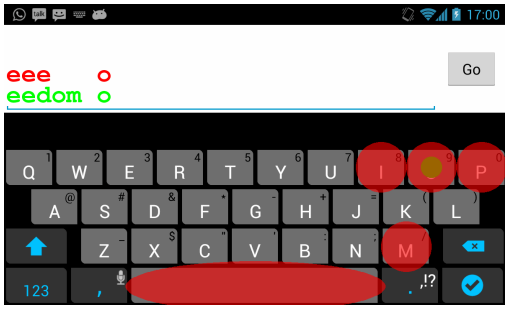


(a) key press m

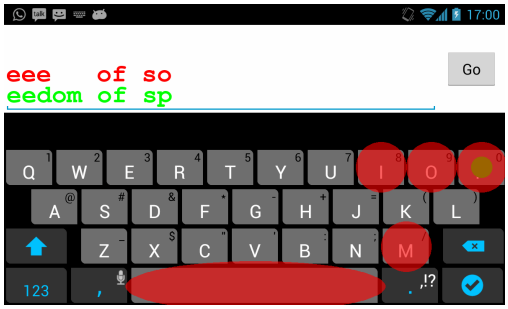


(b) key press n

Figure 6.12: key press m(left) and n(right)

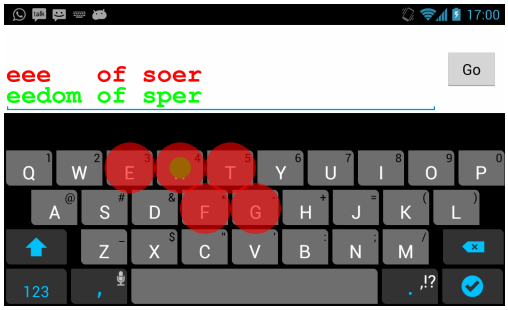


(a) key press o

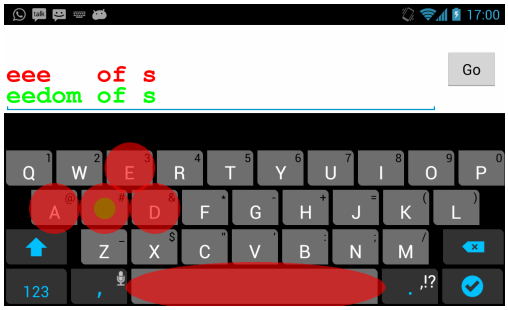


(b) key press p

Figure 6.13: key press o(left) and p(right)

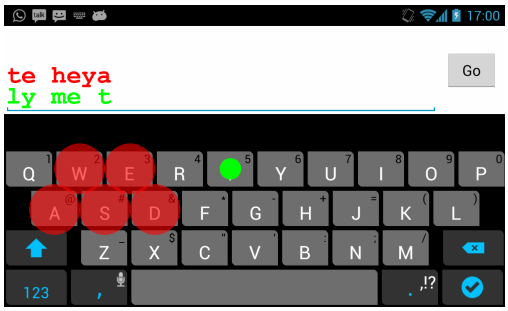


(a) key press r



(b) key press s

Figure 6.14: key press r(left) and s(right)



(a) key press t (b) key press y

Figure 6.15: key press t(left) and y(right)

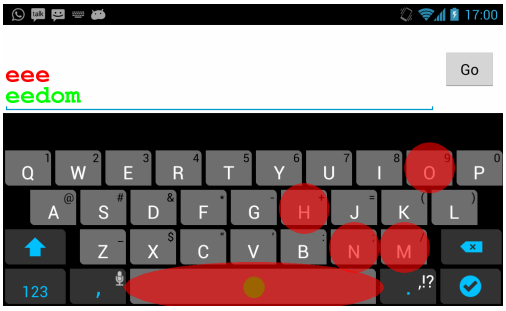


Figure 6.16: key press space)

In the above figures, the green denotes the ground truth while the red represents the top-5 candidates. As you can see, the ground truth is included in the top-5 candidates in most of the figures. We also notice that the top-5 candidates are relatively close to each other and they form a cluster together. For instance, in figure 6.9, key press 'd' classifies the observation as 'e' while the ground truth is 'd', however, the prediction and ground truth are clustered in the same region. Based on this behavior, we believe that a semantic analysis or natural language processing can be applied to future explore the model and improve its performance.

6.4 Model Comparison

In this section, we are going to analyze our models with respect to various aspects, such as combinations of sensors, model structures and types of models. To keep our evaluation criteria unified throughout the experiments, we use single key approach for our feature extraction and accuracy is our evaluation metrics.

6.4.1 Sensor Comparison in Sequential Model

In figure 6.17, the model has about 43% accuracy when only accelerometer readings are used. Similarily, the model has approximitely 43% accuracy when only gyroscope readings are used. Surprisingly, the fused sensor readings improves the model accuracy from 43% to 48%, it contradicts the conclusion of [15] that gyroscopes give the best performance while fusing gyroscope and accelerometer readings decreases the model performance.

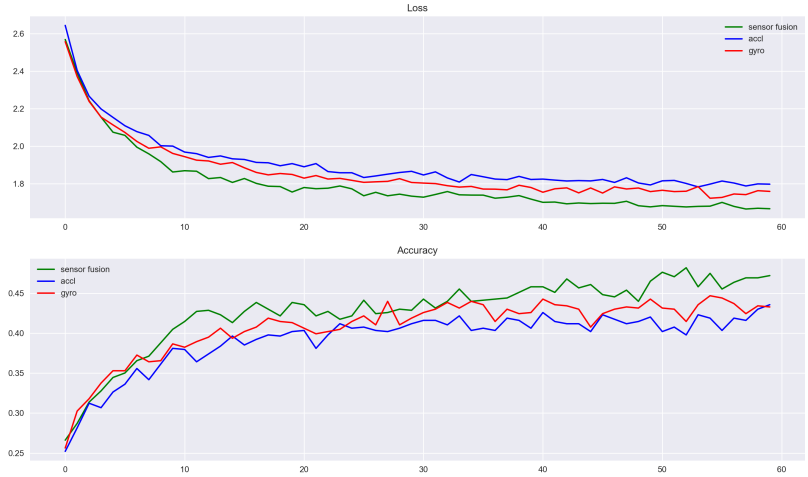


Figure 6.17: Sensor performance comparison

6.4.2 Effectiveness of views in Multi-view Learning

Instead of using one view, multi-view learning takes in multiple views that satisfy the consensus principle and the complementary principle. In our work, we propose two different approaches for crafting our views and they are the followings:

- ***Two views*** - We treat each sensor as an individual view of the model,
- ***Six views*** - Each directional axis of the sensor is treated as a view, and both sensors are used thus we have 6 views in total.

As shown in table 6.2, the model with two views performs better than the model with six views in both early and late fusion. The sequential model has the highest accuracy of 48.2% and we expected multi-view learning models to outperform sequential model, however, this is not observed in our results as the sequential model still performs the best. Additionally, early fusion model has better performance than late fusion model.

	Two Views	Six Views
Early Fusion	0.481	0.463
Late Fusion	0.445	0.429

Table 6.2: View effectiveness comparison

6.4.3 Classification Model Evaluation

We selected several classification models list below and compared their performance based on their the overall classification accuracy:

- Sequential Deep Neural Network (s-DNN)
- Multi-view Learning with Early fusion (e-ML)
- Multi-view Learning with Late fusion (l-ML)
- Random Forest(RF)

- Support Vector Machine (SVM)

As shown in table 6.3, the sequential model performs the best and has an accuracy of 48.2% when accuracy is the evaluation metrics. Similarly, the multi-view learning with early fusion model has the highest top-5 categorical accuracy 87%. In all instances, changing the metrics from accuracy to top-5 categorical accuracy increases the performance of all models by approximately 40%, and it is expected since now we treat all top 5 predictions as successful observations. Another observation is that our three deep neural network models outperform Random Forest and Support Vector Machine in both accuracy and top-5 categorical accuracy.

Evaluation Metrics	s-DNN	e-ML	l-ML	RF	SVM
Accuracy	**48.2%**	48.1%	44.5%	37.4%	36.1%
Top-5 categorical accuracy	86.8%	**87%**	84.1%	77.8%	74.3%

Table 6.3: Overall classification performance

6.4.4 Conclusion

In this chapter, we evaluated different sets of features and we found out that light sensor reading drastically decreases our model performance due to its low sampling rate as we discussed before. We then tested all possible combinations of features with various Δt and Ω, and discovered the best set of parameters for both single key and bigram approach. Based on our analysis for both single key and bigram approach, we realized that single key outperformed bigram in all scenarios, thus we decided to further our studies with only single key approach.

After the best features were chosen, we evaluated the different evaluation metrics accuracy and top-5 categorical accuracy. Later, the classification report was presented to demonstrate the problem of class imbalance in our data and we found out that large class dominated small classes. We then listed all the visualization aids for the results of top-5 categorical accuracy and displayed their spatial corrections to the ground truth.

After the analysis of the evaluation metrics, we studied the effectiveness of various combinations of sensors, and we observed that fused sensor data performed better than sensor was used

alone in the model. Next, we analyzed the performance of views in multi-view learning setting including two views and six views. We noticed that models with two views performed better than models with six views, also models with early fusion achieved higher accuracy than models with late fusion. In the last section, we analyzed 5 different classification models. To our surprise, the sequential model achieved 48.2% accuracy and performed better than both multi-view learning models. With the evaluation metrics being top-5 categorical accuracy, multi-view learning with early fusion achieved the highest accuracy of 87%. Another interesting observation was that all of our deep neural networks outperformed Random Forest and Support Vector Machine models.

Chapter 7

Conclusion and Future Work

In our work, we attempted to explore the feasibility of applying deep learning to text inference attacks. However, in recent years, deep learning has taken over the field of machine learning and it has been adopted extensively in many areas of machine learning especially computer vision. Many inexplicable problems and difficult challenges now can be improved or solved with deep learning techniques. On the other hand, text inference is a new research area of side channel attack on smartphones and it has gathered more attention gradually due to the improvement of hardware equipped in our smartphones. Nonetheless, only a few studies have been conducted related to text inference attacks and most of them are strictly limited to infer keystrokes on numeric keyboards. Among all of the previous work, only a few have successfully inferred keystrokes on the QWERTY Keyboards using conventional classification models such as Random Forest and Support Vector Machine. In our work, we are the first to implement deep learning framework to infer keystrokes on QWERTY Keyboards to our knowledge, and we believe that deep learning is an effective and not well explored framework in the research of text inference attacks.

We suggested that the performance of the model could be improved by fusing the sensor readings together. Thus, we analyzed the possible sensors we could use for our experiments and we came up with accelerometer, gyroscope and light sensor. Then an Android native application was designed to facilitate our data collection process. All the sensor readings were saved in CSV format and later transferred to our computers for further analysis. We then came up with two different feature extraction techniques, single key press and bigram. In order to capture the latent properties of two adjacent key presses, bigram was extracted from single key presses and we expected it was a good fit to preserve the hidden properties of two adjacent presses.

After a few rounds of experiments, we derived to our conclusion that bigram was less effective than single key approach, and it may have potential to outperform single key approach if we have enough data and experiment different features. Moreover, we also observed that light sensor cause the model to perform poorly because of its low sampling rate, and this is a hardware limitation exists in Android ecosystem.

We compared the effectiveness of different combinations of sensors and found out that fused sensor data performed the best, which it is a contradiction to the work of Al-Haiqi et al. [15] that gyroscope alone performs the best. Additionally, the performance of different views in multi-view learning was explored, and we observed that two views performs better than six views, multi-view learning with early fusion achieved higher accuracy than multi-view learning with late fusion.

Eventually, we compared all of our models, namely sequential deep neural network, multi-view learning with early fusion, multi-view learning with late fusion, Random Forest and Support Vector Machine. Our results suggest that all of our three deep learning models performed better than RN and SVM. Among our deep learning models, the sequential model had the high accuracy when only the observation with highest probability considered as a successful prediction. Similarly, the multi-view learning with early fusion performed the best when observations with top 5 probabilities considered as successful predictions.

7.1 Directions of Future Work

In our work, we believe that our model can still improve its performance drastically considering its 48.2% accuracy and 87% top-5 categorical accuracy. The list of future work is presented below:

- As we mentioned before, semantic analysis or natural language processing can be applied to our top-5 candidates. If we know 87% of key presses can be successfully classified within its top-5 candidates, a word can be reconstructed by calculating the probability of each key press forming a word based on the English word frequency. For example, if the first and second letter are predicted as 'a' and 'r', then next letter has a high probability of being 't'

and 'e' if the word only has three letters.

- Regarding our feature extraction approaches, we only considered single key and bigram and they are in the time domain. A good direction of future work can be extracting features from frequency domain and statistical domain, and those approaches have been adopted by previous work. We believe that by extracting various features may increase our model's performance.

- In our work, the timestamp of each key press is recorded as part of the ground truth. However, this is not practical in real-world application. Another direction of future work can be inferring the timestamp of individual press, then proceed to infer the keystroke.

- Lastly, as we mentioned before, we hope that we can identify different users by their typing patterns. In the future, we can build a model that can identify user purely based on the sensor readings.

We believe our work addresses the feasibility of implementing deep learning framework on text inference attacks, and we think this will be the future direction of side channel attacks.

References

[1] D. Burke, "Android: celebrating a big milestone together with you," 2017.

[2] idc, "Os data overview," 2017, last accessed 24 April 2019. [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview

[3] F. Foerster, M. Smeja, and J. Fahrenberg, "Detection of posture and motion by accelerometry: a validation study in ambulatory monitoring," *Computers in Human Behavior*, vol. 15, no. 5, pp. 571–583, 1999.

[4] J. Engelbrecht, M. J. Booysen, G.-J. van Rooyen, and F. J. Bruwer, "Survey of smartphone-based sensing in vehicles for intelligent transportation system applications," *IET Intelligent Transport Systems*, vol. 9, no. 10, pp. 924–935, 2015.

[5] A. Mahfouz, T. M. Mahmoud, and A. S. Eldin, "A survey on behavioral biometric authentication on smartphones," *Journal of information security and applications*, vol. 37, pp. 28–37, 2017.

[6] L. Sun, Y. Wang, B. Cao, S. Y. Philip, W. Srisa-An, and A. D. Leow, "Sequential keystroke behavioral biometrics for mobile user identification via multi-view deep learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 228–240.

[7] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "Mole: Motion leaks through smartwatch sensors," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 155–166.

[8] C. Song, F. Lin, Z. Ba, K. Ren, C. Zhou, and W. Xu, "My smartphone knows what you print: Exploring smartphone-based side-channel attacks against 3d printers," in *Proceedings of the*

2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016, pp. 895–907.

[9] M. Sabra, A. Maiti, and M. Jadliwala, "Keystroke inference using ambient light sensor on wrist-wearables: a feasibility study," in *Proceedings of the 4th ACM Workshop on Wearable Systems and Applications*. ACM, 2018, pp. 21–26.

[10] M. Hussain, A. Al-Haiqi, A. Zaidan, B. Zaidan, M. M. Kiah, N. B. Anuar, and M. Abdulnabi, "The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks," *Pervasive and Mobile Computing*, vol. 25, pp. 1–25, 2016.

[11] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion." *HotSec*, vol. 11, pp. 9–9, 2011.

[12] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: password inference using accelerometers on smartphones," in *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. ACM, 2012, p. 9.

[13] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2012, pp. 113–124.

[14] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith, "Practicality of accelerometer side channels on smartphones," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 41–50.

[15] A. Al-Haiqi, M. Ismail, and R. Nordin, "Keystrokes inference attack on android: A comparative evaluation of sensors and their fusion," *Journal of ICT Research and Applications*, vol. 7, no. 2, pp. 117–136, 2013.

[16] S. Narain, A. Sanatinia, and G. Noubir, "Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning," in *Proceedings of the 2014 ACM*

conference on Security and privacy in wireless & mobile networks. ACM, 2014, pp. 201–212.

[17] Y. Song, M. Kukreti, R. Rawat, and U. Hengartner, "Two novel defenses against motion-based keystroke inference attacks," *arXiv preprint arXiv:1410.7746*, 2014.

[18] T. Nguyen, "Using unrestricted mobile sensors to infer tapped and traced user inputs," in *2015 12th International Conference on Information Technology-New Generations*. IEEE, 2015, pp. 151–156.

[19] C. Shen, S. Pei, Z. Yang, and X. Guan, "Input extraction via motion-sensor behavior analysis on smartphones," *Computers & Security*, vol. 53, pp. 143–155, 2015.

[20] D. Hodges and O. Buckley, "Reconstructing what you said: Text inference using smartphone motion," *IEEE Transactions on Mobile Computing*, vol. 18, no. 4, pp. 947–959, 2019.

[21] Android, "Sensors overview," 2019, last accessed 24 April 2019. [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[23] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," *arXiv preprint arXiv:1304.5634*, 2013.

[24] Android, "Sensor manager," 2019, last accessed 05 May 2019. [Online]. Available: https://developer.android.com/reference/android/hardware/SensorManager

[25] D. Ping, X. Sun, and B. Mao, "Textlogger: inferring longer inputs on touch screen using motion sensors," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2015, p. 24.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.