Arab American University

Faculty of Graduate Studies

Android malware prediction based deep learning approaches: dimensionality reduction and data transformation.

Prepared By

TaqiEddin Fathi Ahmad Alqam

Supervisor By

Dr. Yousef Draghmeh

This Thesis Was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Data Science and Business Analytics

January / 2023

Thesis Approval

Android malware prediction based deep learning approaches: dimensionality reduction and data transformation.

By

TaqiEddin Fathi Ahmad Alqam

This thesis was defended successfully on 25 – 2 – 2023 and approved by:

Committee members		Signature
1. Dr. Yousef Draghmeh	(Supervisor)	
2. Prof. Mohammad Awad	(Internal examiner)	
3. Dr. Amjad Hawash	(External examiner)	

Declaration

I declare that the thesis titled " Android malware prediction based deep learning approaches: dimensionality reduction and data transformation. " is my work, and has been composed solely by myself and does not contain work from other researchers, and has not been submitted for any other degree or scientific work except the reference is made.

Name: TaqiEddin Alqam

Date: January, 2023

Signature: 

Dedication

I thank God who inspired me with strength and patience to finish my master's degree in a journey that lasted for two years. I would like to dedicate this thesis to my mother, who encouraged me to complete my studies and obtain a master's degree, and to my father, who was supportive and encouraging to me throughout my life.

To my brothers and sisters for their unconditional love and support, which has motivated me throughout my life.

To my close friends, who are ten friends. We met at school, and today I am finishing my master's degree and we are still friends, and they are still a source of joy and support for me.

To my dear university, the Arab American University, in which I lived my master’s journey and spent two years in it, the details of which I will not forget throughout my life.

I know that all words are not enough to express my love for you, but I hope that you will find in this dedication some thanks for standing with me throughout my life.

Acknowledgment

I would like to use this space to express my deep gratitude to Dr. Yousef Draghmeh for his advices, help, and valuable time; he spent reviewing and correcting my work. Dr. Yousef Draghmeh provided useful suggestions and advice that have had an important effect and helped in overcoming many obstacles in preparing this work in the best way possible.

Abstract

Android malware prediction based deep learning approaches: dimensionality reduction and data transformation

Prepared By: TaqiEddin Fathi Ahmad Alqam

Supervisor By: Dr. Yousef Draghmeh

Android is free, open-source and the most popular mobile operating system. Android's worldwide market share was 72.22% in the fourth quarter of 2020, and although it dropped to 71.8% by the end of 2022, it is still well in front. Recent Android malware defenses, which detects dangerous data of malware based machine learning, have become a significant issue in information security research due to their importance in keeping devices secure. Traditional machine learning approaches are limited in their ability to learn complicated representations in high-dimensional domains. Furthermore, the success of machine learning models relies heavily on training data, and as Android apps evolve and software engineering advances, these trained models are likely to become outdated. This research develops a hybrid Android app classification model into benign and harmful based Convolutional Neural Networks. The model combines dimensionality reduction and data conversion into images using Image Generator for Tabular Data algorithm. To evaluate the model, the CICMalDroid 2020 data set was used. The proposed model achieved a high accuracy of 94.38% in the binary classification (benign and harmful) and 95.83% in the multiple classification.

Keywords: Android, Android malware, CICMalDroid 2020, dimensionality reduction, data transformation, Image Generator for Tabular Data (IGTD), Convolutional Neural Network (CNN)

Table of Contents

List of Figures

List of Tables

List of Abbreviation

Abbreviation	Description
API	Application Programming Interface
ANN	Artificial Neural Network
APK	Android Package Kit
C&C server	Command and Control server
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAE	Deep Autoencoders
DBN	Deep Belief Network
DNN	Deep Neural Network
DT	Decision Tree
GAN	Generative Adversarial Network
GNN	Graph Neural Networks
IGTD	Image Generator for Tabular Data
KNN	K-Nearest Neighbors
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
NMF	Non-Negative Matrix Factorization
OS	Operating System
RF	Random Forest
SL	Simple Logistic
SMS	Short Message Service
SVM	Support Vector Machines
TAN	Tree Augmented Naïve Bayes
XML	Extensible Markup Language

List of Equations

Chapter 1

Introduction

1.1 Overview

Android, empowered by Google, the free and open-source operating system has the biggest market share among smartphone operating systems[1]. According to [2], Android topped the market in the last quarter of 2020, with 72.22% of the global market share, and although its share declined at the end of 2022, it maintained its lead by 71.8%. Due to Androids openness, popularity, and the massive evolution in the mobile world [3], it has been widely adopted as the operating system of choice for smartphones, tablets, and even Internet of Things (IoT) devices.

The Android operating system is organized in a hierarchical fashion, and its construction included the use of software packages that were based on the Linux kernel. These software packages included the hardware abstraction layer, C and C++ libraries, and the Android Runtime environment. Each layer is composed of a large number of subsystems that are connected to one another through system calls.

Android applications are accessible from potentially malicious third-parties software besides the official Android Market due to the current evolutionary process, wide distribution, and the open-source nature of Android operating system [4]. This third-party software is called malware or malicious software, and it is a program or a file that hurts the system of a device. These malignant programs can steal, encrypt, change, corrupt user data or delete sensitive data, or even monitor user activities without authorization.

Researchers at Kaspersky [5] have discovered a trend in the last years about a large number of malignant applications downloaded from google play store, which is the main app store for android mobile devices. Also, according to another research, anti-malwares was able to find

almost five million malignant Android applications in 2020-21[6]. Hence, we find that the main goal of protection experts is to keep devices safe for use and protect them from hackers and malicious programs, and therefore, malware detection programs are developed to find and reject it before downloading.

Due to the development of the Android operating system and the issuance of new versions of it, many of the systems that were trained using data extracted from the Android operating system become outdated and ineffective due to the development of malicious programs.

Therefore, in this research, we will propose to build a new model and train it using new data, as it is effective in terms of training time and accuracy. It depends on data transformation and deep learning techniques such as Convolutional Neural Network (CNN), VGG16, Resnet50 and Deep Automatic Encoder (DAE).

1.2 Objectives

The objective of this thesis is to develop an efficient model for binary and multiclass malware classification. This will be achieved by using a new and reliable dataset to train the model. The aim is to obtain the highest possible accuracy while minimizing training time. To achieve this objective, the following specific goals will be pursued:

1.To conduct a comprehensive review of the literature on malware classification and identify the current state-of-the-art techniques and models.

2.To use a new dataset of malware samples that is representative of current malware threats and is suitable for training and evaluating the proposed model.

3. To design and implement a new malware classification model that is based on deep learning techniques and is optimized for efficiency and accuracy.

4.To evaluate the performance of the proposed model using standard metrics for binary and multiclass classification, and compare it with state-of-the-art models and techniques.

5.To analyze the trade-offs between accuracy and training time, and identify strategies for optimizing the model for specific use cases.

The objective of this thesis is to make a significant contribution to the field of malware classification by developing a new model that is efficient, accurate, and capable of handling both binary and multiclass classification tasks.

1.3 Research problem

The use of Android devices has increased exponentially in recent years, making them an attractive target for cybercriminals. Android malware is a significant threat to the security and privacy of users, as it can compromise sensitive information and cause financial losses. Prior research in android malware classification has mostly focused on dividing data into two categories: malicious and benign. While this approach has helped in detecting malware samples, it has not been effective in correctly categorizing them. As a result, appropriate precautions to protect Android devices may not be taken.

Moreover, many of the existing methods for android malware classification have low accuracy, with some achieving almost high accuracy, but at the expense of consuming significant time and resources. Furthermore, the data set used in many related studies is outdated (2010-2015) and extracted from old Android versions that are no longer in use. Therefore, there is a need to improve the accuracy of android malware classification by utilizing a more diverse and updated dataset and developing a new method that achieves high accuracy while minimizing the consumption of time and resources.

To address this problem, the thesis will explore the following research questions:

1. How can we develop a new classification method that achieves high accuracy while minimizing the consumption of time and resources?

2. How does the proposed method compare with existing methods in terms of accuracy, time, and resource consumption?

3. How effective is the proposed method in detecting and categorizing android malware samples in real-world scenarios?

This thesis aims to contribute to the development of more effective and efficient methods for android malware classification, which can help in improving the security and privacy of android users.

1.4 Proposed solution

Deep learning with its various structures such as recurrent neural networks, deep belief networks and convolutional neural network has achieved promising results in many applications, such as speech recognition, image recognition, face recognition, and natural language understanding [7].

Convolutional neural networks (CNN) have the best performance in image recognition among deep learning models [8]. CNN has a significant advantage over classical machine learning and other models in its capacity to extract higher-level image data. There are input layers, convolution layers, pooling layers, and fully linked layers in a CNN.

So, we use the power of convolutional deep learning in image analysis in malware detection, because the visualization technology has been introduced in malware detection in recent years [9]. Malware visualization technology refers to converting some elements of malware into an image form for detection.

In this work, we investigate building a new model based on data transformation and deep learning techniques such as Convolutional neural network (CNN), VGG16, Resnet50 and Deep Auto encoder (DAE).

We created a model that reduces the dimensionality of the data using DAE. Then, IGTD is used to convert the data into images that are used to train an ensemble of CNNs to be able to classify malware binary and multiclass. To the best of our knowledge no one has proposed this methodology for classifying Android malware, and most researchers have focused their research on binary classification only.

This work uses public real data generated from high reputation lab [10]. The data were collected from December 2017 to December 2018, and the dataset is intentionally spanning between five distinct categories: Adware, Banking malware, SMS malware, Riskware, and Benign. Also, the methodology will be applied using binary classification and multiple classification with comparison between them for reducing the data dimensions and minimizing the prediction process time while keeping the accuracy as high as possible.

1.5 Contribution

The contribution of this work is using the IGTD for data transformation and deep learning approaches to reduce the dimensionality of Android malware data, which achieves the following benefits:

1. Reducing the dimensionality of the data means less training time and less computational resources and increases the overall performance of machine learning algorithms, and this agrees with the work in [11].

2. Reducing the dimensionality of the data avoids the problem of overfitting and takes care of multicollinearity between the features, and this agrees with the work in [12].

3. Using the IGTD, we can rearrange similar features next to each other which enables us to visualize the data.

4. to the best of our knowledge, the proposed model which depends on reducing dimensions and converting data into images using IGTD and then classifying them using a set of CNN is a new model.

Chapter 2

Literature Review

2.1 Overview

In this part of the thesis, we provide an overview of the research that has been done on the Android system as well as the methods to identify the used features in analyzing Android malware. We also identify the literature that addresses the topics of machine learning and deep learning in identifying Android malware, in addition to visualizing the behavior of Android malware and comparing it with the current work.

2.2 Android

The 14-year-old Android operating system is the most widespread in the world, and given that it is an open-source operating system [13], and by the end of 2022, the number of applications has reached more than 2.7 million on Google Play, but 37% of them are considered low quality according to AppBrain [14].

The Android system consists of a hierarchical structure, and software packages were used to build it based on the Linux kernel, in addition to the hardware abstraction layer, C and C++ libraries, and Android Runtime environment. Each layer contains many subsystems, which are linked using system calls.

Android applications are often written using the Java programming language and then compiled and placed in APK file (Android package) using “aapt” (Android Asset Packaging Tool).

Android uses virtual machine called Dalvik, which employs unique bytecode. On Android, regular Java bytecode cannot be executed. So, Android’s “dx” utility converts Java Class files into Dalvik executable files [15]

2.3 Android Applications

The Application framework is an abstract layer that serves as a basis for building apps that make use of the underlying reusable libraries and packages to facilitate user interaction with the device. Android Package (APK) files are used for distribution [16].

Each application may consist of one or more of the four sorts of components that make up an APK, and these components exchange data with one another by means of messages called Intents [17]. Specifically, the four components that make up the Android application framework are:

1. Content Providers: Components that enables application to share data and access the data of other applications
2. Activities: It manages the lifecycle of applications and provides an interface for users to use the application, as each activity deals with a single user action. For example, WhatsApp application might have one activity that shows a list of chats, another activity to the list of calls, and another activity for showing the new status, and each activity consists of extensible set of Views to build user interface, including lists, buttons, text boxes...etc.
3. Broadcast receivers: Broadcast receivers are a quiescent component of an Android app that does not display a user interface and runs in the background even when the app is closed. Broadcast receivers are activated when a specific event occurs, such as receiving calls or a message, or even the battery is about to run out, and its effect is shown to the user through a notification that appears on the screen.
4. Services: Services in Android are a special component that facilitates an application to run in the background in order to perform long-running operation tasks. The prime aim of a service is to ensure that the application remains active in the background so that the user can operate multiple applications at the same time, like playing songs or uploading and

downloading files. Android apps may utilize services for inter process communication (IPC) [15].

Using the Application framework, developers create apps in Java or Kotlin and then bundle all of their code, data, and resources into a single ZIP file called APK, which consists of:

1. AndroidManifest.xml file: This describes and provides information about the capabilities of the Android application, also declares Various application components like services, broadcast receivers, ...etc.
2. Classes.dex file: It contains application code usually written in Java or Kotlin, which compiled into Dalvik Executable format to execute on Dalvik Virtual Machine (or Android Runtime for Android newer versions).
3. Resources.arsc file: which is XML file for precompiled application resources.
4. Resources folder(/res): directory of application resources that the application will use in runtime like icons, layout, images etc.
5. Libraries(/lib) file: An optional directory that contains executed code intended to run the application on a specific processor type.
6. META-INF: file containing the app's certificate, APK signature and some required resources.

2.4 Android Malware

There is a very large number of malicious applications in android OS, and their number has exceeded 4.18 million applications [18], but in this section we only explain a set of harmful applications, which will be also covered in our dataset. The set of the harmful applications are as follows:

1. Adware: Adware is a sort of software that displays a barrage of advertisements to users based on their online activities (browsing history, search terms entered, etc.), and these

advertisements persist even after the user has stopped the application. Additionally, Adware allows attackers to steal sensitive information about users.

2. Banking malware: Mobile banking malware is a subset of Trojan-based malware that imitates legitimate banking software or a website interface in order to steal login credentials. Where this malware attacks devices and steals sensitive banking information such as the account number, user name, and password for the banking application and transmit stolen data to a command and control (C&C) server [19].
3. SMS malware: Malware that sends text messages uses the service that sends text messages as its medium of operation in order to steal text messages and use them in assaults. The malicious code is first uploaded to the hosting sites used by the attackers and then associated with the SMS. They utilize the command and control (C&C) server to manage the attack instructions, which include sending malicious SMS messages, intercepting legitimate SMS messages, and stealing data.
4. Riskware: The term “riskware” is used to describe legal software that might be used maliciously. As a result, it may mutate into another kind of malware, such adware or ransomware, which adds additional features via the installation of infected programs. These programs aren’t made with malicious intent, but they do have components that might be exploited for harm. The Riskware software may be classified as malware if it were used maliciously.

2.5 Malware analyses

Previous studies have used different types of analyses to identify malicious android applications and these analyses can be categorized into static analyses, dynamic analyses and hybrid analyses.

2.5.1 Static analyses

Static analysis is used to detect malicious applications without executing them in an Android device or emulator, as it scans parts of the application by analyzing code segments [16].

Three strategies are used in this type of analysis where distinct tags are placed to identify malicious applications and this is called signature-based strategy, and then permission requests are used to distinguish malicious applications that have not been identified from the signature-based strategy and this is known as signature-based strategy Permission and finally it uses a component-based strategy that disassembles the application and examines its critical components to determine if it is malicious or not [20][21].

The advantages of static analysis are the low computation cost, less time consuming and low resource consumption [16]. Despite these advantages, the absence of genuine execution routes and appropriate execution circumstances are the primary downsides of using static analysis. In addition to this, there are issues with the incidence of code obfuscation as well as dynamic code loading [22].

There are many malwares detection approaches that use machine learning models based on static analysis. Some research focuses on Manifest file and combine static analysis using Android Asset Packaging Tool to extract features from Manifest file and machine learning model. The study [23] achieved 93.90% accuracy in malware prediction in system named DREBIN, which was trained offline and then transferred to smartphones for direct predicting. The study [24] demonstrated a technique based on machine learning (One- Class Support Vector Machine) using manifest file for offline training, although the cost of the analysis was low, the accuracy was also low. Another lightweight approach was realized in paper [25] , by combining static analysis for some manifest file (permissions and intent filters) to train basic machine learning for malware detection with total accuracy 90%.

Also, researchers in[26] proposed a model based on the use of basic machine learning models such as (Random Forest, Decision Tree, Kneighbors, etc.) and training them with APK files after converting them into audio files. This approach achieved an average accuracy among the different models of up to 97.9%.

The researchers[27] used an approach based on three parallel machine learning models (KNN, random forest, and J48) named Mlifdect, and they obtained an accuracy of up to 99.7% in distinguishing between benign and malicious Android apps. In the same way in paper [28], three machine learning models (SVM, J48 and Random Forest) were used in to detect malicious applications, and the random forest achieved the highest accuracy, reaching 92.4%.

using Non-Negative Matrix Factorization (NMF) for reducing the dimensionality of features from API Calls to train basic machine learning models, the researchers [29] found that K Nearest Neighbors (KNN) classifier achieved the highest accuracy 98.87% in this approach. And some research has focused on classifying one type of malware as the approach proposed in the paper [29] to detect Ransomware, their approach was based on one of the powerful machine learning models (SVM) using swarm algorithm to tune model hyperparameters and they were able to achieve high performance in terms of G-mean (97.5%).

Other researchers depended on deep learning based on static analysis features in order to achieve more robust models for malware detection and family classification [30]. Convolutional neural network based on several datasets was used in study [31] to detect malware and they achieved an accuracy of 99%. Also, Authors [32] used convolutional neural network based on static analyses of the opcode sequence from the dex class in the APK file. They passed the result from CNN to a multi-layer perceptron (MLP), which is the final classification layer and the result is the prediction if that record is a malware. The authors try three different datasets; the highest accuracy was 98% but the accuracy mean was 88%. A study

[33] proposed an approach based on converting the features extracted from the APK file into images, and then used the convolutional neural network to classify the malicious files, and this approach obtained an accuracy of up to 97.7%.

The researchers [34] obtained a mean accuracy 83.56%, using a model that integrates Natural Language Processing (NLP), biLSTM, and Convolutional Neural Network (CNN). The model detects malware using opcode sequences, and by utilizing CNN and a lightweight classifier to examine common API call graph patterns that can result in data leaks, the researchers [35] were able to achieve an accuracy of 91.27 percent.

Researchers [36] used permissions and API function calls features in their proposed approach, which relies on deep web beliefs in discerning whether applications are malicious or not and achieved about 93.96% accuracy. Also, in research [37], a DBN-based deep learning model is used to train features, and SVM algorithm is used for classification, Accuracy ranged between 92 and 97.5% where models from static analysis and deep learning are combined, and using features extracted from API call blocks the researchers [38] proposed a model based on Deep Belief Network for android malware detection with accuracy of 96.66%.

Using static and dynamic features the researchers[39] implemented malware detection model using deep learning and ensemble classifiers, which classify the app for only two categories (benign & malicious). Mean, while the researchers [40] used features extracted from APK file in balanced datasets from 2015 & 2016 to build multimodal deep learning model for two classes (benign & malicious) malware detection and this approach achieved 98% accuracy.

Although graph-based deep learning is vulnerable to fraud by adding dummy relationships that prevent malware detection, the researchers [41] proposed a model based on graph neural networks (GNN) and generative adversarial network (GAN) and achieved an accuracy of nearly 98% in classifying Malware can be divided into two categories, benign and malicious.

Table 1: Summary of the researches that used static analysis using machine learning and deep learning, and the methodology used in each of them.

Static Analysis (deep learning)					
Paper	**Year**	**Methodology**	**Features**	**Result**	**Review**
[32]	2017	CNN	Opcode Sequence	Three different datasets High 98%, mean 88%	The Authors [32] used convolutional neural network based on static analyses of the opcode sequence from the dex class in the APK file, they passed the result from CNN to a multi-layer perceptron (MLP), which the final classification layer and the result is the prediction if that record is a malware. The authors try three different datasets; the highest accuracy was 98% but the accuracy mean was 88%.
[31]	2020	CNN	Opcode Sequence	Using multi–data set comparison 99%	A convolutional neural network based on several datasets was used in study [31] to detect malware and they achieved an accuracy of 99%.
[34]	2021	model was built that integrates Natural Language Processing (NLP), biLSTM, and Convolutional	Opcode Sequence	mean accuracy 83.56%	the researchers [34] obtained mean accuracy 83.56%, using a model was built that integrates Natural Language Processing

		Neural Network (CNN)			(NLP), biLSTM, and Convolutional Neural Network (CNN) to detect malware using opcode sequences.
[35]	2022	CNN and a lightweight classifier	API Calls	91.27%	By utilizing CNN and a lightweight classifier to examine common API call graph patterns that can result in data leaks, the researchers[35] were able to achieve an accuracy of 91.27 percent.
[39]	2022	Deep learning and ensemble classifiers	Static & dynamic features	98%	using static and dynamic features the researchers[39] implement malware detection model using deep learning and ensemble classifiers, which classify the app for only two categories (benign & malicious)
[41]	2022	graph neural networks (GNN) and generative adversarial network (GAN)	Permission and Intent & graph impeding	98%	Although graph-based deep learning is vulnerable to fraud by adding dummy relationships that prevent malware detection, the researchers[41] proposed a model based on graph neural networks

					(GNN) and generative adversarial network (GAN) and achieved an accuracy of nearly 98% in classifying Malware can be divided into two categories, benign and malicious.
[36]	2016	Deep Belief Network	Static Features of Permission & API calls	93.96%	researchers [36]used permissions and API function calls features in their proposed approach, which relies on deep web beliefs in discerning whether applications are malicious or not and achieved about 93.96% accuracy
[40]	2019	deep learning	Features from APK like manifest file, a dex file, and a .so file	98%	the researchers[40] use features extracted from APK file in balanced datasets from 2015 & 2016 to build multimodal deep learning model for two classes (benign & malicious) malware detection and this approach achieved 98% accuracy
[37]	2016	Deep Belief Network	features selected from Requested permissions-used	92 – 97.5%	In this research[37], a DBN-based deep

			permissions-sensitive API calls-Actions-app components		learning model is used to train features, and SVM algorithm is used for classification, Accuracy ranged between 92 and 97.5% where models from static analysis and deep learning are combined.
[33]	2018	CNN	Extract features from the transferred images	97.7%	A study [33] proposed an approach based on converting the features extracted from the A P K file into images, and then used the convolutional neural network to classify the malicious files, and this approach obtained an accuracy of up to 97.7%.
[42] compare	2019	DAE+CNN	Permissions-requested permissions-filtered intents-restricted API calls-hardware features-code related features-suspicious API calls	99.82%	The researchers[42] proposed a model based on the combination of deep autoencoder (DAE) for dimensionality reduction and a convolutional neural network (CNN) for app classification, and this approach achieved an accuracy of 99.82%.
[38]	2016	DBN	Static Features from API call blocks	96.66 %	using features extracted from API call blocks the

					researchers[38] proposed a model based on Deep Belief Network for android malware detection with accuracy of 96.66%
Static Analysis (Classical Machine Learning)					
[26]	2022	basic machine learning models	APK files after converting them into audio files	97.9%.	The researchers[26] proposed a model based on the use of basic machine learning models such as (Random Forest, Decision Tree, Kneighbors, etc.) and training them with APK files after converting them into audio files. This approach achieved an average accuracy among the different models of up to 97.9%.
[43]	2021	basic machine learning models	permissions and API calls	97%	The research[43] employs permissions and API calls and a genetic algorithm for features selection. The Features are then fed to several machine learning (ML) algorithms, including J48, decision tree, random forest, and naïve Bayes. The highest accuracy achieved was around 97%.

[44]	2021	Try basic machine learning models an SVM achieved the best performance	APK files for Ransomware and benign apps	G-mean (97.5%) **Accuracy is not indicated**	Some research has focused on classifying one type of malware as the approach proposed in the paper [44]to detect Ransomware, their approach was based on one of the powerful machine learning models (SVM) using swarm algorithm to tune model hyperparameters and they were able to achieve high performance in terms of G-mean (97.5%)
[27]	2017	KNN, randomforest, and J48	Features selected from decompress APK files to *AndroidManifest.xml* and *dex* files	99.7%	The researchers[27] used an approach based on three parallel machine learning models (KNN, random forest, and J48) named Mlifdect, and they obtained an accuracy of up to 99.7% in distinguishing between benign and malicious Android apps.
[28]	2015	SVM, randomforest, and J48	APIs Class feature	92.4%	[28] Three machine learning models (SVM, J48 and Random Forest) were used in to detect malicious applications, and the random forest achieved the

					highest accuracy, reaching 92.4%.
[23]	2015	Basic machine learning	Manifest file	93.9%	by combining static analysis using Android Asset Packaging Tool to extract features from Manifest file and machine learning model, the study [23] achieve 93.90% accuracy in malware prediction in system named DREBIN, which was trained offline and then transferred to smartphones for direct predicting.
[25]	2013	Basic machine learning	some manifest file (permissions and intent filters	90%	a lightweight approach realized in paper [25] , it also combines static analysis for some manifest file (permissions and intent filters) to train basic machine learning for malware detection with total accuracy 90%.
[24]	2012	One- Class Support Vector Machine	manifest files	Low accuracy Recall 85% F1-score 30%	the paper[24] demonstrate a technique based on machine learning (One-Class Support Vector Machine) using manifest file for offline training, although the cost of the analysis was low, the accuracy was also low.

[29]	2020	Basic machine learning (SVC, LR, RF, KNN)	features from API Calls	98.87%	using Non-Negative Matrix Factorization (NMF) for reducing the dimensionality of features from API Calls to train basic machine learning models, the researchers[29] found that K Nearest Neighbors (KNN) classifier achieved the highest accuracy 98.87% in this approach.

2.5.2 Dynamic analyses

Unlike static analysis, dynamic analysis gives an insight into the behavior of Android applications at runtime [45], by observing application behavior to detect malicious applications by performing actions on applications using a virtual device or a real device [22]

Although the output is less abstract than static analysis, the main advantage of this type of analysis is detecting dynamic code load detection and knowing the behavior of the application during runtime. The code paths that are executed are a subset of each path that runs and can Access it and every event that can occur must be activated to monitor any harmful behavior of the application [46].

There are many drawbacks in this type of analysis due to the overhead and large resources required to perform it compared to static analysis. In addition, there is a possibility that the malicious part of the application will not run during its analysis at runtime, as recently malware

is trying to identify this. Type of analysis and avoid running malicious code and therefore this analysis may be vulnerable to evasion [47].

To overcome the shortcomings of static analysis, the researchers tried to use basic machine learning models based on dynamic analysis, so we find that the researchers [48] proposed ServiceMonitor that Detects mobile malware dynamically. This approach uses Markov chain model to find out how system services access resources and build features to identify the application behavior with 96% accuracy using random forest classifier. Also [49] [50] relies on Android malwares behavior like CPU, battery, memory and network traffic, where an open-source tool named "MIET" was the proposed approach by [49] , this tool collects Android malwares behavior data like CPU, battery, memory and network traffic. The data was used in machine learning detection models (KNN, SVM, Random Forest, AdaBoost and Naïve Bayes) and achieved 85.4% accuracy.

Also, the authors in [50] proposed C4.5 algorithm which is an optimized decision tree [51] to analyze android application network traffic and identify android malwares. The model was built based on Drebin dataset [52] and achieved high accuracy of 97.89%. The study [53] used random forest classifier on features extracted in run time of android applications like features in CPU, battery, memory, network traffic and permissions. With good tuning for the model hyperparameters, their model achieved over 99 percent in general accuracy.

In addition, some researchers depend on dynamic permissions features and base machine learning. For instance, the researchers[54] extract 123 dynamic permissions features and evaluate base machine learning models including Naïve Bayes (NB), Decision Tree (J48), Random Forest (RF), Simple Logistic (SL), and k-star using it and they find out that SL got the highest accuracy of 99.7% among classifiers in detect android malwares.

The paper [55] proposed a dynamic approach using emulator to run Android application and collect features of malware in runtime to fed base machine learning models for malware detection and Family Classification, where the KNN in malware detection and the DT in family Classification and accuracy rates was 85% and 72% respectively.

Few researchers depended on deep learning based on dynamic analysis features in order to achieve more robust models for malware detection and family classification. For instance, using static and dynamic features the researchers[39] implement malware detection model using deep learning and ensemble classifiers, which classify the app for only two categories (benign & malicious). And Component Traversal is a dynamic analysis method proposed by [56], which based on extract features Linux kernel system calls from android applications and build a directed graphs to fed it into deep learning framework to classify it into benign and malware categories with 93.6% accuracy.

Table 2: Summary of the researches that used dynamic analysis using machine learning and deep learning, and the methodology used in each of them.

Dynamic Analysis (deep learning)					
[39]	2022	Deep learning and ensemble classifiers	Static & dynamic features	86%	using static and dynamic features the researchers[39] implement malware detection model using deep learning and ensemble classifiers, which classify the app for only two categories (benign & malicious)
[56]	2017	ANN	system calls	93.6%	Component Traversal is a dynamic analysis method proposed by [56], which based on extract features Linux kernel system calls from android applications and build a directed graphs to fed it into deep

					learning framework to classify it into benign and malware categories with 93.6% accuracy.
Dynamic Analysis (Classical Machine Learning)					
[49]	2018	KNN, SVM, Random Forest, AdaBoost and Naïve Bayes	Android malwares behavior data like CPU, battery, memory and network traffic	85.4%	An open-source tool named "MIET" was the proposed approach by [49] , this tool collects Android malwares behavior data like CPU, battery, memory and network traffic. The data was used in machine learning detection models (KNN, SVM, Random Forest, AdaBoost and Naïve Bayes) and achieved 85.4% accuracy
[50]	2019	C4.5 algorithm	android application network traffic	97.89%	another approach in paper [50] was proposed using C4.5 algorithm which is an optimized decision tree[51] ,to analyze android application network traffic to identify android malwares. The model was built based on Drebin dataset[52] and achieved high accuracy of 97.89%
[48]	2017	Markov chain, random forest	device performance such as CPU and Memory behavior	96%	Researchers [48] proposed ServiceMonitor that Detects mobile malware dynamically. This approach uses Markov chain model to find out how system services access resources and build features to identify the application behavior with 96% accuracy using random forest classifier.
[54]	2017	Naïve Bayes, Decision Tree	dynamic permissions	99.7	the researchers[54] extract 123 dynamic permissions features and

		(J48), RF, Simple Logistic			evaluate base machine learning models including Naïve Bayes (NB), Decision Tree (J48), Random Forest (RF), Simple Logistic (SL), and k-star using it and they find out that SL got the highest accuracy of 99.7 among classifiers in detect android malwares
[55]	2022	Base machine learning models KNN & DT	dynamic permissions	85 and 72% respectively.	The paper [55] proposed a dynamic approach using emulator to run Android application and collect features of malware in runtime to fed base machine learning models for malware detection and Family Classification, where KNN in malware detection and DT in Family Classification and accuracy rates was 85 and 72% respectively.
[53]	2013	random forest	dynamic features	99%	the study [53] used random forest classifier on features extracted in run time of android applications like features in CPU, battery, memory, network traffic and permissions. With good tuning for model hyperparameters their model achieved over 99 percent in general accuracy

2.5.3 Hybrid analyses

The hybrid analysis method combines the advantages of static analysis and dynamic analysis in order to cover the shortcomings in the two methods. The static analysis cannot detect

complex codes, and malicious applications have developed themselves recently, as they can discover the default environment used by dynamic analysis to find malwares and hide its malicious properties about it.

The use of hybrid forensics will improve the efficiency and accuracy of forensics and increase the ability to identify malicious applications [57], [58], and it can detect even the most complex malware threats, but it is a costly and time-consuming approach[15].

Researchers tried to use basic learning models based on hybrid analysis. As researchers[59] proposed hybrid model combines between malware genome and Drebin for static analysis and CICMalDroid dataset for dynamic analysis to fed machine learning model (gradient boosting (XGB)), which achieved 99% detection accuracy. While the researchers [60] proposed a hybrid malware detection mechanism “TAN” (Tree Augmented naïve Bayes) based on static and dynamic features (API calls, permissions and system calls). Their method was effective with 97% accuracy, but it takes a long time.

The paper [61] proposed an Android malware detection engine named ``DroidBox'' based on hybrid analysis technique to extract features from each app, which fall under one of three types: required permissions, sensitive APIs, and dynamic behaviors and employ deep belief network for malware classification with high accuracy of 96.76%.

Also, the researchers [62] used the Hybrid technique for malware detection, which mixes static and dynamic features and fed DNN with different features separately. Their results prove that their model delivered high accuracy.

The study [63] suggested a hybrid method for detecting malware variants that combines many sorts of characteristics (static & dynamic) and utilize deep learning models for android malware detection and family classification, which achieved 95% and 90%, respectively.

Table 3: Summary of the researches that used hybrid analysis using machine learning and deep learning, and the methodology used in each of them.

Hybrid Analysis (deep learning)					
[61]	2016	DBN	permissions, sensitive APIs, and dynamic behaviors	96.76%.	paper [61] proposed an Android malware detection engine named ``DroidBox'' based on hybrid analysis technique to extract features from each app, which fall under one of three types: required permissions, sensitive APIs, and dynamic behaviors and employ deep belief network for malware classification with high accuracy of 96.76%.
[62]	2020	DNN	Static feature (API Calls) & dynamic features (Manifest Permission, System Command and Intent-Filter)	96.1 for dynamic 88.9 for static	researchers [62] used the Hybrid technique for malware detection, which mixes static and dynamic features and fed DNN with different features separately. Their results prove that their model delivered high accuracy
[63] Compare	2019	CNN	API calls, opcodes	95% for malware detection and 90% for family classification	Study [63] suggested a hybrid method for detecting malware variants that combines many sorts of characteristics (static & dynamic) and utilize deep

					learning models for android malware detection and family classification, which achieved 95% and 90%, respectively.
Hybrid Analysis (Classical Machine Learning)					
[59]	2020	gradient boosting (XGB)	Dynamic and static features, 311 application samples (165 benign & 146 malware)	99%	Researchers[59] proposed hybrid model combine between malware genome and Drebin for static analysis and CICMalDroid dataset for dynamic analysis to fed machine learning model (gradient boosting (XGB)), which achieved 99% detection accuracy.
[60]	2020	" (Tree Augmented naïve Bayes)	static and dynamic features (API calls, permissions and system calls).	97%	the researchers[60] proposed a hybrid malware detection mechanism "TAN" (Tree Augmented naïve Bayes) based on static and dynamic features (API calls, permissions and system calls). Their method was effective with 97% accuracy, but it takes a long time

2.6 limitations

The aforementioned research has shortcomings. For example, we find that the data set that was used in a number of the related studies is considered old 2010-2015 and was extracted from

old Android versions and is no longer used now, in [23] [24] [25] [28] [36], and it is strange that some of the new research came back and used it again such as [43] in 2021.

An additional limitation is that prior research has only divided data into two categories: malicious and benign. While this has helped in detecting malware samples, it has not helped in correctly categorizing them such as [23][25] [37] [41] [54], which in turn has reduced the likelihood that appropriate precautions will be taken to protect Android devices.

Many of the methods that were used had low accuracy (below 90%), especially in the classification of malicious application, such as in [24] [34] [39] [49] [55] [62].

Some research has relied on one type of feature, which is considered insufficient to classify malware, such as [32] [31] [34] which relied on Opcode Sequence, or [35] which relied only on API Calls. The study [60] achieved almost high accuracy results, but the proposed method consumes much time and resources.

Chapter 3

Methodology

3.1 Overview

This chapter introduces the methodology that will be used to achieve the goal of this thesis. The chapter explains the data set that is used in the research and the preprocessing steps in terms of data cleaning, data integration, and data reduction.

Also, this chapter includes the process of converting the data after reducing it to images using IGTD, and then we reach the final stage, which is the using ensemble of CNNs in order to classify the data. **Figure 1** shows the mechanism used in the research, and each step will be explained extensively in this chapter.

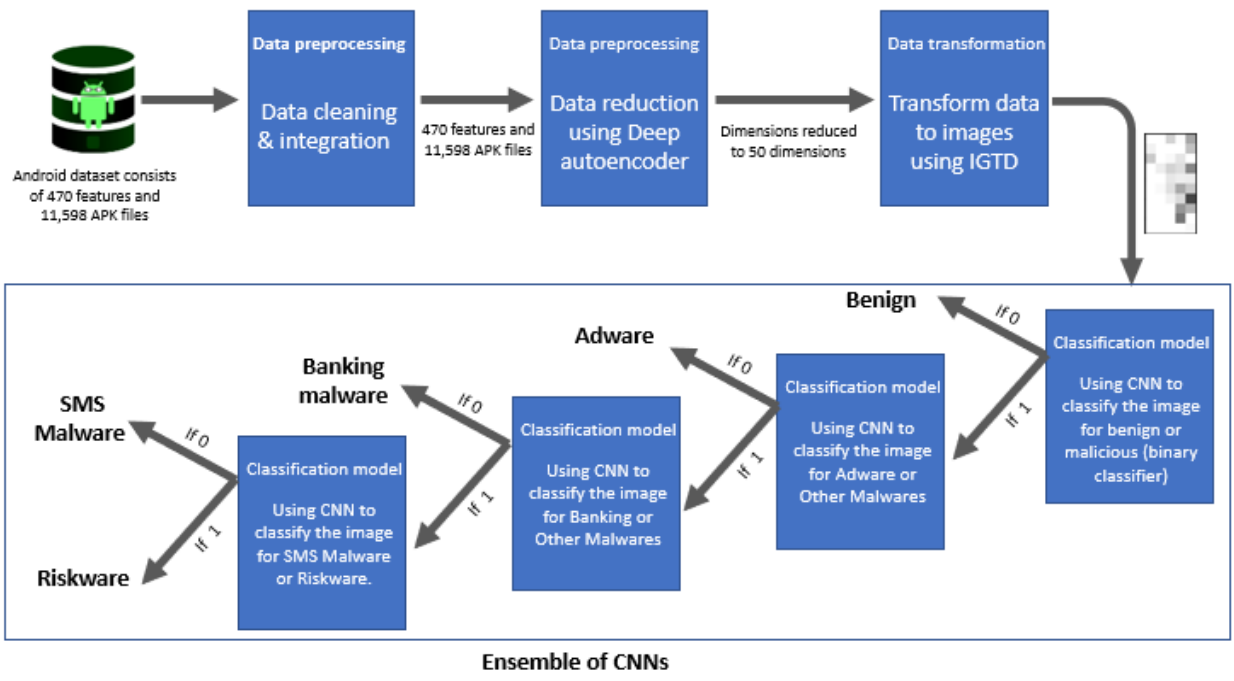


Figure 1: The main structure for the methodology

3.2 Dataset

This work uses public real data [10] generated from high reputation lab. The data were collected from December 2017 to December 2018. With the help of the VirusTotal service, the Contagio security blog, Advanced Micro Devices, MalDozer, and other datasets utilized by recent research contributions, the researchers were able to compile a collection of more than 17,341 Android samples. The researchers analyzed the collected data dynamically using CopperDroid, a VMI-based dynamic analysis system. The final remaining successful 11598 Android samples were published.

Specifically, we make use of one of the published csv files that includes 470 extracted features for 11,598 APK files, such as the frequency of system calls, bindings, and composite behaviors. The dataset is intentionally spanning between five distinct categories: Adware, Banking malware, SMS malware, Riskware, and Benign as **figure 2** shows.

The dataset is designed to be used for research purposes, particularly in the development of machine learning models for Android malware detection. It is released under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license, which allows for the dataset to be used and shared for non-commercial purposes as long as proper attribution is given and any derivative works are released under the same license.

Overall, the CICMALDROID 2020 dataset provides a valuable resource for researchers and practitioners in the field of cybersecurity, specifically in the development of machine learning models for Android malware detection and classification.

Figure 2: The total number of records in the dataset and number of records in each class.

3.3 Data Preprocessing

The dataset should be examined before starting to try the proposed approach, because it may contain problems that may affect accuracy in detecting and classifying malware, such as missing values, redundancy, and uneven distribution [64]. With these problems still in the dataset, machine learning cannot be relied upon to give us a reliable, high-accuracy model for malware detection and classification. So, in order to get rid of these problems, we will follow these steps in preparing the dataset:

3.3.1 Data cleaning: In this step, the data set will be checked if it contains missing values and deal with it either by repairing or deleting it.

3.3.2 Data integration: It means putting together data from different sources, getting rid of duplicate data, and doing correlation analysis.

3.3.3 Data reduction: In this step we aim to get much smaller representation than the size of the original data, and we will use an autoencoder to achieve this goal.

Auto encoder:

The auto encoder neural network is an algorithm for unsupervised learning that uses backpropagation to use the same input as a target output [65].

Auto encoder has a symmetric structure consists of two parts: the encoder and the decoder. The encoder part converts high-dimensional data to low-dimensional data, while the decoder part converts low-dimensional data to high-dimensional data. **Figure 3** shows the structure of an autoencoder used in this thesis.

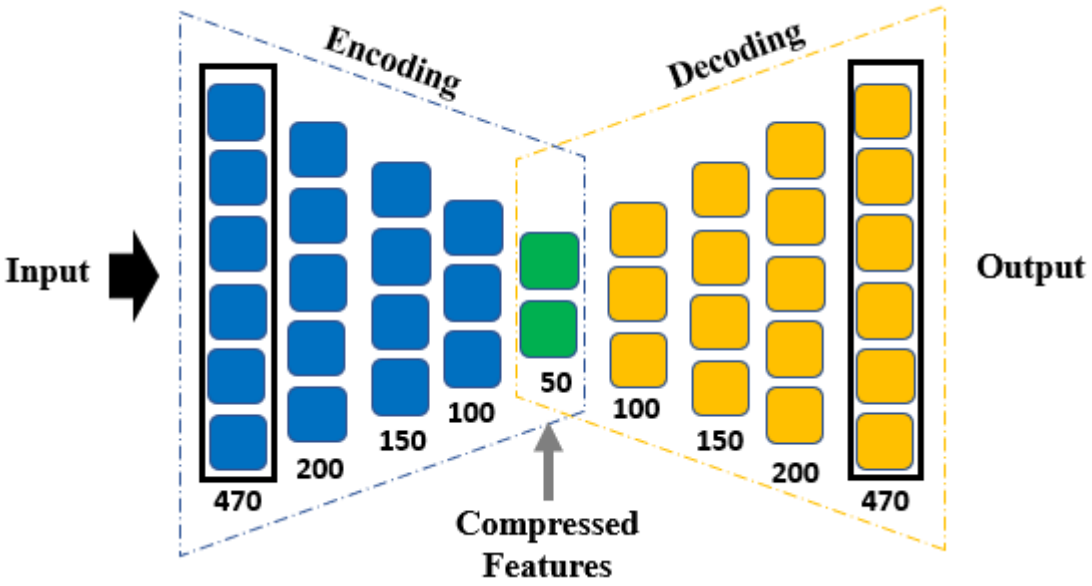


Figure 3: Structure of autoencoder with input and output size 470 and 7 hidden layers including the compressed 50 features

Training the auto encoder model is a bit tricky. During the training phase, the whole encoder-decoder section is trained against the same input as an output of decoder [64]. Features will be compressed in the intermediate layer as we go through the convergent and divergent layers in order to obtain the desired output.

After the encoder has been trained enough, as measured by a decrease in error values throughout a sufficient number of repetitions, the trained encoder section is used to generate the latent features for visualization and utilizing in the next stage of the model [66].

3.3.4 Data transformation:

This part is about converting or structuring the data into a more convenient form that can be utilized for further processing. For this section, we use Image Generator for Tabular Data (**IGTD**) algorithm[67]. This algorithm transforms tabular data into pictures by allocating features to pixel coordinates such that comparable features are near together.

The method seeks an optimal assignment by minimizing the difference between the ranking of distances between features and the ranking of distances between their assigned picture pixels, to overcome the hurdle of assuming no spatial relationship between features in tabular data. Thus, we can overcome the hurdle that assumes that there is no spatial relationship between the features in the tabulated data.

let's point to our dataset with size M by N, using symbol X, with x_i point to the i^{th} row and x_j point to the feature. our goal is to transform each sample of X into $N_r * N_c$ image (2D Array and $N_r * N_{c=} N$)). To do that, we calculate the pairwise distance between features using distance measures like Euclidean distance and ranked the pairwise distances ascendingly. So, we gave the small distance a small rank & the large distance a large rank and put the diagonal equal zeros to create a symmetric rank matrix R.

Then, we calculate distance between pairs of pixels i and j using distance measures and rank these distances ascendingly to form a distance matrix called Q and set the diagonal to be zeros, so Q will be symmetric pixel rank matrix. The pixels in the image are concatenated row by row to form the order of pixels in Q.

In order to convert tabular data into images, each feature must be allocated a pixel location in the image, by assigning each ith feature (the i^{th} row and column) in R to i^{th} pixel in Q.

To overcome the difference between the two matrices we used an error function in equation (1) that measures the difference between $r_{i,j}$ and $q_{i,j}$, such as the absolute difference in equation (2) or the squared difference in equation (3).

$$\text{error}(Q, R) = \sum_{i=2}^{N} \sum_{j=1}^{i-1} \text{diff}\,(r_{i,j}, q_{i,j}) \qquad (1)$$

$$\text{absolute difference} = |r_{i,j} - q_{i,j}| \qquad (2)$$

$$\text{squared difference} = (r_{i,j} - q_{i,j})^2 \qquad (3)$$

So, we reorder the features synchronously by swap the positions of features to reduce the error iteratively by searching for suitable feature swaps [67].

3.4 ML model fitting and Evaluation

3.4.1 Introduction

A significant development has occurred in the field of machine learning after the emergence of biologically inspired computational models called the artificial neural network (ANN), and with the continuation of development in this field, the convolutional neural network (CNN) appeared, which relies on (ANN) in Its structure was created with the aim of finding difficult patterns in images.

Artificial neural network (ANN)

An artificial neural network (ANN) is a computational model that was inspired by the neural networks in the human brain. It consists of many interconnected computational nodes (neurons) [68].

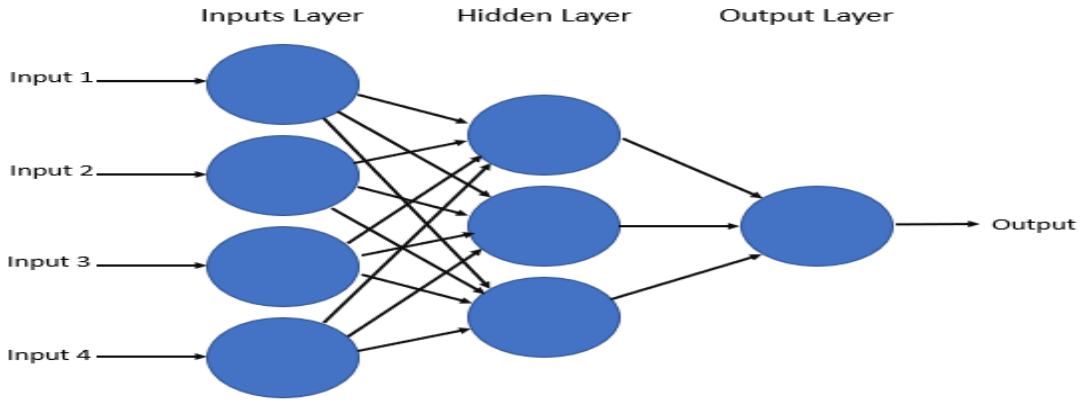


Figure 4: the basic structure of the forwarded artificial neural network

Figure 4 shows the basic form of the artificial neural network, which consists mainly of three layers (inputs, hidden and outputs). The input layer distributes the inputs to the hidden layer and finally to the output layer through the edges that connect the neurons in each layer. Where edges and neurons have a weight and an error value that is modulated during the learning process.

Convolutional neural network (CNN)

A convolutional neural network (CNN) is a sort of artificial neural network with a similar structure, consisting of neurons that strive for self-improvement via learning [69]. CNNs are especially good for recognizing objects, classes, and classes by identifying patterns in images [70].

The main significant distinction between CNNs and conventional ANNs is that CNNs are predominantly used in the area of image pattern recognition. This enables us to embed image-

specific characteristics into the network's architecture, making it more suited for image-centric tasks, while also lowering the number of parameters necessary to build up the model.

3.4.2 CNN Architecture

As mentioned earlier, the inputs to CNN are images, and the CNN structure is designed to best suit this type of data. The neurons that compose the layers of the CNN are organized in three dimensions, according to the spatial dimensionality of the input (height and breadth) and the depth. The depth is not the entire number of layers inside an ANN, but rather the third dimension of an activation volume [70]. Unlike conventional ANNS, the neurons in each layer will only connect to a tiny portion of the layer below.

CNNs consist of three distinct sorts of layers. These layers are convolutional, pooling, and fully-connected. When these layers are stacked, CNN architecture is generated, as shown in Figure 5, the structure of CNN consists of two convolutional layers, two max pooling layers, one flatten layer and final fully connected layers.

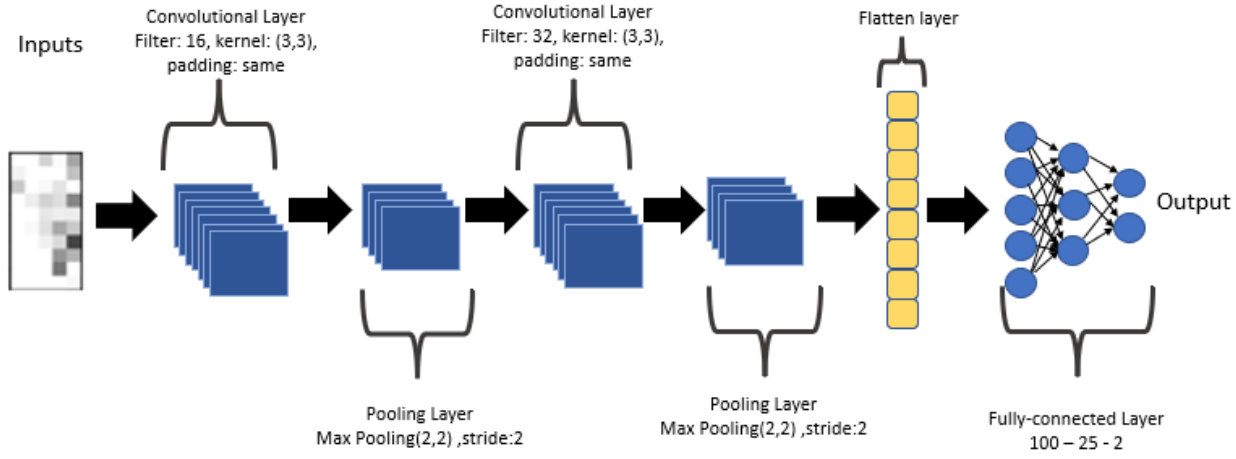


Figure 5: CNN structure of two convolutional layers, two max pooling layers, one flatten layer and final fully connected layers

The above example CNN's fundamental functioning can be split down into four important parts:

1. The **input layer** is at the beginning like other (ANN) and contains the values of the pixels in the image.
2. **The Convolutional layer** plays an essential role in CNN, as the parameters use a learnable kernel that contains filters [71], (Filters are explained later in this section).

   Mathematically, convolution is the sum of the products of two matrices. As we can see in Figure 6, supposing we have image X and filter Y, the output of the process of converting image X using filter Y is output Z.

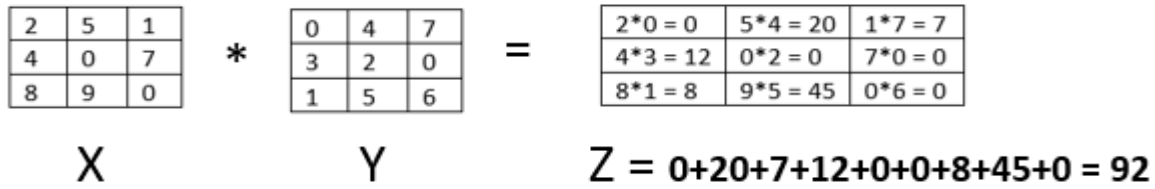


Figure 6: Convolve the image 'X' using filter 'Y' example

Filters are arrays with n dimensions and typically square or cubic matrices, which used to quantify how closely an input patch or area matches a feature is [72]. The filter functions as a single template or pattern that, when convolved over the input, identifies similarities between the stored template and various locations/regions in the input image and identify spatial patterns like edges by identifying variations in the image's intensity levels.

So, the values in the output matrix show the intensity change with respect to the input image's columns along the horizontal axis. For example, if the value of the output image is 0, this means that there is no change in the density between the columns in the input image, but if there is a numerical value for the output, this means that there is a change in the density between the columns as we see in Figure **7**.

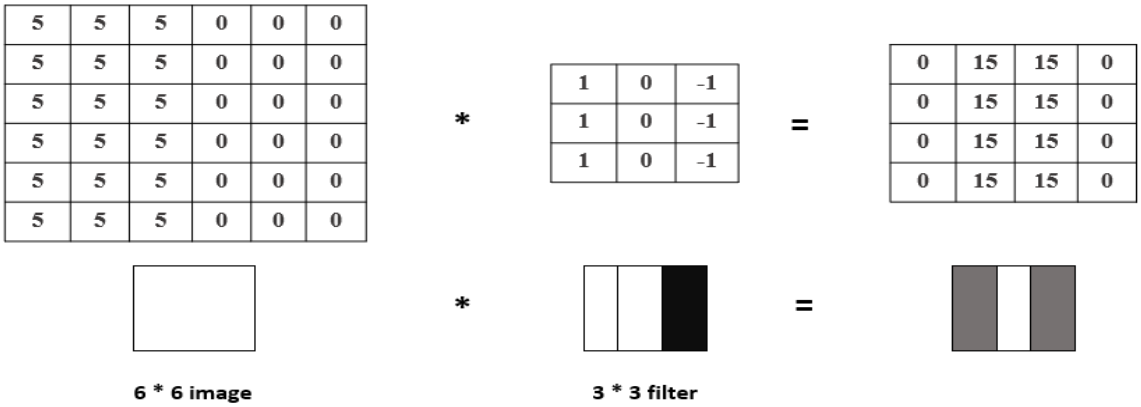


Figure 7: Vertical edge detection of 6*6 input image using 3*3 filter

To find out the size of convolved output, we make use the formula in equation (4), where **I** represent the input image size, **F** represent the filter size, which usually an odd size to get a central position.

$$\text{The size of convolved output} = \frac{(I - F) + 2P}{S + 1} \ldots\ldots (4)$$

P represent the padding size and we use padding, because every time we apply a convolution operator the input image shrinks[73]. And by continuing the convolution process, a lot of information will be lost because of image shrinking. Figure 8 shows the result after padding the image missioned in Figure 7.

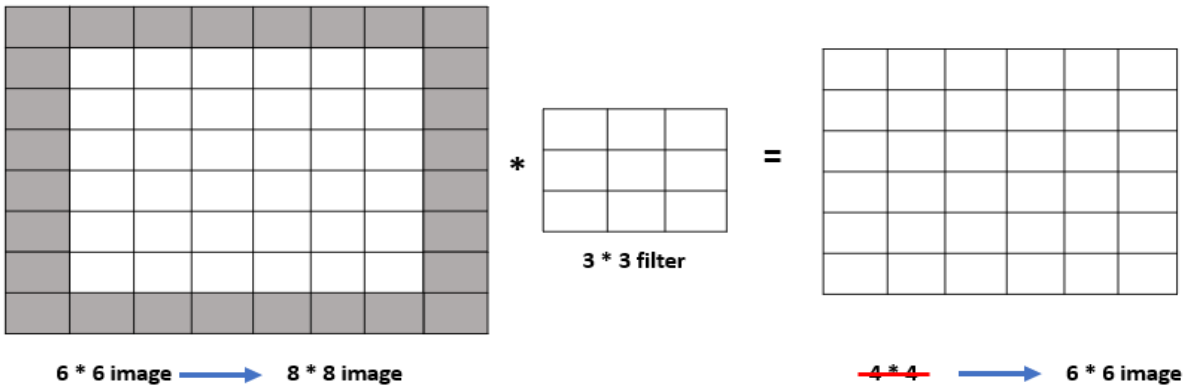


Figure 8: Padding the input 6*6 image to avoid shrinking in the output

There is another problem that will appear without padding, as during the convolution, the pixels in the corners and edges are deemed only once. Thus, it is used much less in the output. Thus, we have lost a lot of information in the edge of the image.

S refers to the stride, which describes the speed at which the filter traverses the pixels of the input image both horizontally and vertically during convolution [74]. If the result of the previous equation is not a whole number, the stride has been improperly adjusted, and the neurons will not fit neatly across the input.

3. Pooling Layer

Pooling layers try to progressively lower the dimensionality of the representation by determining whether or not a given image area has the desired feature, hence reducing the number of parameters and computing complexity of the model [75].

The pooling layer acts over every input activation map, scales its dimensionality, and collects aggregate data (min, max, average & global) [76].

The "max" and the "average" are the two most prominent aggregate functions used in pooling [77], where Max pooling means in a simple language if any of the patches say something firmly about the presence of a particular feature, then the pooling layer counts that feature as ‘detected’, while Average pooling means If one patch asserts something emphatically, while the others disagree, the average pooling determines the average. **Figure 9** shows an example that illustrates how the two methods work.

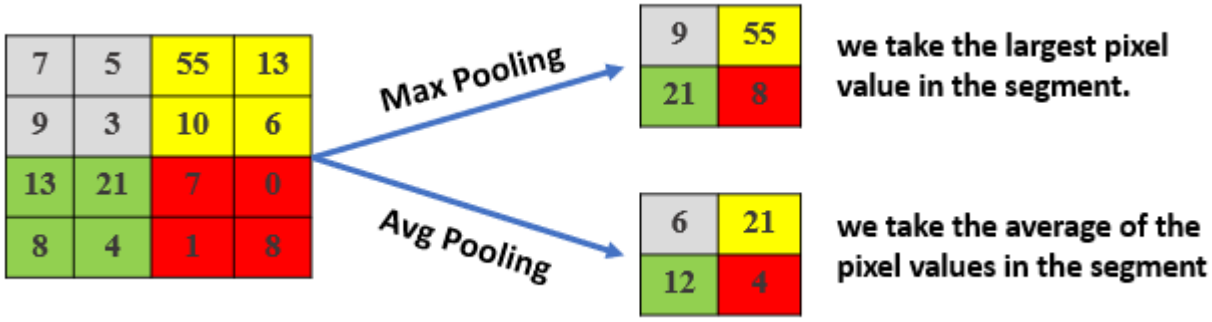


Figure 9: Difference between Max pooling and Average pooling.

4. Fully-connected layer

The fully-connected layer comprises neurons that are directly linked to the neurons in the two neighboring layers, but are not connected to any layers within those layers. This is comparable to how neurons are placed in conventional ANNs **Figure 4.**

3.4.3 VGG16 Algorithm

The VGG16 (Visual Geometry Group 16) algorithm is a deep learning architecture that was proposed by researchers from the Visual Geometry Group at the University of Oxford in 2014 [78]. It is a type of convolutional neural network (CNN) that has achieved state-of-the-art performance on various computer vision tasks, such as image classification, object detection, and segmentation. The VGG16 algorithm is widely used in both academia and industry due to its high accuracy.

The VGG16 architecture consists of 16 layers as **figure (10)** shows, including 13 convolutional layers, and 3 fully connected layers. The convolutional layers are divided into 5 blocks, where each block contains one or more convolutional layers followed by a max-pooling layer. The first two blocks have 2 convolutional layers each, while the last three blocks have 3 convolutional layers each. All the convolutional layers in the VGG16 architecture have a filter

size of 3x3, a stride of 1, and are padded with 1 pixel on both sides to maintain the spatial resolution of the input image.

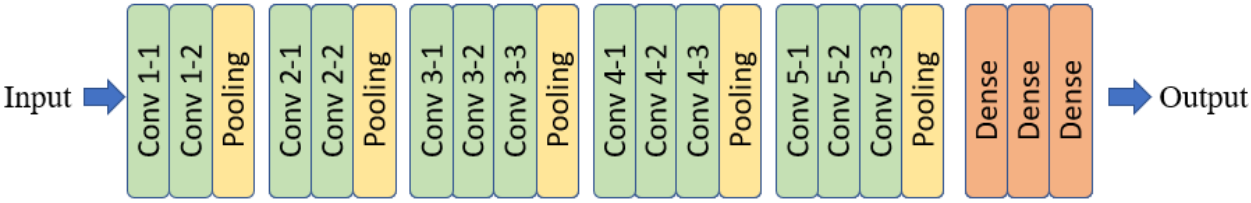


Figure 10: VGG16 algorithm structure

The fully connected layers in the VGG16 architecture have 4096 neurons each, followed by a final softmax layer with 1000 neurons, which corresponds to the number of classes in the ImageNet dataset, on which the algorithm was trained [78]. The VGG16 algorithm uses the cross-entropy loss function to optimize the model parameters during training.

The VGG16 architecture is characterized by its depth and the use of small filters, which allows it to learn more complex features from the input image. The architecture also uses a large number of parameters, with a total of about 138 million parameters, which makes it computationally expensive. However, the use of small filters allows the VGG16 architecture to have fewer parameters than other deep learning architectures, such as AlexNet [79], while achieving higher accuracy.

Compared to traditional CNNs, the VGG16 algorithm has several advantages. First, the use of small filters and a large number of layers allows it to learn more complex features from the input image, which can improve the accuracy of the model. Second, the VGG16 architecture is highly modular, which makes it easy to modify and adapt to different tasks. Finally, the VGG16 algorithm has achieved state-of-the-art performance on various computer vision tasks, which demonstrates its effectiveness [78]

However, the VGG16 architecture also has some disadvantages. First, it is computationally expensive due to the large number of parameters, which can make it difficult to train on low-end hardware. Second, the VGG16 algorithm requires a large amount of training data to achieve high accuracy, which can be a limitation in some applications. Finally, the VGG16 algorithm is highly sensitive to overfitting, which can occur when the model is trained on a small dataset [78]

3.4.4 Resnet Algorithm

The ResNet50 algorithm is a deep convolutional neural network architecture developed by Microsoft Research Asia in 2015. It is one of the most popular and widely used deep learning models for image classification tasks. The ResNet50 architecture consists of 50 layers, hence its name, and uses residual connections to mitigate the problem of vanishing gradients, which can occur in very deep neural networks. In this section, we will provide a full explanation of the ResNet50 algorithm, including its architecture, mathematical equations, and advantages and disadvantages.

The ResNet50 architecture, as shown in **figure (11),** is based on a series of residual blocks, which are composed of convolutional layers, batch normalization, and rectified linear units (ReLU) [80]. The residual block introduces a shortcut connection that bypasses one or more convolutional layers and adds the output of those layers to the input of the block. This residual connection allows the network to learn the residual mapping, or the difference between the input and output of the block, rather than the full mapping. This approach helps to alleviate the vanishing gradient problem and allows for deeper networks to be trained. The architecture also includes global average pooling and a fully connected layer for classification.

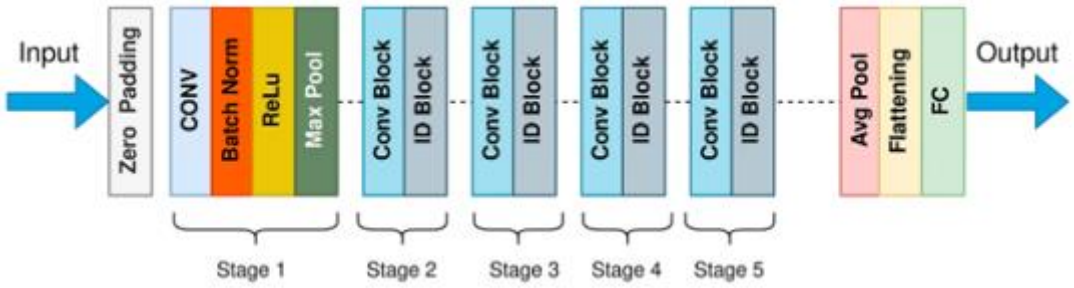


Figure 11: ResNet50 algorithm structure

One of the main advantages of the ResNet50 algorithm is its ability to train very deep neural networks. The residual connections enable the network to learn the residual mapping and bypass the vanishing gradient problem. This allows for the training of deeper and more accurate models. Another advantage is that ResNet50 has achieved state-of-the-art results on various benchmark datasets for image classification, such as ImageNet and CIFAR-10 [81].

However, the main disadvantage of the ResNet50 algorithm is its high computational cost and memory requirements. The deep architecture and large number of parameters require significant computational resources, which can limit its practical applications. Additionally, the use of residual connections can also make the network more difficult to interpret and visualize

Recent research has focused on improving the efficiency and effectiveness of the ResNet50 algorithm. For example, the ResNeXt architecture introduced by Facebook AI Research in 2017 uses a group convolutional layer instead of a regular convolutional layer, which can improve the model's accuracy and efficiency. Other research has explored the use of ResNet50 for transfer learning, object detection, and other computer vision tasks.

The ResNet50 algorithm is a powerful deep learning model for image classification tasks. Its use of residual connections enables the training of deeper and more accurate models, but its high computational cost and memory requirements can limit its practical applications. Ongoing

research is focused on improving the efficiency and effectiveness of the algorithm, and its use is likely to continue to grow in the field of computer vision [82].

3.4.5 Model Evaluation

To evaluate our model, we will use:

1.**Confusion matrix**, which is a summary method for model prediction result on a classification problem, which is in our case benign or malicious app. The matrix shows the actual result versus the prediction result of the model [83].

	Predicted: Yes	Predicted: No
Actual: Yes	TP	FN
Actual: No	FP	TN

Figure 10: confusion matrix

As **Figure 10** shows the confusion matrix has many terms, which we will explain using our data as an example.

a. **True positives (TPs):** True positives are cases when the model predict the app as benign and it is actually a benign app.
b. **True negatives (TNs)**: Cases when the model predict the app as malicious and it is actually a malicious app.
c. **False positives (FPs):** When the model predict the app as benign and it is actually a malicious app.
d. **False negatives (FNs):** When the model predict the app as malicious and it is actually a benign app.

e. **Precision (P)**: When yes is predicted, how often is it correct, and it can be calculated using equation (5).

f. **Recall (R)**: Among the actual yeses, what fraction was predicted as yes, and it can be calculated using equation (6).

$$Precision = \frac{TP}{TP+FP} \text{ ……..(5)}$$

$$Recall = \frac{TP}{TP+FN} \text{……(6)}$$

2. **Roc Curve**

A ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a binary classification model, which shows the relationship between the true positive rate (TPR) and false positive rate (FPR) at different classification thresholds. The TPR is the proportion of true positives (i.e., correctly predicted positive instances) among all actual positive instances, and it can be calculated using equation (7), while the FPR is the proportion of false positives (i.e., incorrectly predicted positive instances) among all actual negative instances, and it can be calculated using equation (8).

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP+FN} \text{……..(7)}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP+TN} \text{……..(8)}$$

In a ROC curve, the x-axis represents the FPR and the y-axis represents the TPR. The curve is created by plotting the TPR against the FPR at various threshold settings. The ROC curve can then be used to determine the optimal threshold for a given model by identifying the point on the curve that is closest to the top-left corner, which represents perfect classification.

One common metric derived from the ROC curve is the area under the curve (AUC), which ranges from 0.0 to 1.0 and provides a measure of the overall performance of the model. An

AUC value of 1.0 indicates perfect classification, while an AUC value of 0.5 indicates that the model's performance is no better than random guessing.

Chapter 4

Results and Discission

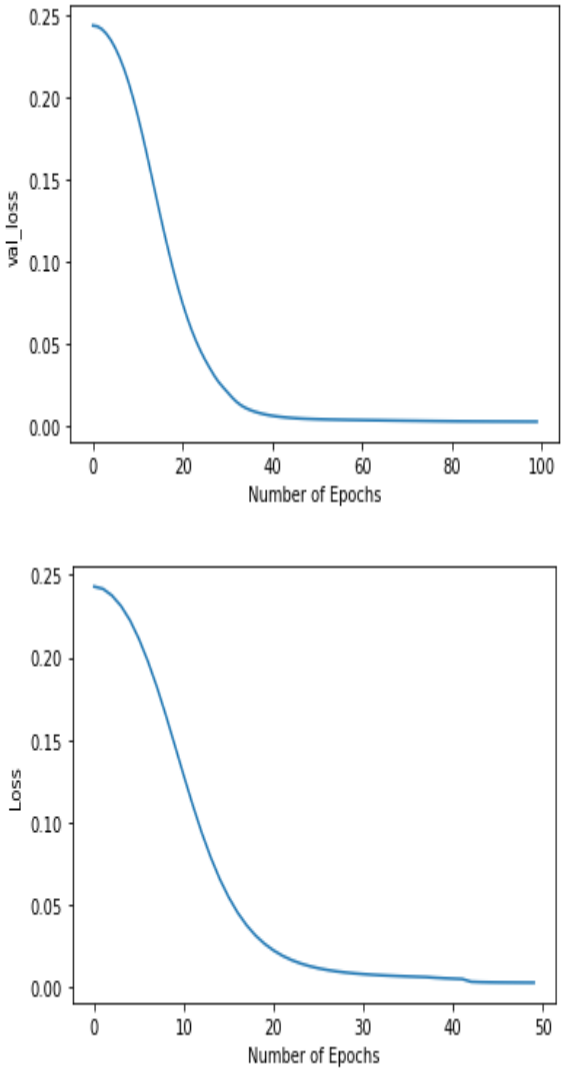
4.1 Data preprocessing results

4.1.1 Data cleaning: In this step, the data was checked for missing values and we found out that there are **no missing values.**

4.1.2 Data integration: as we mentioned before in chapter 3, the data was a single csv file that contains 471 features (including dependent target) and 11526 records divided into five classes. In this part, we check for duplicate records and drop **72 duplicated records.**

4.1.3 Data reduction: by using the structure in **figure 3,** we trained our autoencoder model to reduce the data dimensionality from 470 to 50 features.

As we can see in **Figures 11 (a, b)**, the model maintains its stability even with the increase in the number of epochs. The loss continues to decrease gradually and smoothly and the accuracy reached 97.57%.



Figures 11: (a) autoencoder loss function comparing with 100 epochs, (b) autoencoder loss function comparing with 50 epochs.

4.1.4 Data transformation: figure (12 a) shows Rank matrix of Euclidean distances between 50 features. The rank value is indicated by the grey level. The 50 features with the largest variations are included for calculating the matrix, **figure 12 b** shows the Rank matrix of Euclidean distances calculated between all pairs of pixels in a 5 by10 image based on their coordinates. To generate the order of pixels in the matrix, the pixels from the image are concatenated row by row.

Figure 12 c shows the distance rank matrix for features after optimization and rearrangement, while **figure 12 d** shows the decrease in the error value in the optimization process compared to the number of iterations.

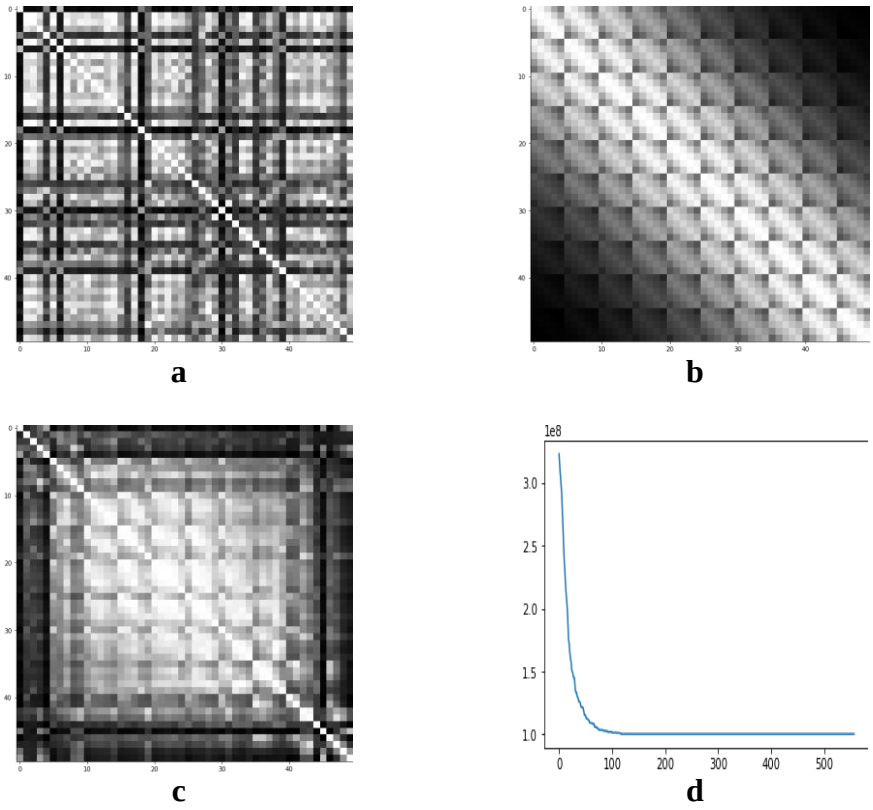


Figure 12: (a) Rank matrix of Euclidean distances between 50 features. (b) Rank matrix of Euclidean distances between all possible pixel pairs in a 5x10 image. (c) Distance rank matrix for features after optimization and rearrangement. (d) decrease in the error value in the optimization process compared to the number of iterations.

Figure 13 shows an example of the resulting images from this part, and we notice a difference in the intensity of the gray scale according to the value of the feature, and the features have been arranged according to their proximity depending on the Euclidean distance between them and to obtain the lowest value of error**.**

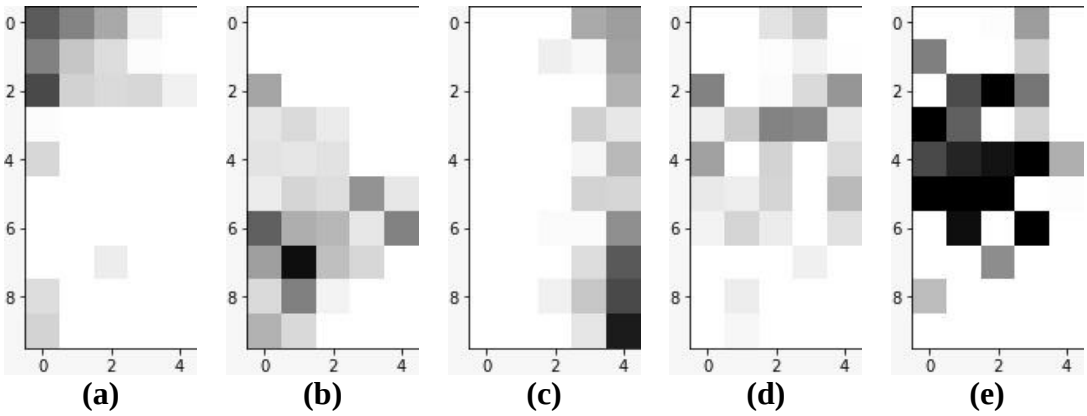


Figure 13: (a) example of benign image, (b) example of Adware image, (c) example of Banking malware image, (d) example of SMS Malwares image, (e) example of Riskware image.

4.2 Model evaluation (CNN, VGG16, Resnet50):

As table 5 shows, our model achieved high accuracy, where the accuracy was 94.38% for binary classification (benign and malware), and 95.83% as an average accuracy for our hybrid model for multiclassification.

Also, the average precision and the average recall were high with values 94.75% and 98%, respectively. This means that among the actual and predicted target values, our model was able to predict it correctly with high percentage accuracy, comparing with other algorithms like VGG16 and ResNet50 which achieved lower accuracy with values 88.32% and 82.45%, respectively, and the reason is that these algorithms have big structure and contains many parameters to learn and this won't be possible with small size dataset which causes underfitting situation.

The training time was low for each model and even the total time was low 1150 sec (almost 20 min) comparing with VGG16 with total time 2319 sec and ResNet50 with total time 2338 sec, using a device have the characteristics shown in table 4.

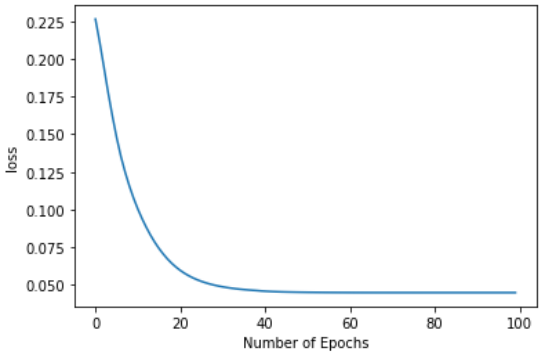
Table 4: characteristics of the device used to train the model.

Characteristic	Value
Type	Hp workstation
OS	windows 10 pro-64
CPU	Intel(R) Core i5-6500 CPU at 3.20 GHz
RAM	24 GB
Storage	256 SSD

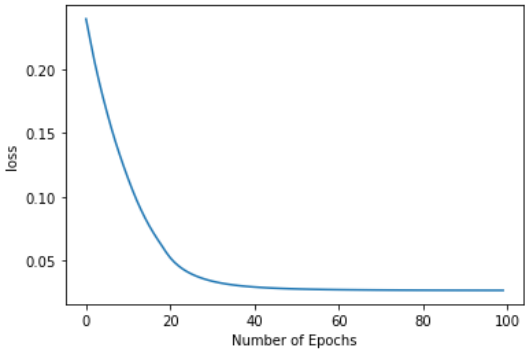
Table 5: The training and testing size of each model and classification report details

Model	Training	Testing	Precision	Recall	Accuracy	F1 score	Training Time
(Benign, Malware)	9220	2306	90	98	94.38	94	447 sec
(Adware, Other Malwares)	1714	429	95	99	94.64	97	69 sec
(Banking, Other Malwares)	3202	801	97	98	97.13	98	84 sec
(SMS Malwares, Other Malwares)	2107	527	97	99	97.15	99	73 sec
average			**94.75**	**98**	**95.83**	**97**	
Total training time							**1150 sec**
VGG16 (Benign, Malware)	9220	2360	89	87	88.32	88	2319
Resnet50 (Benign, Malware)	9220	2360	84	78	82.45	81	2338

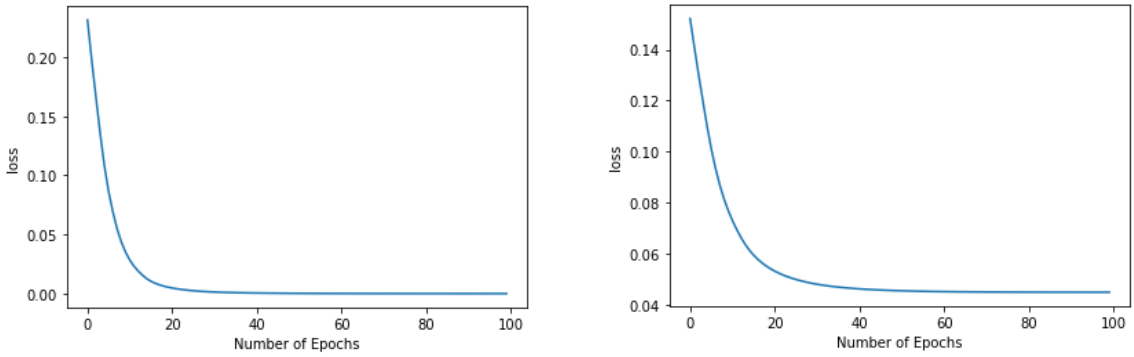
Figure 14 shows, the loss in the four models continues to decrease gradually and smoothly, which shows the stability of the models and non-existing of overfitting, while figure 18 shows the ROC curve with the performance of a classification models(CNN, VGG16, and ResNet50) at all classification thresholds and the CNN achieved the highest performance of 94.



a



b



c **d**

Figure 14: (a) loss curve of the CNN classifies for (Benign, Malware), (b) loss curve of the CNN classifies for (Adware, Other Malwares) (c) loss curve of the CNN classifies for Banking, Other Malwares) (d) loss curve of the CNN classifies for (SMS Malwares, Other Malwares).

During our attempt to train the model to reach the best structure for classifying malicious software, we used more than one structure as shown in Table 6, and we found that with the increase in the number of convolutional layers, number of filters and the increase of layers in the fully connected layer all lead to an increase in training time and it also yielded an inverse result in terms of accuracy, as the number of layers increases, the accuracy decreases because we need a larger volume of data to teach the model.

Table 6: Different approaches of CNN classification model, accuracy and training time for each approach

Approach	Accuracy	Training time(sec)
Original model: Conv 16 – Max pooling – Conv 32 -Max pooling -Flatten layer – Fully connected layer (100 – 25 – 2)	**94.38%**	**447**
Conv 8 – Max pooling – Conv 16 -Max pooling - Conv 32 -Max pooling - Flatten layer – Fully connected layer (100 – 25 – 2)	**92.97%**	**756**
Conv 8 – Max pooling – Conv 16 -Max pooling - Conv 32 -Max pooling - Flatten layer – Fully connected layer (400 – 200 - 100 – 25 – 2)	**92.84%**	**886**
Conv 32 – Max pooling – Conv 32 -Max pooling - Conv 64 -Max pooling - Flatten layer – Fully connected layer (100 – 25 – 2)	**89.33%**	**1341**
Conv 32 – Max pooling – Conv 32 -Max pooling - Conv 64 -Max pooling - Flatten layer – Fully connected layer (400 – 200 - 100 – 25 – 2)	**89.1%**	**1507**

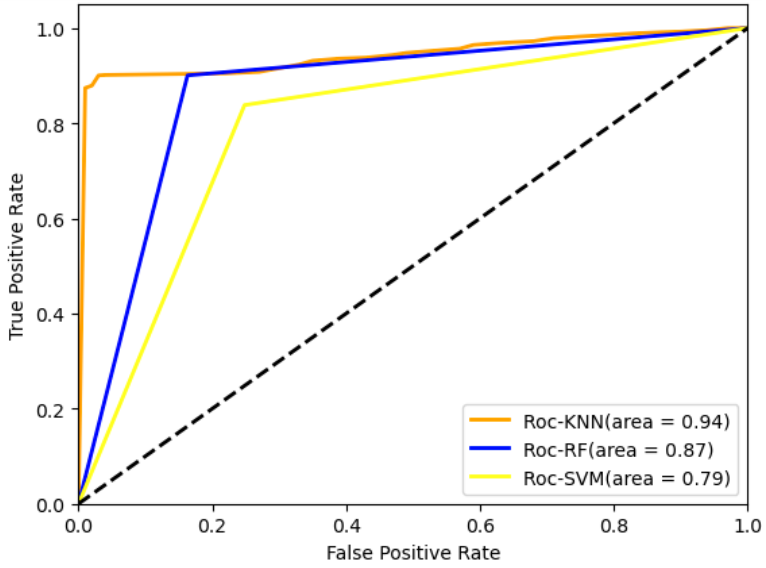


Figure 17: ROC curve for CNN, VGG16 and ResNet50 algorithms

4.3 Discussion

When comparing the methodology used in this thesis, we find that there is no other research used the same methodology on the topic of Android malware. The part related to the use of IGTD to convert data into images and then classify it using CNN was used for the first time in the classification of genes [67] As for the part of reducing the dimensions using the autoencoder and then classifying the data using CNN, it was used by [42] and got a high accuracy of up to 99.82%, but only for the binary classification and the multiple classification was not mentioned.

There are many researchers who used CNN in static analysis of Android malware, such as [32], and it combined CNN with MLB and achieved 98% accuracy on binary classification, and did not address multiple classification either.

There are also [33] where they converted data extracted from the APK file into color images and classified them binary using CNN and obtained an accuracy of 97.7 and did not apply the multiple classification.

By integrating Natural Language Processing (NLP), biLSTM, and Convolutional Neural Network (CNN), the researchers [27] obtained mean accuracy 83.56% in malware distinction.

We noticed that the researchers [42] achieved high rates in the classification of the data binary into harmful and benign, but they did not address the multiple classification. How much there are researchers such as [27] have achieved lower rates in accuracy than the methodology used in this thesis, as it is according to the best of our knowledge None of the researchers addressed this methodology in classifying Android malware.

Chapter 5

Conclusion

Android is the most popular platform for those who want a high-performance operating system that can operate on today's most popular mobile devices. According to [2], Android dominated the market in the fourth quarter of 2020 with 72.22% of the worldwide market share, and despite a fall in market share towards the end of 2022, it maintained a 71.8 percent dominance. Android applications are accessible from potentially malicious third parties besides the official Android Market due to the current evolutionary process, wide distribution, and open-source nature of Android, so this will bring malware designed with adaptive and success-based learning to improve the efficacy of attacks. As a result, developing efficient tools to assess and detect malware application risks is critical.

Based on the aforementioned problem, this thesis proposes a hybrid model that integrates dimensionality reduction, data transformation, and the use of ensemble of CNN for binary classification and multiple classification. The model achieved a high accuracy of 94.38% in binary classification (benign or malicious) and multiple classification. (Benign, Adware, Banking malware, SMS Malwares and riskware) achieved an accuracy of 95.83%.

In future work, we will extract useful features to generate a database of Android malware using dynamic analysis and investigate more successful algorithms to evaluate and identify more complex Android malware.

References

[1] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm, "KronoDroid: Time-based hybrid-featured dataset for effective android malware detection and characterization," *Comput Secur*, vol. 110, Nov. 2021, doi: 10.1016/j.cose.2021.102399.

[2] "Global mobile OS market share 2022 | Statista." https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (accessed Jan. 25, 2023).

[3] D. Arp *et al.*, "Dos and Don'ts of Machine Learning in Computer Security."

[4] P. Faruki *et al.*, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 2, pp. 998–1022, Apr. 2015, doi: 10.1109/COMST.2014.2386139.

[5] D. Bhusal and N. Rastogi, "Adversarial Patterns: Building Robust Android Malware Classifiers," Mar. 2022, [Online]. Available: http://arxiv.org/abs/2203.02121

[6] "Mobile malware evolution 2020 | Securelist." https://securelist.com/mobile-malware-evolution-2020/101029/ (accessed Jan. 25, 2023).

[7] U. S. Shanthamallu and A. Spanias, "Machine and Deep Learning Applications," in *Machine and Deep Learning Algorithms and Applications*, U. S. Shanthamallu and A. Spanias, Eds., Cham: Springer International Publishing, 2022, pp. 59–72. doi: 10.1007/978-3-031-03758-0_6.

[8] A. Peryanto, A. Yudhana, and R. Umar, "Convolutional Neural Network and Support Vector Machine in Classification of Flower Images."

[9] N. Sharma, R. Sharma, and N. Jindal, "Machine Learning and Deep Learning Applications-A Vision," *Global Transitions Proceedings*, vol. 2, no. 1, pp. 24–28, Jun. 2021, doi: 10.1016/j.gltp.2021.01.004.

[10] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2020, pp. 515–522. doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.

[11] D.-J. Choi and H. Park, "A HYBRID ARTIFICIAL NEURAL NETWORK AS A SOFTWARE SENSOR FOR OPTIMAL CONTROL OF A WASTEWATER TREATMENT PROCESS," 2001.

[12] C. Chen *et al.*, "Deep Learning on Computational-Resource-Limited Platforms: A Survey," *Mobile Information Systems*, vol. 2020. Hindawi Limited, 2020. doi: 10.1155/2020/8454327.

[13] V. Kouliaridis and G. Kambourakis, “A comprehensive survey on machine learning techniques for android malware detection,” *Information (Switzerland)*, vol. 12, no. 5, 2021, doi: 10.3390/info12050185.

[14] “Number of Android applications on the Google Play store | AppBrain.” https://www.appbrain.com/stats/number-of-android-apps (accessed Dec. 15, 2022).

[15] M. Ibrahim, B. Issa, and M. B. Jasser, “A Method for Automatic Android Malware Detection Based on Static Analysis and Deep Learning,” *IEEE Access*, pp. 1–1, Nov. 2022, doi: 10.1109/access.2022.3219047.

[16] A. Kapratwar, “Static and Dynamic Analysis for Android Malware Detection,” San Jose State University, San Jose, CA, USA, 2016. doi: 10.31979/etd.za5p-mqce.

[17] V. G. Teodorovici, “Modern embedded computing,” *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 1, pp. 59–60, Jan. 2013, doi: 10.1145/2413038.2413059.

[18] “20 Scary Mobile Malware Statistics to Keep in Mind.” https://safeatlast.co/blog/mobile-malware-statistics/#gref (accessed Dec. 15, 2022).

[19] “Figures of the year.”

[20] W.-C. Hsieh, C.-C. Wu, and Y.-W. Kao, “A study of android malware detection technology evolution,” in *2015 International Carnahan Conference on Security Technology (ICCST)*, 2015, pp. 135–140. doi: 10.1109/CCST.2015.7389671.

[21] P. Faruki *et al.*, “Android Security: A Survey of Issues, Malware Penetration, and Defenses,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 998–1022, Dec. 2015, doi: 10.1109/COMST.2014.2386139.

[22] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, “A Review of Android Malware Detection Approaches Based on Machine Learning,” *IEEE Access*, vol. 8, pp. 124579–124607, 2020, doi: 10.1109/ACCESS.2020.3006143.

[23] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket,” 2014. [Online]. Available: http://dx.doi.org/

[24] J. Sahs and L. Khan, “A machine learning approach to android malware detection,” in *Proceedings - 2012 European Intelligence and Security Informatics Conference, EISIC 2012*, 2012, pp. 141–147. doi: 10.1109/EISIC.2012.34.

[25] R. Sato, D. Chiba, and S. Goto, “Detecting Android Malware by Analyzing Manifest Files,” *Proceedings of the Asia-Pacific Advanced Network*, vol. 36, no. 0, p. 23, Dec. 2013, doi: 10.7125/apan.36.4.

[26] P. Tarwireyi, A. Terzoli, and M. O. Adigun, “BarkDroid: Android Malware Detection Using Bark Frequency Cepstral Coefficients,” 2022.

[27] X. Wang, D. Zhang, X. Su, and W. Li, “Mlifdect: Android malware detection based on parallel machine learning and information fusion,” *Security and Communication Networks*, vol. 2017, 2017, doi: 10.1155/2017/6451260.

[28] Westyarian, Y. Rosmansyah, and B. Dabarsyah, “Malware detection on Android smartphones using API class and machine learning,” in *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, Aug. 2015, pp. 294–297. doi: 10.1109/ICEEI.2015.7352513.

[29] A. Roy, D. S. Jas, G. Jaggi, and K. Sharma, “Android Malware Detection based on Vulnerable Feature Aggregation,” in *Procedia Computer Science*, Elsevier B.V., 2020, pp. 345–353. doi: 10.1016/j.procs.2020.06.040.

[30] U.-H. Tayyab, F. B. Khan, M. H. Durad, A. Khan, and Y. S. Lee, “A Survey of the Recent Trends in Deep Learning Based Malware Detection,” *Journal of Cybersecurity and Privacy*, vol. 2, no. 4, pp. 800–829, Sep. 2022, doi: 10.3390/jcp2040041.

[31] D. Li, L. Zhao, Q. Cheng, N. Lu, and W. Shi, “Opcode sequence analysis of Android malware by a convolutional neural network,” in *Concurrency and Computation: Practice and Experience*, John Wiley and Sons Ltd, Sep. 2020. doi: 10.1002/cpe.5308.

[32] N. McLaughlin *et al.*, “Deep android malware detection,” in *CODASPY 2017 - Proceedings of the 7th ACM Conference on Data and Application Security and Privacy*, Association for Computing Machinery, Inc, Mar. 2017, pp. 301–308. doi: 10.1145/3029806.3029823.

[33] T. H.-D. Huang and H.-Y. Kao, “R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based AndroiD Malware Detections,” in *2018 IEEE International Conference on Big Data (Big Data)*, Dec. 2018, pp. 2633–2642. doi: 10.1109/BigData.2018.8622324.

[34] D. Dang and F. di Troia, “Malware Classification Using Long Short-Term Memory Models Mark Stamp ‡,” 2021.

[35] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, “MAPAS: a practical deep learning-based android malware detection system,” *Int J Inf Secur*, vol. 21, no. 4, pp. 725–738, Aug. 2022, doi: 10.1007/s10207-022-00579-6.

[36] Z. Wang, J. Cai, S. Cheng, and W. Li, “DroidDeepLearner: Identifying Android malware using deep learning,” in *2016 IEEE 37th Sarnoff Symposium*, 2016, pp. 160–165. doi: 10.1109/SARNOF.2016.7846747.

[37] X. Su, D. Zhang, W. Li, and K. Zhao, “A Deep Learning Approach to Android Malware Feature Learning and Detection,” 2016, doi: 10.1109/TrustCom/BigDataSE/ISPA.2016.69.

[38] S. Song and Y. Tong, Eds., *Web-Age Information Management*, vol. 9998. in Lecture Notes in Computer Science, vol. 9998. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-47121-1.

[39] P. Musikawan, Y. Kongsorot, I. You, and C. So-In, “An Enhanced Deep Learning Neural Network for the Detection and Identification of Android Malware,” *IEEE Internet Things J*, p. 1, 2022, doi: 10.1109/JIOT.2022.3194881.

[40] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, “A multimodal deep learning method for android malware detection using various features,” *IEEE Transactions on*

Information Forensics and Security, vol. 14, no. 3, pp. 773–788, Mar. 2019, doi: 10.1109/TIFS.2018.2866319.

[41] R. Yumlembam, B. Issac, S. M. Jacob, and L. Yang, "IoT-based Android Malware Detection Using Graph Neural Network With Adversarial Defense," *IEEE Internet Things J*, 2022, doi: 10.1109/JIOT.2022.3188583.

[42] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J Ambient Intell Humaniz Comput*, vol. 10, no. 8, pp. 3035–3043, Aug. 2019, doi: 10.1007/s12652-018-0803-6.

[43] J. Lee, H. Jang, S. Ha, and Y. Yoon, "Android malware detection using machine learning with feature selection based on the genetic algorithm," *Mathematics*, vol. 9, no. 21, Nov. 2021, doi: 10.3390/math9212813.

[44] I. Almomani *et al.*, "Android Ransomware Detection Based on a Hybrid Evolutionary Approach in the Context of Highly Imbalanced Data," *IEEE Access*, vol. 9, pp. 57674–57691, 2021, doi: 10.1109/ACCESS.2021.3071450.

[45] M. Bierma, E. Gustafson, J. Erickson, D. Fritz, and Y. Ryn Choe, "Andlantis: Large-scale Android Dynamic Analysis."

[46] L. Richter, "Common Weaknesses of Android Malware Analysis Frameworks." [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro.html

[47] X. Wang, Y. Yang, and Y. Zeng, "Accurate mobile malware detection and classification in the cloud," *Springerplus*, vol. 4, no. 1, p. 583, 2015, doi: 10.1186/s40064-015-1356-1.

[48] M. Salehi and M. Amini, "Android Malware Detection using Markov Chain Model of Application Behaviors in Requesting System Services," Nov. 2017, [Online]. Available: http://arxiv.org/abs/1711.05731

[49] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and G. Wang, "Mal-Warehouse: A Data Collection-as-a-Service of Mobile Malware Behavioral Patterns," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2018, pp. 1503–1508. doi: 10.1109/SmartWorld.2018.00260.

[50] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Journal of Network and Computer Applications*, vol. 133, pp. 15–25, May 2019, doi: 10.1016/j.jnca.2018.12.014.

[51] I. S. Damanik, A. P. Windarto, A. Wanto, Poningsih, S. R. Andani, and W. Saputra, "Decision Tree Optimization in C4.5 Algorithm Using Genetic Algorithm," in *Journal of Physics: Conference Series*, Institute of Physics Publishing, Sep. 2019. doi: 10.1088/1742-6596/1255/1/012012.

[52] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket,” 2014. [Online]. Available: http://dx.doi.org/doi-info-to-be-provided-later

[53] M. S. Alam and S. T. Vuong, “Random forest classification for detecting android malware,” in *Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCom 2013*, 2013, pp. 663–669. doi: 10.1109/GreenCom-iThings-CPSCom.2013.122.

[54] A. Mahindru and P. Singh, “Dynamic permissions based android malware detection using machine learning techniques,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Feb. 2017, pp. 202–210. doi: 10.1145/3021460.3021485.

[55] S. Shakya and M. Dave, “Analysis, Detection, and Classification of Android Malware using System Calls,” 2022.

[56] S. Hou, A. Saas, L. Chen, and Y. Ye, “Deep4MalDroid: A deep learning framework for android malware detection based on Linux kernel system call graphs,” in *Proceedings - 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops, WIW 2016*, Institute of Electrical and Electronics Engineers Inc., Jan. 2017, pp. 104–111. doi: 10.1109/WIW.2016.15.

[57] S. K. Muttoo and S. Badhani, “Android malware detection: state of the art,” *International Journal of Information Technology*, vol. 9, no. 1, pp. 111–117, 2017, doi: 10.1007/s41870-017-0010-2.

[58] B. Baskaran and A. Ralescu, “A Study of Android Malware Detection Techniques and Machine Learning.”

[59] R. B. Hadiprakoso, H. Kabetta, and I. K. S. Buana, “Hybrid-Based Malware Analysis for Effective and Efficiency Android Malware Detection,” in *Proceedings - 2nd International Conference on Informatics, Multimedia, Cyber, and Information System, ICIMCIS 2020*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020, pp. 8–12. doi: 10.1109/ICIMCIS51567.2020.9354315.

[60] R. Surendran, T. Thomas, and S. Emmanuel, “A TAN based hybrid model for android malware detection,” *Journal of Information Security and Applications*, vol. 54, Oct. 2020, doi: 10.1016/j.jisa.2020.102483.

[61] Z. Yuan, Y. Lu, and Y. Xue, “DroidDetector: Android Malware Characterization and Detection Using Deep Learning,” 2016.

[62] R. B. Hadiprakoso, I. K. S. Buana, and Y. R. Pramadi, “Android Malware Detection Using Hybrid-Based Analysis Deep Neural Network,” in *2020 3rd International Conference on Information and Communications Technology, ICOIACT 2020*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020, pp. 252–256. doi: 10.1109/ICOIACT50329.2020.9332066.

[63] J. Zhang, Z. Qin, H. Yin, L. Ou, and K. Zhang, “A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding,” *Comput Secur*, vol. 84, pp. 376–392, Jul. 2019, doi: 10.1016/j.cose.2019.04.005.

[64] P. Dangeti, *Statistics for machine learning : build supervised, unsupervised, and reinforcement learning models using both Python and R*.

[65] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science (1979)*, vol. 313, no. 5786, pp. 504–507, 2006.

[66] L. Zamparo and Z. Zhang, “Deep Autoencoders for Dimensionality Reduction of High-Content Screening Data,” Jan. 2015, [Online]. Available: http://arxiv.org/abs/1501.01348

[67] Y. Zhu *et al.*, “Converting tabular data into images for deep learning with convolutional neural networks,” *Sci Rep*, vol. 11, no. 1, Dec. 2021, doi: 10.1038/s41598-021-90923-y.

[68] S. Mclean Cabaneros, J. K. Calautit, B. R. Hughes, and C. Author, “A Review of Artificial Neural Network Models for Ambient Air Pollution Prediction Author names and affiliations.”

[69] W. L. Hakim *et al.*, “Convolutional neural network (CNN) with metaheuristic optimization algorithms for landslide susceptibility mapping in Icheon, South Korea,” *J Environ Manage*, vol. 305, p. 114367, 2022, doi: https://doi.org/10.1016/j.jenvman.2021.114367.

[70] Y. Y. Ghadi, A. A. Rafique, T. al Shloul, S. A. Alsuhibany, A. Jalal, and J. Park, “Robust Object Categorization and Scene Classification over Remote Sensing Images via Features Fusion and Fully Convolutional Network,” *Remote Sens (Basel)*, vol. 14, no. 7, Apr. 2022, doi: 10.3390/rs14071550.

[71] A. Agrawal and N. Mittal, “Using CNN for facial expression recognition: a study of the effects of kernel size and number of filters on accuracy,” *Vis Comput*, vol. 36, no. 2, pp. 405–412, 2020, doi: 10.1007/s00371-019-01630-9.

[72] Y. Y. Ghadi, A. A. Rafique, T. al Shloul, S. A. Alsuhibany, A. Jalal, and J. Park, “Robust Object Categorization and Scene Classification over Remote Sensing Images via Features Fusion and Fully Convolutional Network,” *Remote Sens (Basel)*, vol. 14, no. 7, Apr. 2022, doi: 10.3390/rs14071550.

[73] M. Dwarampudi and N. V. S. Reddy, “Effects of padding on LSTMs and CNNs,” Mar. 2019, [Online]. Available: http://arxiv.org/abs/1903.07288

[74] S. Ahlawat, A. Choudhary, A. Nayyar, S. Singh, and B. Yoon, “Improved handwritten digit recognition using convolutional neural networks (Cnn),” *Sensors (Switzerland)*, vol. 20, no. 12, pp. 1–18, Jun. 2020, doi: 10.3390/s20123344.

[75] F. Demir, A. M. Ismael, and A. Sengur, “Classification of Lung Sounds with CNN Model Using Parallel Pooling Structure,” *IEEE Access*, vol. 8, pp. 105376–105383, 2020, doi: 10.1109/ACCESS.2020.3000111.

[76] R. Azadnia, M. M. Al-Amidi, H. Mohammadi, M. A. Cifci, A. Daryab, and E. Cavallo, "An AI Based Approach for Medicinal Plant Identification Using Deep CNN Based on Global Average Pooling," *Agronomy*, vol. 12, no. 11, p. 2723, Nov. 2022, doi: 10.3390/agronomy12112723.

[77] N. Akhtar and U. Ragavendran, "Interpretation of intelligence in CNN-pooling processes: a methodological survey," *Neural Comput Appl*, vol. 32, no. 3, pp. 879–898, 2020, doi: 10.1007/s00521-019-04296-5.

[78] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Sep. 2014, [Online]. Available: http://arxiv.org/abs/1409.1556

[79] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," Jul. 2012, [Online]. Available: http://arxiv.org/abs/1207.0580

[80] Y. Ma *et al.*, "Lungbrn: A smart digital stethoscope for detecting respiratory disease using bi-resnet deep learning algorithm," in *BioCAS 2019 - Biomedical Circuits and Systems Conference, Proceedings*, Institute of Electrical and Electronics Engineers Inc., Oct. 2019. doi: 10.1109/BIOCAS.2019.8919021.

[81] A. Wang, M. Wang, H. Wu, K. Jiang, and Y. Iwahori, "A novel LiDAR data classification algorithm combined capsnet with resnet," *Sensors (Switzerland)*, vol. 20, no. 4, Feb. 2020, doi: 10.3390/s20041151.

[82] U. Saeed *et al.*, "Portable UWB RADAR Sensing System for Transforming Subtle Chest Movement into Actionable Micro-Doppler Signatures to Extract Respiratory Rate Exploiting ResNet Algorithm," *IEEE Sens J*, vol. 21, no. 20, pp. 23518–23526, Oct. 2021, doi: 10.1109/JSEN.2021.3110367.

[83] J. Xu, Y. Zhang, and D. Miao, "Three-way confusion matrix for classification: A measure driven view," *Inf Sci (N Y)*, vol. 507, pp. 772–794, 2020, doi: https://doi.org/10.1016/j.ins.2019.06.064.

الملخص

نظام التشغيل أندرويد ، المدعوم من جوجل ، هو أكثر أنظمة تشغيل الأجهزة المحمولة شيوعًا كما أنه مجاني ومفتوح المصدر. بلغت حصة أندرويد في السوق العالمية 72.22 بالمائة في الربع الرابع من عام 2020 ، وعلى الرغم من انخفاضها إلى 71.8 بالمائة بحلول نهاية عام 2022 ، إلا أنها كانت لا تزال في المقدمة.

أصبحت دفاعات أندرويد الضارة ، التي تحلل البيانات الخطيرة للبرامج الضارة لتمييزها ، مشكلة مهمة في أبحاث أمن المعلومات نظرًا لأهميتها في الحفاظ على أمان الأجهزة.

أساليب التعلم الآلي التقليدية محدودة في قدرتها على تعلم التمثيلات المعقدة في المجالات عالية الأبعاد ، على الرغم من استخدام عدد من النماذج لتحديد البرامج الضارة في أنظمة أندرويد. علاوة على ذلك ، يعتمد نجاح نماذج التعلم الآلي بشكل كبير على بيانات التدريب ، ومع تطور تطبيقات أندرويد وتطور هندسة البرمجيات ، فمن المرجح أن تصبح هذه النماذج المدربة قديمة.

في هذا البحث ، بناءً على التحليل الثابت ، تم استخدام مجموعة بيانات CICMalDroid 2020 لتطوير نموذج هجين يجمع بين تقليل الأبعاد وتحويل البيانات إلى صور باستخدام IGTD ، ثم استخدام مجموعة CNNs للتصنيف الثنائي والتصنيف المتعدد.

حقق هذا النموذج دقة عالية بلغت 94.38٪ في التصنيف الثنائي (حميدة وضارة) و 95.83٪ في التصنيف المتعدد.