Detection of Android Adware using Transfer Learning with Computer Vision

By

KABOMBO KATUTWA

A Dissertation submitted to the University of Zambia in partial fulfilment of the requirements for the award of the degree in Master of Engineering in Information and Communications Technology Security

THE UNIVERSITY OF ZAMBIA

LUSAKA

2023

DECLARATION

I, the undersigned, hereby truthfully declare that I am the sole author of this report, and all its contents are my original work that has before not been presented at any learning institution for an award. I further acknowledge that similar work has been done but not the same as this. All the work of other persons used to complete this project has been duly acknowledged and properly referenced.

Student Name: …….

Signature: ……..

Date: ……

Supervisor Name: ……

Signature: ……..

Date:.……

CERTIFICATE OF APPROVAL

This dissertation by KABOMBO KATUTWA is approved as fulfilling the partial requirements for the award of the degree of Master of Engineering in Information and Communications Technology (ICT) Security by the University of Zambia

Examiner 1

Name:…… Signature:……..

Date:……..

Examiner 2

Name:…… Signature:……..

Date:……..

Examiner 3

Name:…… Signature:……..

Date:……..

Chairperson

Name:…… Signature:……..

Date:……..

ACKNOWLEDGEMENT

Firstly, I give praise and thanks to God almighty for giving me the opportunity to undertake this study as well as for the many blessings that have enabled this to be a success. I am extremely grateful to Dr. Dani E. Banda and Mr. Julien Shabani who helped enormously and guided me with their knowledge and experience throughout my research to completion.

DEDICATION

I dedicate this research work to my wife Nsofwa Chansa Katutwa and our three children who have been my source of strength and motivation as I pursued this research study.

To my late parents, Fred Kabombo Katutwa and Queen Nalwimba Katutwa (May Their Souls Rest In Peace) for always encouraging me to do the very best in whatever I do and for giving me the very best, I owe every success I score to them. I wish you were here.

ABSTRACT

Android malware is a dominant threat category for consumers with a vast majority of this malware being Adware. Adware is a type of malware that displays unsolicited pop-up advertisements that are used to generate revenue and can also lead to installation of trojan horses, spyware, and other malicious software. Over the recent years there has been an increase in the number and complexity evasive malware, especially Adware. Despite this, there has not been a lot of focus put on the Adware malware family. Malware authors are using obfuscation techniques to make their malware difficult to detect using conventional static and dynamic methods of analysis, this has led to the rapid adoption of artificial intelligence and machine learning approaches to overcome these challenges. Malware binaries can be visualized as images, with the observation that for many malware families, the images belonging to the same family appear very similar in layout and texture. These images can then be used as input to train deep neural networks to detection malware. The goal of this research is to detect android Adware using visual representation of an android application package (APK) Dalvik Executable (DEX) file and providing it as an input image to a pre-trained neural network can successfully detect android Adware apps. Using the CICMalDroid2020 dataset obtained from the University of New Brunswick Canadian Institute of Cybersecurity as training, validation and test datasets for my experiments, the DEX files in each APK file are first extracted from the APK files and then converted to grayscale images. The obtained images are then applied as input to deep neural networks that use selected pre-trained models namely VGG16, ResNet50, InceptionV3, EfficientNet v2 and MobileNet v2 to detect android Adware. Using the confusion matrix to calculate each model's accuracy, recall, precision, and f-1 score, the performance results of each trained model are compared amongst all the models used in this research. The highest performance measurement of 92 % detection rate on the f1- score performance metric was achieved by the MobileNet v2 model. The results obtained in my research reveal that computer vision and transfer learning perform adequately in the detection of android Adware apps from benign apps.

Keywords: Android Adware, Computer Vision, Deep learning, pre-trained model, Convolutional Neural Network (CNN), Transfer learning

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ADR	Attack Detection Rates
AMD	Android Malware Dataset
API	Application Programming Interface
APK	Android Package
APP	Mobile Application
CART	Classification and Regression Trees
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
DBN	Deep Belief Network
DEX File	Dalvik Executable file
DNN	Deep Neural Network
GAP	Global Average Pooling
GIST	Generalized Search Tree
IMCFN	Image based Malware Classification using Fine-tuned Convolutional Neural Network Architecture
iOS	iPhone Operating System
IOT	Internet of Things
KNN	K-Nearest Neighbors
LDA	Linear Discriminant Analysis
LR	Logistic Regression
MALWARE	Malicious Software
MLP	Multilayer Perceptron
NB	Naïve Bayes
OHA	Open Handset Alliance
OS	Operating System
PIN	Personal Identification Number
UI	User Interface
RF	Random Forest
RGB	Red, Green Blue,
RNN	Recurrent Neural Network

SMS	Short Message Service
SVM	Support Vector Machine
TCN	Time Convolutional Neural Network
TPU	Tensor Processing Unit
VGG	Visual Geometry Group
XML	Extensible Markup Language

CHAPTER ONE: INTRODUCTION

1.1 Introduction

People use their smartphones in both their personal and professional lives today, making them an essential aspect of life. There are an estimated 2.6 billion active smartphone users [1]. The rise in smartphone users has also led to an increase in malicious programs targeting mobile devices, that is, mobile malware. Malware is a program that has an intention of causing harm to the operating system kernel or some security sensitive application or data without the user's consent [1]. Criminals attempt to take advantage of flaws on other people's smartphones for their own gain. Additionally, over the past years' malware authors have become less recreational-driven and more profit-driven as they are actively searching for sensitive, personal, and enterprise information [2].

There has been an increase in the usage of handheld devices with the android platform having the larger market share, in 2018 the worldwide mobile application (app) downloads were approximated to be 194 billion [3]. With the growth in the adoption of technology, malware has become an increasing problem.

Android is the most popular smartphone operating system in the world as of 2018[4]. Since its release, sales of smartphones running on the android operating system have grown strongly over the years [4]. Because of this popularity, android presents itself as an attractive attack platform for adversaries looking to maximize their impact on victims [5].

Applications like WeChat, TikTok, and mobile banking applications are used in our daily lives and continue to play an increasingly important role. Most of these applications have access to users' private information such as their location, debit/credit card, and contact information. Almost all applications access the users' private data, although this provides users with better personalized services it may also result in information leakage of private data and economic loss. As cited by [6], Dogru et al. state that android malicious applications keep on emerging continuously, and this security concern has gained increasing attention in both industry and academic fields.

Android operating system had 72% mobile operating system market share for the period May 2019 to May 2020 [7].

By mobile operating system market share, android is the dominant player and with this in combination with its liberal and open application marketplaces (compared to Apple's locked-down iOS application ecosystem), has meant that it has quickly become the mobile platform of choice for malware authors [8].

McAfee's Mobile threat report of quarter one 2019 reported that mobile malware continues to increase in complexity and scope [9]. There are more types of threat actors targeting mobile platforms than just those hoping to increase advertising sales. It has also been observed that stealing banking or other financial credentials or PINs from mobile devices is also on the rise, as more and more people embrace the convenience of mobile banking and payments.

1.2 Background

To address the plethora of malware and its associated security concerns, researchers are constantly developing and upgrading malware detection systems. Mobile malware detection is the process of classifying unknown mobile applications into benign and malicious [10].

There are different types of malicious software and one of these is called Adware. Adware is a form of malware that downloads and displays unwanted advertisements, which are often offensive and always unsolicited [11].

Advertisements are used to promote or sell a product, an idea, or a service. The advent of the internet, and later the smartphone, has pushed marketing strategies towards digitizing advertisements as it is the new norm, and the digital channel is rapidly growing with no signs of slowing down [24]. According to work by Shahzad et al [26] the Adware problem is growing continuously due to the profound monetary gains for Adware developers, and it is largely agreed upon that user awareness of Adware and its possible negative effects is still minimal.

Avast a global leader in digital security and privacy, reported that in 2021, Adware continued to be the most significant threat on android phones and tablets, with 45% of mobile threats being Adware in the first five months of that year [12].

1.2.1 Android Operating System

Android is a Linux-based mobile operating system, and it was developed by a consortium of companies known as the Open Handset Alliance (OHA), with the primary contributor and commercial marketer being Google [13]. Since its first release, the Android operating system has undergone tremendous development. The

year 2008 saw the formal release of Android version 1.0 to the general market. With the Android 1.5 Cupcake release in 2009, the tradition of naming Android versions after confectionery was born [13].

1.2.2 Android Architecture

Android is an open source, Linux-based software stack created for a wide array of devices and form factors [14]. An Android app is made up of an application package. The following describes the components of an android package:

a) Android Package (APK)

Bakour et al [15] describes the android APK anatomy as containing the following files and folders.

i. **AndroidManifest.xml**

It is one of the most important files in the android application, and this file is the first part that is read by the operating system when running any android application.

ii. **Classes.dex**

Dex code is an optimized bytecode for android applications that contains multiple constructs like file header, string table, local variable list, class definition table, method list etc.

iii. **Resources.arsc**

It is a file containing the application's resources in a binary format.

iv. **Lib/folder**

This folder contains the native code libraries.

v. **Assets/folder**

The assets (i.e., images, files, etc.) can be placed in this folder and it will be accessed using AssetManager.

vi. **Res/folder**

The app's resources (like icons, music, images etc.) are placed in this directory.

vii. **META-INF/folder**

It contains the application's certificate and composes of three essential files namely, MANIFEST.MF, *. SF, and *.RSA [16]

1.2.3 Computer Vision

Computer vision (CV) is a field of study focused on the problem of helping computers to see [17]. [17] further goes on to describe CV as a multidisciplinary field that could broadly be called a subfield of artificial intelligence and machine learning, which may involve the use of specialized methods and make use of general learning algorithms [17].

The goal of computer vision is to teach computers how to make sense of these pixels the way humans (and other creatures) do, or even better [18].

Computer vision is a subfield of Artificial Intelligence (AI) that tries to mimic the human visual system; while it used to rely on an expert systems approach, today it's done with machine learning [18].

Figure 1.1 illustrates the relationship of artificial intelligence and computer vision. It shows that computer vision is a subfield of both artificial intelligence and machine learning, with the latter also a subfield of AI.

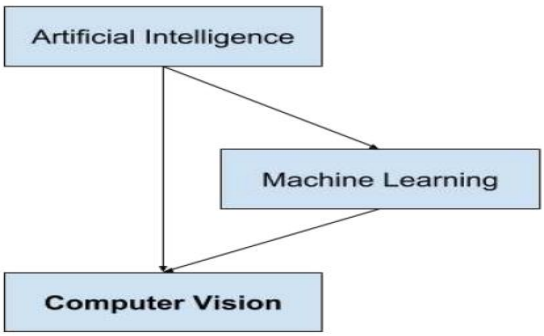


Figure 1.1 Relationship of Artificial Intelligence and Computer Vision[1]

Overall, the intent of computer vision is to enable a computer system to recognise what is contained in a digital image.

Based on the literature reviewed, I established that numerous work has been conducted in developing effective and efficient computer vision-based malware detection mechanisms. Almomani et al [67], introduced an efficient and automated computer vision-based android malware detection model which precluded the need for pre–designed features extraction process while generating accurate predictions of malware images that are inexpensive and quickly detected.

[1] Brownlee, B., Deep Learning for Computer Vision, 1st ed. Machine Learning Mastery

Bensaoud et al [19] state that researchers and practitioners can understand malware better by visualizing malware binaries as images since the patterns within such images become clearly visible.

1.2.4 Deep Learning

Deep learning is a type of machine learning that has advanced rapidly in the past few years, due to improvements in processing power and deep learning techniques [20]. Usually, deep learning refers to deep, or many-layered, neural networks, which excel at performing very complex, often historically human-centric tasks, like image recognition and language translation [20]. One of the key deep learning neural networks applied in computer vision problems is the Convolutional Neural Networks (CNN). With respect to security, several attempts have been made to transfer deep learning's application from the domain of image recognition or natural language processing into malware detection [21]. Finding patterns within images can be performed well by deep learning [22].

a) Convolutional Neural Networks (CNN)

CNNs, or ConvNets, are quite like regular neural networks, they are still made up of neurons with weights that can be learned from data and each neuron receives some inputs and performs a dot product [23]. Figure 1.2 illustrates three-layer convolutional neural network

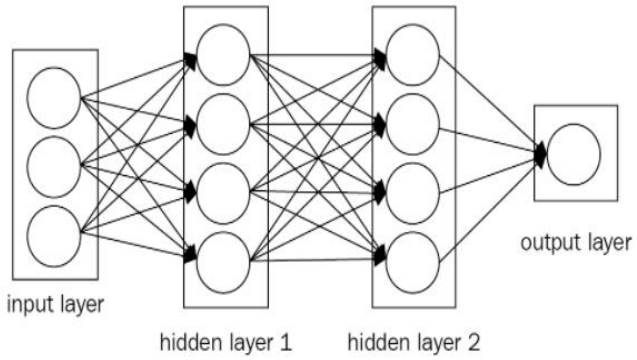


Figure 1.2 A regular three-layer neural network[2]

In a CNN each hidden layer has a set of neurons, and every neuron is fully connected to all the other neurons in the previous layer as illustrated above. In a single layer, each neuron is completely independent, and they do not share any connections. The

[2] Sewak, M., Karim, M. and Pujari, P., 2018. Practical Convolutional Neural Networks. 1st ed.

output layer, which is the last fully connected layer, contains class scores in the case of an image classification problem.

Hasegawa et al. as cited in [77] proposed a lightweight malware identification technique leveraging Convolutional Neural Network (CNN) to analyse raw APKs.

In [78], a novel efficient deep learning network with multi-streams for android malware family classification was proposed by obtaining the input data for a convolutional neural network (CNN) in string format from some main files or sections contained in each android malicious app.

b) Transfer Learning

Transfer learning is the idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones [45]. It is a machine learning technique that leverages the knowledge learnt from one research problem to resolve a different but related task. It results in quick- model development and might also improve final performance of a model. Transfer learning is a relatively new area of research (since 1995) in machine learning whose aim is to extract knowledge that was learned for one task and use it for a target task [24].

In my research, I use transfer learning by utilizing five pre-trained CNNs that have successfully achieved great performance on the ImageNet database [25]. These models are Visual Geometry Group (VGG)16, ResNet50, Inception v3, EfficientNet v2 and MobileNet v2.

1.2.5 Malware Analysis

Malware analysis is the study of malware's behavior. The objective of malware analysis is to understand the working of malware and how to detect and eliminate it. It involves analyzing the suspect binary in a safe environment to identify its characteristics and functionalities so that better defenses can be built to protect an organization's network [26].

Jung et al. [27] emphasized that to detect malicious code, static analysis and dynamic analysis are used. In [28], Singh explains that malware analysis is being used both for detecting malware and classification of malware. Academic researchers have extensively studied the android malware detection problems [90].

a) Android Adware Detection

The main target of Adware is to generate revenue for the publisher, additionally some types of Adware also collect information regarding a user's activities without consent and this type of Adware is also called Spyware [29].

As cited in [30] the Adware problem is growing continuously due to the profound monetary gains for Adware developers. The authors further went on to observe that major commercial antivirus tools try to detect instances of Adware by relying on static or dynamic analysis such as signature-based and heuristic approaches (which were developed for detection of viruses), but these techniques have a deficiency in detecting unknown or new instances and can be bypassed in different ways. Alani et al [31] presented a machine learning system called AdStop that detects Android Adware. They used features from the network flow of outgoing and incoming traffic to an Android device.

Dobhal et al [32], carried out research work on Android Adware. Their work had several objectives, with particular interest to my thesis is their investigation on the performance of the machine learning algorithms for binary classification of the adware instances.

1.3 Problem Statement

Because of their popular use due to its free and open-source nature, Android devices present themselves as an attractive attack platform for numerous types of malware with the following problems:

a) Proliferation of malware targeted at Android devices.
b) Malware authors using obfuscation techniques to make their malware difficult to detect using traditional static and dynamic methods of analysis
c) Numerous zero-day malware that easily evades signature-based malware detection.
d) Highly specialized skills required and the huge volume of data for malware analysts to review that is time consuming and may lead to analysis fatigue.

Bagui et al [33] highlighted the need to develop better methods of detecting Adware due to the rise of more and complex evasive malware, specifically Adware as, very little focus has been put on the Adware family compared to other types of malware.

1.4 Research Aim

To detect Android Adware from benign apps using computer vision and pre -trained deep learning models.

1.5 Objectives

The objectives of this research are:

a) To examine Android Adware apk files to extract bytecode and convert it to a grayscale image.
b) Investigate transfer learning and computer vision techniques in Android Adware detection.
c) Compare the training accuracy against the validation accuracy obtained during the training of each selected pre-trained neural network.
d) Evaluate the performance of selected pre-trained neural networks in detecting Android Adware.

1.6 Research Questions

To achieve my research objectives, I will attempt to answer the following questions:

a) Can computer vision be applied in the detection of Android Adware?
b) Are the selected pre-trained deep neural network models effective in the detection of Android Adware?
c) Which pre-trained deep neural network models used in (b) above will produce the best performance metric outcomes in the detection of android adware from benign apps?

1.7 Research Contributions

This research aims to contribute to the body of knowledge by firstly carrying out research work that leverages computer vision and transfer learning in the detection of android adware apps from benign android apps. To my knowledge obtained through the various reviewed literature, previous work in this area has not utilized computer vision and transfer learning to detect android adware apps from benign apps. Secondly, I evaluate the performance of selected pre-trained convolutional neural networks to determine which of these pre-trained models performs the most effectively and efficiently in detecting android adware from benign apps.

1.8 Significance of Study

This research is important in that it will aid malware analysts in their efforts of analysing Android Adware from benign apps. In addition, it will reduce the time of

carrying out malware analysis and does not require expert knowledge in feature extraction due to the ability of deep learning models utilizing automatic extraction of features on samples being tested. Though a lot of work has been done on Android malware detection in general, very little focus has been put on the adware family [33], my research therefore expands on the work in this area. Additionally, based on the literature reviewed we have not come across research work that has used transfer learning and visualization to detect android adware apps from benign apps hence the motivation to carry out research in this area.

1.9 Scope of Study

The scope of the research will be as follows:

a) My research will be focused on Adware on the Android platform.
b) The dataset used will focus solely on Android Adware and benign files that are free and publicly available.
c) The research will utilize selected pre-trained deep neural networks.

1.10 Ethical Considerations

During my research:

a) I will ensure adequate level of confidentiality of the research data is maintained.
b) I shall ensure that there is no deception or exaggeration about the aims and objectives of the research that I will conduct.
c) I shall make sure that any form of communication in relation to the research should be done with honesty and transparency.
d) I will adhere to the regulations of the University of Zambia as I carry out my research.

1.11 Summary

In this chapter, I start by introducing the work that will be carried out in this study. I begin by looking at the background to the plethora of the Android malware problem. The aim, objectives and research contributions in this study were then outlined. Finally, I give the significance of this study and the scope of the research work to be conducted. I concluded this chapter with the outline of the ethical considerations that I abide by during the process of carrying out this work.

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction

This section presents different literature reviewed from various sources as part of the literature study. The sources included journals, thesis, dissertation work and textbooks as well as selected articles from the internet.

Mobile malware was almost non-existent before the official release of the Android platform in 2008 [34]. The first Android malware to come into existence in early August 2010 was dubbed FakePlayer [35]. AndroidOS.FakePlayer is a Trojan horse that attempts to send premium-rate short message service (SMS) messages to predetermined numbers [35].

With the plethora of application development on mobile platforms and the increase in the number of mobile threats, there has been significant growth in research work in the field of mobile malware, specifically Android malware. In this thesis, I review research work that has been done in areas related to the work done in my thesis. These are malware analysis, Android malware, malware visualization, and deep learning.

2.2 Android Malware

This section looks at various literature on Android malware in general and then narrows down to Android Adware.

With the rapid development of mobile platforms and the rise in the number of mobile threats, the number of studies on mobile malware, specifically Android malware, has steadily increased. [34].

Several techniques and methods have been introduced over the years for the detection or classification of Android malware. I will examine literature on Android malware detection and classification based on features extracted from static and dynamic malware analysis and the use of computer vison to visualize malware samples and then apply the resulting images to machine learning and deep leaning techniques.

Dhalaria et al. [36] performed Android family classification by using AndroidManifest.xml and classes.dex files represented using grayscale images which were used as input to a convolutional neural network (CNN) to classify the malware

images. In their experiments, the results indicated that classes.dex file gives better results as compared to the AndroidManifest.xml.

Fang et al. [37] highlight that traditional static method for classification is easily affected by confusion and reinforcement, while the dynamic method is expensive in computation. To overcome this, they used an approach that was divided into four parts, these being file extraction, then converting the extracted DEX file followed by features extraction, and classification.

Their experimental results show that the Android malware family classification method they proposed that was based on DEX file section features performed with high classification efficiency with precision, recall, and F1-score reaching 94%.

Naway et al. [38] implemented a model that analyzed Android malware using static analysis to provide five different feature sets of permission combinations, API calls, intent filters, valid certificates, and the presence of APK files in the asset folder. All the extracted features were transformed into a feature vector that was applied to a deep neural network. The research work achieved a precision of 95.35% on correctly classified apps within all classified apps. The ratio (recall) of correctly identified benign samples was 95.31% and an F1-Score of 95.31%.

2.3 Computer Vision Approaches

Image classification is a fundamental problem in computer vision [41]. In the early stage, Haralick et al. [42] describes some easily computable textural features based on gray tone spatial dependencies and illustrates their application in category-identification tasks.

In 2011 Nataraj et al [43] proposed a simple yet effective method for visualizing and classifying malware using image processing techniques. Malware binaries were visualized as gray-scale images, with the observation that for many malware families, the images belonging to the same family appear very similar in layout and texture.

The image classification method does not require any disassembly or execution of the actual malware code. Moreover, the image textures used for classification provide more resilient features in terms of obfuscation techniques, and for encryption [43].

Image based detection and classification proved to be effective, because it leverages the structural similarity between the known and new malware samples [27]. Moreover, visual analytics helps analysts to recognise patterns in malwares' code and behaviour, thus helping them to come up with better results [27].

Han et al. [44] suggested visualizing malware using op-code sequences to detect and classify malware samples. They used image matrices to visually represent malware which assisted in detecting features of malware and in finding similarities between different samples swiftly. Their initial step was to disassemble the binary file using IDA Pro or OllyDgb and dividing the op-code sequence into blocks. Then they used two hash functions and for each block of op-code sequence, computed a coordinate and the RGB value using the two hash functions. After this they plotted all the RGB values to their corresponding coordinates in a matrix of dimension 8 by 8 to get an image matrix. They used "selective area matching" for calculating similarities between image matrices and evaluated their model on malware samples from 10 different families. Their results showed that image matrices of malware from the same family had higher similarity score than with malware from different family, hence they concluded that image matrices could effectively classify malware families.

Gennissen [45] proposed a system for detecting Android malware where ten (10) types of images were created by adding domain knowledge to image conversion. With the Dalvik opcode and API information, they converted classes.dex into a fractal shaped Hilbert curve image and used a CNN model with two layers. An accuracy of up to 92% was reported.

Jung et al. [46] proposed an Android malware detection method where Android applications were represented as grayscale images. The first step involved a Dex file processing step through unzipping an APK file and extracting the classes.dex from it. This was followed by parsing the header section of classes.dex, to obtain the offset of data section. They then created a file data_section.dex whose contents are the data section. They proceeded to read each byte from data_section.dex and considered it as a grayscale pixel value in the 8-bit range, that is, each pixel value ranges from 0 to 255. A grayscale image whose width is determined according to file size was created. The height of the image depended on the file size of

data_section.dex 0-padding bits are attached to the last row to make the image rectangle. Lastly, the created images were used as training and test data to build a CNN model. Due to the use of the data section only, instead of the whole DEX file the image size of this proposal was smaller than that of the conventional one, due to this they concluded less data loss, which in turn would enhance the performance accuracy.

Gao et al. [47] proposed an effective malware classification framework based on malware visualization and semi-supervised learning. Their framework utilized three approaches, namely malware visualization, feature extraction, and use of a classification algorithm. The first step was to process binary files directly through visual methods, without assembly, decompression, and decryption. Then the global and local features of the gray image were extracted, and the visual image features extracted were fused overall by a special feature fusion method to eliminate the exclusion between different feature variables. Finally, an improved collaborative learning algorithm to continuously train and optimize the classifier by introducing features of inexpensive unlabeled samples was used. This framework was evaluated over two extensively researched Malimg and Microsoft benchmark datasets. Their results showed that compared with traditional machine learning algorithms, the improved collaborative learning algorithm can not only reduce the cost of sample labeling but also can continuously improve the model performance through the input of unlabeled samples, thereby achieving higher classification accuracy.

Thakuri et al. [48] carried out work detecting Android malware using its image sections. In their work, Android malware images were created using different sections of malicious applications. Four types of malware images generated were the android manifest, classes.dex, resources, and certificate files. To extract features from the malware images the Generalized Search Tree (GIST) algorithm was used. To perform the classification various machine learning classifiers such as Support Vector Machines (SVM), K-Nearest Neighbors, Random Forest, and Naïve Bayes are used in [48]. Their work compared the results obtained by these different classifiers resulting in the model GIST +SVM outperforming all other classifiers and attained the classification accuracy of 92.7% on malware images.

Malware images are used as patterns to visualization and automatic classification [43]. There have been many studies using image classification techniques for malware detection [49]. Nowadays, image classification through machine learning has achieved brilliant achievement and the malware mapping to the image field, will make people more intuitive to observe the malware segment information and other features [49].

2.4 Deep Learning Approaches

Deep learning has been very successfully utilized for various computer vision tasks, such as object identification, using different CNN architectures [50]. In their paper Yosinski et al. [51], present their findings on how the lower layers act as conventional computer-vision feature extractors, such as edge detectors, while the final layers work toward task-specific features.

Sun et al. [52] classified unknown Android malware into malware families by employing a deep learning-based classification approach with the code images of the malware. They constructed the deep learning classifier by reusing the feature-extracting layers of a Convolutional Neural Network (CNN) which has been successfully trained for other image classification tasks on a large dataset. They developed a malware classifier that makes use of code images transformed from binary bytecodes of Android malware and employs a pre-trained CNN to learn discriminative patterns from the images. Unlike many existing static methods, their classification method in does not require any disassembly analysis of Android malware, and thus more resilient to obfuscation techniques.

Convolutional Neural Networks (CNNs) have shown superior performance compared to traditional machine learning techniques, specifically in tasks such as image classification [52]. This motivated Prima et al. [52] to propose a CNN-based architecture for malware classification. With the malicious binary files represented as grayscale images they used a pre-trained deep neural network VGG16 that was trained on the ImageNet dataset [25] and adapting the last fully connected layer to their malware family classification task.

Xiao et al. [53] used deep learning to do feature extraction automatically after they displayed the binary malware as entropy graphs and then used Support Vector Machine (SVM) algorithms to classify the malware based on the extracted features.

To train a highly effective malware classifier for static malware classification [54] proposed a method that leverages transfer learning. The author was able to show that transfer learning showed the highest classification accuracy, lowest false positive rate, highest true positive rate and highest F1-score compared to classical machine learning algorithms such as Naive Bayes, K-Nearest Neighbor, Linear Discriminant Analysis, Random Forest, XGBoost and Support Vector Machine with Linear and Radial Kernels respectively.

Khan et al. [55] performed analysis for transfer learning for malware classification using ResNet and GoogleNet with their data preparation pipeline and the models ResNet 18,34,50,101,152 were reported to have achieved an accuracy of 83%,86.51%,86.62%,85.94% and 87.98% respectively. The accuracy for GoogleNet was reported to have reached 84%. For their work two sorts of datasets were utilized for training and validation of the models. One of the datasets was downloaded from Microsoft which is the combination of 10,868 records and these records are binary records. These records are also divided into nine distinct categories. The second dataset, considered, contains three-thousand benign files. The datasets were initially in the form of EXE files, which were then converted into opcode, which was then converted into images.

Mohammed et al. [56] demonstrated the use of deep learning that includes transfer learning-based approach in malware detection tasks and showed that the features learned by the deep CNN models (ResNet) performed significantly better than the hand-crafted GIST features proposed in the literature, and slightly better than the features learned by the proposed shallow CNN model. They further created a new dataset called MaleX which contains around one million malware and benign windows executable samples created for large-scale malware detection and classification experiments.

Singh et al. proposed in [57] a system called SARVOTAM that is defined as Summing of neurAl aRchitecture and VisualizatiOn Technology for Android Malware classification. This system was developed to convert the malware non-intuitive features into fingerprint images to extract the quality information.A CNN was fine-tuned to automatically extract the rich features from visualized malware thus eliminating the feature engineering and domain expert cost.SARVOTAM was

augmented by combining traditional classifiers like K-Nearest Neighbours (KNN),Support Vector Machine (SVM) and Random Forest (RF) to recommend prominent Android file structure features for malware identification and classification. From this investigation it was observed that the CNN-SVM model outperformed original CNN as well as CNN-KNN, and CNN-RF. They observed that malware images formed using certificate and android Manifest files (CR+AM) offer a lightweight and much precise option for malware identification. The proposed system was evaluated against the DREBIN dataset [58]. This dataset contains 5560 applications from 179 different malware families. The results showed that combination of CNN and SVM was found to be most suited and even surpassed generic CNN in identification and classification of Android malware families. CNN-SVM achieved the classification accuracy of 92.59%. Figure 2.1 depicts the SARVOTAM methodology for classification of Android malware that was used in [57].

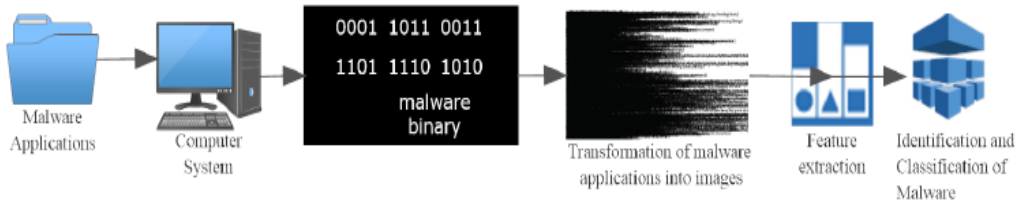


Figure 2.1 An illustration of SARVOTAM methodology for classification of Android malware[3]

In [59], the authors used the Xception ImageNet pre-trained deep neural network model that is trained on the ImageNet [25] dataset to classify malware images. The features called bottleneck features were extracted from the first layer up to the Global Average Pooling layer. The transfer learning technique that was used involved replacing the softmax layer of the pre-trained CNN to a new randomized weight softmax layer which was used to train the new softmax layer to classify the malware images. Based on the results reported in [59] it was demonstrated that the validation accuracy of the Xception model on the Malimg Dataset and the Microsoft malware datasets was higher (99.04% and 99.17% respectively) with lower log loss

[3] Singh, J., et al., Deep Feature Extraction and Classification of Android Malware Images. Sensors, 2020

compared with the previous winner of the VGG model on malware image classification.

Zhenlong Yuan et al. as cited in [38], explored the use of deep learning in Android malware identification by using static and dynamic features extracted from APK files. The authors developed a deep belief network (DBN) model for classification utilizing small dataset and achieved high detection accuracy.

Naway et al. [38] proposed a system for Android malware detection using the deep neural network, which uses permissions combination, intent filters, invalid certificate, the existence of APK file in asset folder, and API calls as features to construct a deep neural network (DNN) that can identify malicious from benign ones. They noted that the best accuracy was achieved when all features are used in the classification process, their results are tabulated in the table 2.1 which shows the results of their experiments.

Table 2.1 Performance results of the DNN using different features set[4]

	Features Set	**Accuracy**
1	All Features Set	95.31%
2	APIs Only	90.10%
3	Permissions Combination Only	82.29%
4	Intent-filters Only	69.79%
5	APIs + Permissions combination + Invalid Certificate + Presence of APK file in asset Folder	93.22%
6	APIs + Intent-filters + Invalid Certificate + Presence of APK file in asset folder	91.14%
7	Permissions Combination+ Invalid Certificate + Presence of APK file in asset Folder	83.33%

[4] Naway, A., LI, Y., 2019. Using Deep Neural Network for Android Malware Detection. International Journal of advanced studies in Computer Science and Engineering (IJASCSE) VOLUME 7 ISSUE 12, 2018, pg. 9-18.

Additionally, the performance of the proposed system was compared with four (4) common machine learning methods Decision Tree (DT), K-Nearest Neighbor (KNN), Random Forest (RF), and Support Vector Machines (SVM) using the same setting applied to evaluate DNN. The observation that they made on the proposed system exceeded the common machine learning methods in all performance metrics. This observation has been seen in [57] supporting the superior performance of deep neural networks in malware classification.

Dhalaria and Gandotra [60], proposed a design of a CNN model which was applied on the grayscale images (of different sizes) obtained from AndroidManifest.xml and classes.dex file extracted from APK files. The CNN automatically extracted features from the images. Their experiments were conducted on a dataset consisting of grayscale images of 1747 Android application with 13 malware families. The results of these experiments demonstrated that the images created through classes.dex files gave better classification accuracy as compared to the images created using AndroidManifest.xml files. It is also established from their study that for both types of files, as the image size increased, the classification accuracy also increased.

Zhao and Qian as cited by [60] introduced a static technique which does not require knowing the source code of application. With the help of decompilation they mapped the opcodes and API function into an RGB image. They employed CNN to detect Android malware families. The results were demonstrated based on accuracy, precision, recall, and f-measure which were 90.67%, 93.36%, 93.95% and 93.56% respectively.

Zou et al. [61], came up with ByteDroid an automatic Android malware detection CNN. It applied multiple convolutional kernels to learn the sequential patterns of bytecode. ByteDroid was more effective at detecting malware applied with trivial obfuscation, however it had poorer performance in defending other obfuscation techniques such as string encryption, class encryption and reflection because these techniques affect the strings and bytecode sequences [61].

In [62], the proposed method deals with the opcode and the bytecode in the APK files and analyses various API sequences features by utilizing bi-directional Long Short-Term Memory (Bi-LSTM) network to implement a classification model. Each unit in the extracted behaviour sequence was inventively represented as a vector,

which allowed Droidetec to automatically analyse the semantics of sequence segments and eventually find out the malicious code. Experiments with 9616 malicious and 11982 benign programs from the malware samples used were from AMD (Android Malware Dataset) [63], [64], a carefully labeled dataset that includes comprehensive profile information of malware, and the benign samples were from Google Play store show that Droidetec reached an accuracy of 97.22% and an F1-score of 98.21%.

Kim et al. as cited by [65], proposed a multi-modal deep learning malware detection model, which extracted multiple feature types to reflect the attributes of the Android application from various aspects and uses feature extraction methods based on presence or similarity to refine these features and to achieve effective feature representation in malware detection. Permissions, components, environment, strings, Dalvik opcode sequences, API call sequences, and shared library function opcode features are the seven types of static features that were extracted. Each type of feature was used respectively to train the initial network of the corresponding deep neural network. Training results of the initial network was then used to train the final network. The dataset was composed of 13075 malicious samples and 19747 benign samples with the model achieving a detection accuracy of 98%.

Zen et al. [66] proposed two end-to-end Android malware detection methods based on deep learning. They used a method of resampling the raw bytecodes of the classes.dex files of Android applications as input to deep learning models. These models were trained and evaluated in a dataset containing eight thousand benign applications and eight thousand malicious applications. Experiments showed that the proposed methods could achieve 93.4% and 95.8% detection accuracy respectively. They compared their approach with existing methods and concluded that their method was not limited by input file size, required no manual feature engineering and the resource consumption was low.

Su et al. [67] proposed DroidDeep, a malware detection approach for the Android platform based on the deep learning model. To implement this, they first extracted permissions, used permissions, sensitive API calls, opcodes, and application component features through static analysis. They then, developed the deep learning model to learn features from the extracted artifacts of the Android apps. Finally, the

learned features were used to detect unknown Android malware. In their experiment they used 3,986 benign apps and 3,986 malware. They reported to have achieved a 99.4% detection accuracy using DroidDeep.

Alotaibi et al. [68] proposed a framework based on deep residual long short-term memory (LSTM) network to identify and classify malware variants. The framework was divided into four major stages: raw data static extraction, feature extraction sets, vector space embedding, and a residual long short-term memory model process. Their framework-imposed constraints on the deep learning architecture to capture dependencies between features extracted from the Android package kit (APK) file. These feature sets were mapped to a vector space to process the input sequence using a sequence model based on the residual LSTM network. To evaluate the performance of the proposed framework, several experiments are conducted on the Drebin dataset [69],[70] which contains 129,013 applications. From their paper it was reported that the framework could achieve a 99.32% detection accuracy.

Almomani et al. [71], proposed the use of transfer learning to proficiently classify benign Android apps from android malware apps using 16 different fine-tuned deep learning-based CNN algorithms namely Xception, VGG16, VGG19, DarkNet53, MobileNetv2, ResNet 101, AlexNet, ResNet50, ResNet18, Inceptionv3, DarkNet19, ShuffleNet, Places 365-GoogleNet, NasNetMobile, GoogleNet, and SqueezeNet. Firstly, the bytecodes of the "classes.dex" files extracted from the Android benign and malware apps were converted to color and grayscale visual images before forwarding them to the developed CNN algorithms for classification. Subsequently the detection efficiency of the proposed Android Malware Detection model was examined and evaluated using the imbalanced benchmark Leopard Android dataset that composes 14733 samples of malware apps and 2486 samples of benign apps. Finally, different experimental scenarios were conducted using balanced and imbalanced Android samples of color and grayscale images generated from the Leopard dataset. They reported detection accuracy reached 99.40% for balanced samples and 98.05% for imbalanced samples.

Vasan et al. [72] designed a hybrid deep learning-based framework for malware detection. They proposed an approach for multi-class malware classification. Their algorithm converts the raw malware binary into both grayscale and color images and

utilize the fine-tuned CNN architecture previously trained with ImageNet dataset [25] which has more than 10 million images. Furthermore, during the fine-tuning, data augmentation method was used to improve the performance of their Image based Malware Classification using Fine-tuned Convolutional Neural Network Architecture (IMCFN) algorithm. They reported empirical evidence of achieving an accuracy of 98.82% in Malimg malware dataset and more than 97.35% for IoT-android mobile dataset.

From [73] the authors conducted experiments to compare the IMCFN performance with VGG16, ResNet50 and InceptionV3 already trained on ImageNet dataset [25]. As cited by [73] these state-of-the art architectures have previously been used to solve the multi-class malware family classification problem [74]– [76]. For this experiment, they reported an overall classification accuracy of 97.12% for VGG16 which represented a significant decline from their IMCFN accuracy of 98.82%, 98.61% for ResNet50 and 98.65% for InceptionV3 which represented decline from the IMCFN accuracy of 98.82%.

The authors in [77] proposed a flexible, innovative, and scalable deep learning-based detection mechanism leveraging Recurrent Neural Network (RNN), particularly Gated Recurrent Unit (GRU) to identify multi-class attacks effectively in an Android environment. For malware detection they used datasets comprised of 38842 APKs, that is, 3081 benign APKs had been collected from Androzoo [79] and 801 Malware APKs from Android Malware Dataset (AMD) [80]. AMD contains 10 diverse classes of malware (i.e., Backdoor, Trojan, Trojan-Banker, Trojan-clicker, Trojan-Dropper, Trojan-SMS, Trojan-Spy, Adware, HackerTool, Ransomware) with 71 distinct malware families [77]. The authors in [77] merged 6 different classes of trojan into single trojan class thus, the complete distribution of dataset was across 3 different classes including benign, Backdoor and Trojan. The distribution of the dataset used in [77] is shown in figure 2.2.

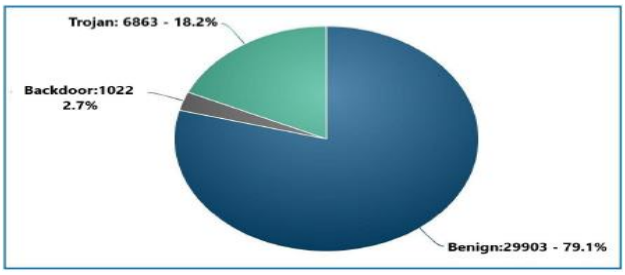


Figure 2.2 The Dataset Distribution Graph for Proposed scheme in [77][5]

Kim et al. [78] proposed 1D convolution filter-based classification network for classifying Android malware families from raw malware samples without any data preprocessing by investigating which parts (files or sections) in the malware sample are more effective for the classification of Android malware families. Their approach involved receiving input data in the form of strings from each part of a malicious Android file's manifest file, certificate file, and the sections of executable code. After this they connected the data in series and built a 1D convolution filter-based deep learning model to finally classify malicious apps into related families. They reported that they had observed that the Basic-1D-CNN with a low network depth and small parameters did show higher performance than 2D CNN based models namely SARVOTAM, EfficientNetB0, VGG16, ResNet50, InceptionV3, and LeNet. An accuracy of 93.2% was reported to have been achieved from experimental results done using the 1D convolution filter-based classification network with multi-streams.

The author [29] in his thesis performed experiments to gauge the difficulty in identifying each family of Adware from the benign apps without ads and benign apps with ads. He observed that the deep neural network multilayer perceptron (MLP) identifies each family with higher accuracy compared to traditional machine learning approaches.

Ding et al. [81] designed an approach that directly extracted bytecode file from an Android APK file, and then converted the bytecode file into a two-dimensional bytecode matrix. This was then followed by using a CNN, to train a detection model

[5] I. Bibi, et al. "A Dynamic DL-Driven Architecture to Combat Sophisticated Android Malware," in IEEE Access, vol. 8 in IEEE Access, vol. 8, pp. 129600-129612, 2020

and apply it to classify malware. CNNs can automatically learn features of bytecode file which can be used to recognize malware [81].

The authors in [82] developed a coloR-inspired convolutional neuRal networks (CNN)-based AndroiD malware Detection (R2-D2) system. Their system involved converting the bytecode of classes.dex from Android archive file to RGB color code and storing it as a color image with fixed size. The color image then provided input to the convolutional neural network for automatic feature extraction and training. In their paper they reported results showing that R2-D2 system worked well in detecting known Android malware and even unknown Android malware.

Yadav et al. [83] presented an EfficientNet-B4 CNN-based method for Android malware detection. It used image-based malware representations of the bytecode obtained from Android DEX files as input to the EfficientNet-B4 network to extract relevant features. The extracted features are then passed through a GAP layer and fed into a softmax classifier. They reported obtaining a 95.7% accuracy in separating malware from benign images. Their model was compared with 25 state-of-the-art pre-trained CNN-based models. Additionally, they also performed large-scale learning with SVM and RF classifiers and stacking in combination with CNN models.

Zhang et al. [84] proposed an Android malware detection model based on the time convolution neural network (TCN). Their approach used an image-based analysis method to avoid the malicious applications, which may have symmetric encryption, and confusion to evade the traditional static analysis process. Firstly, they directly read and fused the XML files and two DEX files to create gray image data sets with different feature combinations. Secondly, they then carried out experiments by using two-dimensional CNN and MobileNet v2 to analyze the differences of the feature combinations. From their results they found that adding visual features of XML files can reduce the dependence of the malware detection model on single DEX file features. This achieved a detection accuracy of 95.44%, precision of 95.45%, recall rate of 95.45%, and F1-Score of 95.44% for the TCN model. The experimental results showed that adding XML files is beneficial for Android malware detection [84].

2.5 Related Work

Dobhal et al. [32], analyzed the pertinence of machine learning based solutions to detect Android malware, particularly Adware. They used several machine learning algorithms namely Logistic Regression (LR), Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN), Classification and Regression Trees (CART), and Naive Bayes (NB). They used two approaches, one for binary classification and the other for multi-class classification. In the binary classification a maximum average f1-score of 94 % was achieved.

Ren et al. [39] developed MobiSentry, a lightweight defence system for malware classification and categorization on smartphones. Apart from conventional static features such as permissions and API calls, MobiSentry also uses the N-gram features of operation codes (n-opcode), which can be used to deal with known obfuscation techniques like rename obfuscation.

In [40] an Android malware detection system was proposed. The system was developed with the help of permissions, APIs, and with the presence of different key app information such as, the dynamic code, reaction code, native code, cryptographic code, database, etc. as features to train and build a classification model just by using various machine learning techniques which automatically distinguish malicious Android apps (malware) from the legitimate ones.

In the thesis [29] Suresh conducted investigation in analyzing Android Adware using binary and multiclass classification scenarios. In the experiments, he utilized machine learning supervised learning models namely Random Forest, Adaboost, Support Vector Machine (SVM) and a deep learning-based model called Multilayer Perceptron (MLP). In the binary classification experiments the author reported that using dynamic features alone for detecting Adware is not sufficient as the accuracy was only about 76%. However, when experimented with combined features he observed an improved accuracy value of 84%. In addition, he observed that though the overall accuracy for binary classification is higher for static features.

Suresh et al. [30], determined that there are some inherent difficulties in trying to distinguish Adware from benign. For one, strong (but simple) static features appear to yield models that overfit the training data. The fact that many benign apps display

ads in a very similar manner as Adware-infected apps was deemed to be a complicating factor.

Alani et al. [31] proposed an Android Adware detection system based on machine learning. They used features from the network flow of outgoing and incoming traffic to on an Android device.

In [33] research was carried out on detecting Android Adware using machine learning. The aim of their research was to identify the ten most important network traffic features for classification of the individual Adware families and improving the classification of the individual Adware families. This was in addition to identifying the ten most frequent network traffic features for classification of the Adware category.

2.6 Summary

In this chapter I gave a comprehensive literature review of numerous research work that has been done using computer vision techniques, deep learning and transfer learning in the classification and detection of Android malware and adware.

The chapter reviews several different approaches to detecting and classifying malware on conventional computer platforms and more specifically on the Android platform. The chapter highlighted different image processing techniques and malware visualization approaches.

Furthermore, the chapter provided insights on various research work showing the leveraging of transfer learning, which is a sub field of deep learning, in Android malware analysis. Lastly the chapter, looks at work that has been done in Android Adware detection and classification by various researchers.

CHAPTER THREE: METHODOLOGY

3.1 Research Design

In carrying out my research I used an experimental approach as it provided the appropriate method to answer the aim and objectives of this study. To answer the questions of the research, Adware and benign Android Application APK files were collected. Once collected the APK files were pre-processed to extract the DEX file from each APK file in the dataset. Each DEX file was then converted to a grayscale image to analyse and classify the Android Adware from Android benign applications using transfer learning.

The steps taken in the research are as outlined below and explained in detail in this chapter.

a) Data Collection and Preprocessing.
b) Adware and benign APK files image conversion.
c) Training of selected pre-trained neural network models using transfer learning.
d) Testing and evaluating the model.

Figure 3.1 illustrates my proposed approach in this study.

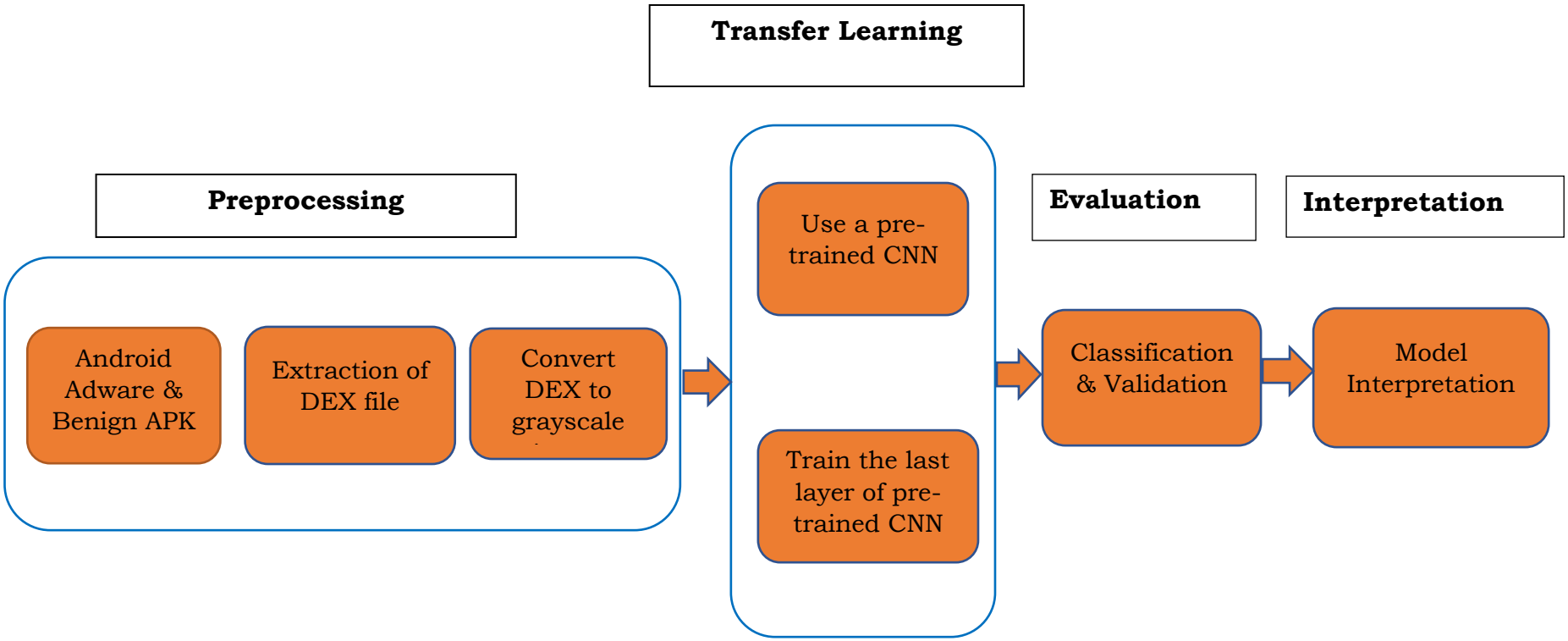


Figure 3.1 Proposed Research Approach

3.2 Preprocessing

3.2.1 Data Collection

The collected Android APK files were obtained from the Canadian Institute of Cybersecurity at New Brunswick University from the dataset named CICMalDroid2020 [85] – [86]. This sample was compiled between December 2017 and December 2018 and is publicly available for use by researchers.

The collected samples files were a total of three thousand and forty-eight (3,048) Adware and benign APK files. During the preprocessing some APK files were corrupted resulting in a lower number of files used in the experiments. The successfully processed files were broken down into, one thousand five hundred and fifteen (1515) android adware and one hundred and eight (108) benign APK files.

The files were downloaded using the available "Download Dataset" link on the Canadian Institute of Cybersecurity web site. The link navigates to a webform requesting user details of the person requesting to download the dataset, the webform described here is illustrated in figure 3.2. Once submitted the available APK files namely adware, banking, benign, riskware and short messaging service malware are presented for download as depicted in figure 3.3.

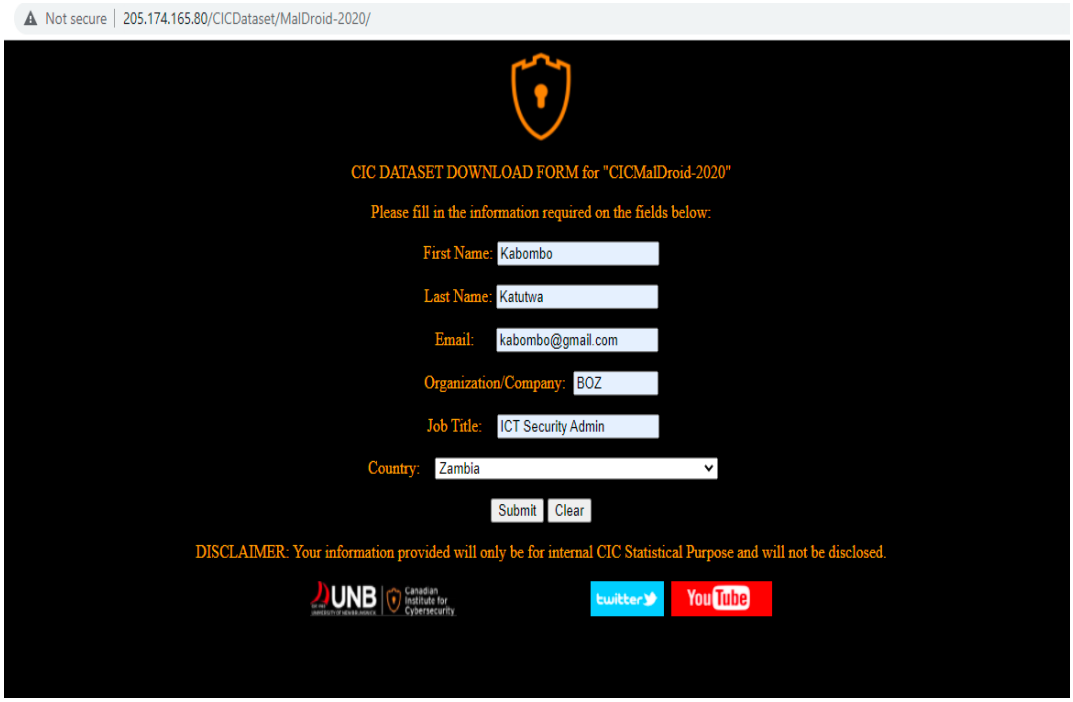


Figure 3.2 CIC Dataset Download form

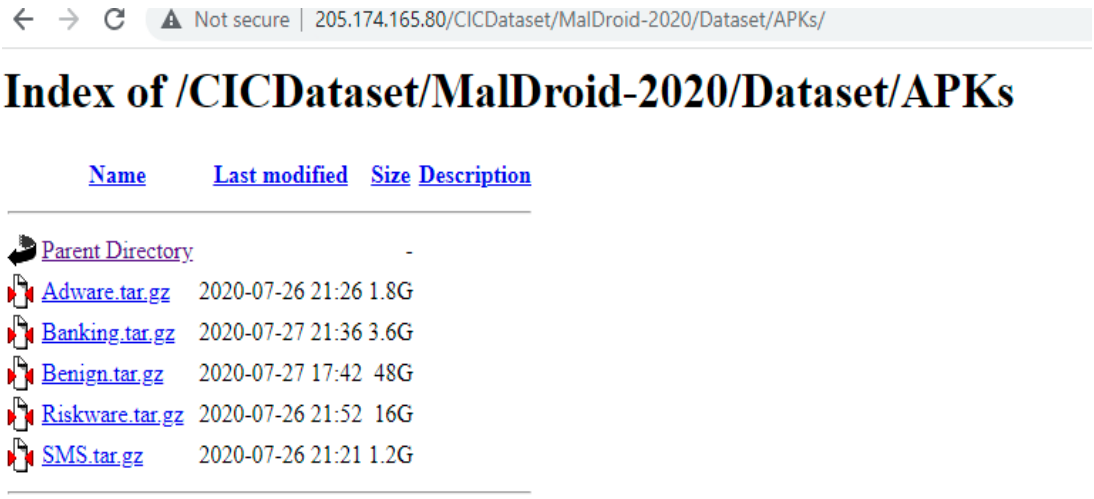


Figure 3.3 Link to download Dataset

To validate the downloaded sample files, a random selection was performed by picking a few of the APK files and analysing and confirming their Adware and benign status on the cloud-based malware analysis engines namely VirusTotal [87]and Hybrid Analysis [88].

3.2.2 Extraction of DEX files

To extract the classes.dex files from the collected dataset, the APK files were placed into separate directories of Adware and Benign files respectively. Using scripts developed using python, the APK files were converted to files with a zip file extension. The code to achieve this is shown in Figure 3.4 where the code is used to convert the android application package files to zip files

The next step was to extract the classes.dex files from the previously obtained zip files. This was achieved by traversing the directories storing the zipped adware and benign samples, extracting the classes.dex file and renaming each DEX file with the file name of the respective zipped file. Figure 3.5 depicts the python code used to achieve this.

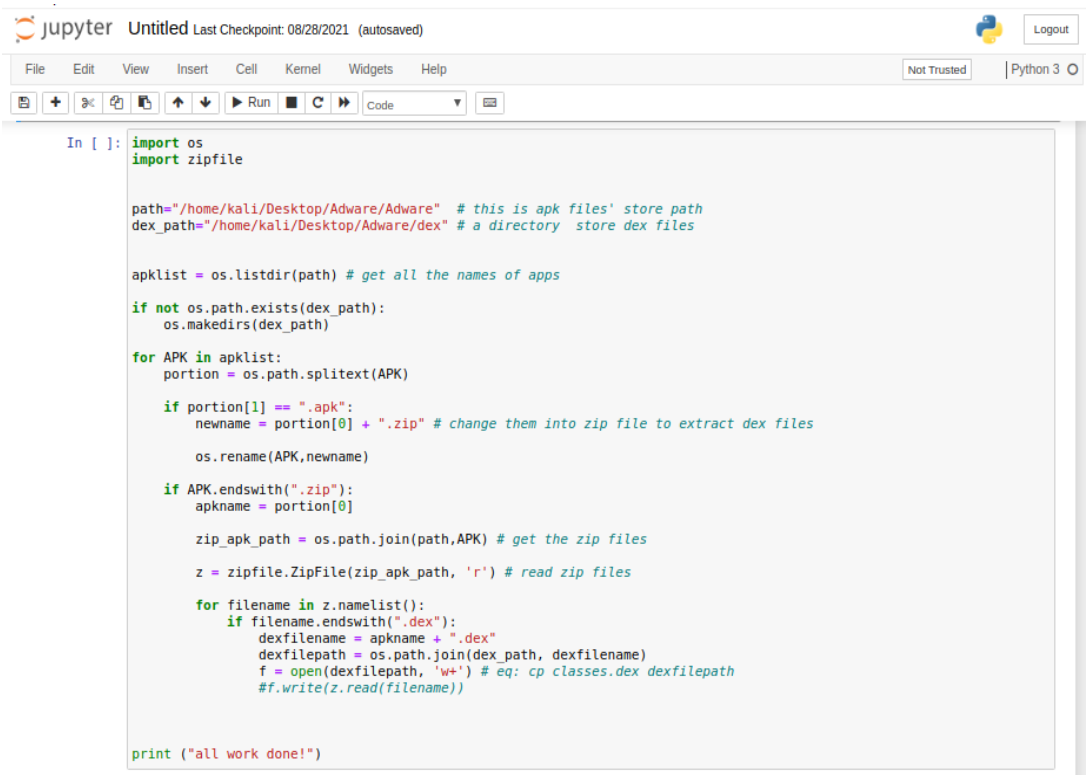


Figure 3.4 Python script to convert APK files to zip files

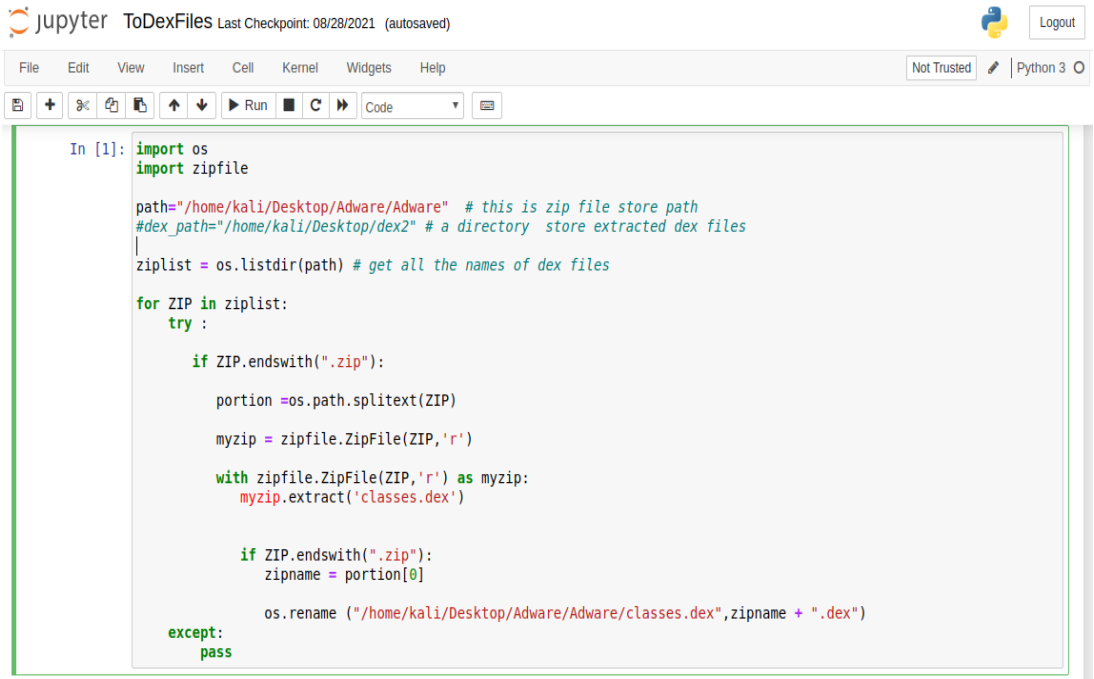


Figure 3.5 Python script to extract of DEX files

3.2.3 Conversion of DEX file to Image

Once the DEX files were extracted from each APK file in the dataset, they are processed and converted to portable network graphics (PNG) format grayscale

images. Each classes.dex file is represented as an array and then using the imageio class in the python programming language is converted to a grayscale image and saved as portable network graphics file. Figure 3.6 shows the python script that implements the DEX to image conversion.

jupyter ToImageScript Last Checkpoint: 08/28/2021 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Code

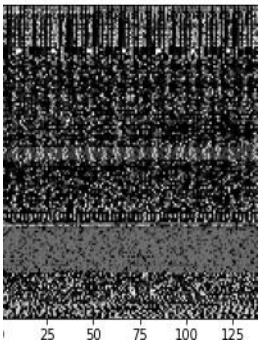
In [1]:

```
import os
import scipy
import array
import numpy
import imageio


path="/home/kali/Desktop/Adware/dex2"  # this is zip file store path
#dex_path="/home/kali/Desktop/dex2" # a directory  store extracted dex files

imglist = os.listdir(path) # get all the names of dex files

for IMG in imglist:


  try :

     portion = os.path.splitext(IMG)
     imgname = portion[0] + ".png"
     if IMG.endswith(".dex"):
          f=open(IMG,'rb');
          ln = os.path.getsize(IMG);
          width = 256;
          rem = ln%width;
          a = array.array("B");
          a.fromfile(f,ln-rem);
          f.close();
          k=int((len(a)/width))
          g = numpy.reshape(a,(k,width));
          g = numpy.uint8(g)
          imageio.imwrite(imgname,g);

  except:
        pass
```

Figure 3.6 Python script to convert DEX files to grayscale images

The image representation in figure 3.7 shows a sample of the converted Adware and benign DEX files.

(a) Adware



(b) Benign

Figure 3.7 Sample grayscale images

Once the images were ready, they were split into training, validation, and testing dataset per class. The ratio used is as shown in the table 3.1.

Table 3.1 Dataset distribution ratio

	Training Set	**Validation Set**	**Testing Set**
Adware Images	70%	20%	10%
Benign Images	70%	20%	10%

As the models were developed, the grayscale file image sizes were aligned to the default input image size of each pre-trained model that was used in this research as indicated in the table 3.2. The default sizes were retrieved from the documentation of the Keras and TensorFlow application package interfaces (APIs).

Table 3.2 Image input size

SNo	**Pre-Trained Model**	**Image input Size**
1.	VGG16	224 * 224
2.	ResNet 50	224 * 224
3.	Inception v3	299 * 299
4.	EfficientNet v2	300 * 300
5.	MobileNet v2	224 * 224

3.2.4 Dataset Labeling

To carry out this research I used supervised learning, and this requires labeling of that data samples of our dataset. The grayscale image files are placed in a directory structure with each class of files in its own directory as illustrated in figure 3.8

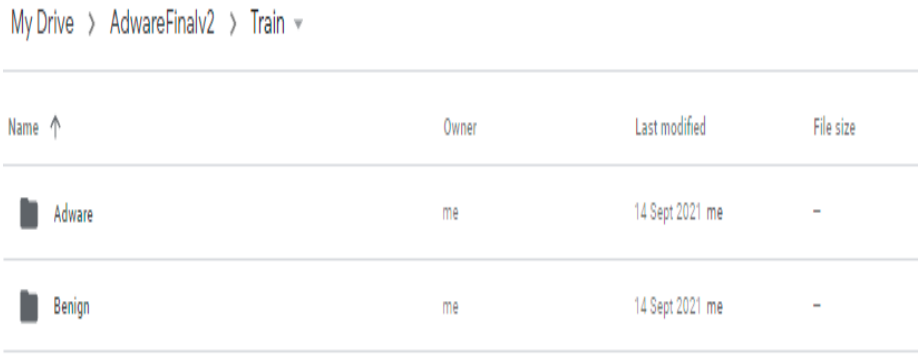


Figure 3.8 Dataset Folder layout

3.3 Transfer Learning

3.3.1 Proposed Models

The models used in the study were built using pre-trained models that were trained on a large-scale dataset called ImageNet [25]. It is a large database or dataset of over 14 million images. It was designed by academics intended for computer vision research. The models used are VGG16, ResNet50, Inception v3, EfficientNet v2 and MobileNet v2.

For the VGG16 model, I downloaded the weights trained on ImageNet [25]. The rest of the pre-trained models the features of the model are downloaded without each model's respective classification layer, this is as shown from figure 3.9 to figure 3.13 for each of the pre-trained models used in my research. The weights of each pre-trained model are frozen so that the knowledge learnt from previous training is leveraged and on the last layer is used to classify new image categories, in my research these are the adware and benign grayscale images. I

```
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

Figure 3.9 Freezing weights of VGG16 model

```
model = tf.keras.Sequential([
    hub.KerasLayer("https://tfhub.dev/google/imagenet/resnet_v1_152/feature_vector/5",
                   trainable=False),  # Can be True, see below.
    tf.keras.layers.Dense((len(folders)), activation='softmax')
])
model.build([None, 224, 224, 3])  # Batch input shape.
```

Figure 3.10 Freezing weights ResNet 50 model

```
model = tf.keras.Sequential([
    hub.KerasLayer("https://tfhub.dev/google/imagenet/inception_v3/feature_vector/5",
                   trainable=False),  # Can be True, see below.
    tf.keras.layers.Dense((len(folders)), activation='softmax')
])
model.build([None, 299, 299, 3])  # Batch input shape.
```

Figure 3.11 Freezing weights of Inception v3 model

```
model = tf.keras.Sequential([
    hub.KerasLayer("https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_b3/feature_vector/2",
                   trainable=False),  # Can be True, see below.
    tf.keras.layers.Dense((len(folders)), activation='softmax')
])
model.build([None, 300, 300, 3])  # Batch input shape.
```

Figure 3.12 Freezing weights of EfficientNet v2 model

```
model = tf.keras.Sequential([
    hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4",
                   trainable=False),  # Can be True, see below.
    tf.keras.layers.Dense((len(folders)), activation='sigmoid')
])
model.build([None, 224, 224, 3])  # Batch input shape.
```

Figure 3.13 Freezing weights of MobileNet v2 model

Figures 3.14 to 3.18 depict the summary of each of the pre-trained models that are to be trained. It can be seen from each model's summary that the models only trained on the parameters of the dense layer as the rest of the parameters are non-trainable as we had frozen the weights of the pre-trained model. From each figure (figure 3.14 to 3.18) it can be observed that the dense layer parameter value is equal to the trainable params value.

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 2)                 50178
=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
```

Figure 3.14 VGG 16 Model Summary

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 2048)              58295232
_________________________________________________________________
dense (Dense)                (None, 2)                 4098
=================================================================
Total params: 58,299,330
Trainable params: 4,098
Non-trainable params: 58,295,232
_________________________________________________________________
```

Figure 3.15 ResNet 50 model Summary

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 2048)              21802784
_________________________________________________________________
dense (Dense)                (None, 2)                 4098
=================================================================
Total params: 21,806,882
Trainable params: 4,098
Non-trainable params: 21,802,784
_________________________________________________________________
```

Figure 3.16 Inception v3 model summary

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1536)              12930622
_________________________________________________________________
dense (Dense)                (None, 2)                 3074
=================================================================
Total params: 12,933,696
Trainable params: 3,074
Non-trainable params: 12,930,622
_________________________________________________________________
```

Figure 3.17 EfficientNet v2 model summary

```
model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 2)                 2562
=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

Figure 3.18 MobileNet v2 model summary

3.3.2 Reducing overfitting

To train the models I set the number of epochs to a maximum of one hundred (100), this was arrived at after numerous instances of hyperparameter tuning. To control the likelihood of experiencing overfitting during the training of the models I used a technique known as early stopping. Early stopping is a method that allows the specifying of an arbitrary large number of training epochs and stopping training once the model performance stops improving on a holdout validation dataset.

In the experiments the early stopping was on the validation loss value as shown in figure 3.19.

```
es =EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
```

Figure 3.19 Early stopping parameters

The second control that was implemented to minimize overfitting was the use of the validation dataset. The objective was to have a set of images that the model has never seen during training to establish how well it would predict images that it has never previously seen.

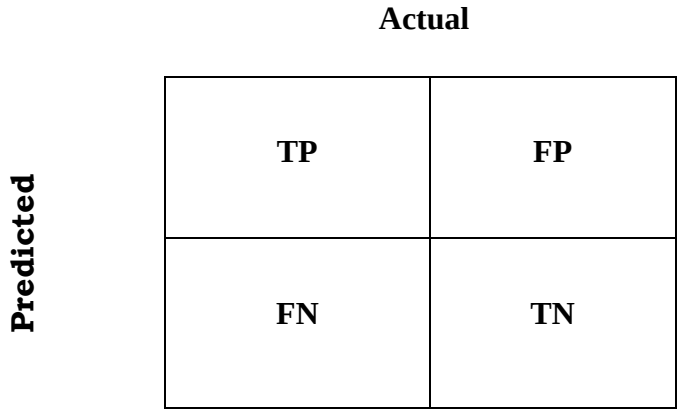
3.3.3 Training

The next step was to conduct transfer learning on the reshaped and resized images, where the base convolutional network already contains features that are generically useful for classifying images. The entire pre-trained model is not retrained, but the final classification part of each pre-trained model is specific to the original

classification task, and subsequently specific to the set of classes (Adware and Benign) on which the models are trained.

3.4 Evaluation

Training and validation accuracy were initially used to evaluate the performance of the models. However, it was established that a much better way to evaluate the performance of a classifier is to look at the confusion matrix. The general idea is to count the number of times instances of 'class A' are classified as 'class B'. Confusion matrix is a method used to evaluate a model's performance. In the matrix, classes are scored based on the instances of correct classification for a given class [89]. To evaluate the performance of models in this research I used a confusion matrix to derive recall, precision, F1-score and accuracy metrices. The equations and a short description of these metrices are given below.



TP = True Positive

FP = False Positive

FN = False Negative

TN = True Negative

Figure 3.20 An illustration of a Confusion Matrix

3.4.1 Quality Metrics

The following terms were used to quantify the results of the model using the confusion matrix:

a) **TP:** an image is an ADWARE image and classifier marks it is as ADWARE
b) **TN:** an image is a BENIGN image and classifier marks it is as BENIGN
c) **FP:** an image is a BENIGN image and classifier marks it is as ADWARE
d) **FN:** an image is an ADWARE image and classifier marks it is as BENIGN

Each of the metrics and their respective formula is defined below:

Accuracy: is the measure of all the correctly identified cases. It is most used when all the classes are equally important.

$$\text{Accuracy} = \frac{TP+TN}{(TP+FP+TN+FN)} \tag{3.1}$$

Precision: is the ratio of the probability that an app is classified as Adware app correctly.

$$\text{Precision} = \frac{TP}{TP+FP} \tag{3.2}$$

Recall: is the ratio of the total malicious apps that are classified as Adware.

$$\text{Recall} = \frac{TP}{TP+FN} \tag{3.3}$$

F-Measure or F1 score: It is a measure of a test's accuracy by considering both the precision and recall scores into a single measure of performance, where an F1-score is usually between 0.0 and 1.0 which closer to 1 is good while closer to 0.0 is poor performance.

$$\text{F –Measure (F1-score)} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.4}$$

On the test dataset using the Keras application package interface the confusion metrics was calculated for each model and the results were used to determine the recall, precision, F1-score and accuracy of each model.

3.4.2 Classification Results

The results of this research are reported using the accuracy of the training and test data. Furthermore, due to the imbalance of our dataset, I report the performance of the model's using precision, recall and F1-score for all the five (5) models that I experimented on in this research.

3.5 Interpretation

With the establishment of the performance metrices the effectiveness of the models was derived from these values, with the highest percentage values indicating the best performing model.

3.6 Development Environment

To perform the research work I used Jupyter Notebook development environment, for the dataset preprocessing, and Google Colaboratory or "Colab" for short was used to develop the models. Additionally, Anaconda Integrated Development Environment (IDE) was also used. This is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. Jupyter Notebook was the main component that was used in this development environment.

a) For the preprocessing, a system with the following specifications Intel (R) Core (TM) i3-6100U CPU @ 2.30GHz, 2304 MHZ, 2 Core(s), 4 Logical Processor(s)
b) 8 GB RAM and 256 Solid State Disk Drive

For the development and testing of the model, Colab was used. It provides free access to computing resources and has the following advantages:

a) It has pre-installed machine learning libraries such as Keras and TensorFlow.
b) It allows saving of work to the cloud.
c) Google Research provides their dedicated graphical processing units (GPUs) and tensor processing units (TPUs) for personal machine learning projects.

To mitigate the detection of the malware samples and the subsequent quarantining or deletion of the files, the preprocessing of the files was also performed on a Linux platform that did not have any anti-malware solution installed on it.

3.7 Summary

In this chapter, I gave an account of the methodology that was followed. In this study, a publicly available Android Adware and benign dataset was collected and preprocessed by extracting bytecode from the APK files of both the Android Adware and benign samples. The bytecode files were then converted to grayscale images, and these were fed as input to train the pre-trained neural networks. The chapter also explains how the selected pre-trained neural networks were used to build and train the models to detect Android Adware from benign files. Lastly the chapter describes the method used to evaluate the performance of the derived models and the development environment used to carry out the experiments in this research.

CHAPTER FOUR: RESULTS AND DISCUSSION

4.1 Experiments

In this chapter, I review the outcome of conversion of Android Adware and Benign APK files to grayscale images. Further I examine the evaluate the performance of the selected pre-trained models in detecting Android Adware and analyse the findings from the experiments that were conducted in this research.

4.1.1 Conversion of DEX files to Grayscale images

The extracted classes.dex files were successfully converted to grayscale images of varying sizes. The images were two (2) – dimensional thus providing appropriate input format for a CNN and do not require feature engineering which requires an expert skillset and is time-consuming.

4.1.2 VGG16 Trained Model

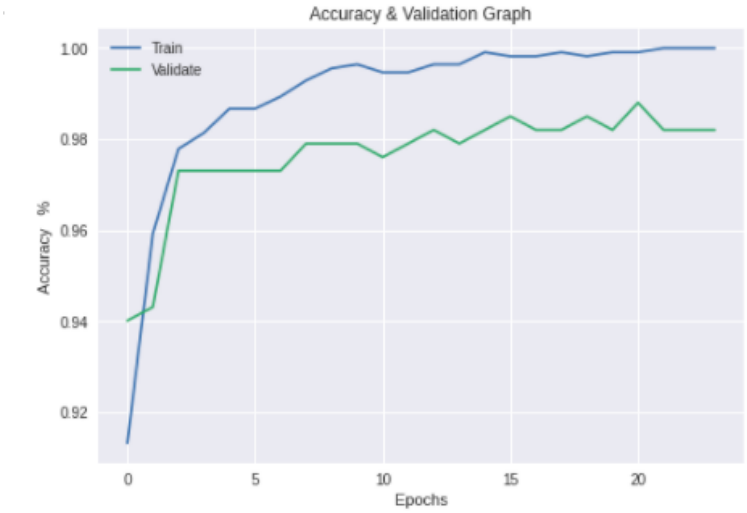


Figure 4.1 VGG16 Training & Validation Accuracy Graph

Fig. 4.1 depicts the training accuracy and validation graph of the VGG16 pre-trained model. The VGG16 model's training accuracy started at 91.59 % and went on to achieve the highest training accuracy of 99%. The validation accuracy started at 94.91% and reached a peak of 98.5%. These values began with a high value because the pre-trained model utilized the existing knowledge learnt previously on the ImageNet [25] database. The difference in the training and validation accuracies are an indication of the minimal model overfitting as the difference in values was not too wide. The training accuracy was derived from the training dataset and the validation accuracy shows how well the model performed on this data (the validation dataset)

that the model has not seen before. To circumvent overfitting the early stopping callback function was triggered, and the model training stopped after twenty-one (21) epochs and a duration of nineteen (19) minutes.

Figure 4.2 VGG16 Training & Validation Loss Graph

Fig. 4.2 depicts the training and validation loss graph of the VGG16 pre- trained model. The training loss indicates how well the model was fitting from the training data, while the validation loss is indicative of how well the model fits new data. The VGG16 model achieved the lowest training loss of 0.0076 at the 21st epoch and the lowest validation loss of 0.0532. It was observed that as the epochs increased the loss values decreased indicating that the model was generalizing the training and validation data well. However, the lack of convergence of the two graphs does indicate some overfitting. Once the validation loss value did not improve over five epochs the early stopping function was triggered.

4.1.3 ResNet 50 Trained Model

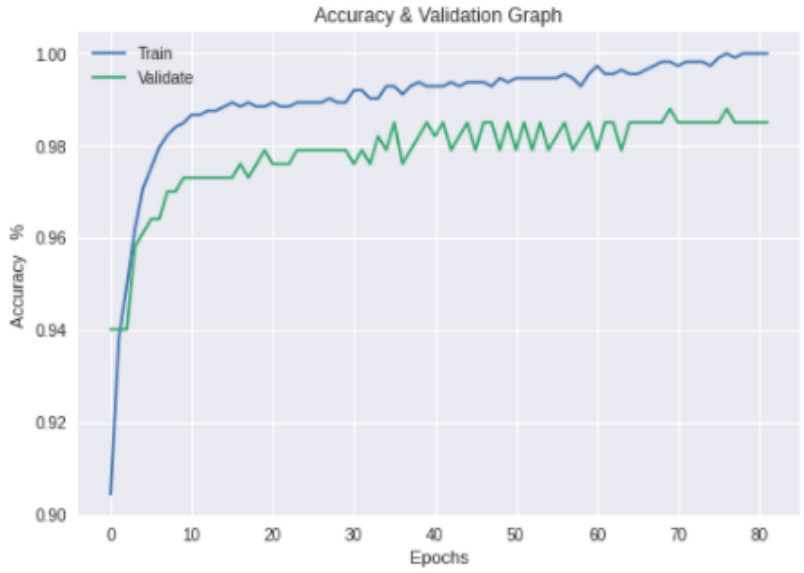


Figure 4.3 ResNet 50 Training & Validation Accuracy Graph

Fig. 4.3 depicts the accuracy and validation graph for the model developed using the ResNet 50 pre-trained model. The ResNet 50 model training and validation accuracy started at 90.43 % and 94.01 % respectively due to utilizing a pre-trained model which has knowledge acquired from previous learning. The training accuracy peaked to 99 % whereas the validation accuracy peaked to 98.5 %. The training duration for the ResNet 50 model in this experiment was forty-three (43) minutes and the training of the model run for eighty-two (82) epochs at which point the early stopping function call was triggered due to non-improvement of the validation loss value of five (5) epochs.

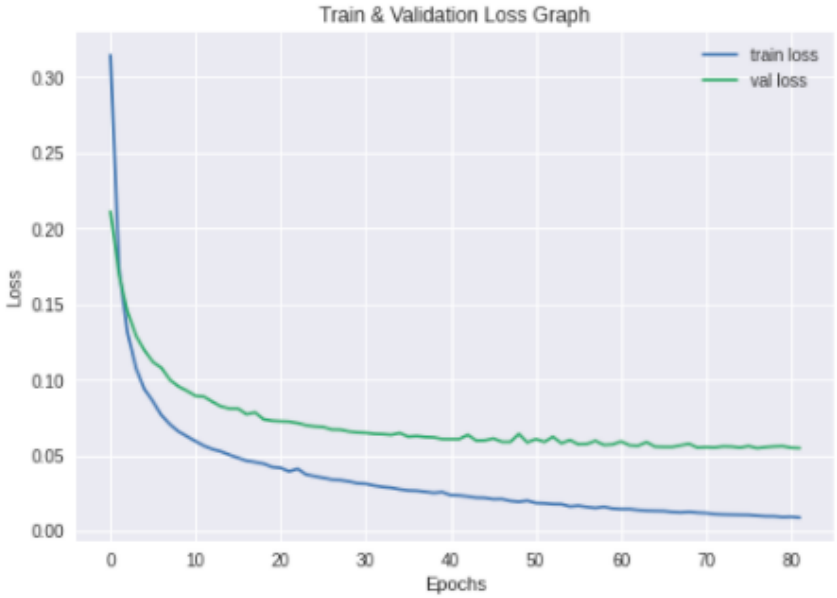


Figure 4.4 ResNet 50 Training & Validation Loss Graph

Fig. 4.4 depicts the training and validation loss graph of the ResNet 50 model. Both the training and validation loss showed a downward trend indicating good generalization of the model. However, low overfitting is still exhibited and can be seen by the gap between the two curves.

4.1.4 Inception v3 Trained Model

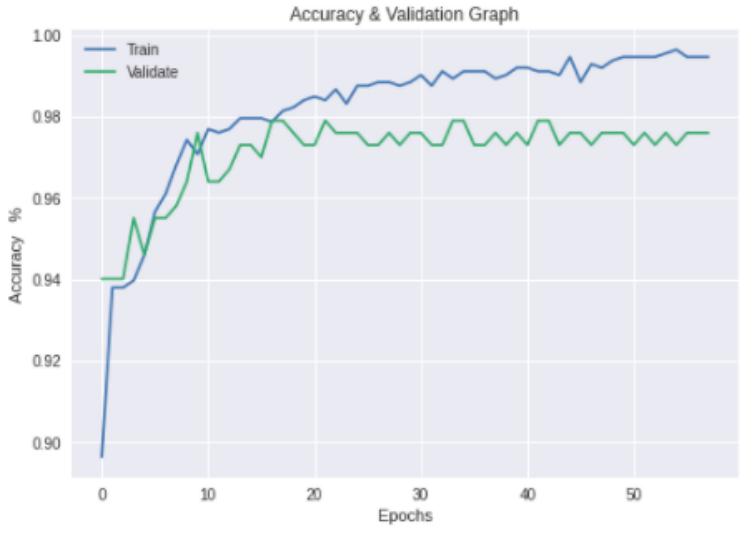


Figure 4.5 Inception v3 Training & Validation Accuracy Graph

Fig. 4.5 depicts the accuracy and validation graph of the model developed using the Inception v3 pre-trained model. The training accuracy achieved for this model reached a maximum of 99% while the maximum value achieved for the validation

accuracy was 97.6%. The difference between these accuracy values is slightly higher than in the VGG16 and ResNet 50 graphs, like the previous experiment results on the VGG16 and ResNet 50 models above this is indicative of minimal overfitting happening on the model. Due to this the model triggered the early stopping function after fifty-eight (58) epochs. The trigger executed after a duration was twenty-nine (29) minutes.

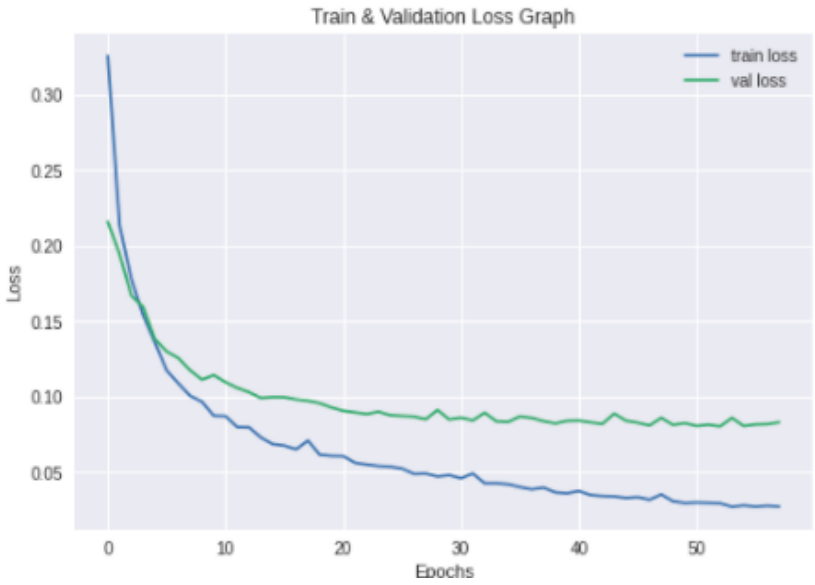


Figure 4.6 Inception V3 Training & Validation Loss Graph

Fig. 4.6 depicts the training and validation loss of the model developed using Inception v3 pre-trained model. As expected, both the training and validation loss showed a downward trend indicating good generalization of our model.

4.1.5 EfficientNet v2 Trained Model

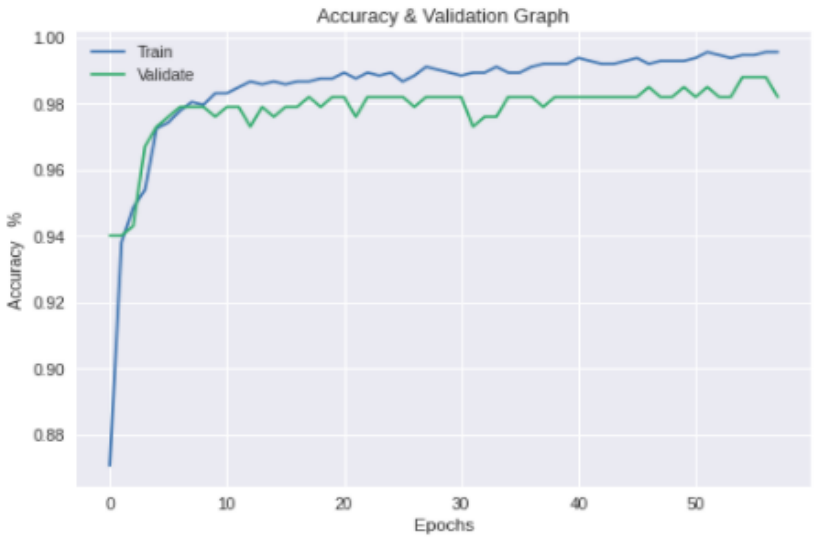


Figure 4.7 EfficientNet Training & Validation Accuracy Graph

Fig. 4.7 depicts the training and validation accuracy for the model developed using the EfficientNet pre-trained model. The training and validation accuracies started at 87.07% and 94.01% respectively. The performance rose steadily reaching a peak 99% for training accuracy and 98% for validation accuracy. Like the Inception v3 based model the training of the model run for fifty-eight (58) epochs at which point the early stopping function call was triggered, it however run for a longer duration of thirty-five (35) minutes.

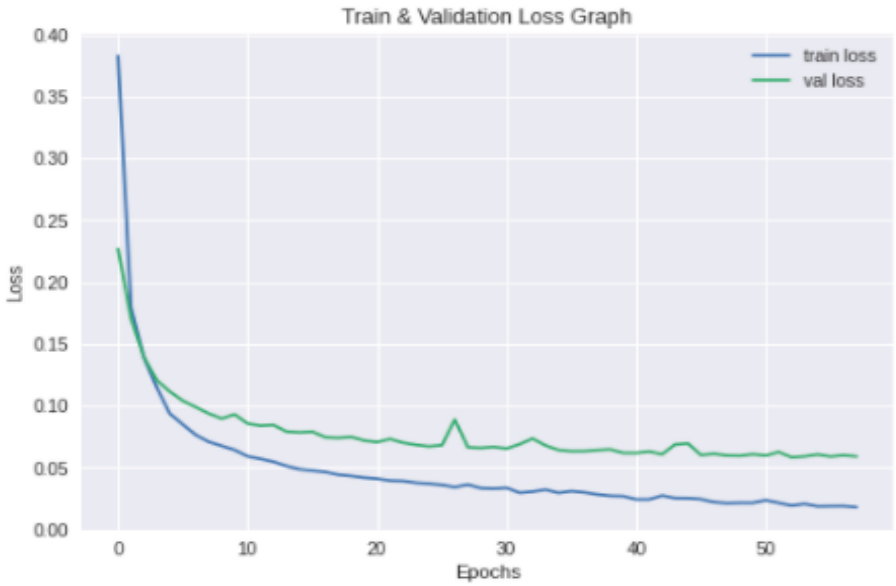


Figure 4.8 EfficientNet Training & Validation Loss Graph

The Figure 4.8 shows the training and validation loss graph of the EfficientNet model. The training loss and validation loss stood at 0.3828 and 0.2268 respectively at 1st epoch. The training loss decreased steadily to 0.0180 at the 58th epoch. Similarly, the validation loss also decreased steadily to 0.0590 at 58th epoch too. This steady decrease of both values is indicative of the model generalizing well on the input training and validation data.

4.1.6 MobileNet v2 Trained Model

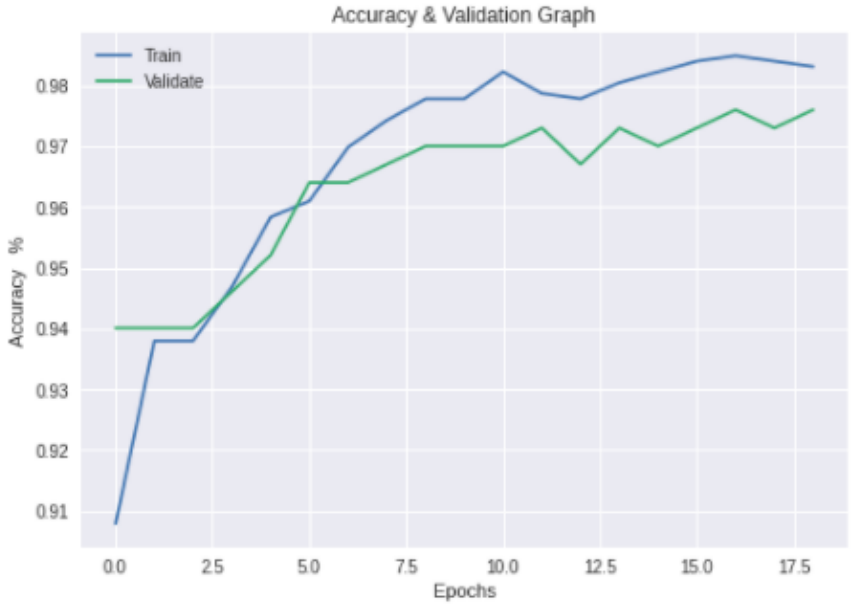


Figure 4.9 MobileNet v2 Training & Validation Accuracy Graph

Figure 4.9 depicts the performance of model developed using the MobileNet v2 pre-trained model. This experiment produced a performance of 98% for training accuracy and 97.6% for the validation accuracy. The training of the model run for nineteen (19) epochs at which point the early stopping function call was triggered after twenty-six (26) minutes.

Figure 4.10 MobileNet v2 Training & Validation Loss Graph

Fig. 4.10 depicts the training and validation loss graph of the MobileNet v2 pre-trained model. Our MobileNet v2 model achieves the lowest training loss of 0.0534 and the lowest validation loss of 0.0886 at the 19th epoch. It was observed that as the epochs increase the loss values decrease indicating that the model is generalizing the training and validation data well.

4.2 Performance Comparison

As highlighted previously, the dataset utilized in this research was imbalanced. This is due to having a higher number of Adware samples compared to the benign samples. Classification accuracy fails on classification problems with a skewed class distribution because of the intuitions developed by practitioners on datasets with an equal class distribution [91]. The reason that classification accuracy fails on classification problems with a skewed class distribution is because this metric is ideal on datasets with an equal class distribution. I therefore leveraged the use of the confusion matrix to overcome the challenge of a skewed dataset. The intention was to provide more insight into not only the accuracy of a predictive model, but also which classes are being predicted correctly, incorrectly, and what type of errors are being made by the model. Table 4.1 shows the matrices calculated using the confusion matrix for each model.

Table 04.1 Pre-Trained Model Performance Values

Pre-Trained Model	Accuracy	Precision	Recall	F1-Score
VGG 16	84%	91%	92%	91%
RESNET 50	81%	89%	91%	90%
INCEPTION V3	82%	90%	93%	91%
EFFICIENTNET	83%	90%	91%	90%
MOBILENET v2	83%	91%	94%	92%
AVERAGE	**83%**	**90%**	**92%**	**91%**

From the results collected during the experiments, it was observed that the accuracy of the different models was comparable even with the F1- score which averaged to ninety-one percent.

The training duration of the models varied with the VGG16 pre-trained model execution time being the least at nineteen minutes and the ResNet 50 pre-trained model took the longest by running for forty-five minutes. The number of epochs also varied with the MobileNet v2 model having the least number of epochs and ResNet 50 having the highest number of epochs. Refer to table 4.2 for the observed duration and epochs of each pre-trained model that was used.

Table 04.2 Pre-Trained Training Duration

Pre-Trained Model Used	Model Execution Duration (minutes)	Epochs
VGG – 16	19 min	21
RESNET 50	45 min	82
INCEPTION V3	29 min	58
EFFICIENTNET	35 min	58
MOBILENET	26 min	19

4.3 Overfitting

It was observed that the models exhibited minimal overfitting, this can also be observed from the validation and accuracy and the loss graphs. This can be attributed to the low number of sample files that I used to train the selected pre-trained models. To reduce this, early stopping was utilized to ensure that the training of the models stopped when there was no improvement on the validation loss of the graphs during training.

4.4 Comparison with prior work

Bagui and Benson [33], use network traffic data available in the CICAndMal2017 dataset by Lashkari et al. [90] to analyze and classify Adware families. They performed feature selection using information gain and classification was performed using traditional machine learning techniques, specifically J48 Decision Tree, Naïve Bayes and OneR. The results of the three classifiers, used as binary classifiers presented a highest average classification rate of 68% and highest average Attack Detection Rates (ADR) of 62.64% for the Adware families using the J48 Decision Tree classifier. This compared to the accuracy and F1-Score in my research had a higher rate of 83% and 91% respectively. This shows that the use of computer vision

in our research shows a significant improvement over previous work for classification of Adware malware done in [33].

When a comparison is made with the proposed solution in the work done in [31], it was observed that their detection accuracy had a higher accuracy and F1 score of 97.08 % and 97.09 % respectively compared to the accuracy and F1-score that was obtained in my research.

Additionally, we compare the work done by Dobhal et al. [32] in binary classification of Android Adware using various machine learning algorithms by illustrating results in the table 4.3.

Table 4.3 Comparison of the proposed system with state-of-the-art approaches

Machine Learning Algorithm	F1 - Score	Accuracy
Logistic Regression (LR)	88%	71.12%
Linear Discriminant Analysis (LDA)	88%	79.28%
K-Nearest Neighbors (KNN)	83%	89%
Classification And Regression Trees (CART)	94%	91.72%
Naive Bayes (NB)	87%	77%
Average in proposed approach	***92%***	***83%***

As can been seen in table 4, apart from the Classification And Regression Trees (CART) algorithm, on the F1 score our approach out performs the other algorithms used in [32].

In Suresh's dissertation [29] experiments were done with machine learning algorithms namely Random Forests (RF), Support Vector Classifier (SVC), Ad boost and MLP respectively showing that using dynamic features alone for detecting Android Adware had an accuracy of about 76%, however this improved when experimented with combined features to an accuracy of about 84%. This does show that my experimental results performed better when compared to the dynamic features approach used by [29]. However, the use of combined features approach in [29] performed better than my approach by 1% as my accuracy results averaged 83 %.

Figure 4.11 shows a bar graph that compares the results on my research with prior related work.

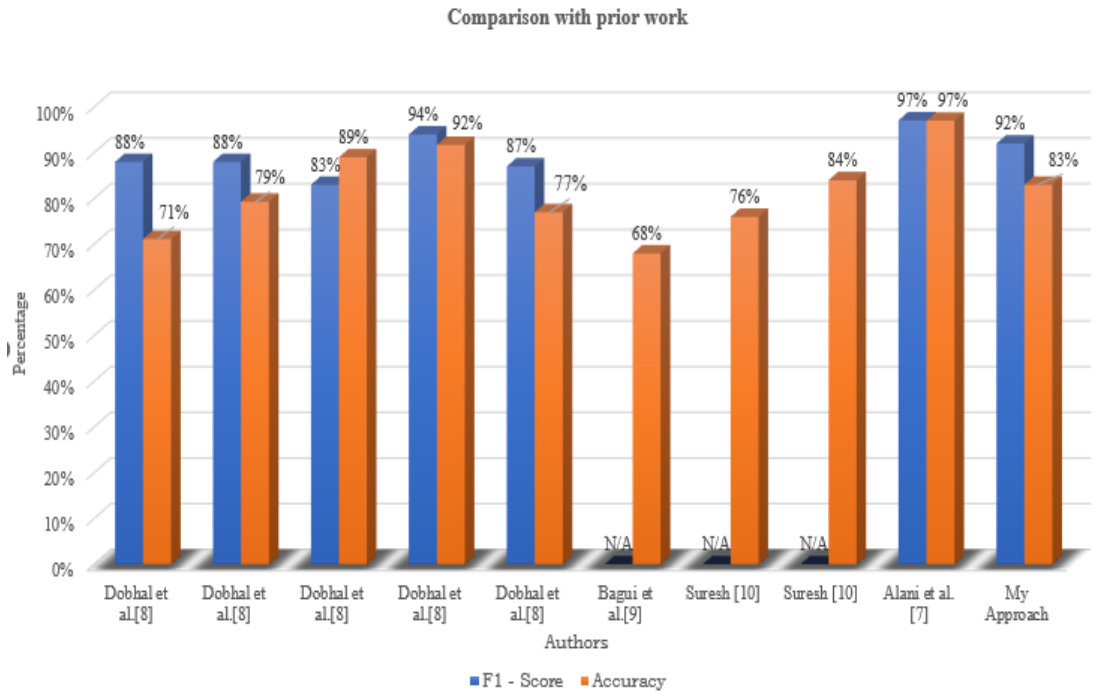


Figure 4.11 Comparison with prior work

4.5 Summary

In this chapter, I presented results of the training of the selected pre-trained neural networks. Firstly, we obtained the training and validation accuracy of each model, however due to the imbalanced dataset used in training each model a more reliable performance evaluation approach was used. I therefore used the confusion matrix to compute the Recall, Precision and F1-score metrics, with the F1-Score being the average metric of the Recall and Precision values. From the results obtained it clearly emerges that the trained models can successfully distinguish between Android Adware and benign apps. This is supported by the average of ninety-two percent value of the F1-score. Lastly in this chapter I also compared my experimental performance results with results observed from related prior work in this area of research.

CHAPTER FIVE: CONCLUSION AND RECOMMENDATION

My research presented an approach to detect Android Adware using visualization and transfer learning. The results obtained show that the proposed approach works well in classifying known Android Adware from Android benign apps, with high performance classification metrics that are comparable and, in some cases, outperforms prior work of detecting Android Adware in the literature that was reviewed.

5.1 Conclusion

In my research I was able to extract the bytecode files from the Android Adware and benign files, these were then successfully converted to grayscale images. The grayscale images were then used to develop models using transfer learning. During model training the training and validation accuracy high results indicated that the learning process of the model was very good. During the training of each model, the training and validation accuracies were compared. The two values showed a slight variance which indicated that some overfitting was present during model training, this was mitigated by fine tuning of each model's hyperparameters. The high values of both the training and validation accuracies provide assurance that the model training progressed well and would perform well in achieving the aim of my research.

The derived performance results, namely the recall, precision and f1-score indicate that using transfer learning and computer vision exemplifies that this approach is adequate in the detection of Android Adware.

I was later able to compare the performance of each of the selected pre-trained neural networks used in my research and determine which model provided the best outcomes in the detection of Android Adware from Android benign apps.

Based on the outcome of the experimental work done in this research we can deduce that the use of visualization, image processing techniques and transfer learning when applied to binary classification of Android Adware and Android Benign apps performs well as can be seen from the accuracy and f1 -Score metric.

In comparison, each pre-trained deep neural network's performance results showed a small variance on all the performance metrics of accuracy, precision, recall and f1-score with the MobileNet v2 obtaining the highest f1-score. This leads me to

conclude that pre-trained models that are created and adequately trained to solve a similar problem or task can be used to successfully detect Android Adware from benign apps when trained using an appropriate dataset. However, in contrast there was notable variation in the time taken to train the models with the VGG16 and ResNet 50 taking the shortest time and taking the longest time respectively.

The experimental study was able to meet the set objectives with limitations noted and recommended as future work and areas of improvement.

The study has shown that indeed transfer learning and computer vision sufficiently achieve the intended objectives that the overall performance is adequate.

From the outcome of my work in this research the overall conclusion is that the use of computer vision adequately detects Adware from benign android apps.

5.2 Recommendations

The work in this research aimed to establish the use of computer vision and transfer learning in the binary classification of Android Adware verses Android benign app. The study was able to show that this approach performs well and is comparable in performance and in some instances outperforms other methods. I therefore recommend for adoption of this approach in classifying and detecting Android Adware from benign apps. Furthermore, because of this work I recommend the use of techniques used in this research in overcoming malware authors' malware anti-detection techniques such as obfuscation techniques. To help improve the results of this research I also recommend regular review and training of newly developed and well performing pre-trained neural networks. I also recommend the use of the approach used in my research to enhance anti-malware solutions that rely on traditional signature-based detection methods.

5.3 Future work

For future work, research can be done that uses the proposed approach to conduct multi-class classification of Android Adware into its various families. In addition, other possible future work would be to combine the DEX file and other files in the APK, the subsequently convert the resulting files into images that are applied to pre-trained CNN. Developing mobile app, based on the work in this research, that can detect Android Adware in real-time while downloading the APK files from the google playstore can be pursued.

REFERENCES

[1] De Wit .J, "Dynamic detection of mobile malware using real-life data and machine learning", Masters Thesis, University of Twente, 2018.

[2] Tam .K, Feizollah .A, Anuar .N.B, Salleh .R, and Cavallaro .L, “The evolution of Android malware and Android analysis techniques”, ACM Computing Surveys (CSUR), vol. 49, no. 4, p. 76, 2017.

[3] https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/[accessed 11th February 2022].

[4] https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ [accessed 11th February 2022].

[5] Sophos Ltd, SophosLabs 2018 Malware Forecast. Oxford, UK: Abingdon Science Park, 2017.

[6] Jiang .X, Mao .B ,Guan .J and Huang .X, "Android Malware Detection Using Fine-Grained Features", Scientific Programming, vol. 2020, pp. 1-2, 2020. Available: https://www.hindawi.com/journals/sp/2020/5190138/. [Accessed 24 June 2020].

[7] Mobile Operating System Market Share Worldwide | Statcounter Global Stats", StatCounter Global Stats, 2020. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide. [Accessed: 23- May- 2020].

[8] Sophos Ltd, SophosLabs 2018 Malware Forecast. Oxford, UK: Abingdon Science Park, 2017.

[9] McAfee, McAfee Mobile Threat Report Q1, 2019. McAfee, 2019.

[10] Lu .N, Li .D, Shi .W, Vijayakumar .P, Piccalilli .F and Chang .V, "An efficient combined deep neural network-based malware detection framework in 5G environment", Computer Networks, vol. 189, p. 107932, 2021. Available: 10.1016/j.comnet.2021.107932 [Accessed 25 June 2022].

[11] Gao .J, Li .L, Kong .P, Bissyandé .T.F and Klein .J, "Should You Consider Adware as Malware in Your Study?," 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2019, pp. 604-608, doi: 10.1109/SANER.2019.8668010.

[12] Avast Reports Continued Dominance of Adware Among Android Threats. 2021. Avast Reports Continued Dominance of Adware Among Android Threats. [online] Available at: <https://press.avast.com/avast-reports-continued-dominance-of-adware-among-android-threats> [Accessed 16 June 2022].

[13] Bommisetty .S, Tamma .R, Skulkin .O and Mahalik .H, Practical Mobile Forensics, 4th ed. Birmingham: Packt Publishing, 2020, pp. 159 - 160.

[14]2020.Online].Available:https://developer.Android.com/guide/platform. [Accessed: 20- Jun- 2020].

[15] Bakour, K., Ünver, H.M. & Ghanem, R. The Android malware detection systems between hope and reality. SN Appl. Sci. 1, 1120 (2019). https://doi.org/10.1007/s42452-019-1124-x

[16]“ Topark-ngarm, P., "Identifying Android Malware Using Machine Learning Based Upon Both Static ”nd Dynamic Features", Masters, Victoria University of Wellington, 2014.

[17] Brownlee, B., Deep Learning for Computer Vision, 1st ed. Machine Learning Mastery., 2020, p. 5 - 6.

[18] Planche .P and Andres .E, Hands-On Computer Vision with TensorFlow 2, 1st ed. Birmingham: Packt Publishing Ltd., 2019, p. 31.

[19] Bensaoud, A., Abudawaood, N., & Kalita, J. (2020). Classifying malware images with convolutional neural network models. International Journal of Network Security, 22(6), 1022-1031.

[20] Saxe, J. and Sanders, H., 2018. Malware data science. 1st ed. San Francisco: No Starch Press, Inc., p.177.

[21] Vu .L.N. and Jung .S, "AdMat: A CNN-on-Matrix Approach to Android Malware Detection and Classification," in IEEE Access, vol. 9, pp. 39680-39694, 2021, doi: 10.1109/ACCESS.2021.3063748.

[22] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, and Jianfei Cai. Recent Advances In Convolutional Neural Networks. Pattern Recognition, 77:354377, 2018.

[23] Sewak .M, Karim .M , and Pujari .P, 2018. Practical Convolutional Neural Networks. 1st ed. MUMBAI: Packt Publishing, p32.

[24] Waizel .A, "Improving the Detection of New Malware Classes using Transfer Learning", Master of Science, BEN-GURION UNIVERSITY OF THE NEGEV, 2015.

[25] "ImageNet," www.image-net.org. https://www.image-net.org/index.php. Accessed 1st January 2022

[26] Mohanta .A and Saldanha .A, 2020. Malware analysis and detection engineering. 1st ed. New–York: Apress, pp.5 - 7.

[27] Jung .J, Choi .J, Cho .S, Han .S, Park .M and Hwang .Y, 2018. Android malware detection using convolutional neural networks and data section images. Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems.

[28] Singh .A, 2017. Malware Classification using Image Representation. Master of Technology. INDIAN INSTITUTE OF TECHNOLOGY KANPUR.

[29] Suresh .S, 2018. Analyzing Android Adware. Master of Science (MS). San Jose State University.

[30] Suresh .S, Di Troia .F, Potika .K and Stamp .M, 2018. An analysis of Android adware. Journal of Computer Virology and Hacking Techniques, 15(3), pp.147-160.

[31] Alani .M and Awad .A, "AdStop: Efficient flow-based mobile adware detection using machine learning", Computers & Security, vol. 117, p. 102718, 2022. Available: 10.1016/j.cose.2022.102718

[32] Dobhal .D, Das .P, Aswal .K, 2019. Detection of Android Adwares by using Machine Learning Algorithms. International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8 Issue-4S, April 2019

[33] Bagui .S and Benson .D , "Android Adware Detection Using Machine Learning", International Journal of Cyber Research and Education, vol. 3, no. 2, pp. 1-19, 2021. Available: 10.4018/ijcre.2021070101.

[34] Abdul Kadir, A. F. (2018). A detection framework for android financial malware. Doctor of Philosophy. THE UNIVERSITY OF NEW BRUNSWICK.

[35] Dunham .K, Android Malware and Analysis, 1st ed. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2015.

[36] Dhalaria .M. and Gandotra .E, 2019. Convolutional Neural Network for Classification of Android Applications Represented as Grayscale Images. International Journal of Innovative Technology and Exploring Engineering, 8(12S), pp.835-843.

[37] Fang .Y, Gao .Y, Jing .F. and Zhang .L, 2020. Android Malware Familial Classification Based on DEX File Section Features. IEEE Access, 8, pp.10614-10627.

[38] Naway .A, Li .Y, 2019. Using Deep Neural Network for Android Malware Detection. International Journal of advanced studies in Computer Science and Engineering (IJASCSE) VOLUME 7 ISSUE 12, 2018, pg. 9-18.

[39] Ren .B, Liu .C, Cheng .B, Guo and Chen .J, "MobiSentry: Towards Easy and Effective Detection of Android Malware on Smartphones", Mobile Information Systems, vol. 2018, pp. 1-14, 2018. Available: 10.1155/2018/4317501

[40] Koli .J. D. “RanDroid: Android malware detection using random machine

learning classifiers.” 2018 Technologies for Smart-City Energy Security and Power (ICSESP). IEEE ,2018.

[41] Koli J. D. “RanDroid: Android malware detection using random machine

learning classifiers.” 2018 Technologies for Smart-City Energy Security and Power (ICSESP). IEEE,2018.

[42] Haralick .R.M, Shanmugam .K, Dinstein .I, Textural features for image classification. IEEE Trans Syst Man Cybern. 1973;3(6):610-621.

[43] Nataraj .L, Karthikeyan .K, Jaco .G and Manjunath .B, "Malware images", Proceedings of the 8th International Symposium on Visualization–for Cyber Security - VizSec '11, 2011. Available: 10.1145/2016904.2016908

[44] Han .K. S, Lim .J. H, and Im .E. G. (2013). Malware analysis method using visualization of binary files. In Proceedings of Research in Adaptive and Convergent Systems ACM.

[45] Gennissen .J, Cavallaro .L, Moonsamy .V and Batina .L, Gamut: sifting through images to detect android malware, Bachelor thesis, Royal Holloway University, London, UK,2017.

[46] Jung .J, Choi .J, Cho .S, Han .S, Park .M and Hwang .Y, "Android malware detection using convolutional neural networks and data section images", Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems, 2018. Available: 10.1145/3264746.3264780

[47] Tan .G, Lan .Z, Xudong .L, Wen .C, Malware detection based on semi-supervised learning with malware visualization[J]. Mathematical Biosciences and Engineering, 2021, 18(5): 5995-6011. doi: 10.3934/mbe.2021300

[48] Thakur .D, Singh .J, Faruki .P, Gera .T, Classification of Android Malware using its Image Sections. International Journal of Advanced Trends in Computer Science and Engineering, volume 9 No. 4, July - August ,2020, doi.org/10.30534/ijatcse/2020/288942020

[49] Chen .H, Du .R , Liu .Z and Xu .H, "Android Malware Classification using XGBoost based on Images Patterns," 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC), 2018, pp. 1358-1362, doi: 10.1109/ITOEC.2018.8740537.

[50] Sarkar .D, Bali .R and Ghosh .T, Hands-on transfer learning with Python, 1st ed. Birmingham: Packt Publishing, 2018, p. 170.

[51] Yosinski .J, Clune .J, Bengio .Y and Lipson .H., 2014. How transferable are features in deep neural networks?.Advances in Neural Information Processing Systems 27, pages 3320-3328. Dec. 2014, arXiv preprint arXiv:1411.1792.

[52] Prima .B and Mohammed B (2020). USING TRANSFER LEARNING FOR MALWARE CLASSIFICATION. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLIV-4/W3-2020. 343-349. 10.5194/isprs-archives-XLIV-4-W3-2020-343-2020.

[53] Xiao, Guoqing, Jingning .L, Yuedan .C and Li .K 2020. MalFCS: An Effective Malware Classification Framework with Automated Feature Extraction Based on Deep Convolutional Neural Networks. Journal of Parallel and Distributed Computing 141: 49–58.

[54] Chen .L. (2018). Deep Transfer Learning for Static Malware Classification. ArXiv, abs/1812.07606.

[55] Khan .R.U, Zhang .X & Kumar .R, Analysis of ResNet and GoogleNet models for malware detection. J Comput Virol Hack Tech 15, 29–37 (2019). https://doi.org/10.10h 15, 29–37 (2019). https://doi.org/10.1007/s11416-018-0324-z

[56] Mohammed .T. M, Nataraj .L, Chikkagoudar .S, Chandrasekaran .S, & Manjunath .B. S. (2021). Malware Detection Using Frequency Domain-Based Image Visualization and Deep Learning. arXiv preprint arXiv:2101.10578

[57] Singh .J, Thakur .D, Ali .F, Gera .T, and Kwak .K.S, "Deep Feature Extraction and Classification of Android Malware Images," Sensors, vol. 20, no. 24, p. 70 13, Jan. 2020, doi: 10.3390/s20247013.

[58] Arp .D, Spreitzenbarth .M, Hübner .M, Gascon .H, Rieck .K , Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 23–26 February 2014; pp. 23–26.

[59] Lo .W, Yang .X and Wang .Y, "An Xception Convolutional Neural Network for Malware Classification wit" Transfer Learning," 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 2019, pp. 1-5, doi: 10.1109/NTMS.2019.8763852.

[60] Dhalaria .M and Gandotra .E, "Convolutional Neural Network for classification of Android applications represented as Grayscale Images," International Journal of Innovative Technology and Exploring Engineering, vol. 8, no. 12S, pp. 835–843, 2019.

[61] Zou .K, Luo X, Liu .P, Wang .W, Wang .H (2020) ByteDroid: Android Malware Detection Using Deep Learning on Bytecode Sequences. In: Han W., Zhu L., Yan F. (eds) Trusted Computing and Information Security. CTCIS 2019.

Communications in Computer and Information Science, vol 1149. Springer, Singapore. https://doi.org/10.1007/978-981-15-3418-8_12

[62] Hochreiter .S and Schmidhuber .J, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[63] Ma .Z, Ge .H, Wang .Z, Liu .Y, and Liu .X, “Droidetec: Android malware detection and malicious code localization through deep learning,” 2020, arXiv:2002.03594. [Online]. Available: http://arxiv.org/abs/2002.03594

[64] Li .Y, Jang .J, Hu .X, and Ou .X, “Android malware clustering through malicious payload mining,” in International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, 2017, pp. 192–214.

[65] Wang .Z, Liu .Q and Chi .Y, "Review of Android Malware Detection Based on Deep Learning," in IEEE Access, vol. 8, pp. 181102-181126, 2020, doi: 10.1109/ACCESS.2020.3028370.

[66] Ren .Z, Wu .H, Ning .Q, Hussain .I and Chen .B, “End-to-end malware detection for Android IoT devices using deep learning,” Ad Hoc Networks. Vol. 101, Apr. 2020, Art. no. 102098.

[67] Su .X, Zhang .D, Li .W, and Zhao .K, “A deep learning approach to Android malware feature learning and detection,” in Proc. IEEE Trust-com/BigDataSE/ISPA, Aug. 2016, pp. 244-25

[68] Alotaibi .A, "Identifying Malicious Software Using Deep Residual Long-Short Term Memory," in IEEE Access, vol. 7, pp. 163128-163137, 2019, doi: 10.1109/ACCESS.2019.2951751.

[69] Arp .D, Spreitzenbarth M, Hübner .M, Gascon .H and Rieck .K, "DREBIN: Effective and explainable detection of Android malware in your pocket", Proc. NDSS, pp. 1-15, 2014.

[70] Spreitzenbarth .M, Freiling .F, Echtler .F, Schreck .T and Hoffmann .J, "Mobile-sandbox: Having a deeper look into Android applications", Proc. 28th Annual. ACM Symposium. Appl. Computer., pp. 1808-1815, 2013.

[71] Almomani .I, Alkhaye .A and El-Shafai .W, "An Automated Vision-Based Deep Learning Model for Efficient Detection of Android Malware Attacks," in IEEE Access, vol. 10, pp. 2700-2720, 2022, doi: 10.1109/ACCESS.2022.3140341.

[72] Vasan .D, Alazab .M, Wassan .S, Naeem .H, Safaei .B and Zheng .Q, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture", Computer Networks, vol. 171, p. 107138, 2020. Available: 10.1016/j.comnet.2020.107138.

[73] Shaha .M and Pawar .M, “Transfer Learning for Image Classification,” in Proceedings of the 2nd International Conference on Electronics, Communication and Aerospace Technology, CECA 2018, 2018, pp. 656–660.

[74] Rezende .E, Ruppert .G, Carvalho .T, Theophilo .A, Ramos .F, and De Geus .P, “Malicious Software Classification Using VGG16 Deep Neural Network’s Bottleneck Features,” in Advances in Intelligent Systems and Computing, 2018.

[75] Rezende .E, Ruppert .G, Carvalho .T, Ramos .F and De Geus .P, “Malicious software classification using transfer learning of ResNet-50 deep neural network,” in Proceedings - 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, 2018.

[76] Das .S, “CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more,” Medium, 2017.

[77] Bibi .I, Akhunzada .A, Malik .J, Iqbal .J, Musaddiq .A and Kim .S, "A Dynamic DL-Driven Architecture to Combat Sophistica”ed Android Malware," in IEEE Access, vol. 8, pp. 129600-129612, 2020, doi: 10.1109/ACCESS.2020.3009819.

[78] Kim .H.I, Kang .M, Co . S. J and Choi S. I, "Efficient Deep Learning Network With Multi-Streams for Android Malware Family Classification," in IEEE Access, vol. 10, pp. 5518-5532, 2022, doi: 10.1109/ACCESS.2021.3139334.

[79] Allix .K, Bissyandé . T. F, Klein .J, and Traon Y. Le, “AndroZoo: collecting millions of Android apps for the research community," in Proc. 13th Int. Workshop Mining Software. Repositories MSR, 2016, pp. 468-471, doi:10.1145/2901739.2903508.

[80] Wei . F, Li . Y, Roy . S, Ou .X, and Zhou .W, “Deep ground truth analysis of current Android malware," in Proc. Int. Conf. Detection Intrusions malware, Vulnerability Assessment. Springer, 2017, pp. 252-276.

[81] Ding .Y, Zhang .X, Hu .J. et al. Android malware detection method based on bytecode image. J Ambient Intell Human Comput (2020). https://doi.org/10.1007/s12652-020-02196-4

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

[83] Yadav .P, Menon .N, Ravi .V, Vishvanathan .S, Pham .D.T, EfficientNet Convolutional Neural Networks-based Android Malware Detection”, Computers & Security (2022),doi: https://doi.org/10.1016/j.cose.2022.102622

[84] Zhang .W, Luktarhan .N, Ding .C, Lu .B, Android Malware Detection Using TCN with Bytecode Image. Symmetry 2021, 13, 1107.https://doi.org/10.3390/sym13071107

[85] Mahdavifar .S, Kadir .A.F.A, Fatemi .R, Alhadidi .D, Ghorbani .A.A, Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning, The 18th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC), Aug. 17-24, 2020.

[86] Mahdavifar .S, Alhadidi .D, and Ghorbani A.A (2022). Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder, Journal of Network and Systems Management 30 (1), 1-34

[87]"VirusTotal", Virustotal.com, 2022. [Online]. Available: https://www.virustotal.com/. [Accessed: 28- Aug- 2021].

[88]"Free Automated Malware Analysis Service - powered by Falcon Sandbox", Hybrid-analysis.com, 2022. [Online]. Available: https://www.hybrid-analysis.com/. [Accessed: 28- Aug- 2021].

[89] Géron .A, Hands-on machine learning with Scikit-Learn and TensorFlow, 2nd edition. Sebastopol, CA: O'Reilly, 2019, p. 92.

[90] Lashkari . A.H, Kadir .A.F.A , Taheri .L, and Ghorbani Ali .A, “Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification”, In the p^{ro}ceedings of the 52nd IEEE International Carnahan Conference on Security Technology (ICCST), Montreal, Quebec, Canada, 2018.

[91] Brownlee .J, "Failure of Classification Accuracy for Imbalanced Class Distributions", Machine Learning Mastery, 2022. [Online]. Available: https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/. [Accessed: 30- Mar- 2022].

APPENDIX

Ethical Clearance

THE UNIVERSITY OF ZAMBIA
DIRECTORATE OF RESEARCH AND GRADUATE STUDIES

Great East Road Campus | P.O. Box 32379 | Lusaka10101 | Tel: +260-211-290 258/291 777
Fax: (+260)-211-290 258/253 952 | **E-mail**: director.drgs@unza.zm | Website: www.unza.zm

APPROVAL OF STUDY

IORG No. 0005376
NASRECREC IRB No. 00006465

30th January, 2023

REF NO. NASREC-2023-JAN - 007.
Mr. Kabombo
Katutwa,
The University of Zambia,
School of Natural Sciences,
P.O. Box 32379,
LUSAKA.

Dear Mr. Katutwa,

RE: "DETECTION OF ANDROID ADWARE USING TRANSFER LEARNING WITH COMPUTER VISION"

Reference is made to your protocol dated as captioned above. NASREC resolved to approve this study and your participation as Principal Investigator for a period of one year.

REVIEW TYPE	ORDINARY REVIEW	APPROVAL NO. NASREC-2023- JAN- 007
Approval and Expiry Date	Approval Date: 30TH January , 2023	Expiry Date: 29th January, 2024
Protocol Version and Date	Version - Nil.	29th January, 2024
Information Sheet, Consent Forms and Dates	• English.	To be provided
Consent form ID and Date	Version - Nil	To be provided
Recruitment Materials	Nil	Nil
Other Study Documents	Questionnaire.	

Specific conditions will apply to this approval. As Principal Investigator it is your responsibility to ensure that the contents of this letter are adhered to. If these are not adhered to, the approval may be suspended. Should the study be suspended, study sponsors and other regulatory authorities will be informed.

CONDITIONS OF APPROVAL

- No participant may be involved in any study procedure prior to the study approval or after the expiration date.
- All unanticipated or Serious Adverse Events (SAEs) must be reported to NASREC within 5 days.
- All protocol modifications must be approved by NASREC prior to implementation unless they are intended to reduce risk (but must still be reported for approval). Modifications will include any change of investigator/s or site address.
- All protocol deviations must be reported to NASREC within 5 working days.
- All recruitment materials must be approved by NASREC prior to being used.
- Principal investigators are responsible for initiating Continuing Review proceedings. NASREC will only approve a study for a period of 12 months.
- It is the responsibility of the PI to renew his/her ethics approval through a renewal application to NASREC.
- Where the PI desires to extend the study after expiry of the study period, documents for study extension must be received by NASREC at least 30 days before the expiry date. This is for the purpose of facilitating the review process. Documents received within 30 days after expiry will be labelled "late submissions" and will incur a penalty fee of K500.00. No study shall be renewed whose documents are submitted for renewal 30 days after expiry of the certificate.
- Every 6 (six) months a progress report form supplied by The University of Zambia Natural and Applied Sciences Research Ethics Committee as an IRB must be filled in and submitted to us. There is a penalty of K500.00 for failure to submit the report.
- When closing a project, the PI is responsible for notifying, in writing or using the Research Ethics and Management Online (REMO),both NASREC
- and the National Health Research Authority (NHRA) when ethics certification is no longer required for a project.
- In order to close an approved study, a Closing Report must be submitted in writing or through the REMO system. A Closing Report should be filed when data collection has ended and the study team will no longer be using human participants or animals or secondary data or have any direct or indirect contact with the research participants or animals for the study.
- Filing a closing report (rather than just letting your approval lapse) is important as it assists NASREC in efficiently tracking and reporting on projects. Note that some funding agencies and sponsors require a notice of closure from the IRB which had approved the study and can only be generated after the Closing Report has been filed.
- A reprint of this letter shall be done at a fee.

- All protocol modifications must be approved by NASREC by way of an application for an amendment prior to implementation unless they are intended to reduce risk (but must still be reported for approval). Modifications will include any change of investigator/s or site address or methodology and methods. Many modifications entail minimal risk adjustments to a protocol and/or consent form and can be made on an Expedited basis (via the IRB Chair). Some examples are: format changes, correcting spelling errors, adding key personnel, minor changes to questionnaires, recruiting and changes, and so forth. Other, more substantive changes, especially those that may alter the risk-benefit ratio, may require Full Board review. In all cases, except where noted above regarding subject safety, any changes to any protocol document or procedure must first be approved by NASREC before they can be implemented.

Should you have any questions regarding anything indicated in this letter, please do not hesitate to get in touch with us at the above indicated address.

On behalf of NASREC, we would like to wish you all the success as you carry out your study.

Yours faithfully,

Dr. Mususu Kaonda

VICE-CHAIRPERSON
THE UNIVERSITY OF ZAMBIA NATURAL AND APPLIED SCIENCES RESEARCH ETHICS COMMITTEE - IRB

cc: Director, Directorate of Research and Graduate Studies
Assistant Director (Research), Directorate of Research and Graduate Studies
Assistant Registrar (Research), Directorate of Research and Graduate Studies

Publication

Computer Science and Engineering 2022, 12(1): 30-38
DOI: 10.5923/j.computer.20221201.03

Detection of Android Adware Using Transfer Learning with Computer Vision

Kabombo Katutwa*, Dani E. Banda, Julien Shabani

Department of Electrical and Electronics Engineering, University of Zambia, Lusaka, Zambia

Abstract Adware, or advertising-supported malware, is a term used to describe unwanted software that displays advertisements. Adware can infect and root-infect a device, forcing it to download specific adware types and allowing attackers to steal personal information therefore making it a major threat category for consumers. Little research has been conducted on android adware analysis. In this research, the goal is to leverage the use of visualization and transfer learning to detect android adware, to show that a pre-trained neural network can be used to detect android adware from benign application package (APK) files and to compare the performance of selected pre-trained neural networks in accomplishing this task. Using the CICMalDroid2020 dataset, the Dalvik Executable (DEX) files were extracted from APK files and then converted to grayscale images. These images were then used as input to pre-trained deep neural networks that have been trained on the ImageNet dataset. The pre-trained models used are VGG16, ResNet 50, Inception v3, EfficientNet v2 and MobileNet v2. The results show a maximum performance measurement 92% on the F1 – score metric which was achieved by the MobileNet v2 model.

Keywords Android Adware, Computer Vision, Deep learning, Pre-trained model, Convolutional Neural Network (CNN), Transfer learning

Kabombo Katutwa, Dani E. Banda, Julien Shabani, Detection of Android Adware Using Transfer Learning with Computer Vision, Computer Science and Engineering, Vol. 12 No. 1, 2022, pp. 30-38. doi: 10.5923/j.computer.20221201.03.

PUBLICATION URL: http://article.sapub.org/10.5923.j.computer.20221201.03.html