T.R.
ONDOKUZ MAYIS UNIVERSITY
INSTITUTE OF GRADUATE STUDIES
DEPARTMENT OF COMPUTATIONAL SCIENCES

ANDROID MALWARE DETECTION USING AUDIO AND IMAGE DATA TRANSFORMATION

Ph.D. Thesis

Oğuz Emre KURAL

Supervisor
Prof. Dr. Erdal KILIÇ

SAMSUN
2023

DECLARATION OF COMPLIANCE WITH SCIENTIFIC ETHIC

I hereby declare and undertake that I complied with scientific ethics and academic rules in all stages of my doctoral thesis, that I have referred to each quotation that I used directly or indirectly in the study, and that the works I have used consist of those shown in the references, that it was written in accordance with the institute writing guide and that the stated cases in article 3, section 9 of the Regulation for TÜBİTAK Research and Publication Ethics Board were not violated.

Is an Ethics Committee Necessary?

☐ Yes

☒ No

11/07/2023
Oğuz Emre KURAL

DECLARATION OF THE THESIS STUDY ORIGINALITY REPORT

Thesis Title: ANDROID MALWARE DETECTION USING AUDIO AND IMAGE DATA TRANSFORMATION

As a result of the originality report taken by me from the plagiarism detection program on 19/06/2023 for the thesis titled above:

Similarity ratio : %15

Single source rate : %1 has been released.

19/06/2023
Prof. Dr. Erdal KILIÇ

ÖZET

SES VE GÖRÜNTÜ DÖNÜŞÜMÜ KULLANILARAK ANDROID KÖTÜCÜL YAZILIM TESPİTİ

Oğuz Emre KURAL
Ondokuz Mayıs Üniversitesi
Lisansüstü Eğitim Enstitüsü
Hesaplamalı Bilimler Ana Bilim Dalı
Doktora, Temmuz / 2023
Danışman: Prof. Dr. Erdal KILIÇ

Mobil cihazlar sahip oldukları donanımlar ve onlar için geliştirilen uygulamalarla hayatımızda yeni bir dönemi başlatmıştır. Bu cihazlarda oluşabilecek bir güvenlik açığı kişisel bilgilerinin çalınmasına, gizlilik ihlallerine ve hatta finansal zarara neden olabilir. Bu nedenle bu cihazlarda kullanılacak uygulamaların güvenilir ve güvenli olması oldukça önemlidir. Bu amaçla, bu tez kapsamında, Android kötücül yazılım tespiti ve aile sınıflandırması için ses ve görüntü tabanlı yöntemler üzerine araştırmalar yapılmıştır. Görüntü tabanlı yaklaşımda, Android uygulama dosyalarını ikili diziler olarak ele alan uçtan uca bir yöntem önerilmiştir. Yöntemde her bir örnek için gri seviye görüntü temsilleri oluşturularak, eğitim ve test süreçleri CNN ile gerçekleştirilmiştir. Kötücül yazılımların görüntü temsilleri dosyaların ikili diziler olarak ele alınması ile oluşturulabildiği gibi, çıkartılan statik özniteliklerin matris formunda ele alınması ilede oluşturulabilir. Bu nedenle statik öznitelik seti kombinasyonlarının sınıflandırma başarımına etkisi üzerinede çalışılmıştır. İlk olarak Android uygulama dosyalarından elde edilen dört farklı öznitelik setinin muhtemel tüm kombinasyonları ele alınarak sınıflandırma başarımına etkileri araştırılmıştır. On farklı sınıflandırma algoritması ile tüm kombinasyonlar değerlendirilerek, etkin öznitelik seti kombinasyonları belirlenmiştir. Ardından öznitelik seti kombinasyonları ile RGB görüntüler oluşturularak CNN ile eğitim ve test süreçleri gerçekleştirilmiştir. Elde edilen sonuçlarda farklı öznitelik seti kombinasyonları ile 99% üzeri başarım elde edilebildiği görülmüştür.

Android uygulamalarının görüntü temsillerinin yanı sıra, ses temsilleri de oluşturulabilir. Literatürde ses tabanlı yaklaşımlar görüntü tabanlılara göre daha az yaygın olmakla bilikte, benzer şekilde yüksek başarımlar elde edebilmektedir. Bu kapsamda, Android kötücül yazılım aile tespiti problemi bir müzik kategori sınıflandırma problemi gibi ele alan ses tabanlı bir yaklaşım önerilmiştir. Bu amaçla ilk olarak Android uygulama dosyaları bir ses dosyasına dönüştürülerek ses tabanlı öznitelikleri çıkarılmıştır. Ardından dört farklı öznitelik seçim algoritması ile ayrıştırıcılığı yüksek öznitelikler belirlenmiş ve sınıflandırma işlemleri gerçekleştirilmiştir. Sekiz sınıflı bir veri seti üzerinde yapılan deneylerde aile tespiti 96.6% başarım ile gerçekleştirilmiştir. Tez çalışmasının sonunda çalışmada gerçekleştirilen yöntemlerle ilgili tartışmalar yapılmıştır.

Anahtar Sözcükler: Android Kötücül Yazılım Tespiti, Aile Sınıflandırması, Ses-Tabanlı, Görüntü-Tabanlı, Makine Öğrenmesi

ABSTRACT

ANDROID MALWARE DETECTION USING AUDIO AND IMAGE DATA TRANSFORMATION

Oğuz Emre KURAL
Ondokuz Mayıs University
Institute of Graduate Studies
Department of Computational Science
Ph.D., July / 2023
Supervisor: Prof. Dr. Erdal KILIÇ

Mobile devices have started a new era with their hardware and various software developed for them. A security vulnerability in these devices could lead to the theft of personal information, breaches of privacy, and even financial loss. Therefore, ensuring that the apps downloaded to the devices are reliable and safe is very important. For this purpose, within the scope of this thesis, an investigation into the image and audio-based approaches for Android malware detection and family classification is conducted. In the image-based approach, an end-to-end method is proposed that treats Android application files as binary sequences. In the method, grayscale image representations were created for each sample, and training and testing processes were carried out with CNN. Image representations of malware can be made by treating the files as binary sequences or the extracted static features in matrix form. For this reason, the impact of static feature set combinations on classification performance is also investigated. Initially, all possible combinations of four different feature sets obtained from Android application files are considered, and their effects on classification performance are investigated. Effective feature set combinations are determined by evaluating all combinations with ten different classification algorithms. Subsequently, RGB images are created with feature set combinations, and training and testing processes are carried out using CNN. In the results obtained, it was seen that with different feature set combinations, a performance above 99% could be obtained.

In addition to image representations of Android applications, audio representations can also be created. Although audio-based approaches are less common than image-based ones in the literature, they can achieve similarly high classification accuracies. In this context, an audio-based method is proposed, treating the Android malware family detection problem as a music category classification problem. Android application files were converted to audio files, and their audio-based attributes were extracted. Then, features with high discrimination were determined with four different feature selection algorithms, and the classification processes were carried out. Family detection was performed with 96.6% accuracy in experiments on an eight-class data set. At the end of the thesis, discussions were made about the methods used in the study.

Keywords: Android Malware Detection, Family Classification, Audio-Based, Image-Based, Machine Learning

ACKNOWLEDGEMENT

I want to express my sincere thanks to my advisor, Prof. Dr. Erdal Kılıç, for his knowledge and guidance throughout my graduate education process. My sincere thanks also go to my teacher, Prof. Dr. Sedat Akleylek, who generously shared his experiences. I want to thank my esteemed teacher Prof. Dr. Mustafa Ulutaş for sharing his valuable views during my thesis journey.

I also want to thank my thesis committee members, Assoc. Prof. Dr. Rafet Durgut and Assoc. Prof. Dr. Recep Sinan Arslan, for their contributions to my thesis with their suggestions.

I would like to thank my dear friends Durmuş Özkan Şahin, Meryem Soysaldı Şahin, and Kadir Kaya, with whom we often share our knowledge and experiences in our academic life.

I want to thank my precious wife, Eda Kural, who has always made me feel her support in every moment of my life and encouraged me to continue on my journey whenever I feel hopeless. I would also like to thank my son, Kuzey Alp Kural, who patiently waited for our time together and offered me his support with his little hands. Lastly, I would like to thank my Mum, Dad, and Brother for always supporting and caring for me.

Oğuz Emre KURAL

CONTENTS

SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS

$AAPT2$	: Android Asset Packaging Tool
APK	: Android Package Kit
API	: Application Programming Interface
$ARSC$	: Android Resources
$AXML$	: Android XML
$BFCC$	: Bark Frequency Cepstral Coefficients
CNN	: Convolutional Neural Network
$DESCR$	: Descriptive Relevance
$DISCR$	: Discriminative Relevance
DEX	: Dalvik Executable
$GFCC$	: Gammatone Frequency Cepstral Coefficients
$GIST$	: Global Image Feature
GML	: Graph Modeling Language
$IDFEC$	: Inverse Document Frequency Excluding Category
JAR	: Java Archive
JFS	: Joint Feature Score
KNN	: K-Nearest Neighbours
LBP	: Local Binary Pattern
$MFCC$	: Mel-Frequency Cepstral Coefficients
$ODEX$	: Optimized Dalvik Executable
RF	: Random Forest
RFF	: Relevance Frequency
RGB	: Red-Green-Blue
SDK	: Software Development Kit
SVM	: Support Vector Machine
VGG	: Very Deep Convolutional Networks
TGF	: Term Global Frequency
XML	: Extensible Markup Language

FIGURES LEGENDS

TABLES LEGENDS

1 INTRODUCTION

Mobile devices are in almost every aspect of our daily life. It touches people's lives in many areas, like education, health, transportation, and personal activities. More and more people are becoming mobile device users every day. According to Statista's report (*Statista — Forecast number of mobile users*, 2023), it was reported that there were 7.1 billion users worldwide in 2021. In the same report, it is predicted that this number will reach 7.3 billion in 2023 and 7.5 billion in 2025. In such a broad market, the Android operating system has the largest share based on mobile operating systems. According to another Statista report (*Statista — Global mobile os market share*, 2023), Android is the market leader, with 71.4% in the first quarter of 2023. This market share makes the platform a honeypot for application developers and cyber attackers. While developers develop different paid and free applications and present them to people, cyber attackers target people with the same method. Although the Android platform is an open-source platform, it causes more people to contribute, but it also causes malicious people to take advantage of the vulnerabilities they detect.

Applications developed for Android can be published in different third-party markets along with the official market offered by Google. Generally, unofficial markets do not take precautions regarding published applications' reliability. This provides the appropriate environment to distribute their applications for developers who target users with methods such as repackaging. Although the Google Play Store (*Google play store*, 2023) takes measures with its review process, they are not always sufficient. According to a report published by McCaffe in 2021 (*McAfee Mobile Threat Report.*, 2023), malware published on the market was downloaded by more than 700000 users before it was noticed and removed by Google. The increasing number of apps makes the detection of malicious applications more difficult. According to another report by AVTest (*AV-ATLAS - Malware & PUA*, 2023), an average of 3 malicious applications are seen every second. Moreover, with the increase in number, malware is using more sophisticated methods every day (Elsersy, Feizollah, & Anuar, 2022). These observations reveal the need to develop faster, more efficient, and more advanced techniques for Android malware detection.

Android malware detection is performed in three different ways according to the feature extraction approach (Meijin et al., 2022). These are static analysis, dynamic analysis, and hybrid analysis. Static analysis is an analysis method performed by analyzing Android files and extracting information such as API calls, permissions, and opcode sequences. It is faster and generally safer because the analysis is done without running suspicious files. Dynamic analysis is the analysis method in which the application is run in a sandbox environment, and its behavior is monitored. Data such as resource usage, network traffic, and API requests are obtained by monitoring the behavior of the application during operation. Then, inferences are made from the obtained data with different methods. Both methods have different advantages and disadvantages. Static analysis approaches can provide full code coverage because they examine application files. It's also safer and faster overall, as analysis is performed without running the application. However, it is generally more vulnerable to first-day attacks and obfuscation techniques. In order to overcome this deficit, in recent years, static analysis approaches have been made more resilient by applying data transformation techniques and deep learning methods. Although static analysis methods evaluate the application through its files, they cannot predict its behavior during operation. On the other hand, dynamic analysis methods make determinations by monitoring applications at runtime. It detects behavior patterns by observing the behaviors of applications, such as resource consumption, system calls, API calls, and network usage. Since they monitor applications at runtime, they are more resistant to first-day attacks and obfuscated malware. However, a sandbox-like analysis environment should be created to perform dynamic analysis, and applications should be monitored for a certain period. This period varies according to the process of the application performing malicious behaviors. In addition, specific steps need to be taken to trigger the malicious behavior of many malware (Maniriho, Mahmood, & Chowdhury, 2022). This makes dynamic analysis methods costly, slow, and difficult to perform. Finally, hybrid analysis is a combination of static and dynamic analysis methods to make malware detection more accurate and efficient.

Existing approaches are supported by adapting innovative methods applied in different fields. For example, techniques used in document classification in text mining can be adapted to malware detection (M. Sharma, Chawla, & Gajrani, 2016;

Suarez-Tangil, Tapiador, Peris-Lopez, & Blasco, 2014). Similarly, with the representation of data in various domains, audio, and image-based approaches can also be used in malware detection (Vasan, Alazab, Wassan, Safaei, & Zheng, 2020; Farrokhmanesh & Hamzeh, 2019). Within the scope of this thesis, Android applications are represented in other domains, and new methods are proposed for Android malware detection.

1.1 Motivation and Contribution

Android malware poses various threats, including the theft of personal data, altering or deleting device files, monitoring user activities, and causing financial damage through banking applications. In addition to detecting malware, it is important for researchers to understand which family they belong to take appropriate precautions (B. Wu et al., 2021). However, this process is both time-consuming and costly. For this reason, it is essential to produce effective and cost-effective methods to solve these problems. Different static and structural information can be obtained when Android applications are handled statically. Application files can be taken as binary, and information such as permissions, API calls, and intent filters can be obtained by processing the files. Using all of the received information often leads to inefficient use of computational resources. For this reason, choosing the efficient ones from the feature sets reduces the cost of classification and increases the classification performance. When the application files are handled as binary, the data is usually represented in different domains. Transforming data into other domains paves the way for adapting techniques applied in different fields to malware detection and examining data differently. It has been observed that especially audio and image conversion techniques can achieve effective results in malware detection. Along with these observations, this thesis study sought answers to the following research questions in Android malware detection and family detection.

- Can malware be detected by converting binary sequences from APK files to grayscale images?
- In static analysis, different attribute groups can be obtained from application files. How do these attribute groups affect classification performance when used individually or together?

- Classifiers play a crucial role in static malware detection models. Which classification algorithms show higher performance in static analysis?
- How does the conversion of feature sets obtained by static analysis to RGB image affect classification performance?
- Can Android malware family detection be performed by representing binary sequences from APK files as audio signals?
- Which features are more effective in audio-based Android malware family detection?

In response to the first research question, a structure that reads the classes.dex file as binary and performs end-to-end Android malware detection with the CNN model proposed (Kural, Sahin, Akleylek, Kılıç, & Ömüral, 2021). Grayscale images were obtained from binary sequences, and detections were made with the CNN network. Experiments were carried out with a data set containing more than 24000 samples, and the results were shared.

In response to the second and third research questions, experiments were conducted on Android malware detection with four different feature sets (Kural, Sahin, & Kiliç, 2023). Considering all possible combinations of feature sets, the feature sets with the highest contribution were determined. The results were validated by experiments performed on ten different classifiers and two different data sets. In addition, classification algorithms that produce successful results in Android malware detection have been determined. In the results, it was seen that API calls made more successful results than other attribute sets. Responding to the fourth research question, experiments were carried out with RGB image representations of feature set combinations. Image representations are created by giving different sets of features belonging to the same application to different color channels of RGB images. Classification performances are shared by giving comparative results. In the results, it has been seen that the classification performance is over 99% when the right feature sets are selected.

Finally, in response to research questions five and six, the Android classes.dex file is represented by an audio file in .wav format (Kural, Kiliç, & Aksaç, 2023). Audio-based features are extracted from the data and converted to audio by making audio-based feature extraction. Then, audio-based features with high discrimination are determined by applying four different feature reduction methods. Within the scope

of the study, the attributes with high discrimination and those with low discrimination were shared. To the best of our knowledge, this is the first study to examine feature selection and its effects on classification performance in audio-based Android malware family classification.

1.2 Organization

The rest of the thesis is organized as follows. In Chapter 2, we provided information on Android file structure and tools used in malware analysis. At the same time, the performance criteria used in the studies and the literature research on the field studied were presented. In Chapter 3, we discuss our evaluations on Android malware detection and provide a detailed account of our preliminary investigations.

In Chapter 4, we examined the effects of feature set combinations on Android malware detection. Furthermore, we generated image representations from feature set combinations and shared results on training and testing processes using a CNN model.

Lastly, in Chapter 5, we introduced an audio-based Android malware detection model. We also employed four feature reduction methods on the audio-based features, identifying those most beneficial in Android malware family classification.

2 BASIC STRUCTURES AND RELATED WORKS

This section provides an overview of the structure of Android applications and the tools and datasets commonly employed in Android malware analysis. Following this, we discuss the performance metrics utilized during the classification stages. Finally, we provide a detailed review of relevant studies in the literature, focusing on their methodologies and findings.

2.1 Android Platform

Android applications are distributed with package files called APKs. An APK comprises several key components often utilized in static analysis, including AndroidManifest.xml, Classes.dex, Resources.arsc files, and directories like META-INF, Assets, res, and lib.

The AndroidManifest.xml file is the root file where all application components are declared. This file encapsulates critical details such as the application's package name, the application's name, the SDK version it's built with, the hardware prerequisites, and the permissions it needs to operate properly. Its primary role is to inform the system about all application components, such as activities, services, receivers, and providers. AndroidManifest.xml is frequently utilized in static analysis-based methods such as permission analysis and data transformation-based methods. The Classes.dex file is the file that contains the compiled Java bytecode of Android applications. This file is frequently used in code analysis and data transformation-based methods. Information such as component features, code dependencies, and string information can be accessed by analyzing this file. The Resources.arsc file is the file that contains the compiled resources of the application. It includes the precompiled versions of files such as strings, colors, and styles. The META-INF directory contains certificate and signature information related to packaging and security. It is responsible for verifying, signing, and maintaining the integrity of APKs. Assets and res directories are directories that contain application resources. The assets directory includes structures such as image, audio, and text files, while the res directory includes structures such as layouts and drawables. Finally, the lib directory contains compiled or shared libraries in

applications and usually includes subdirectories separated by device architectures.

2.1.1 Android Malware Analysis Tools

This section describes the tools available for static and dynamic Android malware analysis.

AAPT2: AAPT2 is the official build tool for Android (*AAPT2 — Android Studio — Android Developers*, 2023). AAPT2 takes resources such as images, user interface layouts, and language strings and parses them, indexes them, and compiles them. AAPT2 is a critical part of the Android application development process.

Androguard: Androguard is a static analysis tool that provides information about APK, DEX, XML, and ARSC files (*Androguard*, 2023). It allows Disassemble/Analysis/Modification operations with DEX, ODEX, APK, AXML, and ARSC files. It was developed with Python.

Apktool: It is a 3rd party tool developed to reverse engineer Android applications (*Apktool*, 2023). It can be used in decoding and repackaging processes of Android application packages.

ClassyShark: ClassyShark is an analysis tool that enables binary analysis in Android applications (*ClassyShark: Android and Java Bytecode viewer*, 2023). The class interface and its members in binary executables allow displaying of important information such as dex count and dependencies. It supports library formats such as .dex, .aar, .so and executables such as .apk, .jar, and .class. It is an open-source tool developed by Google.

Cuckoo: Cuckoo Sandbox is an open-source system for malware analysis (*Cuckoo Sandbox: Automated Malware Analysis*, 2023). It allows monitoring and inspection of different data, such as network traffic (including SSL/TLS encrypted), resource usage, and API calls of the applications under investigation. Apart from the Android platform, analyses can be made for different platform malware.

DeGuard: It is a tool that reverses different obfuscation operations performed in Android applications (*DeGuard: Statistical Deobfuscation for Android*, 2023). Removing the obfuscation effect allows practical analysis for applications.

dex2jar: It is a tool for converting classes.dex file to JAR file (*dex2jar: Tools to work with android .dex and java .class files*, 2023). Converting the codes in bytecode to JAR format makes the codes more human-readable.

JADX: It is a tool that extracts Java source codes from Android Dex and APK files (*JADX: Dex to Java decompiler*, 2023). It facilitates code analysis for researchers by providing opportunities such as editing and coloring on Java source codes. It can be used both from the command line and from the GUI.

MobSF: Mobile Security Framework is an all-in-one malware analysis tool offering static and dynamic analysis (*MobSF: Mobile Security Framework*, 2023). It offers pen-testing and malware analysis on malware developed for Android, iOS, and Windows.

2.2 Android Malware Datasets

This section provides an overview of datasets commonly utilized in Android malware detection.

AMD: Shared by (Wei, Li, Roy, Ou, & Zhou, 2017), AMD contains 24553 Android malware samples collected between 2010 and 2016. The dataset consists of 71 families and 135 variants.

AndroZoo: Shared by (Allix, Bissyandé, Klein, & Le Traon, 2016), AndroZoo contains 22,775,299 applications collected from different application markets. The first version of the data set was shared with 3.1 million applications in 2016 and is still being updated today.

CIC-AndMal2017: As shared by (Lashkari, Kadir, Taheri, & Ghorbani, 2018), CIC-AndMal2017 includes 429 malicious and 5065 benign samples from different sources. Although more than 4000 malicious and 6000 benign samples were examined during the dataset's creation, not all were included due to low sample quality. While the benign samples were collected between 2015-2017, the malware samples started to be collected in 2012. The dataset contains samples from 42 families in 4 main categories (Adware, Ransomware, Scareware, SMS Malware).

CIC-MalDroid2020: The CIC-MalDroid2020 dataset shared by (Mahdavifar,

Alhadidi, & Ghorbani, 2022) includes 17341 Apps, consisting of 4 malware and one benign category. The samples in the dataset were collected between December 2017 and December 2018. 11598 of the App samples were processed through a series of processes and were shared as operable.

Drebin: As shared by (Arp et al., 2014), Drebin contains 5560 Android application samples from 179 malware families. App samples were collected between August 2010 and October 2012.

Malgenome: Malgenome contains 1260 Android malware samples from 49 malware families (Zhou & Jiang, 2012). App samples were collected between August 2010 and October 2011. The Malgenome dataset is the first publicly available Android malware dataset shared by researchers.

2.3 Performance Measures

In classification tasks, the performance of a classifier is measured using the confusion matrix. The confusion matrix is a table showing the ability of a classification model to accurately predict classes. A confusion matrix is shown in Table 2.1.

Table 2.1. Confusion matrix

		Predicted	
		Malware	**Benign**
Actual	**Malware**	TP	FN
	Benign	FP	TN

Based on the classification results, a confusion matrix comprises four outcomes, which include:

- True Positive(TP): Samples that the model predicts positively and are actually positive.
- True Negative(TN): Samples that the model predicts negatively and are actually negative.
- False Positive(FP): Samples that the model predicts as positive but are actually negative.
- False Negative(FN): Samples that the model predicts as negative but are actually

positive.

These four components can be used to calculate model performance metrics such as accuracy, precision, recall, and F-measure. The accuracy metric is the ratio of correctly classified samples to the total number of samples. The mathematical representation of accuracy is shown in Equation 2.1.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Although accuracy can provide meaningful insights into model performance when dealing with balanced datasets, it may be insufficient when used alone for imbalanced datasets. In such scenarios, employing precision, recall, and F-measure metrics can offer a more accurate evaluation. Precision and recall metrics are where the positive class is at the forefront. Precision refers to how much of the positively predicted data is genuinely positive. Recall, on the other hand, refers to how much of the truly positive data is predicted as positive. The equations of precision and recall are given in Equation 2.2 and Equation 2.3.

$$precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$recall = \frac{TP}{TP + FN} \tag{2.3}$$

F-measure is calculated by taking the harmonic mean of precision and recall values. The mathematical expression of F-measure is shown in Equation 2.4.

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.4}$$

2.4 Literature Review

The most commonly used methods for Android malware detection are static feature extraction followed by the use of machine learning approaches as a decision maker. This process involves examining an application's source code or application binary without actually executing it, and identifying potential malicious patterns or behaviors. In these models, researchers test innovations in efficient feature extraction

and machine learning approaches to increase performance. After the feature extraction stage, methods such as data integration, data reduction, and feature weighting are being researched to use the available features more effectively and to use machine learning resources more efficiently. Researchers test the data obtained from these methods with classification algorithms, comparing different classifiers and newly proposed classification models, whether single or hybrid. The higher performance of the developed models has led malware developers to find new ways, such as obfuscation and encryption, to evade detection, especially by static analysis-based methods. Researchers applied data-transformation techniques to adapt static analysis methods, which are weaker against first-day attacks and obfuscation techniques. New approaches are proposed based on data representation in different domains, such as audio and image, using data transformation techniques. Researchers prefer image transformation-based strategies not only because they can resist obfuscated malware but also because they offer end-to-end malware detection solutions using deep learning methods, such as CNN.

Inspired by these successful applications of image and audio transformations, researchers realized that the Android malware and malware family detection model is very similar to other classification problems, such as music classification and genre recognition, due to the nature of the problem being dealt with. This similarity has led researchers to innovative approaches based on audio and image transformations.

Given this evolution, this study incorporates approaches based on image and audio transformations. We examine studies that employ classical image processing techniques following image transformation, along with image-based deep learning models that provide end-to-end solutions. Similarly, audio-based malware detection techniques and their applications are also discussed in the context of the existing literature.

In image transformation-based approaches, researchers have proposed methods in which image representations are extracted with image processing techniques such as GIST and LBP and evaluated with different classifiers.

(Nataraj, Karthikeyan, Jacob, & Manjunath, 2011) proposed a new method for visualization and automatic malware classification in 2011. In the proposed method,

malware files are read in binary format and converted to gray-level images. The malware detection problem was then considered a visual classification problem. GIST, which uses the wavelet decomposition of the images, is used to obtain the texture features of grayscale images. Experiments were carried out for classification using the kNN algorithm with Euclidean distance. In the experiments performed with the data set containing 9458 samples from 25 families, the most successful result was reported as 98%. The proposed method is the first study in the literature to use image-based approaches for malware detection.

(Kumar, Sagar, Kuppusamy, & Aghila, 2016) conducted a study in 2016 that deals with apk files in different color formats. They extracted GIST features by converting apk files to images in grayscale, RGB, CMYK, and HSL formats. Their experiments with Decision Trees, KNN, and Random Forest algorithms on a small data set reported the most successful result as 91%. The best results in the study were obtained with grayscale images and the Random Forest algorithm.

(Luo & Lo, 2017) proposed a methodology for classifying malware images based on their binary image and extracting local binary pattern (LBP) features. The LBP is implemented on the malware images to extract useful features in pattern or texture classification. To do this, malware read in binary format was first converted to grayscale images. Then, every nine consecutive image pixels are rearranged to form a 3x3 grid to extract the LBP features. The extracted LBP features were classified by machine learning algorithms such as SVM and KNN, and the best result was reported as 93.17%. Experiments were carried out with a dataset containing 12000 malware.

In a study conducted by (Darus, Ahmad, & Ariffin, 2019), Android malware detection was investigated through two different approaches to the classes.dex file. Firstly, the entire classes.dex file was converted into an 8-bit grayscale image. Secondly, only the data field within classes.dex was extracted and converted into a grayscale image. The purpose of this was to investigate whether classification performance would be affected if only the data portion of the classes.dex file was processed. After image transformation, image attributes are extracted with a GIST descriptor. A data set containing 300 benign and 418 malware samples was used in the experiments. The obtained feature vectors were classified with XGBoost, KNN, and RF classifiers,

and six results were obtained for two scenarios. The results reported that the highest performance was obtained using the data section in classes.dex with 74%.

(Fang, Gao, Jing, & Zhang, 2020) proposed a new method for classifying Android malware families based on DEX file section features. In their approach, attributes are extracted in both domains by converting the dex file to image and plain text. After the image transformation, both GIST features and texture attributes, as well as color moments and attributes, were obtained from RGB images. With the plain text transformation, text features were obtained with the Simhash algorithm. Then, it was planned to determine the family by classifying the features obtained. Considering the small number of samples in the classes for the classification stages, it was decided that the SVM classifier was suitable for the problem. However, the fact that the features obtained from the data set carry different types of data in other domains has revealed that it would be more appropriate for each feature set to have its kernel and parameters. To this end, the researchers proposed the Multiple Kernel Learning structure. Experiments were conducted with different kernels and parameters to determine the most accurate kernel combination. A 96% F-score performance was achieved in the experiments by selecting 15 families containing more than 200 samples of the AMD dataset.

In another study they conducted in 2021 using the same dataset (J. Singh et al., 2021), experiments were carried out for different classifier and feature fusion strategies by extracting handcrafted features from grayscale images with techniques such as GIST, LBP, and GLCM. In the first stage of the study, which was carried out in two phases, results were obtained with KNN, SVM, and Random Forest algorithms, and in the second stage, with CNN and feature fusion strategies. They reported by comparing the results obtained with GLCM-SVM, GIST-SVM, LBP-SVM, GLCM-KNN, GIST-KNN, LBP-KNN, LCM-RF, GIST-RF, LBP-RF pairs with the experimental results obtained with feature fusion strategies. The results reported that they achieved the highest performance, with 93.24%, with images converted from Certificate and Android Manifest files and Feature Fusion SVM.

(Ünver & Bakour, 2020) proposed machine learning-based Android malware detection using image-based local and global features. The Manifest, Dex, and ARSC files were extracted from the Android application files and converted to grayscale

images. The method extracts local (SIFT, SURF, KAZE, ORB) and global features (Colour Histogram, Hu Moments, Haralick Texture) from the images and classifies them with different machine learning approaches. Experiments were carried out for local and global features for scenarios where manifest, dex, and arsc files are used separately and together. The experiments showed that the AdaBoost classifier reached 98% accuracy when the global features were extracted from the images produced from the Manifest, Dex, and Manifest-Dex-ARSC files. When local features were extracted, it was seen that the AdaBoost algorithm achieved 98% success with the set produced from Manifest-DEX-ARSC files. When an overall review is made for local features, it has been reported that the results obtained using the Manifest file are lower than those obtained with DEX and Manifest-DEX-ARSC. In all experiments, it was reported that the worst result was obtained with the ORB local feature. Among the local features, it was seen that KAZE gave the best result. In addition, it was stated in the examinations made for the computational times that the system can calculate under 0.018s.

(Bakour & Ünver, 2021) proposed a new model that hybridizes image-based features with deep learning techniques. The model, called DeepVisDroid, extracts image-based local features such as SIFT, SURF, ORB, and KAZE and image-based global features such as Hu moments and Haralick texture to train a convolutional neural network model in multiple scenarios. Four data sets were created during the image conversion stages by considering the manifest.xml, resources.arsc, and classes.dex files in the apk files. The experiments were carried out separately for each, and the results were reported. Tests were carried out with state-of-the-art methods like VGG16 and ResNet using a two-class data set containing 9700 samples. In the results, it was revealed that the model can achieve high results with 98% accuracy.

(Shahid Alam, 2023) proposed a method that performs feature extraction on gray-level images with Gabor filters. Gabor filters were applied with the sliding window method to look for malicious behavior patterns in images representing applications. To reduce the extracted features and contribute to the performance, feature selection was made with a greedy method, Recursive Feature Elimination (RFE). Since the classifier used in RFE is an essential criterion in the results obtained with RFE, preliminary experiments were carried out with different classifiers on a 50-sample data set. As a result of the experiments, it was concluded that the most efficient classifier for RFE is

linear SVM. In the classification experiments performed with Naive Bayes, AdaBoost, and linear SVM after the selections, it was stated that the feature selection provides +5% success for all classifications. It was noted that Naive Bayes showed the best result with 99.43% accuracy in classifications. A balanced data set containing a total of 1402 samples was used in the experiments.

(Bakır & Bakır, 2023) introduced a novel model that employs autoencoders for feature extraction. By generating image representations for each application, the researchers performed feature extraction on images utilizing ANN-based, CNN-based, and VGG19-based autoencoders. The experiments involved classifying a dataset of 3000 benign and 3000 malware applications using different machine-learning algorithms. The best result obtained in the experiments was 98.56% accuracy with the VGG19-based encoder and LR.

The image representations of the applications can be produced directly from the binary sequences of the files, as well as at the end of the static feature extraction process from the application files. Attributes such as permissions, API calls, and opcode sequences that can be extracted from applications can be vectorized for each application and converted to image representations in different formats.

(Ganesh et al., 2017) proposed a method that analyzes permission patterns with CNN networks by converting application permissions to binary images. Considering the application permissions for each apk according to the presence-absence status, 138 permissions were extracted. Then, the permission vectors were converted into matrix form, and 12x12 binary images were obtained. In the training stages, well-known networks such as LeNet, GoogleNet, ImageNet, and AlexNet were trained using the Android Malware Genome and Drebin datasets. The experiments reported that the best result was obtained with LeNet using a dataset containing 2000 malware and 500 benign samples. The reason why LeNet outperformed more complex networks, such as GoogleNet, with a 93% success rate, was interpreted as the fact that 12x12 images were not complicated enough for these networks. Within the scope of the study, an Android mobile application called Droidscreen was developed. Although the application works on mobile devices, the apps considered for evaluation are sent to a remote server, and the evaluation result is sent back to the device.

In 2018, (Y.-X. Ding, Zhao, Wang, & Wang, 2018) studied detecting Android malware using API calls. They parsed the .dex file to obtain the API calls and converted the call sequences into a two-dimensional image. In experiments, the DREBIN dataset was used to evaluate the effectiveness of the approach. The experiments conducted with CNN, SVM, KNN, and RF showed that the most successful result, with 90% accuracy, was achieved when CNN was utilized.

(Vu & Jung, 2021) proposed a framework called AdMat for Android malware detection and classification using deep learning. AdMat characterizes Android applications as adjacency matrices, which are then used as input images for a Convolutional Neural Network (CNN) model. First, API methods used in applications were extracted to create neighborhood matrices. Considering that the total number of API methods would be very high, only 219 widespread API calls were considered. Then, the calls were represented as graphs with GML for API calls. Neighborhood matrices were created with the obtained GML and converted to NxN matrices. In experiments with Drebin and AMD datasets, the best results for malware detection and Family classification were reported as 97% and 92% F1-scores, respectively.

(Arslan & Tasyurek, 2022) proposed an image-based method for Android malware detection to minimize resource consumption. The method has obtained a QR-like image for each example by considering the features processed from the AndroidManifest.xml file. Considering 349 different features, the areas where the feature is present are coded as black, and the areas where it is absent are coded as white. A CNN model with three convolution layers, constructed with the obtained images, was trained, and the results were compared with classical machine learning methods. The results show that the proposed method outperforms classical machine learning methods such as SVM and RF with a test accuracy of 96.2%.

(Tasyurek & Arslan, 2023) proposed a fast and efficient approach called RT-Droid. Combining the QR code-like visual transformation model they had previously proposed with YOLO V5, they proposed a low resource consumption approach that could detect 0.019 seconds. They conducted experiments with different networks using the Drebin and Genome datasets for malicious samples and the Arslan dataset for benign samples. The results indicated that their proposed model

outperformed similar networks, operating at speed 5.5 times faster. The peak performance of the model was documented as a 97% F-score. Comparative analyses revealed that their proposed model required 25 times less memory than grayscale image conversion.

Image representations can be created using static attributes, as well as converting application files directly to images. To this end, researchers proposed approaches that read the application files as binary and convert them to image representations in different formats.

(Hsien-De Huang & Kao, 2018) proposed a Color-inspired CNN-based Android malware detection system named R2-D2. The system uses .dex files by converting them to RGB images. When the similarity was measured with the Levenshtein distance metric within the same family after the transformation, it was seen that the similarity within the family was high. This shows that the proper methods can classify samples. CNN-based methods were chosen to be used in the training and testing stages. However, since the converted images are not natural images, it is predicted that filters such as classical 3x3 and 5x5 may perceive unrelated data as related. To prevent this situation, 1x1 convolutions were used in the Inception-V3 network. Experiments were carried out with 2 million samples collected between January and August 2017, and it was reported that the system achieved 93% accuracy with 829356 samples.

(Yen & Sun, 2019) proposed a method that visualizes apk files by performing multiple data transformations. The method is based on the processing of Java code. For this reason, firstly, using dex2jar and jad tools, java code was obtained from the dex file. The importance of code words is calculated to avoid the effect of obfuscation techniques such as renaming. Word importance groups were created after calculating the importance of each code word with TF-IDF from the Java code. Simhash and djib2 algorithms place the data on the x and y axes in the R, G, and B channels. After the visual transformation, training, and testing processes were carried out with CNN. The experiments were conducted with the data set containing 1440 APK files. Researchers reported the best result as 92.67% accuracy.

(Y. Ding, Zhang, Hu, & Xu, 2020) proposed a method that reads dex files as bytecode and classifies them with CNN. The proposed method is compared with similar

studies made with classical machine learning algorithms (Peiravian & Zhu, 2013), and its advantages and disadvantages are revealed. In the process, firstly, image conversion from binary code was made. Since the sizes of the files in the dataset are different, the image sizes after the conversion were also other. To balance this situation and make it ready for training and testing with CNN, the images have been resized to 512x512. Then, using the Drebin dataset, CNN was trained with 3962 malware and 1000 benign samples. In the experiments, it was stated that the proposed method achieved 94.4% accuracy. Compared to the classical techniques, it was noted that the method obtained better results than the decision tree but showed 1.7% lower classification performance than SVM. However, compared to classical machine learning algorithms, the method's advantages are that it does not require prior domain knowledge and does not require feature engineering. In addition, it was reported that the system performed highly in experiments with polymorphic encryption applied malware, which is generally difficult to classify by static methods.

(Zou, Luo, Liu, Wang, & Wang, 2020) proposed an Android Malware detection method that processes bytecode sequences obtained from the data section of the dex file. In the proposed method, the header and index sections in the Dex file were ignored, and bytecode was obtained only from the data section. Large bytecode sequence sizes cause high computational costs in training processes. Therefore, it was desired to set a sequence limit that keeps TPR at reasonable levels while avoiding high computational costs. For this purpose, starting from the 500KB limit, the TPR value of the dataset was monitored for different values, and the 1500KB data size was determined as the limit for bytecode sequences. Thus, only samples below 1500KB were included in the training stages. While the sequences were trained with CNN, four datasets were used, the largest of which was 10479 samples. The best result obtained from the experiments was reported as 96.9% accuracy. The best result obtained in the experiments with the 10479 sample dataset, which includes samples with seven different obfuscation techniques, was reported as 92.17%. After the experiments, the misclassified samples were evaluated manually, and the inferences on the data set and the method were shared. Testing with Virustotal for 69 faulty samples classified as malicious in the reviews was benign. When the misclassified malicious samples were examined manually, it was seen that the average sequence size of the samples was below 500KB.

(Feng, Shen, Chen, Wang, & Li, 2020) proposed a two-layer system capable of performing Android malware detection, category detection, and family detection. A proposed model uses permissions, intent, and component information in the first layer as features. Using a fully connected neural network, 95.22% accuracy was achieved in this layer. Applications classified as benign in the first layer were evaluated in the second layer for a secondary control. Network traffic features are used as features in the second layer. A new method called CACNN, which uses CNN and AutoEncoder as a cascade, is proposed in the evaluation. In the experiments performed for malware detection (two classes), category detection (4 classes), and family detection (40 classes) in the second layer, 99.3%, 98.2%, and 71.48% accuracy were obtained, respectively. When the proposed method is evaluated, although it takes a secondary measure against the false-negative samples in the first layer with its two-layer structure, it does not offer anything for the false-positive samples in the first layer.

(W. Zhang, Luktarhan, Ding, & Lu, 2021) proposed a new Android malware detection model based on a temporal convolution network (TCN) that combines the visual features of the XML file with the data section of the DEX file. The proposed method creates four gray-scale image datasets with different combinations of texture features by fusing XML files and DEX files. The image size is then unified and input to the designed neural network with three different convolution methods for experimental validation. The experimental results show that adding XML files benefits Android malware detection. The detection accuracy of the TCN model is 95.44%, precision is 95.45%, recall rate is 95.45%, and F1-Score is 95.44%. Compared with other methods based on the traditional CNN or lightweight MobileNetV2 model, the method proposed in this paper, based on the TCN model, can effectively utilize bytecode image sequence features, improve the accuracy of detecting Android malware and reduce its computation.

(Gerardi, Iadarola, Martinelli, Santone, & Mercaldo, 2021) evaluates the effectiveness of image-based malware detection techniques in the real world by testing the resilience of these approaches when morphed samples are considered. The authors present a tool called DexWave, which automatically injects perturbations techniques targeting the smali code representation of Android applications to induce the classifier to incorrectly predict the family to which the input malware belongs. With two

perturbation techniques called NopsBombing and StringBombing, various instructions and strings were added to the original code, attempting to mislead a CNN network with 90% detection accuracy. In the results, 20 of 59 samples could be detected as malicious in the first experiments, while only nine were evaluated as malicious after perturbation procedures.

(Daoudi et al., 2021) proposed DEXRAY, which visualizes binary sequences as one-dimensional images. Considering that a square or rectangular visual transformation would cause a loss of information carried by the binary sequence, they represented binary expressions in a vector form. The proposed method conducted training and tests with a 2-layer 1-D CNN model by representing dex files with (1, 128x128) vectors. In the experiments, results were obtained for different scenarios, and the results for all scenarios were shared in detail. First, results were obtained with Drebin (15558 samples) and AndroZoo (158K samples), and comparisons were made with state-of-the-art studies. In the results, it was reported that DEXRAY could detect with 0.96 F1-score on the Drebin dataset. It was stated that the model achieved an F1-score of 0.98 in tests performed on experiments that it had not seen before. In the next step, it was said that DEXRAY could achieve successful results when experiments were carried out with obfuscated samples at different rates. It was also reported that downsizing caused up to 5% performance loss in experiments performed at different sizes to compare the initial (1,128x128) size.

(Yadav, Menon, Ravi, Vishvanathan, & Pham, 2022) proposed a two-stage deep learning framework for detecting Android malware and classifying its variants using image-based malware representations of the Android DEX files. The framework uses the EfficientNetB0 convolutional neural network to extract relevant features from the malware color images. The extracted features are then passed through a global average pooling layer and fed into a stacking classifier. The stacking classifier employs linear support vector machine and random forest algorithms as base-level classifiers and logistic regression as the meta-level classifier. The proposed method achieved an accuracy of 100% in binary classification and 92.9% accuracy in 5-class classification and 88.6% in 4-class classification. The framework is platform-independent and can handle both unpacked and packed malware.

(Almomani, Alkhayer, & El-Shafai, 2022) proposed an automated vision-based AMD model composed of 16 fine-tuned CNN algorithms for efficient and quick detection of Android malware attacks. Working with both RGB and grayscale images, fine-tuned Xception, VGG16, VGG19, DarkNet53, MobileNetV2, ResNet101, AlexNet, ResNet50, ResNet18, InceptionV3, DarkNet19, ShuffleNet, Places365, GoogleNet, NasNetMobile, GoogleNet, and SqueezeNet networks are trained. Experiments were conducted with the Leopard Mobile dataset, and experiments were carried out with 14733 malicious and 2486 benign applications. A sub-balanced data set of 2486 or 2486 was created to examine the proposed model's performance on balanced and unbalanced data sets. Considering all parameters, the best results for the balanced data set were reported as 99.40% F1-score in RGB images and 99.20% F1-score in grayscale images. In the unbalanced dataset, the best performance was reported as 93.1% in RGB images and 92.7% in grayscale images as F1-score.

(X. Li, Tang, Christo, Zhao, & Li, 2022) proposed an Android malware detection method based on RGB image conversion using binary combinations of different files. Dex, manifest, and META-INF files containing signature information were used and converted to RGB images to represent each application. In the conversion, the bytecode sequences of the files are combined and divided into three equal parts. Then, 224x224x3 color images were obtained by placing each piece in the image's R, G, and B channels. During the training and testing phases, experiments were carried out with AndMal2017, CICMalDroid, and Drebin datasets using the VGG16 network. It has been reported that the best result is 96% accuracy obtained in the experiments performed with the AndMal dataset. Compared to similar studies with AndMal2017, it was stated that the method achieved +4% better results.

(Ye, Zhang, Li, Tang, & Lv, 2022) proposed an Android malware detection model that speeds up the training and testing phases using lightweight CNN. A fast and efficient model based on MobileNetV2 is proposed by comparing the number of parameters and training and test results of networks such as VGG16, InceptionV3, and ResNet. Using the CIC-AndMal-2020 dataset, classes.dex, AndroidManifest.xml, and resources.arsc files in the APK were converted to RGB images, creating a separate channel. In experiments with other CNN models, it was stated that the best and fastest results were obtained with MobileNetV2. It was noted that the lowest accuracy values

were obtained with VGG16. As a result of the experiments, it was stated that the model created is more effective because the VGG16 and other large models have much more parameters and require much more machine resources.

(Liu, Wang, Zhang, & Song, 2023) proposed the ImageDroid framework, which can perform malware detection, family detection, and obfuscated software detection with an image-based approach. By ignoring the header and index sections of the classes.dex file analyses were performed only in the data area. In image conversion, three-channel RGB images are preferred instead of one-dimensional grayscale images, allowing more data to be carried in the same size images. Heat maps were created on the images using the Grad-CAM algorithm to detect the visual parts that play a vital role in detecting malware. The Grad-CAM algorithm believes that the last feature maps of the convolution layer contain the most valuable data about the input data. This approach determines the parts that play a key role in detecting maliciousness by obtaining scores on feature maps. Experimental results were obtained for different scenarios using three malware datasets with VGG, GoogleNet, and ResNet methods. As a result of the research, 97.2% success was achieved in malware detection, 95.1% in family detection, and 94.6% in obfuscated malware detection.

(Kinkead, Millar, McLaughlin, & O'Kane, 2021) proposed a novel method for explaining the predictions of a CNN model used for Android malware detection. The technique identifies locations in an Android app's opcode sequence deemed important by CNN and contributes to malware detection. Then, the sections highlighted by the method were compared with the state-of-the-art explainability method LIME. It shows that the areas considered most malicious by CNN match closely to those considered most malicious by LIME. This indicates that CNN can successfully focus on potentially malicious parts of applications. The researchers say that the proposed method can help security analysts detect areas of applications that exhibit malicious behavior.

In another study that processes certain parts of the dex file, (H. Zhu, Wei, Wang, Xu, & Sheng, 2023) proposed a framework called MADRF-CNN. In the research, different parts of the dex file were handled, and it was evaluated that the header section did not contain enough data for detection and the data area was larger and more complex than necessary. Considering these criteria, the operations performed on the dex file

discuss six parts consisting of String_ids, Type_ids, Proto_ids, Field_ids, Method_ids, and Class_ids sections. These extracted sections and the AndroidManifest.xml file were converted to RGB images for classification stages and used. Considering that some deep learning methods do not evaluate global context information, receptive fields, and reuse of pixel attributes, they proposed a CNN variant called MADRF-CNN, which uses a combination of max-pooling and average-pooling. By performing experiments with 2507 malicious and 1417 benign applications from the VirusShare dataset, the results were reported for both different input and network combinations. The results show that the best results were obtained with the dex file and MADRF-CNN with 95.9% F-measure. The best result that can be obtained with the manifest file is 76.3% F-measure.

(J. Singh, Thakur, Ali, Gera, & Kwak, 2020) proposed a system called SARVOTAM (Summing of neurAl aRchitecture and VisualizatiOn Technology for Android Malware identification) for identifying and classifying Android malware using deep learning and optimization algorithms. The system automatically converts non-intuitive malware features into fingerprint images and uses a fine-tuned Convolutional Neural Network (CNN) to extract rich features from visualized malware. Different image datasets were created using all combinations of APKs' Certificate, AndroidManifest.xml, classes.dex, and resources.arsc files (15 in total). In the training and testing stages, the softmax layer of CNN was replaced with machine-learning techniques such as KNN, SVM, and RF. The results showed that the combination of CNN and SVM was better for identifying and classifying Android malware families than the combinations of CNN, CNN-KNN, and CNN-RF. The CNN-SVM combination achieved a classification accuracy of 92.59% using Android certificates and manifest file images. The experiments were done using the DREBIN dataset.

(Marwaha et al., 2023) visualized all available combinations of AndroidManifest.xml, Assets, AppObjectFactories, and AppCertification files and compared their effects on binary classification. They trained VGG16 networks for 15 combinations of 4 files and shared the results with metrics such as accuracy, F1-score, and ram usage for each. With 94.01% accuracy and 88.12% F1-score, the best results were obtained using manifest and certificate files. In the experiments, RGB images

converted from the Drebin dataset were used for network training.

Although image transformation-based approaches are widely researched, audio transformation-based approaches are only recently becoming widespread. By converting the classes.dex file obtained from APK files to audio, studies that detect malware and families use different audio-based attribute groups.

(Nataraj et al., 2021) investigated the interoperability of different attribute sets in malware detection. Audio features, image similarity features, and some static and statistical features were extracted, and orthogonality research was conducted on the interoperability of these feature sets. The orthogonality of two feature sets to each other was investigated with the JFS metric. JFS calculates based on the ratio of common errors of two attribute sets. If the common errors of the two feature sets are low, it is thought that the other will classify the samples that one cannot classify. In this case, the JFS value of the two attribute sets will converge to 1. In the opposite case, the JFS value will converge to 0 as errors will be common. In the tests, the best results were obtained with 97% accuracy with audio-based attributes and Nearest Neighbor. It has also been seen that audio-based features are more orthogonal with other features. As a result, it is seen that the calculation of orthogonality is important in the proper investigation of feature fusion. On the other hand, the results show that JFS is insufficient for observing unbalanced datasets.

(Mercaldo & Santone, 2021) conducted audio-based Android malware detection and family classification. Their proposed model consisted of two cascaded systems that worked consecutively. The first model determined whether the incoming sample was malicious or benign, and if the decision indicated maliciousness, the second model was used for family classification. The dataset used in the study included 24,553 malware samples (71 families) and 25,447 benign samples. The benign samples were obtained by downloading applications from various categories of official stores and testing them using Virus Total. To perform classification, the classes.dex files within the APK files were converted into .wav format, and classical audio-based features were extracted to obtain a feature vector for each application. The classification stages involved using k-Nearest Neighbors (kNN), Support Vector Machines (SVM), Logistic Regression, and a 4-layer Neural Network. All classifiers underwent 5-fold cross-validation. The

most successful result was the Neural Network, with a classification accuracy of 0.952 and a family detection accuracy of 0.922.

(Casolare, Iadarola, Martinelli, Mercaldo, & Santone, 2021) proposed an Android malware family detection method using audio signal processing. The suggested approach has applied audio-based feature extraction steps by converting the classes.dex file to an audio file in .wav format. To observe the parsing of the extracted features, the features are visualized with boxplots for different families. After each application's feature vectors were obtained, the training and testing processes were carried out using RF, SGD classifiers, and two MLPs based on backpropagation. The results of the experiment with the data set containing 4796 samples with ten classes, which were created by combining different data sets, were shared. The best results were obtained with the MLP-2 with 90% F-measure and 98.8% accuracy.

In their study (Tarwireyi, Terzoli, & Adigun, 2022), researchers utilized a low-level BFCC feature extraction approach, representing the human hearing structure more closely than MFCC. This choice was prompted by the observation that the proper form of the transformed APK files resembled noise, hence the need for noise-resistant attributes. The BFCC algorithm derives coefficients by segmenting the waveform into parts and applying a series of high and low-pass filter combinations. Experiments were conducted with CICAndMal2017 and CICMalDroid2020 datasets, with data divided into 80% for training and 20% for testing. The researchers tested the performance of 23 different Machine Learning algorithms and reported that the best results were obtained with the RandomForest and ExtraTrees algorithms, achieving a 98.6% F-measure. Conversely, the worst general performance was reported with the GaussianNb algorithm.

In 2023, (Tarwireyi, Terzoli, & Adigun, 2023) proposed a multi-audio feature-fusion approach for detecting Android malware as they noticed insufficient work in audio-based malware detection. This approach involved using MFCC, GFCC, and BFCC together to obtain a 102-feature data vector. This study used the biologically inspired GFCC coefficient for the first time in malware detection. The CICAndMal2017 and CICMaldroid2020 datasets were used, and experiments were conducted using 23 different classical machine-learning methods. They achieved the

best results by using all the features together (Classic + MFCC + BFCC + GFCC) with the 102-feature LGBM classifier, and the reported best result was a 99.65% F-measure on the CICMalDroid2020 dataset.

Apart from audio and image-based studies, different studies on Android malware detection can be summarised as follows: (Atacak, Kılıç, & Doğru, 2022) proposed a hybrid model that combines CNN and ANFIS (Adaptive network-based fuzzy inference system). They employed CNN for feature extraction and ANFIS as a decision-maker within their methodology, subsequently comparing these techniques with classical machine learning algorithms. In their experiments utilizing two distinct datasets, the highest results reported were a 92% F-score for the Drebin dataset and a 94.6% F-score for the CICMalDroid2020 dataset.

(Calik Bayazit, Koray Sahingoz, & Dogan, 2023) compared the classification performance of extracted feature groups with static and dynamic analysis methods on different data sets. Classification processes were carried out with ten classifiers using 84 dynamic features from the CIC-AndMal2017 dataset and 8115 features from the CIC-InvesAndMal dataset. The experiments performed with RF, DT, three different ANNs, three different MLPs, LSTM and CNN-LSTM reported that the best result for static data was obtained when LTSM was used with 98.75 accuracies. On the other hand, for dynamic features, it has been reported that the best result is 95.3 accuracy obtained with CNN-LSTM. It was shared that the lowest results were 91.64 accuracies obtained with DT in static analysis and 85.3 accuracies obtained with ANN in dynamic analysis.

(Memon, Unar, Ahmed, Daudpoto, & Jaffari, 2023) developed an Android malware detection system based on semi-supervised machine learning. Application permissions and API call logs are used as attributes in the study. A dataset of 54332 applications was used to test the proposed malware detection system. Both Google Play Store and Virus Total were used to create this dataset. A feature set was made by extracting 1488 permissions and API calls from these applications. 80% of the dataset is reserved for training, while the remaining 20% is reserved for testing. In the study, a semi-supervised Naive Bayes algorithm is proposed. In addition to this algorithm, DT, SVM, RF, and KNN algorithms are also used to compare with the proposed

method. As a result of the evaluation of the test set, the highest performance was obtained with the proposed semi-supervised NB algorithm. This result is 93.256% according to the accuracy metric. The results obtained from other classifiers are DT 90.12%, SVM 88.23%, RF 82.45%, and KNN 86.35%, respectively.

(Yilmaz, Taspinar, & Koklu, 2022) proposed a permission-based Android malware detection system. In the study, a dataset consisting of 2854 malicious and 2870 benign applications was used. A feature set was created with 116 permissions extracted from all these applications. The numerical data obtained after the creation of the feature set was classified by machine learning techniques. SVM and NB machine learning techniques were used in the study. As a result of the classifications obtained with ten cross-validations, 92.4% classification success was obtained with the NB algorithm, while 90.9% classification success was obtained with the SVM algorithm.

(Q. Li, Chen, Li, et al., 2023) proposed a model that detects Android malware by interpreting program instructions in library files as genes. The method obtains basic block sequences from the instructions extracted from the library files, and size reduction is performed with information gain. Then, semantic abstraction was performed with word2vec to basic block sequences filtered with information gain. In the experiments performed with the VirusShare 2018 dataset, 8000 malware and 2000 benign samples were used, and experiments were carried out with different classifiers on the data. The results produced using different information gain thresholds and word lengths are shared in detail. It has been reported that the best result obtained in the experimental results was obtained with DNN with 97.51 accuracy.

3 PRELIMINARIES

Permission-based approaches can show high performance in Android malware detection. Vector representations are created for each application from the permissions extracted by processing APK files. Then, classification steps are carried out using classical machine-learning approaches. Using machine learning algorithms makes the number of dimensions in the datasets an essential factor. Working with high-dimensional data causes the training phases to take longer and may negatively affect the classification performance. In this direction, researches on reducing the number of features has been carried out. In (Şahin, Kural, Akleylek, & Kılıç, 2021c), supervised and unsupervised dimension reduction techniques were examined. In the comparisons, it was seen that LDA shortened the training time and increased the classification performance. However, since LDA is a supervised technique, it is unlikely to be applied to a real-time system. Therefore, other approaches that can be applied in real-time systems were investigated. First, a linear regression-based feature selection method was developed (Şahin, Kural, Akleylek, & Kılıç, 2021b). Then, filtering-based feature selection methods were adapted to permission-based Android malware detection (Şahin, Kural, Akleylek, & Kılıç, 2021a). In both approaches, the number of features has been reduced by 80%, and it has contributed to the classification performance. To increase the classification performance in static approaches, besides eliminating low-related features with feature selection methods, high-contribution features can be highlighted with term weighting methods in areas such as text mining. In a similar approach, we adapted 14 different term weighting methods to permission-based Android malware detection (Kural, Sahin, Akleylek, & Kılıç, 2019). The details of the study are given in Section 3.1.

Many preprocessing steps are required to implement permission-based approaches. However, permission-based methods do not offer a solution against obfuscated malware and first-day attacks because they focus only on requested permissions. In this context, a binary image transformation-based framework that does not require preprocessing steps such as feature extraction in malware detection has been researched (Kural et al., 2021). In the proposed framework, gray-level

images were obtained from APK files, and malware detection was carried out with CNN. The details of the study are given in Section 3.2.

3.1 Permission Weighting Approaches in Permission Based Android Malware Detection

Permission information from application manifest files is used for permission-based Android malware detection. In classical permission-based studies, application permissions are handled as binary according to whether the permission is requested. This binary structure is also frequently used in text classification studies. The terms in the texts are treated as binary. Although this indicates the presence of the term in that class, it does not show the importance of the term for the class. For this reason, terms are weighted with different term weighting methods. In this way, besides the existence and non-existence of a word belonging to a certain category, the importance of that term for the relevant class is also expressed. With a similar approach, the importance of each permission for the benign and malware classes also differs. In other words, as in text classification, the features' weighting can positively contribute to class distinction. With this motivation, we conducted a comparative study by adapting the 14 term weighting method to permission-based Android malware detection. The contributions of the study can be summarized as follows.

- 14 term weighting methods frequently used in text classification are adapted to permission-based Android malware detection.
- By performing classification experiments with three classification algorithms, results were obtained for each of the 14 cases. The results are shared in tables comparatively.

3.1.1 Proposed Method

This study adapted 14 different term weighting methods to permission-based Android malware detection. Details of the term weighting methods are given in Section 3.1.1.1. The experiments were conducted using two different datasets. The first dataset is the M0Droid dataset containing 200 benign and 200 malicious samples (Damshenas, Dehghantanha, Choo, & Mahmud, 2015). The second dataset consists of 2000 malware samples randomly taken from the AMD dataset and 1000 benign samples downloaded from Google Play Store (*Google play store*, 2023).

3.1.1.1 Term Weighting Methods

Term weighting is an expression that represents the proximity of a term to the content of a document. In our scenario, each term is permission, and each term weight is how much permission relates to the space of sample applications. Some basic expressions are used when calculating the term weight for a term. These are:

a: Number of malware applications that requested the p_j permission,
b: Number of malware applications that did not request the p_j permission,
c: Number of benign applications that requested the p_j permission,
d: Number of benign applications that did not request the p_j permission,
N: Total number of applications, calculated as $N = (a + b + c + d)$

Using these notations, the term weighting methods and formulas used in the study are given in this section.

IDF: A metric that measures how common a term is in a document type. In the assumption for permissions, the rarer permission is in a positive class, the higher its value (Salton & Buckley, 1988). The IDF is calculated as in Equation 3.1.

$$IDF = log\left(\frac{N}{a+c}\right) \tag{3.1}$$

CHI: CHI is a metric commonly used in statistics that measures how different (independent) data are from an expected distribution. When used for permissions, it indicates how much a p_j permission is associated with a class. The calculation of CHI is shown in Equation 3.2.

$$CHI = N * \left(\frac{((a*d)-(b*c))^2}{(a+c)*(b+d)*(a+b)*(c+d)}\right) \tag{3.2}$$

OR: It is a metric that considers the distribution of the term between classes (Lan, Tan, Su, & Lu, 2008). The OR is calculated as in Equation 3.3.

$$OR = log\left(\frac{a * d}{b * c}\right) \tag{3.3}$$

RF: RF is a metric that focuses on the presence of a term in categories (Lan et al., 2008). Its main logic is that the more a high-frequency term is concentrated in the positive category relative to the negative category, the more it contributes to selecting positive samples from negative samples. The calculation of RF is given in Equation 3.4.

$$RF = log\left(2 + \frac{a}{max(1, c)}\right) \tag{3.4}$$

MI: The amount of knowledge that may be gleaned from one random variable given another is measured by a concept known as mutual information. It is used in both term weighting and feature selection. The calculation of MI is given in Equation 3.5.

$$MI = log\left(N\frac{a}{(a + b) * (a + c)}\right) \tag{3.5}$$

TGF: It is a metric that expresses the total number of times the term is used in classes (Maisonnave, Delbianco, Tohmé, & Maguitman, 2019). Indicates the number of times permission is used in malware and benign instances in weighting permissions. The TGF is calculated as in Equation 3.6.

$$TGF = a + c \tag{3.6}$$

IDFEC: It is a supervised variant of the IDF. Its basic logic is to avoid reducing the weight of terms that appear in the same category more than once (Domeniconi, Moro, Pasolini, & Sartori, 2015). The IDFEC is calculated as in Equation 3.7.

$$IDFEC = log\left(\frac{c+d}{c}\right) \tag{3.7}$$

GSS: The Galavotti-Sebastiani-Simi (GSS) metric was first proposed as a feature selection method (Galavotti, Sebastiani, & Simi, 2000). This metric is proposed as a simplified variation of the CHI metric. This metric is also used in term weighting studies (Maisonnave et al., 2019). It is proposed on the principle that negative instances contribute to the classification contribution of terms. The GSS is calculated as in Equation 3.8

$$GSS = log\left(2 + \frac{a+c+d}{c}\right) \tag{3.8}$$

IG: The amount of information that a term's existence or absence adds to the process of correctly classifying a subject into a category is measured by IG. This metric is used as a feature selection method as well as a term weighting method (Lan et al., 2008; Maisonnave et al., 2019). The feature selection method answers the question of how much information the term contains from the categories. A similar approach is seen in term weighting. The calculation of IG is given in Equation 3.9.

$$\begin{aligned} IG = &\frac{a}{N} * \log\frac{a*N}{(a+c)*(a+b)} + \frac{b}{N} * \log\frac{b*N}{(b+d)*(a+b)} \\ &+\frac{c}{N} * \log\frac{c*N}{(a+c)*(c+d)} + \frac{d}{N} * \log\frac{d*N}{(b+d)*(c+d)} \end{aligned} \tag{3.9}$$

GR: Gain Ratio introduces a normalization term termed intrinsic information in an effort to reduce the bias of IG for heavily branching predictors. The calculation of GR is given in Equation 3.10.

$$GR = \frac{IG}{-\left(\frac{a+b}{N}\right) * \log\left(\frac{a+b}{N}\right) - \left(\frac{c+d}{N}\right) * \log\left(\frac{c+d}{N}\right)} \tag{3.10}$$

DESCR: It expresses how much a term describes a class. The basic logic of the metric is that the more instances a term has in a class, the better a descriptor for that class (Maisonnave et al., 2019). Therefore, the instances passed in a class are calculated as all instances for that class. For permissions, it is the ratio of malware instances where permission is passed to the total number of malware instances. The DESCR is calculated as in Equation 3.11.

$$DESCR = \frac{a}{a+b} \tag{3.11}$$

DISCR: Represents how distinctive a term is for a class. Represents how distinctive a term is for a class (Maisonnave et al., 2019). It works with the logic that the more a term is specific to a class (only in that class), the more distinctive it is for that class. The DISCR is calculated as in Equation 3.12.

$$DISCR = \frac{a}{a+c} \tag{3.12}$$

FDD: It is a metric created by combining DESCR and DISCR metrics (Maisonnave et al., 2019). The FDD is calculated as in Equation 3.13.

$$FDD = \frac{(1+beta)^2 * DISCR * DESCR}{beta^2 * DISCR + DESCR} (beta = 0,477) \tag{3.13}$$

3.1.1.2 Experimental Results

This section reports the classification results obtained from weighting the samples in M0Droid and AMD datasets with 14 different term weighting methods. The classification results obtained with M0Droid are reported in Tables 3.1 and 3.2. The results obtained with the AMD data set are given comparatively in Tables 3.3 and 3.4.

In experiments with the M0Droid dataset, the most successful classification results were obtained with SVM. In classifications made with SVM, the most successful results were obtained with DISCR-weighted data. The performance obtained with DISCR weighted data is 0.873 accuracy and 0.882 F-measure. IG and GR are the weighting techniques that affect the classification performance worst in SVM. In classifications with data weighted with IG and GR, accuracy and Fmeasure decreased by more than -5% compared to other term weighting techniques. It is seen that the results obtained in the experiments with other term weighting methods are pretty close to each other.

The best classification result with KNN was obtained with data weighted with IDF. When using IDF, the performance of the KNN classifier is 0.854 accuracy and 0.850 F-measure. After IDF, the highest performance was achieved with IDFEC, a variant of IDF. In the experiments with the data weighted with IDF in the KNN classifier, +5% better results were obtained than the classification made with binary data.

When the classification results with NB were examined, 0.791 accuracy and 0.804 F-measure were obtained with the NB classifier in the data weighted with the TGF weighting technique. It is seen that the classification performance obtained with NB is lower than the results obtained with SVM and KNN.

When Tables 3.3 and 3.4 are examined, it is seen that the most successful result in the AMD dataset is obtained when classification is made with the SVM algorithm. The best results obtained are 0.963 accuracy and F-measure 0.948. MI and RF are the other two-term weighting methods in which the SVM algorithm performs well. The lowest performances with SVM were obtained in TGF. All methods except TGF and CHI show accuracy over 0.90, while TGF remains at 0.83. All other weighting methods showed similar results with SVM.

When KNN is used, the highest accuracy was obtained with the OR weighting method. The OR weighting method measured 0.959 accuracies and 0.939 F-measure classification performance. The highest performance after OR was obtained when using GR and IG.

The most successful results obtained with the NB algorithm were obtained with

Table 3.1. M0Droid dataset classification results by accuracy

Method	**KNN**	**NB**	**SVM**
Binary	0,8	0,7814	0,8680
IDF	0,8546	0,7596	0,8310
CHI	0,7932	0,7807	0,8235
OR	0,8226	0,7709	0,8151
RF	0,8319	0,7857	0,8437
MI	0,8445	0,7746	0,8201
TGF	0,8084	0,7910	0,8294
IDFEC	0,8512	0,7864	0,8403
GSS	0,8445	0,7779	0,8386
IG	0,7798	0,7892	0,7336
GR	0,7857	0,7854	0,7403
DESCR	0,7815	0,7508	0,8193
DISCR	0,8395	0,7794	0,8739
FDD	0,8008	0,7839	0,8563

Table 3.2. M0Droid dataset classification results by F-measure

Method	**KNN**	**NB**	**SVM**
Binary	0,78	0,7915	0,872
IDF	0,85	0,7673	0,8234
CHI	0,7830	0,7909	0,8367
OR	0,8158	0,7882	0,8325
RF	0,8268	0,7960	0,8455
MI	0,8463	0,7874	0,8429
TGF	0,8079	0,8041	0,8291
IDFEC	0,8459	0,7949	0,8384
GSS	0,8432	0,7882	0,8416
IG	0,7708	0,8069	0,7835
GR	0,7792	0,8068	0,7872
DESCR	0,7866	0,7813	0,8233
DISCR	0,8348	0,7943	0,8823
FDD	0,7996	0,7950	0,8612

FDD and DESCR. With these weighting methods, 0.947 accuracies and 0.924 F-measure performance were obtained. All weighting methods showed very close results when the results obtained with the NB algorithm were examined.

When the results obtained with different classifiers are compared, it is seen that the difference created by term weighting in the results is quite low. It is concluded that a term frequency multiplier is needed for the scenario in which the term weighting

Table 3.3. AMD dataset classification results by accuracy

Method	**KNN**	**NB**	**SVM**
Binary	0,9315	0,9421	0,9632
IDF	0,9300	0,9469	0,9585
CHI	0,9488	0,9459	0,8889
OR	0,9591	0,9452	0,9600
RF	0,9327	0,9446	0,9611
MI	0,9412	0,9451	0,9628
TGF	0,9429	0,9468	0,8335
IDFEC	0,9222	0,9453	0,9576
GSS	0,9336	0,9423	0,9572
IG	0,9488	0,9456	0,9199
GR	0,9511	0,9450	0,9172
DESCR	0,9365	0,9474	0,9235
DISCR	0,9214	0,9453	0,9547
FDD	0,9326	0,9472	0,9545

Table 3.4. AMD dataset classification results by F-measure

Method	**KNN**	**NB**	**SVM**
Binary	0,8925	0,9155	0,9284
IDF	0,897	0,9219	0,9416
CHI	0,9239	0,9213	0,8491
OR	0,9399	0,9201	0,9436
RF	0,8977	0,9198	0,9455
MI	0,9149	0,9198	0,9475
TGF	0,9175	0,9222	0,7880
IDFEC	0,8810	0,9204	0,9399
GSS	0,8984	0,9158	0,9392
IG	0,9237	0,9206	0,8894
GR	0,9271	0,9201	0,8855
DESCR	0,9084	0,9249	0,8889
DISCR	0,8808	0,9202	0,9360
FDD	0,9027	0,9230	0,9362

method is used.

3.2 Apk2Img4AndMal: Android Malware Detection Framework Based on Convolutional Neural Network

With the increasing number of malware targeting Android operating systems, there is a growing need for innovative detection methods to effectively identify these

malicious entities. For these reasons, designing new detection methods attracts the attention of researchers. In the malware detection methods specific to Android operating systems, the features extracted mainly through dynamic or static analysis are evaluated through machine learning, and it is decided whether the application is benign or malicious. While feature extraction is easy in static analysis approaches, they are not sensitive to threats like code obfuscation. However, feature extraction is troublesome in the dynamic analysis approach since the application is run on an actual device or a virtual machine. Also, one of the significant disadvantages of dynamic analysis is that some malicious applications can act as benign applications by hiding their malicious behaviors when running on virtual machines or sandboxes outside the device. These analyses led us to investigate an end-to-end detection method for Android malware detection.

- An alternative framework for Android malware detection without knowing the structure and details of APK files is proposed.
- By transforming each APK file into grayscale images, applications are classified with CNN, which is highly skilled in image classification.
- The proposed framework can be used alone or in hybrid models with dynamic/static analysis models. The detection method can be enriched by using dynamic or static analysis models.

3.2.1 Proposed Framework

In this section, the infrastructure of the proposed framework will be mentioned. There are two basic parts of the proposed framework. The proposed framework is given in Figure 3.1. The first part is the image creation phase. The image creation steps consist of a series of steps that produce a grayscale image by reading the raw APK file. These steps are:

- Read APK file as binary
- Convert binary sequences to numbers (0-255)
- Scale data size to 1×10000 and get 1×10000 vector
- Reshape the 1×10000 to be 100×100
- Save the obtained data as an image file

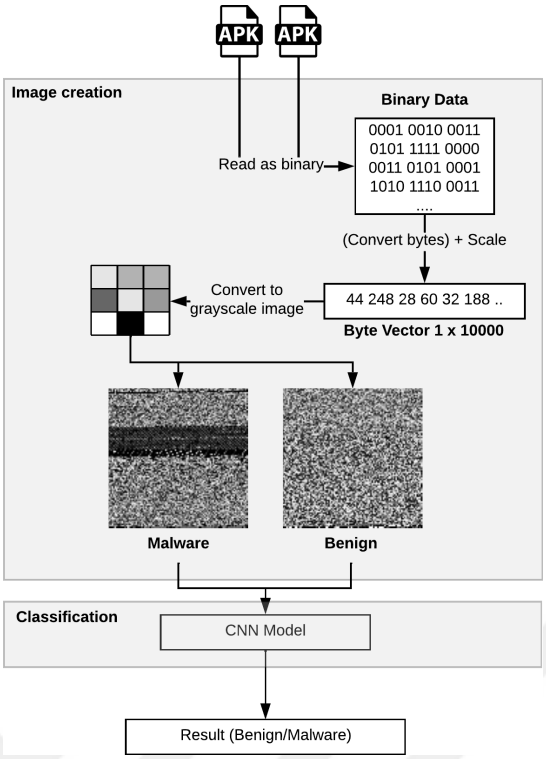


Figure 3.1. The proposed Android malware detection framework

The second part is the classification phase, where the CNN is trained and tested with the generated images. A CNN consisting of 8 hidden layers was created and used for classification at this stage. When the application framework is considered in general, it is seen that it is not necessary to know the general structure of the APK files or the malware analysis techniques for analysis operations. Since the APK files are used as a whole, there is no need to know the files, such as classes.dex, AndroidManifest.xml it contains or what information they have. From this point of view, the application framework we recommend ensures that even people with fundamental knowledge of Android malware analysis are included in the analysis processes.

3.2.2 Experimental Settings

In this section, we detail the experimental setup used in this study, including a description of the utilized dataset and an explanation of the CNN.

3.2.2.1 Used Data Set

In this study, we used Android Malware Dataset (AMD). There is 24588 Android malware in this dataset (Wei et al., 2017). There are also 3000 benign applications

that we have downloaded. Benign applications are downloaded from the APKpure site (*APKPure Android Application Store*, 2023).

3.2.2.2 Convolutional Neural Network

Deep learning, a subset of machine learning techniques, aims to learn the distributed representations of raw data along with its high-level structures using hierarchical layers. Over recent years, with the advent of efficient usage of graphic processors and the decline in hardware costs, deep learning has found its application in various fields, such as voice recognition, natural language processing, and image classification. CNN, a specific type of deep learning technique, has gained significant attention from researchers. CNNs can be utilized for feature extraction (Chen, Jiang, Li, Jia, & Ghamisi, 2016), classification (Wei et al., 2016), and clustering (Hsu & Lin, 2018).

Typically, a CNN comprises three types of layers: convolutional, pooling, and fully connected. Each layer carries out different tasks. Each layer has different tasks. The convolutional layer is responsible for extracting the features of the image using various filters. Following the convolutional layer, a pooling layer is applied, which performs dimensionality reduction by selecting the most prominent features. The final layer, known as the fully connected layer, functions similarly to an artificial neural network, performing the classification task.

3.2.2.3 Experimental Results

In this section, the results obtained in the experiments are interpreted. The study uses deep learning models in different layers and neuron structures. The models used and the performance of each model are given in Table 3.5.

When the results obtained on the network models consisting of different layers are examined, the highest performance obtained in the classification of benign and malicious applications is 94.13%, according to the accuracy metric. The network model with the highest performance consists of 4 layers. The number of neurons in these four layers is 32, 32, 64, and 64, respectively. The result with the lowest classification performance is determined as 85.12%. This result is obtained from the

Table 3.5. Results of the proposed framework

No. of layers	**No. of Neurons**	**Validation Accuracy (%)**	**Test Accuracy (%)**
2	32, 32	99.53	85.12
2	64, 64	99.43	89.5
3	32, 32, 64	99.45	94.05
3	64, 64, 64	99.63	91.3
4	32, 32, 64, 64	99.43	94.13
4	64, 64, 64, 64	99.76	93.32
4	64, 64, 128, 128	99.72	93.61

network model consisting of 2 layers. The number of neurons in each layer in this network model is 32. Similarly, in another network model composed of 2 layers, 89.5% performance is achieved. In general, it is seen that it is not enough to classify this dataset with network models consisting of 2 layers. When the number of layers is 3, two different experiments are performed. The results from these experiments were 94.05% and 91.3%, respectively. When the number of layers is 4, it is seen that the most appropriate classification results are obtained for this data set. In models with four layers, the lowest performance is 93.61%.

4 AN EXTENSIVE EXPERIMENTAL STUDY FOR ANDROID MALWARE DETECTION: INVESTIGATION OF THE EFFECT OF STATIC FEATURE GROUPS ON CLASSIFICATION PERFORMANCE

This chapter presents a comprehensive exploration of various combinations of features frequently employed in static Android malware detection. The impacts of feature sets on classification performance will be examined, employing diverse datasets and classifiers, and the corresponding results will be shared.

4.1 Motivation

A large number of feature sets are extracted in Android malware detection based on static analysis. It is reported that 16 static feature groups are widely used in malware detection in the survey study of (Q. Wu, Zhu, & Liu, 2021). The most preferred among these 16 feature groups are API calls, Native opcodes, and application permissions. In the survey study of (Pan, Ge, Fang, & Fan, 2020), which includes static analysis studies, it is emphasized that researchers mostly prefer API calls and application permissions as static features. In the survey study of (T. Sharma & Rattan, 2021), 10 different static feature groups are considered. It is stated that these are frequently used by researchers in Android malware detection. There are similar situations in the survey studies of (Jusoh et al., 2021) and (Alzubaidi, 2021), and it is seen that different feature set selections are made in the researches. Considering previous studies, determining which feature set(s) is more effective in classification performance in Android malware detection is one of the open problems in this field. Studies examining the effect of static feature groups on classification performance are quite limited. Researchers who want to develop an Android malware detection framework based on static analysis are also unsure about which feature groups they should use, so they usually consider the most commonly used feature group to develop the framework. In this context, this study answers the question of which static feature groups are more effective in classification performance with extensive experiments. In the study, API call signature, Intent filters, Manifest permission, and Command signature feature groups, which are frequently used by researchers in survey studies, are handled, and 15 different static feature combinations

are evaluated under 2 different data sets. In addition, conversion to RGB images is made by combining static feature groups under different conditions. RGB images are used together with convolutional neural network (CNN) techniques to detect malware. By making performance comparisons between the classification of static features directly with classical machine learning techniques and the classification of RGB images with CNN, the question of what advantages and disadvantages these two techniques have compared to each other will be answered.

4.2 Contribution

The main contributions of the study are as follows:

- Many features are extracted with static analysis and evaluated by machine learning. Using all of these features sometimes works well. Even if good results are obtained, giving a large number of inputs to machine learning algorithms will increase the computational cost of the algorithm and reduce its efficiency. In this study, precise results are given to researchers about which static feature groups are more effective in malware classification.
- Four static feature groups are examined in this study. These feature groups are API call signature (ACS), Intent filters (I), Manifest permission (MP), and Command signature (CS). A total of 15 different situations are evaluated, and comprehensive experiments are presented to the researchers.
- Experiments were carried out with 10 different classifiers, and the results were observed for all scenarios. It is observed that the feature groups give similar results in both data sets. In addition, this compatibility is seen in all classification algorithms. With this inference, generalization is made according to the use cases of feature groups.
- Converting APK files to images has been very popular in recent years. In this study, RGB images are obtained with static analysis groups, and the classification of malicious software is made with CNN. When comparing the results based on the direct use of static features with the classification of the images, the highest performance is above 99% in both cases. It is also reported that small-size rendering has little effect on improving classification performance.
- Finally, to see the performance of the feature sets on samples they have not seen

before, the results were shared by giving different data sets in the experiments with CNN in training and testing. The scenarios in which the training was performed with Drebin, the test with the Malgenome dataset, the training with Malgenome, and the test with Drebin were shared as test F-scores.

4.3 Experimental Design

This section consists of 4 subsections. In the first subsection, the metrics used to compare the performance of machine learning algorithms will be discussed. In the second subsection, the datasets used will be introduced. In the third subsection, static feature groups and the combinations created from these groups will be introduced. Finally, in the fourth subsection, the RGB image transformation algorithm obtained by utilizing static properties will be emphasized.

4.3.1 Used Datasets

In this study, we utilized two different datasets: Drebin (Arp et al., 2014) and Malgenome (Zhou & Jiang, 2012). The original versions of these datasets lack benign samples. However, modified versions of these datasets were created by (Yerima & Sezer, 2018) with additional benign samples. The Drebin dataset contains 15036 applications with the added benign samples, 5560 of which are malware and 9476 benign. Similarly, the Malgenome dataset contains 3799 applications, 1260 of which are malware and 2539 are benign.

4.3.2 Feature Set Combinations

In this study, features extracted from APKs with static analysis techniques were used. These features are as follows:

- Applications Permissions (MP): Android apps can request particular permissions, which the user must grant. Depending on the rights it has, the software can access various resources (camera, microphone, location, gallery, etc.) and perform specific activities. Permissions are specified using the uses-permission tags in AndroidManifest.xml.
- API Call Signature (ACS): The Android application development platform has various APIs. These APIs allow apps access to system functions, device

functionalities, and features of other apps. For example, the camera API allows an application to use the device's camera. APIs are used by developers and are available in the form of Java classes and functions.

- Intent Filter (I): Intents allow Android applications to communicate with one another. The intent is a request for an action from another component (activity, service, broadcast receiver, etc.). The Intent Filter specifies which types of Intents an application can process and which components can receive those Intents. It is defined using the intent-filter element in the AndroidManifest.xml file.
- Command signature (CS): There are two kinds of commands. These are the root and botnet commands. Some root commands like "cp," "cat," "kill," and "mount" originated on Unix. Unix administrators use these commands to do various tasks. Because of the Android operating system's Unix background, some root commands can be found in Android applications. Furthermore, malware authors frequently use root commands to manage their target machines. Because of this, root commands are critical for malware identification. The reverse engineering method is used to recover the root instructions utilized by applications from Java source files.

Table 4.1. Feature count distribution across datasets and categories

	Count	
Feature Set	**Drebin**	**Malgenome**
ACS	73	72
MP	113	112
I	23	25
CS	6	6
ACS + MP	186	184
ACS + I	96	97
ACS + CS	79	78
MP + I	136	137
MP + CS	119	118
I + CS	29	31
ACS + MP + I	209	209
ACS + MP + CS	192	190
ACS + I + CS	102	103
MP + I + CS	142	143
ACS + MP + CS + I	215	215

These components are essential building blocks within the Android application file (APK) and ensure applications' functionality, security, and interaction with other

applications. The distribution of these feature groups over the data sets is given in Table 4.1.

All possible combinations were tried to measure the contribution of all feature groups to the classification performance. The feature group consisting of 15 different states, $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$, will be evaluated by classification algorithms. The structure that enables all these combinations to be obtained is shown in Algorithm 1.

Algorithm 1 Obtaining feature combinations

Input: $D[N,k], F[k,2]$

N is total number of applications. $N_1, N_2 \ldots N_N$ represents application ID. k is total number of features. $k_1, k_2 \ldots k_k$ represents name of feature. $D[N,k]$ is full dataset. $F[k,2]$ is the matrix holding the feature name and feature group information.

Output: Obtaining a sub-dataset containing all combinations

Function $ObtainingSubdataset(D[N,k], F[k,2])$
generate all combination list
$number\_of\_feature\_group \leftarrow 4$
$comb\_list \leftarrow \binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$
for $i = 1$ to size($comb\_list$) **do**
    $k\_list \leftarrow \emptyset$
    for $j = 1$ to k **do**
        if is $F[j,2]$ an element of $comb\_list_i$ **then**
            $k\_list \leftarrow k_j$ ▷ $k_j \equiv F[j,1]$
        end if
    end for
    select and save columns in k_list in matrix $D[N,k]$
end for
return all combinations
end Function

4.3.3 Image Transformation

We created image representations for each feature set combination to train the static data with the CNN. By representing each feature set in the combination in a color channel of the RGB image, we obtained visual representations for each sample. While determining the image size, we took the most extensive set in the combination and rounded it to the nearest value that can be represented in square form. For example, for ACS+MP+I, we rounded to the closest square value based on the MP with the highest number of features in the combination. We obtained 11x11 image representations by rounding off the 112 feature number of MP to 121, which is the nearest square value.

The image generation process is shown in Algorithm 2. In representations consisting of a single feature set, we printed the values of the relevant feature set to all channels. We did not create visuals for the 'CS', 'I', and 'I+CS' cases, as the image samples are too small in the samples with only the CS and I feature sets. The created images' color channels and dimensions are given in Table 4.2. The sample images created are shared in Figure 4.1a and Figure 4.1b.

Algorithm 2 RGB image generation

Input: $ACS[N, a], MP[N, b], I[N, c], CS[N, d]$

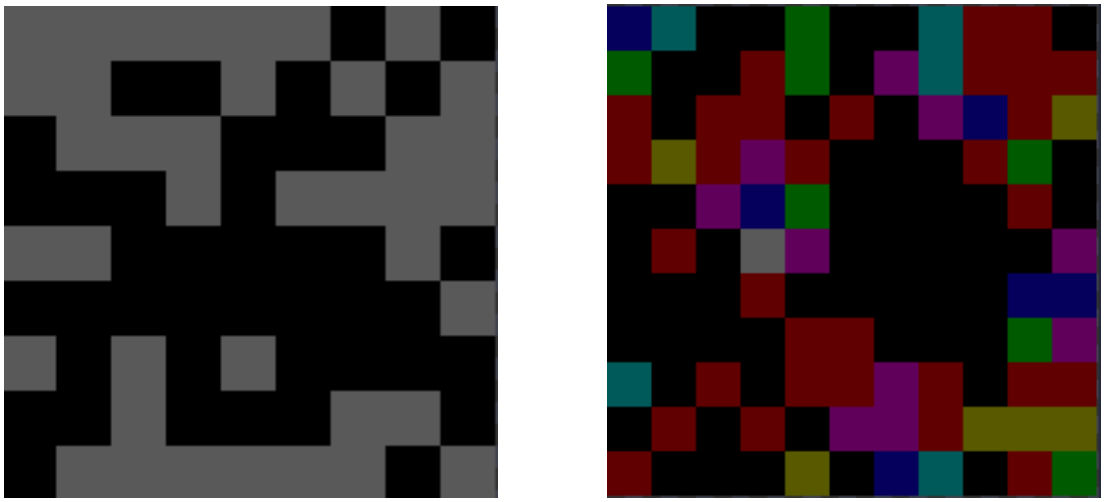
N is total number of applications. $N_1, N_2 \ldots N_N$ represents application ID. a is the total number of API features. b is the total number of MP features. c is the total number of I features. d is the total number of CS features. $D[N, k]$ is full dataset.

Output: Obtaining all combinations of RGB images

Function $ObtainingRGBImage(ACS[N, a], MP[N, b], I[N, c], CS[N, d])$
generate all combination list
$comb\_list \leftarrow \binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$
for $i = 1$ to size($comb\_list$) **do**
    find the max_size (ms) in the $comb\_list(i)$
    round the ms to the nearest perfect square number (sn)
    create a square blank image of size $\sqrt{sn}$
    insert the values of each element in comb_list(i) into channel R, G, B respectively.
end for
return all RGB images
end Function

Table 4.2. Color channel assignments for feature set combinations

	Color Channels			
Feature Sets	R	G	B	Image Size
ACS	ACS	ACS	ACS	9x9
MP	MP	MP	MP	11x11
ACS - MP	ACS	MP	-	11x11
ACS - I	ACS	I	-	9x9
ACS - CS	ACS	CS	-	9x9
MP - I	MP	I	-	11x11
MP - CS	MP	CS	-	11x11
ACS - MP - I	ACS	MP	I	11x11
ACS - MP - CS	ACS	MP	CS	11x11
ACS - I - CS	ACS	I	CS	9x9
MP - I - CS	MP	I	CS	11x11
ACS - MP - I + CS	ACS	MP	I+CS	11x11



(a) Sample image from ACS

(b) Sample image from ACS_MP_I+CS

Figure 4.1. Generated image samples

4.4 Experimental Results

In this section, the results obtained from the data sets will be presented. Since two different datasets are used in the study, the results of each dataset will be explained under the relevant subsections. In the study, results are obtained with 11 different classification algorithms, including traditional machine learning and CNN from deep learning techniques. Predefined parameters are used for conventional machine learning algorithms. For CNN, results were obtained at different layers and parameters. In the following sections, the architecture of the CNN model created will be given. The sklearn (scikit-learn) library is used for traditional machine learning algorithm (Pedregosa et al., 2011; *scikit-learn: machine learning in Python scikit-learn 1.2.2 documentation*, 2023). Keras library is used for CNN (*Keras: Deep Learning for humans*, 2023). All classification algorithms are evaluated with 10-fold cross-validation, and the average is given.

4.4.1 Malgenome Results

The results obtained with the combination of 10 different classifiers and 15 feature sets belonging to the Malgenome dataset are shared in Table 4.3 and Table 4.4. When the individual comparisons of the feature sets were examined, it was observed that the best result was obtained with ACS. After ACS, the most successful result was obtained with MP. Although the feature number of ACS is relatively low compared

to MP (72-112), better results were obtained in all classifiers. The results indicated that both I and CS yielded lower performances compared to ACS and MP. Notably, in the scenario where CS was used independently, the maximum accuracy achieved was 0.786.

Table 4.3. Accuracy-based classification results on Malgenome dataset

	KNN	**C4.5**	**LibSVM**	**NB**	**RF**	**MLP**	**AdaBoost**	**XGBoost**	**LR**	**LDA**
ACS	0,975	0,977	0,984	0,829	0,986	**0,989**	0,976	**0,989**	0,980	0,966
MP	0,966	0,966	0,973	0,639	**0,978**	0,975	0,947	0,973	0,949	0,945
I	0,863	**0,867**	**0,867**	0,501	**0,867**	**0,867**	0,852	0,866	0,858	0,842
CS	0,431	**0,786**	**0,786**	0,761	**0,786**	**0,786**	0,761	**0,786**	0,760	0,762
ACS+MP	0,982	0,982	0,989	0,812	**0,991**	0,990	0,983	**0,991**	0,987	0,980
ACS+I	0,974	0,976	0,985	0,815	0,987	**0,989**	0,976	**0,989**	0,979	0,973
ACS+CS	0,976	0,978	0,987	0,840	0,988	0,989	0,982	**0,990**	0,981	0,973
MP+I	0,962	0,968	0,972	0,664	**0,980**	0,975	0,949	0,976	0,958	0,952
MP+CS	0,978	0,969	0,978	0,641	**0,986**	0,981	0,959	0,982	0,969	0,961
I+CS	0,871	0,891	0,894	0,518	**0,893**	**0,893**	0,866	**0,893**	0,869	0,859
ACS+MP+I	0,983	0,982	0,988	0,813	**0,991**	**0,991**	0,985	0,990	0,987	0,981
ACS+MP+CS	0,985	0,981	0,989	0,817	0,992	0,991	0,985	**0,993**	0,988	0,979
ACS+I+CS	0,979	0,979	0,987	0,822	0,989	0,990	0,983	**0,992**	0,982	0,973
MP+I+CS	0,978	0,972	0,980	0,665	**0,986**	0,982	0,962	0,982	0,972	0,963
ACS+MP+I+CS	0,985	0,982	0,989	0,817	**0,992**	**0,992**	0,983	**0,992**	0,988	0,980

Table 4.4. F-measure based classification results on Malgenome dataset

	KNN	**C4.5**	**LibSVM**	**NB**	**RF**	**MLP**	**AdaBoost**	**XGBoost**	**LR**	**LDA**
ACS	0,972	0,974	0,982	0,823	0,984	0,987	0,973	**0,988**	0,977	0,961
MP	0,961	0,961	0,969	0,638	**0,975**	0,972	0,940	0,969	0,942	0,937
I	0,849	**0,853**	**0,853**	0,491	**0,853**	**0,853**	0,837	0,852	0,843	0,827
CS	0,399	**0,706**	**0,706**	0,691	**0,706**	**0,706**	0,691	**0,706**	0,691	0,692
ACS+MP	0,980	0,980	0,987	0,806	**0,990**	0,988	0,981	**0,990**	0,985	0,977
ACS+I	0,971	0,973	0,983	0,810	0,986	**0,988**	0,973	0,987	0,977	0,970
ACS+CS	0,973	0,975	0,985	0,833	0,986	0,988	0,979	**0,989**	0,978	0,969
MP+I	0,957	0,964	0,968	0,663	**0,978**	0,972	0,942	0,973	0,952	0,946
MP+CS	0,975	0,966	0,975	0,641	**0,984**	0,979	0,954	0,979	0,965	0,956
I+CS	0,851	0,879	**0,882**	0,510	**0,882**	0,881	0,845	**0,882**	0,848	0,839
ACS+MP+I	0,980	0,980	0,986	0,807	**0,990**	0,989	0,983	0,989	0,986	0,978
ACS+MP+CS	0,983	0,978	0,987	0,811	0,990	0,990	0,983	**0,992**	0,986	0,977
ACS+I+CS	0,976	0,977	0,985	0,817	0,988	0,988	0,981	**0,990**	0,980	0,969
MP+I+CS	0,975	0,969	0,977	0,665	**0,984**	0,980	0,957	0,979	0,968	0,957
ACS+MP+I+CS	0,983	0,979	0,988	0,811	0,990	0,990	0,981	**0,991**	0,987	0,978

When the binary combinations were examined, it was seen that the most

successful pair was ACS+MP. The best result of this duo is 0.991 accuracy achieved with both RF and XGBoost. Then the highest result is 0.990 accuracy obtained with ACS+CS. Although the number of features of CS is relatively low, it has been observed to contribute higher than I in the binary combinations formed with ACS and MP. Even between ACS+CS and ACS+MP pairs, the classification performance of CS is very close, although the feature number of CS is relatively low compared to MP (6-112).

Considering the triple and quadruple combinations, the best result was 0.993 accuracy using ACS+MP+CS feature combination and XGBoost classifier. It has also been observed that performance above 0.990 can be achieved in different classification combinations. When examined in general, the results are pretty high, except for the classification results with NB.

When Table 4.3 is considered in general, it is seen that single and double combinations can give very close results to triple and quadruple combinations. The best result across the table is 0.993 accuracy obtained with ACS+MP+CS.

Considering all classification algorithms, it was seen that the highest performances for different combinations were obtained with RF, MLP, and XGBoost algorithms.

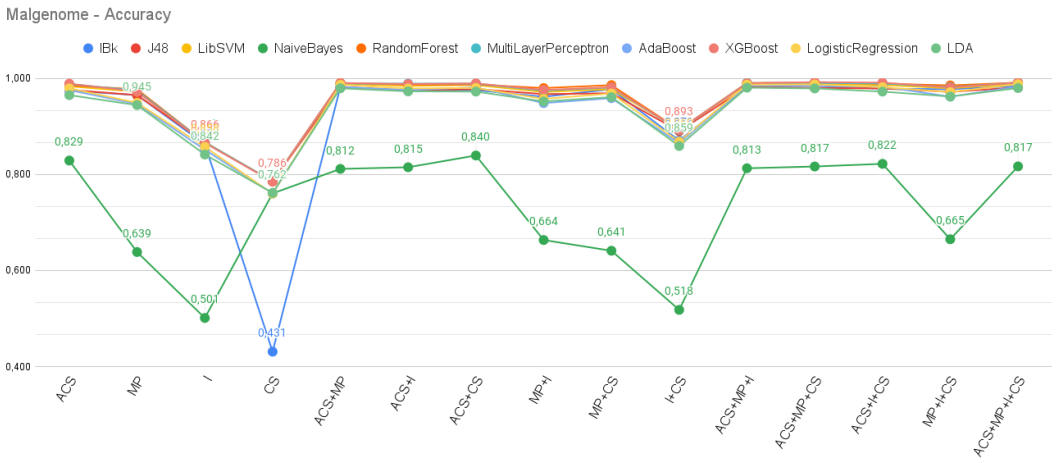


Figure 4.2. Graphical representation of classification results on Malgenome dataset

The classification results of Malgenome are visualized in Figure 4.2. When Figure 4.2 is examined, it is seen that almost all classifiers show similar tendencies

according to their feature sets. The breakdowns in the graph show that nearly all classifiers exhibit identical behavior for sets. For example, ACS performance is higher than MP in all classifiers.

In an effort to evaluate classification performance across various tools, we replicated the ACS features and MLP classifier scenario, previously recognized for its notable success in single evaluations, using Weka (Witten & Frank, 2002). Results revealed that the performance derived from this approach was closely aligned with the results from experiments executed in Python, with only minor differences observed. Detailed classification results are given in Table 4.5.

Table 4.5. Classification results using MLP classifier with ACS features on Malgenome dataset

	TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Mesure**	**Class**
	0,981	0,011	0,978	0,981	0,979	Malware
	0,989	0,019	0,991	0,989	0,990	Benign
Average	0,986	0,016	0,986	0,986	0,986	

4.4.2 Drebin Results

The results obtained with Drebin are shared in Table 4.6 and Table 4.7. When the results obtained with single combinations are examined, it is seen that the most successful results are obtained with ACS, as in Malgenome. After ACS, it was seen that MP, I, and CS gave the most successful results, respectively. The highest accuracy of 0.981 with ACS was achieved using the MLP classifier. The worst results in single combinations were obtained with CS. The highest accuracy revealed by CS was found to be 0.676.

When the binary combinations were examined, the highest results were obtained with ACS+MP with 0.989. When the binary combinations containing CS are discussed, as in Malgenome, it is seen that its contribution to classification performance can be similar or even higher than I, which has more features than itself. When ACS+I and ACS+CS were examined, it was seen that these binary combinations produced identical results. With this evaluation, it increases the preferability of CS with few features. Similar inferences are also seen when MP+CS and MP+I pairs are examined. It was observed that the lowest results in binary combinations were obtained with the

Table 4.6. Accuracy-based classification results on Drebin dataset

	KNN	C4.5	LibSVM	NB	RF	MLP	AdaBoost	XGBoost	LR	LDA
ACS	0,970	0,969	0,972	0,744	0,980	**0,981**	0,952	0,979	0,955	0,944
MP	0,948	0,956	0,952	0,590	**0,965**	0,961	0,920	0,959	0,931	0,915
I	0,762	**0,791**	**0,791**	0,540	**0,791**	**0,791**	0,780	**0,791**	0,781	0,774
CS	0,377	**0,676**	**0,676**	0,651	**0,676**	**0,676**	0,640	**0,676**	0,640	0,640
ACS+MP	0,982	0,978	0,982	0,702	**0,989**	**0,989**	0,961	**0,989**	0,973	0,962
ACS+I	0,972	0,973	0,974	0,749	**0,982**	**0,982**	0,957	0,981	0,961	0,949
ACS+CS	0,971	0,970	0,974	0,747	**0,982**	**0,982**	0,953	0,980	0,955	0,948
MP+I	0,949	0,953	0,953	0,631	**0,966**	0,964	0,925	0,959	0,938	0,922
MP+CS	0,960	0,958	0,959	0,590	**0,971**	0,969	0,926	0,965	0,936	0,925
I+CS	0,755	0,809	0,808	0,539	0,810	**0,811**	0,787	0,809	0,788	0,785
ACS+MP+I	0,979	0,979	0,983	0,715	**0,989**	**0,989**	0,964	**0,989**	0,976	0,963
ACS+MP+CS	0,982	0,978	0,983	0,705	**0,989**	**0,989**	0,962	0,988	0,974	0,962
ACS+I+CS	0,973	0,973	0,975	0,751	0,982	**0,984**	0,958	0,981	0,962	0,952
MP+I+CS	0,958	0,960	0,961	0,631	**0,971**	0,970	0,931	0,964	0,941	0,927
ACS+MP+I+CS	0,982	0,979	0,984	0,715	0,989	**0,990**	0,964	0,989	0,977	0,964

Table 4.7. F-measure based classification results on Drebin dataset

	KNN	C4.5	LibSVM	NB	RF	MLP	AdaBoost	XGBoost	LR	LDA
ACS	0,968	0,967	0,970	0,744	**0,979**	**0,979**	0,948	0,977	0,951	0,939
MP	0,944	0,952	0,947	0,581	**0,962**	0,958	0,913	0,956	0,926	0,907
I	0,739	0,768	**0,769**	0,525	**0,769**	0,768	0,757	0,768	0,758	0,752
CS	0,284	**0,666**	**0,666**	0,520	**0,666**	0,651	0,628	**0,666**	0,629	0,629
ACS+MP	0,980	0,976	0,980	0,702	**0,988**	**0,988**	0,958	**0,988**	0,971	0,959
ACS+I	0,970	0,971	0,972	0,749	**0,981**	0,980	0,954	0,980	0,958	0,945
ACS+CS	0,969	0,968	0,972	0,746	**0,980**	**0,980**	0,950	0,978	0,951	0,943
MP+I	0,945	0,949	0,949	0,628	**0,963**	0,961	0,919	0,956	0,933	0,915
MP+CS	0,957	0,955	0,956	0,581	**0,968**	0,966	0,919	0,962	0,931	0,918
I+CS	0,738	0,788	0,787	0,523	**0,789**	**0,789**	0,764	0,787	0,765	0,762
ACS+MP+I	0,978	0,977	0,981	0,715	**0,988**	**0,988**	0,961	**0,988**	0,974	0,960
ACS+MP+CS	0,981	0,976	0,982	0,705	**0,988**	**0,988**	0,959	0,987	0,972	0,959
ACS+I+CS	0,971	0,971	0,973	0,751	0,981	**0,982**	0,955	0,980	0,959	0,948
MP+I+CS	0,955	0,958	0,958	0,628	**0,969**	0,967	0,925	0,961	0,936	0,920
ACS+MP+I+CS	0,980	0,977	0,982	0,715	0,988	**0,989**	0,962	0,988	0,976	0,962

I+CS pair, which also has the lowest total number of features.

When the triple and quadruple combinations are examined, it is seen that the most successful result is 0.990, which is obtained when all feature sets are together. With the triple combinations ACS+MP+I and ACS+MP+CS, very close results are obtained with 0.989. When all classifiers are evaluated, the most successful results are obtained with RF, MLP, and XGBoost. The lowest results are obtained with NB. When Figure 4.3 is examined, it is seen that all classifiers show similar tendencies. As for the changes

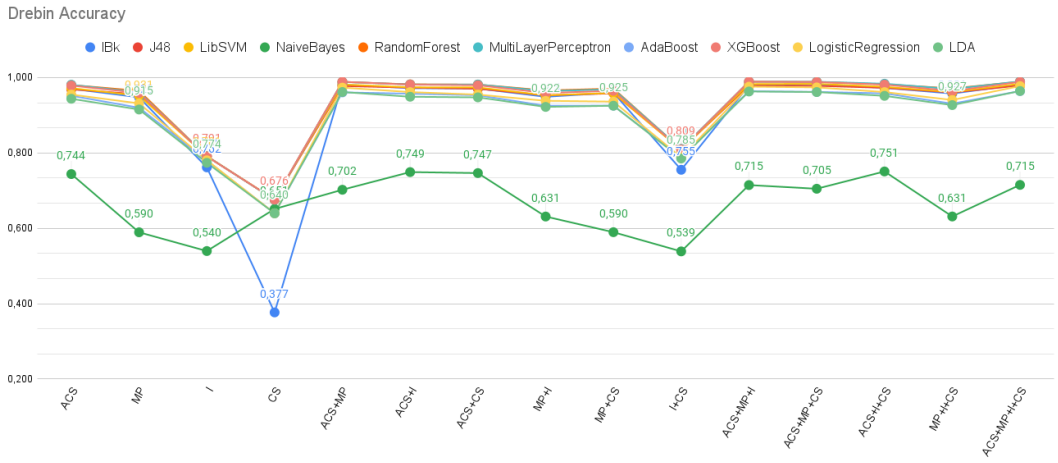


Figure 4.3. Graphical representation of classification results on Drebin dataset

between feature set combinations, they appear to be parallel to the changes observed in the Malgenome dataset (Figure 4.2).

Similar to the results presented in Table 4.5, we conducted experiments on the Drebin dataset using Weka (Witten & Frank, 2002) to investigate how different tools impact the classification performance. For this purpose, we re-implemented the ACS features and the RF classifier scenario using Weka. We found that the results obtained from the Drebin dataset using Weka were closely aligned with those obtained with Python. Detailed classification results are given in Table 4.8.

Table 4.8. Classification results using RF classifier with ACS features on Drebin dataset

	TP Rate	FP Rate	Precision	Recall	F-Mesure	Class
	0,960	0,008	0,986	0,960	0,973	Malware
	0,992	0,040	0,977	0,992	0,984	Benign
Average	0,980	0,028	0,980	0,980	0,980	

4.4.3 CNN Results

Since the images to be trained are small, the CNN model used in the training stages should not be too deep or complex. For this reason, a network of two convolution layers was created to generalize the model without overfitting. Model details are given in Figure 4.4. The training and testing processes of the network were carried out according to each combination in Table 4.2, and the results were shared in Table 4.9 and Table 4.10.

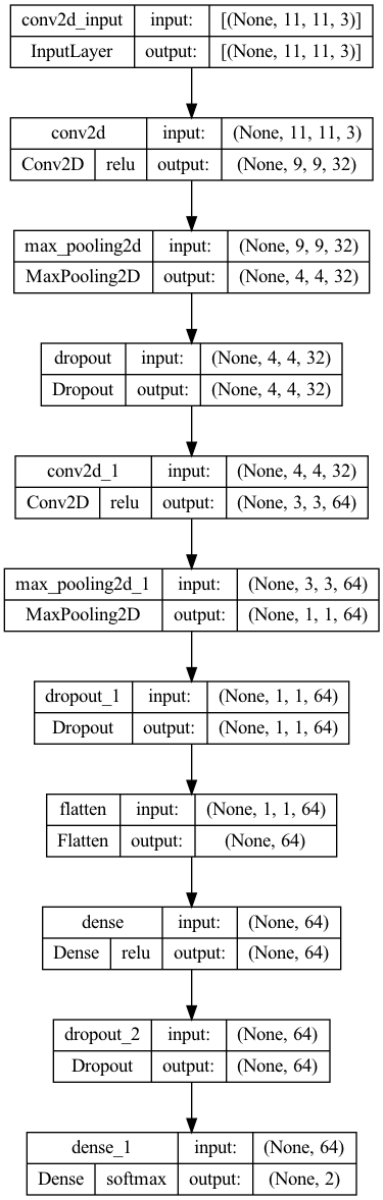


Figure 4.4. Used CNN model

Training and testing processes were carried out in the experiments performed with Malgenome and Drebin datasets and reported. The validation split in the training was determined as 0.2. To better observe the generalization performance of the models, the other dataset was used to test the trained dataset. For example, in the experiments conducted for ACS_MP_CS in the Drebin dataset, the model was tested with the ACS_MP_CS in the Malgenome dataset at the end of the training phase. A similar situation is applied for all combinations, and Tables 4.9 and 4.10 are shared as test_f1 scores.

Dropout was used to prevent memorization in the training of models. In addition,

Table 4.9. Classification results of Drebin dataset on CNN

Name	Runtime	Best Epoch	Best Val Loss	Epoch	Accuracy	Loss	Val Accuracy	Val Loss	F1-score	Test F1
ACS_ACS_ACS	152	25	0,103	30	0,967	0,099	0,963	0,105	0,967	0,983
MP_MP_MP	340	32	0,178	37	0,927	0,188	0,928	0,179	0,927	0,876
ACS_MP	197	30	0,064	35	0,976	0,070	0,978	0,068	0,976	0,984
ACS_I	188	25	0,093	30	0,969	0,089	0,967	0,096	0,969	0,955
ACS_CS	136	18	0,094	23	0,964	0,104	0,969	0,098	0,964	0,987
MP_I	133	16	0,167	21	0,931	0,182	0,941	0,169	0,931	0,834
MP_CS	137	22	0,154	27	0,936	0,156	0,935	0,155	0,936	0,954
ACS_MP_I	176	28	0,060	33	0,979	0,064	0,980	0,066	0,979	0,940
ACS_MP_CS	184	29	0,057	34	0,981	0,058	0,979	0,064	0,981	0,986
ACS_I_CS	218	39	0,073	44	0,976	0,068	0,976	0,077	0,976	0,972
MP_I_CS	169	31	0,142	36	0,946	0,131	0,945	0,143	0,946	0,902
ACS_MP_I+CS	149	26	0,061	31	0,980	0,058	0,980	0,067	0,980	0,963

Table 4.10. Classification results of Malgenome dataset on CNN

Name	Runtime	Best Epoch	Best Val Loss	Epoch	Accuracy	Loss	Val Accuracy	Val Loss	F1-score	Test F1
ACS_ACS_ACS	55	25	0,069	30	0,986	0,052	0,976	0,074	0,986	0,868
MP_MP_MP	65	28	0,130	33	0,938	0,171	0,946	0,147	0,938	0,858
ACS_MP	52	19	0,056	24	0,982	0,054	0,981	0,058	0,983	0,880
ACS_I	55	25	0,054	30	0,981	0,050	0,982	0,058	0,981	0,825
ACS_CS	36	7	0,069	12	0,974	0,081	0,973	0,073	0,975	0,847
MP_I	46	13	0,111	18	0,957	0,130	0,962	0,112	0,958	0,708
MP_CS	44	13	0,099	18	0,961	0,115	0,966	0,103	0,962	0,879
ACS_MP_I	37	7	0,057	12	0,975	0,071	0,982	0,061	0,975	0,889
ACS_MP_CS	58	27	0,039	32	0,991	0,028	0,988	0,044	0,991	0,880
ACS_I_CS	44	11	0,042	16	0,976	0,064	0,982	0,052	0,976	0,797
MP_I_CS	39	8	0,066	13	0,965	0,105	0,978	0,074	0,966	0,787
ACS_MP_I+CS	49	18	0,040	23	0,987	0,039	0,986	0,043	0,987	0,882

early stopping was applied to the models during the training, and the training was terminated if the validation loss value could not improve by five epochs.

When the results for the drebin data set in Table 4.9 are examined, it is seen that the best outcome is 0.981 F-Measure obtained with ACS_MP_CS. It is seen that the same trio received a test F-measure value of 0.986. When all models are examined, it is seen that the lowest accuracy value obtained is 0.927. It has also been observed that combinations containing ACS generally give better results in CNN networks.

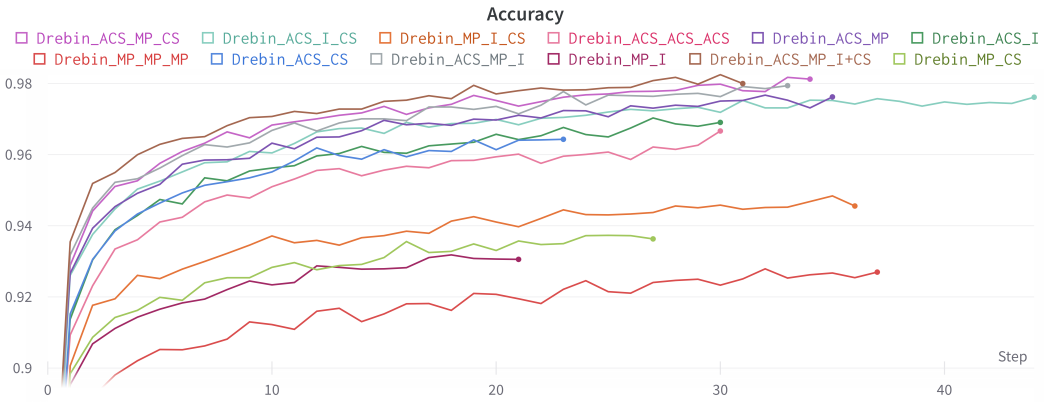


Figure 4.5. Accuracy results for feature set combinations on Drebin dataset

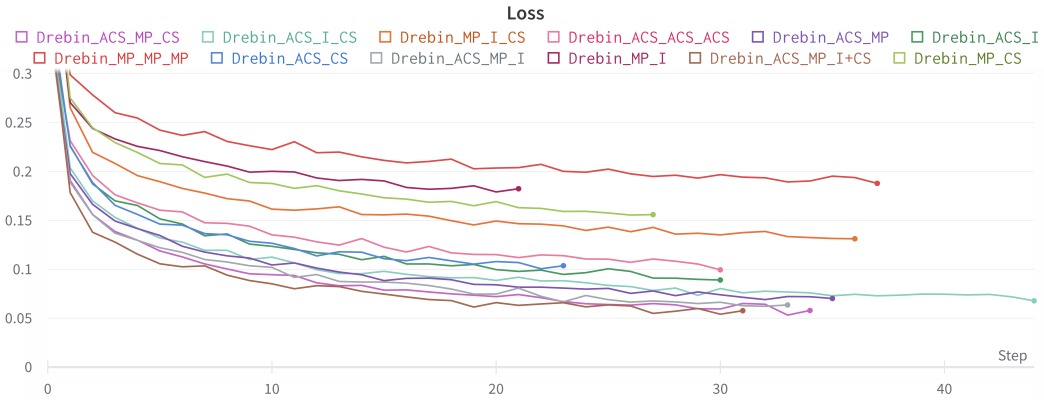


Figure 4.6. Loss results for feature set combinations on Drebin dataset

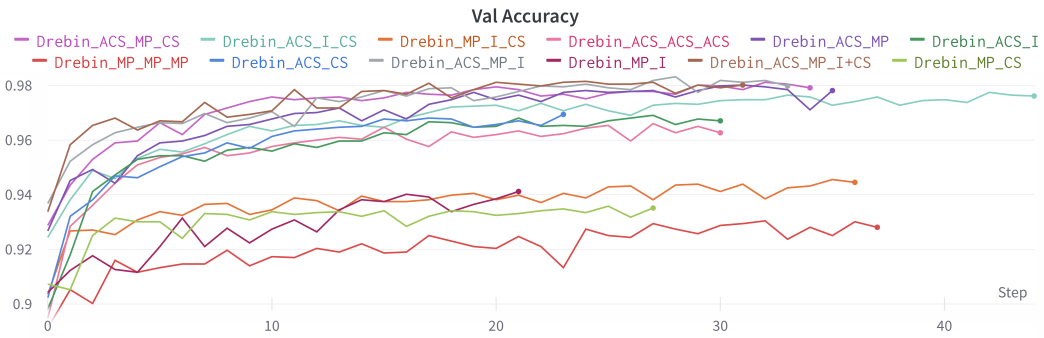


Figure 4.7. Val accuracy results for feature set combinations on Drebin dataset

The observed scores for Malgenome are similar to those of Drebin. However, in this scenario, the training set is much smaller than the test set since the Malgenome datasets are used in the training and the Drebin datasets in the test. Therefore, this scenario is more challenging than the other scenario. In the training made in this scenario, it is seen that the model achieves the best results with ACS_MP_CS. The

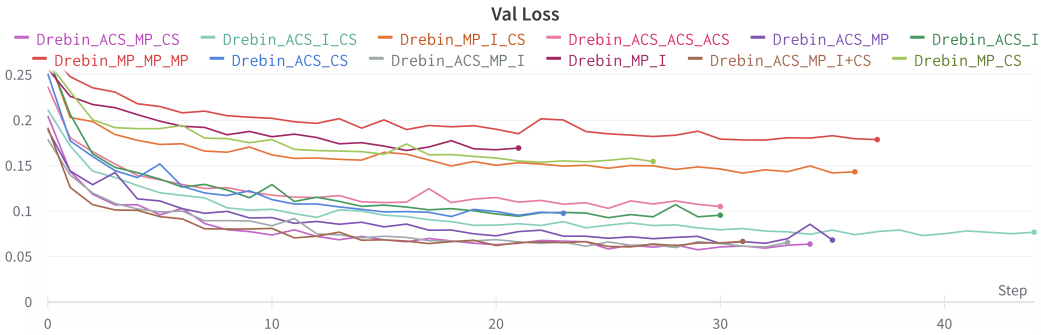


Figure 4.8. Val loss results for feature set combinations on Drebin dataset

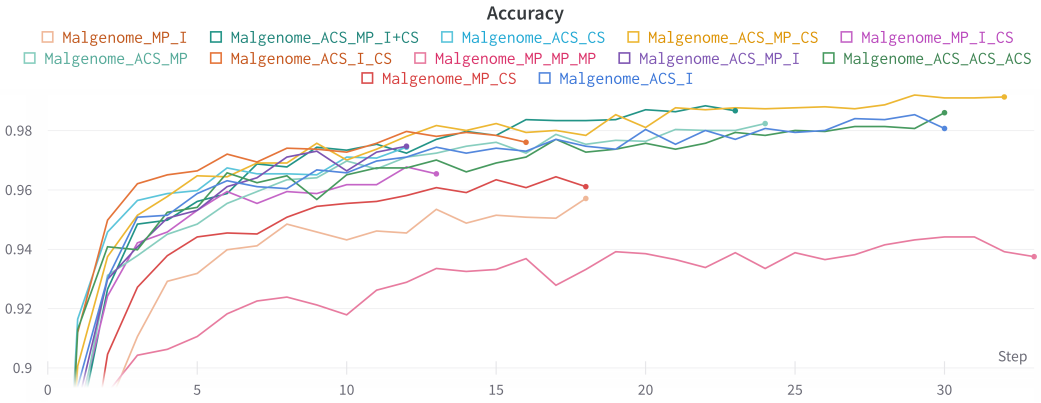


Figure 4.9. Accuracy results for feature set combinations on Malgenome dataset

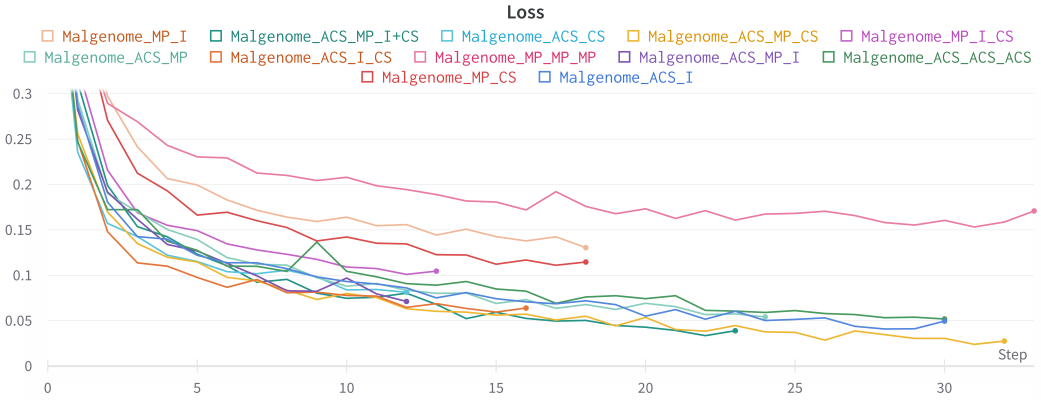


Figure 4.10. Loss results for feature set combinations on Malgenome dataset

model achieved an F-score of 0.991. When the test results are examined, the highest performance is 0.889, obtained with ACS_MP_I. However, many combinations seem to produce outcomes of around 0.88.

Accuracy, loss, validation accuracy, and validation loss graphs that emerged with the training of the models are shared in Figures 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11 and

4.12.

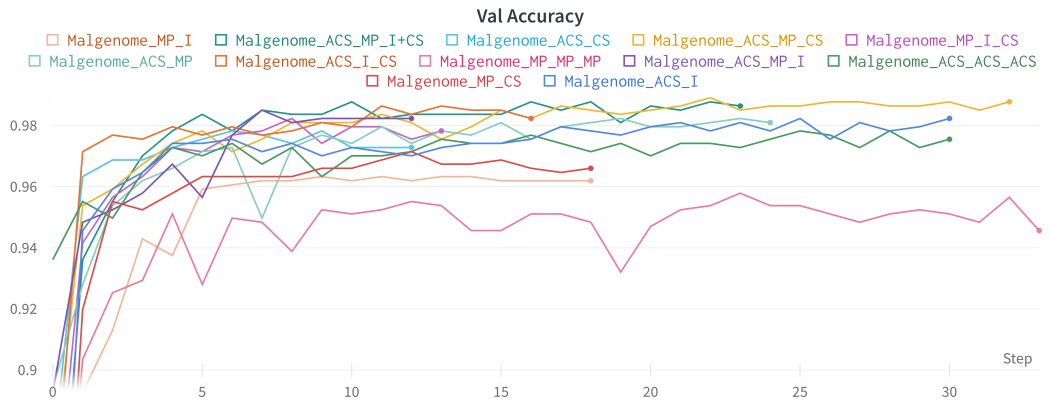


Figure 4.11. Val accuracy results for feature set combinations on Malgenome dataset

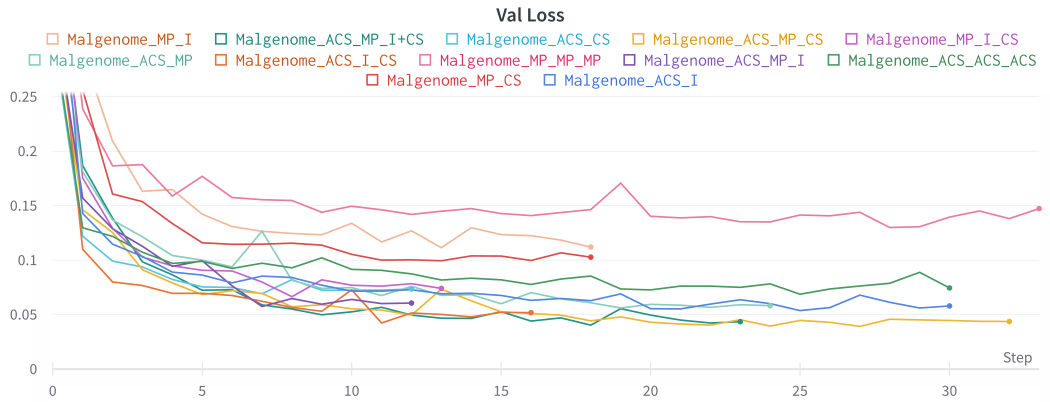


Figure 4.12. Val loss results for feature set combinations on Malgenome dataset

4.4.4 Comparison with previous studies

In this subsection, the results of the studies in the literature will be compared with the results obtained from this study. When the studies in Table 4.12 are examined, researchers perform malware detection using different static feature groups. However, no comparison was generally made with other static feature sets in the selection of the used features. Feature set selections are generally used according to similar studies or considering the relevant feature set to be important. Especially permissions and APIs are frequently preferred by researchers. However, permissions or APIs alone may not provide sufficient classification performance. Better results can be achieved if feature groups such as CS are added to these features. One of the important findings of this

research is the answer to the question of how the classification algorithms perform when used together. With this research, researchers working in static analysis can use more conscious features. For example, in (Giannakas, Kouliaridis, & Kambourakis, 2023), permissions and Intents are used as features. If API calls were preferred instead of Intent, the classification performance would have been higher. Similarly, permissions and Java source codes were evaluated with machine learning techniques in (Milosevic, Dehghantanha, & Choo, 2017). API calls would have been preferable over Java source codes.

Table 4.11. List of studies for comparison

Paper	**Year**	**ID**
(Nissim, Moskovitch, BarAd, Rokach, & Elovici, 2016)	2016	a
(Milosevic et al., 2017)	2017	b
(Tiwari & Shukla, 2018)	2018	c
(H.-J. Zhu et al., 2018)	2018	d
(A. K. Singh, Jaidhar, & Kumara, 2019)	2019	e
(Rathore, Sahay, Nikam, & Sewak, 2021)	2020	f
(Mohamad Arif et al., 2021)	2021	g
(Yilmaz et al., 2022)	2022	h
(Memon et al., 2023)	2023	i
(Giannakas et al., 2023)	2023	j
(Yao, Li, Shi, Liu, & Du, 2023)	2023	k
(Keyvanpour, Barani Shirzad, & Heydarian, 2023)	2023	l

According to the results obtained from this study, the highest performance achieved with traditional machine learning on the Drebin dataset is 99%. This result is obtained by using all static feature groups. However, classification performance is 0.01-0.02 less than this result in results obtained from triple combinations with API. A similar situation is observed in the results obtained from the Malgenome dataset. The highest performance obtained from the Malgenome dataset is obtained from the XGBoost algorithm. This result is obtained with the triple combination. When part of the Drebin dataset is used as training and part as a test, the success rate is 98.1%. On the other hand, when a part of the Malgenome dataset is used for training and some for testing, the success rate is 99.1%. It is seen that the CNN model is quite efficient on RGB images created from static features. When the results obtained from other CNN experiments are evaluated, when the Malgenome dataset is used as training, and

Drebin is tested, a success rate of 88.9% is observed. In the opposite scenario, 98.7% success occurs. It is noteworthy that the highly successful classification of unknown samples using the RGB image-based approach.

Table 4.12. Comparisons with previous studies

ID	Dataset	Feature	Best Alg.	Results(%)	Metric
a	10000 M 30000 B	Permission, Classes.dex file	SVM	98.8	Accuracy
b	200 M 200 B	Permission, Java files	AdaBoostM1	95.6	F1 score
c	669 M 652 B	Permission, API	LR	97.25	Accuracy
d	1065 M 1065 B	Permission, API, System events, URL	Rotation forest	88.26	Accuracy
e	5600 M 8000 B	Permission, API	SVM	99.6	Accuracy
f	5569 M 5721 B	Permission	RF	93.81	Accuracy
g	5000 M 5000 B	Permission	RF	91.59	Accuracy
h	2854 M 2870 B	Permission	NB	92.4	Accuracy
i	27166 M 27166 B	Permission, API	NB	93.26	Accuracy
j	1000 M 1000 B	Permission, Intent	XGBoost	84	F1 score
k	1000 M 1000 B	Permission, Intent, API, Opcode seq.	RF	97.6	Accuracy
l	5560 M 123453B	Hardware comp., API, Permission, Intent	RF	93.9	Accuracy
Ours	5560 M 9476	API, Permissions, Command signature	MLP	99	Accuracy
Ours	5560 M 9476	RGB images	CNN	98.1	Accuracy
Ours	1260 M 2539	API, Permissions, Command signature	XGBoost	99.3	Accuracy
Ours	1260 M 2539	RGB images	CNN	99.1	Accuracy

M: The number of malicious sample, **B:** The number of benign sample

5 APK2AUDIO4ANDMAL: AUDIO BASED MALWARE FAMILY DETECTION FRAMEWORK

This chapter proposes an approach to audio-based Android malware family detection. The processes of converting Android application files into audio format and subsequent extraction of audio-based features will be elaborated. Then, effective features in audio-based malware family detection will be determined by applying different feature selection methods.

5.1 Motivation

In recent years, Android malware detection and classification of Android malware according to their families is one of the important issues. Due to the importance of this issue, many new systems have been proposed and will continue to be proposed in recent years. Despite all these developments, some malware and families can easily bypass these detection systems (Kouliaridis, Barmpatsalou, Kambourakis, & Chen, 2020). Therefore, the need for up-to-date and different detection systems is increasing day by day. Especially considering systems based on static analysis, these systems are mostly vulnerable to zero-day attacks (Yan & Yan, 2018). Therefore, in addition to approaches such as static analysis, malware detection systems based on image and audio processing have been suggested by researchers in recent years (Bijitha & Nath, 2021). In this study, a framework called Apk2Audio4AndMal based on audio processing is proposed. With this framework, APK files can be considered as a more effective system against zero-day attacks, as a different representation system has emerged by converting to audio files. Because vectors with similar properties can often be seen in a static analysis or signature-based systems. However, since the structure will be completely different in audio files, the representation of each malware family or malware may also change. In addition to such advantages, there is no similar study that performs feature selection in audio-based Android malware family classification (Alswaina & Elleithy, 2020). To the best of our knowledge, this will be the first study to examine feature selection and its effects on classification performance in audio-based Android malware family classification.

5.2 Contribution

The main contributions of the study are as follows:

- Android applications were transformed into .wav format audio files and represented using audio-based features. Three features were added to the features used in addition to similar studies.
- Four distinct feature selection methods were employed to examine the impact of feature selection methods in audio-based Android malware family detection.
- Audio-based features were examined, and the features with high and low discrimination in Android malware family detection were determined.

5.3 Proposed Method

This section explains the proposed method for malware family detection and the steps followed to create this method. The stages of the proposed method are shown in Figure 5.1.

Figure 5.1. Audio features based android malware family detection workflow

The raw dataset comprises apk files for malware family detection and requires preparation before the classification stages. To prepare the data, a series of operations were executed for each Android malware sample in the dataset. Initially, the .dex files included in the apk files were exported. Then, the .dex files were read in binary format, and each bit in the data array was replaced with the corresponding signal pattern. Finally, the data was saved in .wav format by appending the appropriate file headers. The flow of converting the .dex file to a .wav format audio file is illustrated in Algorithm 3.

Since the dataset samples are represented as an audio file, they were now ready

to extract audio-based features. Audio-based features were extracted from each sample and saved in a CSV file. The following features were extracted from the audio samples.

- Chromagram is a representation of 12 pitch classes that captures the harmonic and melodic characteristics of sound (Bartsch & Wakefield, 2005; Shah, Kattel, Nepal, & Shrestha, 2019).
- RMS, which stands for root mean square, is a tool that measures the loudness of a sound sample within a window. The resulting value is an average of the total power of the audio sample.
- Spectral centroid indicates the center of mass of the spectrum. It is calculated by the weighted average of the frequencies present in the signal.
- Spectral bandwidth, bandwidth is the difference between the highest and lowest values of frequencies in a sound sample.
- Spectral rolloff is the frequency below which a specified percentage of the total spectral energy
- Zero crossing rate is a measure of how often the signal crosses zero per unit of time. Speech discrimination is frequently used in audio applications such as music genre recognition. It is one of the simplest audio-based features (Giannakopoulos & Pikrakis, 2014).
- Spectral contrast is calculated by averaging the decibel difference between peaks and valleys in each frame in the spectrum. High contrast values indicate clear sound signals, while low contrast values indicate noisy sound signals (Jiang, Lu, Zhang, Tao, & Cai, 2002).
- Flatness refers to how uniformly the frequencies in a spectrum are distributed. In other words, it shows how noisy the sound sample is. Flatness takes a value in the range of 0-1, and as it gets closer to 1, the sound becomes like white noise (Dubnov, 2004).
- Mel spectrogram is the spectrogram in which the frequencies are converted to the mel scale. It represents the sound as a single channel image as it contains time and frequency information at the same time.
- Poly returns the coefficients necessary to fit an nth-order polynomial into the columns of the spectrogram at each frame.
- Tonnetz (tonal network) is a graphical representation of tonal centroid features. It is a useful tool for understanding the harmonic structure of tonal audio.

- MFCC are coefficients that represent sound similar to human perception. It is extensively used in speech and speaker recognition applications.

Algorithm 3 Apk to audio conversion

Input: $Application_1, \ldots, Application_N$
Output: *.wav* files
1: **Function** $Convert2Audio(Application_1, \ldots, Application_N)$
2: **for** $i = 1$ to N **do**
3: export .dex file from apk file for $Application[i]$
4: read .dex file as binary for $Application[i]$
5: add .wav file headers to data
6: create .wav output file
7: replace each bit in the data array with the corresponding signal pattern
8: set .wav parameters (on the 7th, 8th, and 9th lines, the parameter of the .wav file is set)
9: $number\_of\_channels \leftarrow 1$
10: $sample\_width \leftarrow 1$
11: $frame\_rate \leftarrow 32768$
12: write data to file
13: close file
14: **end for**
15: **return** *.wav* **files**
16: **end Function**

The feature vector was extracted for each file and saved with the label indicating the malware family. After the feature extraction, a 4000×32 data matrix was obtained. In the resultant CSV file, each row represents a sample from the dataset, and each column represents an attribute extracted from the samples. At this stage, the CSV file can be used to train machine-learning algorithms for malware family detection. However, it is essential to ascertain whether all the extracted features effectively distinguish between malware families. Consequently, feature selection was performed on the acquired CSV file using CFS-Subset, Information Gain, Gain Ratio, and ReliefF algorithms. After the selections were made, the data were reduced according to the selected features, and reduced data sets were obtained. Details of the feature selection methods used are given in Section 5.3.1. In order to perform family classification, it is necessary to create a model using classification algorithms with the data gathered in CSV files. For this process, classification experiments were performed with KNN, SVM, Logistic, Random Forest, and C4.5 algorithms.

5.3.1 Feature Selection

Suppose FS is a set showing all attributes of a dataset. Finding the best subset that can be selected from this set is called feature selection. The goodness of the selected subset is the situation in which the selection is made in a way that does not adversely affect the classification performance. There are many advantages to using feature selection methods. These can be considered as reducing the computational cost, eliminating the excessive memorization problem, and running machine learning techniques efficiently. Feature selection methods are generally evaluated under 3 groups (Chandrashekar & Sahin, 2014). These are filter-based feature selection methods, wrapper feature selection methods, and embedded feature selection methods. The feature selection methods used in this study are discussed in detail in the subsections.

5.3.1.1 Information Gain (IG)

An attribute's "information gain" (IG) indicates how much data it provides about a given class. The information gain metric employs entropy from the theory of information. In practice, it is calculated based on the difference in entropy before and after the data is separated by an attribute. Equation 5.1 provides a purely mathematical formulation of the IG metric. For feature f, the IG score is found using Equation 5.1.

$$IG(f, D) = Entropy(D) - \left(\sum_{j=1}^{n} \frac{|D_j|}{|D|} * Entropy(D_j) \right) \tag{5.1}$$

In Equation 1, D represents the used dataset, f represents the evaluated feature, and $|Dj|$ represents the number of times the j value passes in f.

5.3.1.2 Gain Ratio (GR)

The IG method becomes biased when the number of distinct values a feature has is large. This is because the number of branches after division is high, and the number of samples under each branch is low. This raises problems such as overfitting. To prevent this situation, the Gain Ratio algorithm normalizes the Information Gain algorithm with SplitInfo and detects features with high representation ability. The Gain Ratio algorithm is shown in Equation 5.2.

$$Gain - Ratio = \frac{IG(f, D)}{SplitInfo} \tag{5.2}$$

5.3.1.3 CFS Subset (CSF)

Correlation-Based Feature Selection (CFS Subset) (Hall, 1999) is a feature selection algorithm that aims to select the best feature set by calculating the correlation between features. The algorithm aims to select a subset from the feature set that has high representativeness (low correlation between them) and high correlation with the class label. The correlation between features is calculated according to Equation 5.3.

$$Cor(X, Y) = \frac{(N * \sum(X_i * Y_i) - (\sum X_i) * (\sum Y_i))}{std(X) * std(Y))} \tag{5.3}$$

The numerator part of the formula expresses the covariance between X and Y features, and the denominator part expresses the product of the standard deviations of the X and Y attributes.

5.3.1.4 ReliefF (RFF)

Relief is a filter-based feature selection algorithm proposed by Kira and Rendell in 1992 (Kira & Rendell, 1992). Although it was a simple and effective method, it could only deal with two-class problems. After developing several intermediate versions, Kononenko 1994 proposed the ReliefF algorithm (Kononenko, 1994), the sixth version of the Relief algorithm (A, B, .., F), which can be applied to multi-class problems. While the ReliefF algorithm works on a multi-class dataset, R samples are selected from the dataset in each m iteration. Then, k Nearest Hits from the same class, and k Nearest Misses for each different class are found for the selected sample. The weights of the attributes are updated according to the values found. As a result of iterations, the updated weights in each round take the result values. The higher the resulting weight values, the more valuable the features are considered for classification.

5.4 Experimental Settings

We performed our experiments on a balanced 8-class dataset obtained from different datasets. We created a homogeneous dataset consisting of 4000 data in total by choosing eight classes with more than 500 samples from AMD (Wei et al., 2017) and Drebin (Arp et al., 2014) datasets. The malware families used in the experiments and which datasets they were taken from are shown in Table 5.1.

Table 5.1. Used dataset

Family	**Count**	**Origin Dataset**
Bankbot	500	AMD
DroidKubgFu	500	AMD
FakeInst	500	AMD
Fusob	500	AMD
Jisut	500	AMD
Mecor	500	AMD
Opfake	500	Drebin
Plankton	500	Drebin

5.5 Experimental Results

The results obtained by applying four different feature selection methods to the dataset are presented in Table 5.2. When the table is examined, it is seen that six features are selected with the CFS-Subset method, 15 features are selected with the Gain Ratio method, 15 features are selected with the Information Gain method, and 16 features are selected with the ReliefF method on total 31 features. It has been observed that all methods select five common features (chroma-stft, rmse, poly, mel spectrogram, mfcc1), and three or more methods select eleven common features. In the experiments, it was observed that all methods selected the Mel spectrogram we added to the feature set, and all three methods except CFS-Subset selected the flatness and contrast. The high selection rate of the added features indicates that their discrimination on the dataset is high.

When the low number of selected features is evaluated, it is seen that zero crossing rate, mfcc6, mfcc8, mfcc11, mfcc12, mfcc13, mfcc15, mfcc16, mfcc19 features are

Table 5.2. Selected features by feature selection algorithms

	ReliefF(16)	**Gain-Ratio(15)**	**CFS-Subset(6)**	**Infogain(15)**
chroma_stft	x	x	x	x
rms	x	x	x	x
spectral_centroid		x		x
spectral_bandwidth				x
rolloff		x		x
zero_crossing_rate				
contrast	x	x		x
flatness	x	x		x
mel spectrogram	x	x	x	x
poly0	x	x	x	x
tonnetz	x			
mfcc1	x	x	x	x
mfcc2	x			
mfcc3	x	x		x
mfcc4	x	x	x	
mfcc5	x			
mfcc6				
mfcc7		x		x
mfcc8				
mfcc9		x		
mfcc10	x	x		x
mfcc11				
mfcc12				
mfcc13				
mfcc14	x			
mfcc15				
mfcc16				
mfcc17				x
mfcc18	x	x		x
mfcc19				
mfcc20	x			

not selected by any algorithm. Tonnetz, mfcc5, mfcc14, and mfcc20 attributes are selected only by the ReliefF algorithm. The feature distributions by families for zero crossing rate, mfcc8, mfcc12, and mfcc19 are given in Figures 5.2, 5.3, 5.4, and 5.5, respectively.

When we examine Figure 5.2, it is seen that the zero crossing rate takes values in the same range for almost all classes. Similarly, when we examine Figure 5.3, it is seen that DroidKongFu in mfcc8 has spread over an area covering four families. When

5.4 and 5.5 are examined, it is seen that the discrimination of these features according to classes is low.

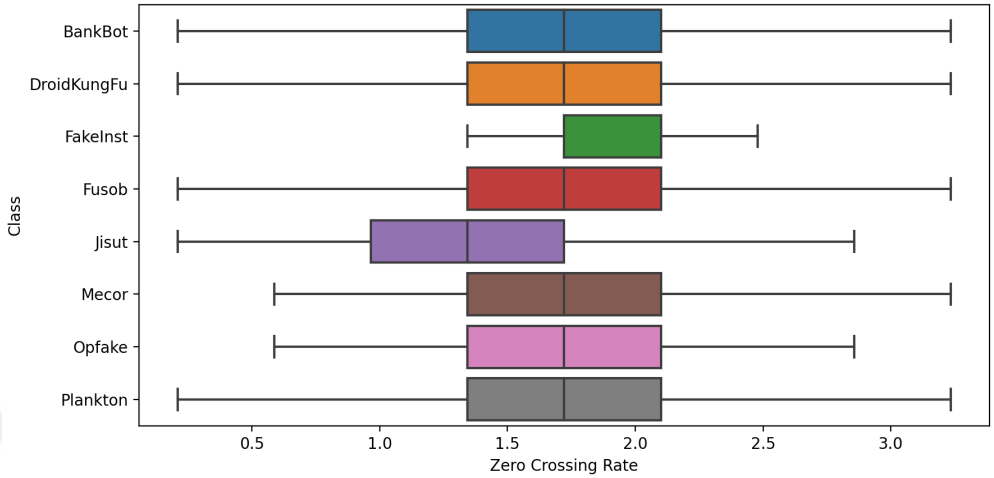


Figure 5.2. Zero crossing rate feature distribution by families - box plot

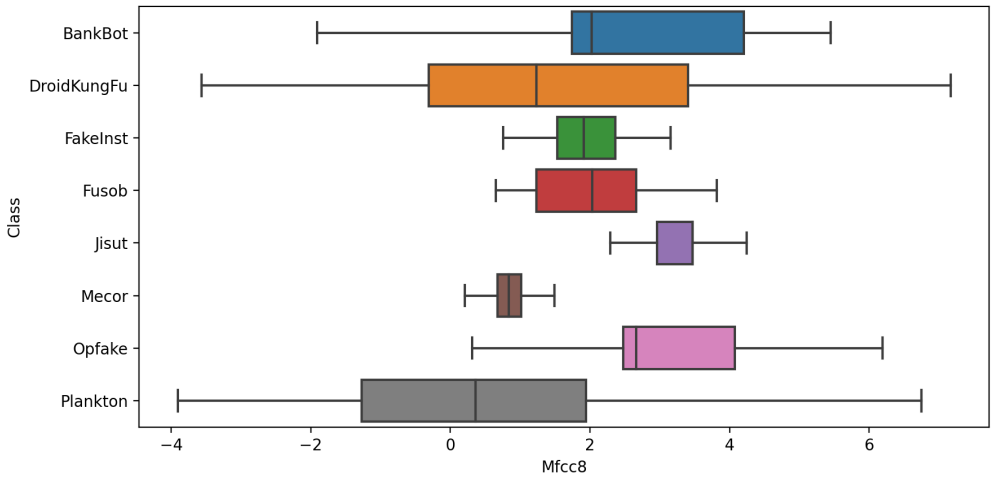


Figure 5.3. Mfcc8 feature distribution by families - box plot

When we examine the features with high discrimination, it is seen that the jisut family is differentiated from all classes in the contrast shown in Figure 5.6. It is seen that the distribution of Bankbot only coincides with DroidKungFu. Similarly, when flatness in Figure 5.7 is examined, it can be seen that Opfake, Jisut, Fusob, and FakeInst families differ from Plankton, Mecor, DroidKungFu and BankBot. Although the mean flatness values for all families are between 0.00042 and 0.00050, families such as Opfake are

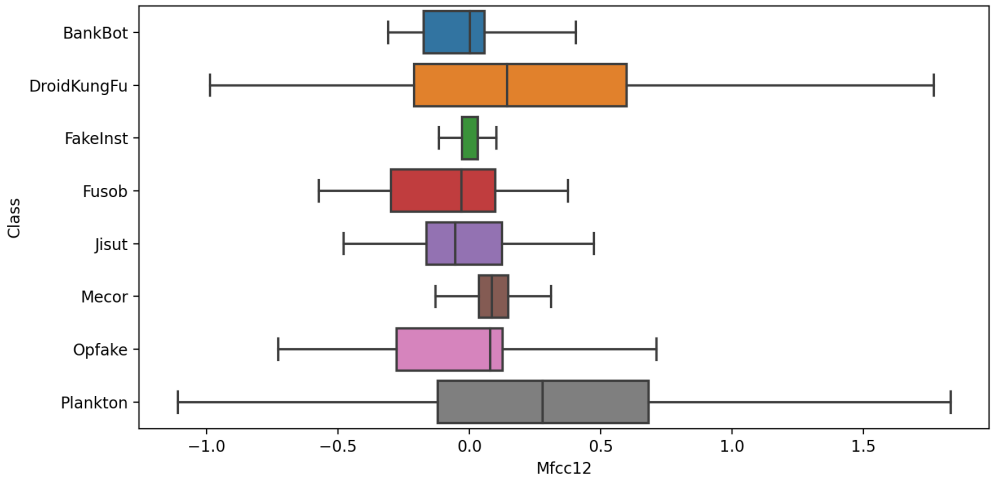


Figure 5.4. Mfcc12 feature distribution by families - box plot

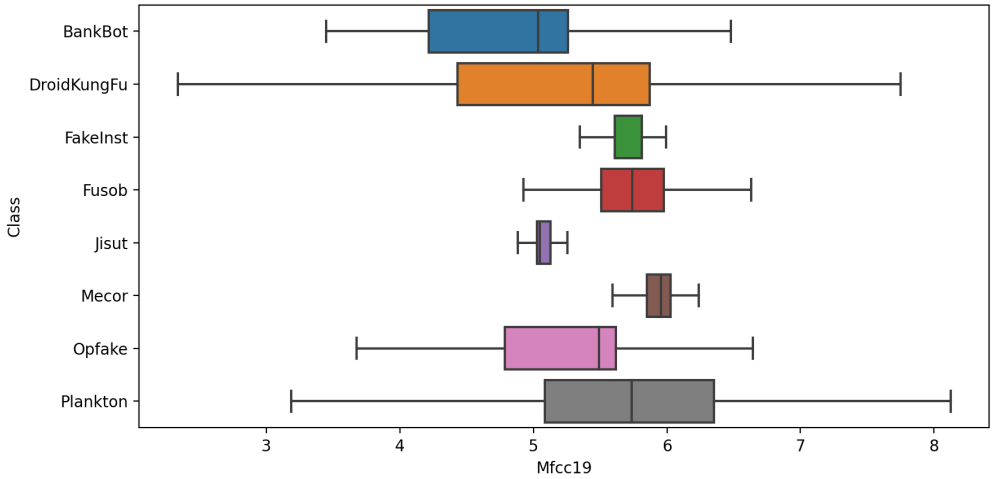


Figure 5.5. Mfcc19 feature distribution by families - box plot

in lower values.

In the feature groups selected by all methods, it is seen that the distribution for the mel spectrogram divides the families into four groups (Figure 5.8). The Jisut family is in the lowest mel spectrogram range, followed by the Opfake, Fusob, and FakeInst families. The Mecor family has a distribution between 40-41. Its specimens were differentiated from other families because of the range it was found. In the Bankbot, DroidKungFu, and Plankton families, the mel spectrogram distribution is above 41. Poly in Figure 5.9 appears to form a mel spectrogram-like family distribution.

In family classification experiments, it was preferred to use Weka (Witten &

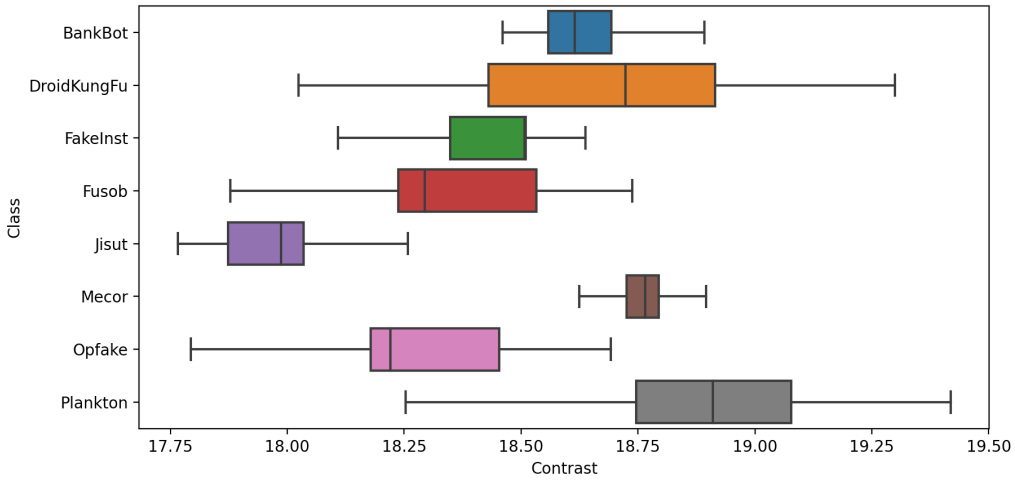


Figure 5.6. Contrast feature distribution by families - box plot

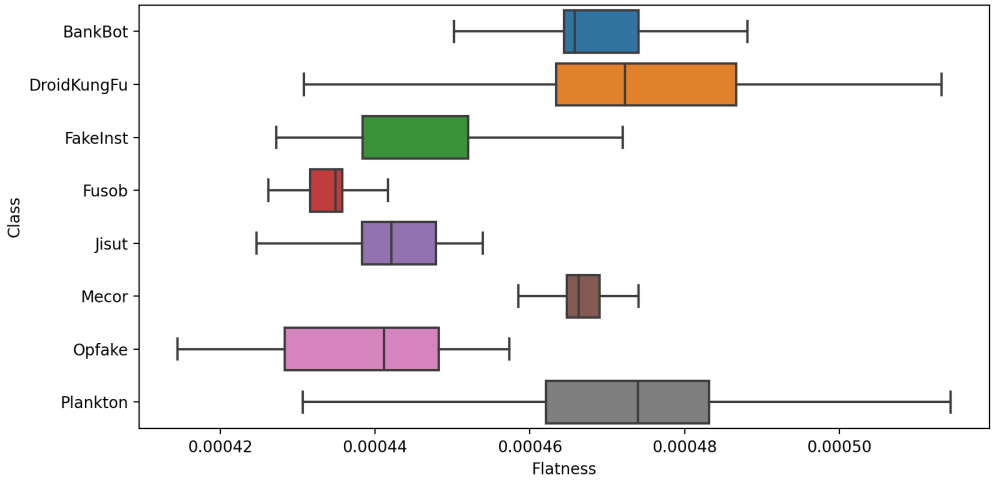


Figure 5.7. Flatness feature distribution by families - box plot

Frank, 2002), which has many classifiers ready. Classification results were obtained for each data set reduced by feature selection algorithms with KNN, Random Forest, C4.5, Logistic, and SMO algorithms. To better understand the generalizability of the models created with the classifiers, 10-fold cross-validation was applied to all classifiers. The results obtained with the classifiers according to the feature selection methods are shown in Table 5.3.

The results obtained with KNN and Random Forest are generally very close. The highest performance in all experiments was obtained using the ReliefF feature selection method and KNN classifier with 0.966 F-measure. The lowest-performing

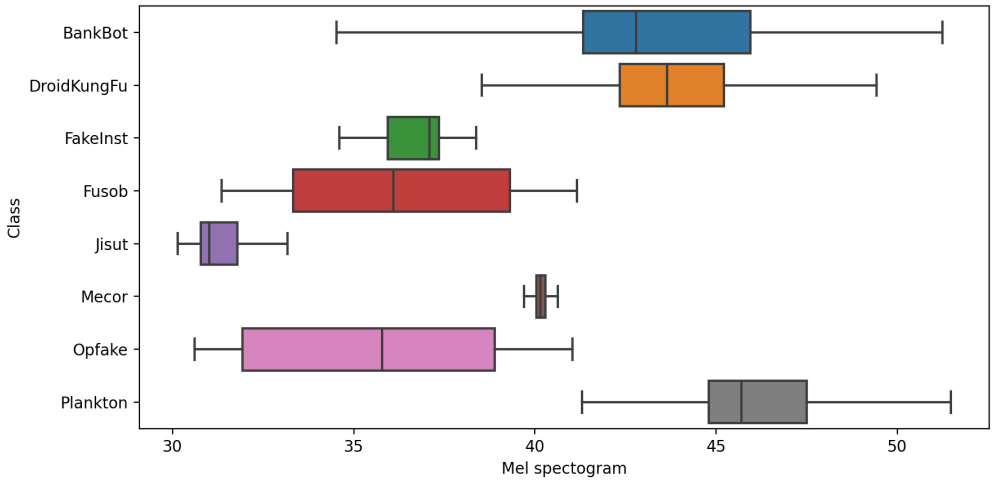


Figure 5.8. Mel spectrogram feature distribution by families - box plot

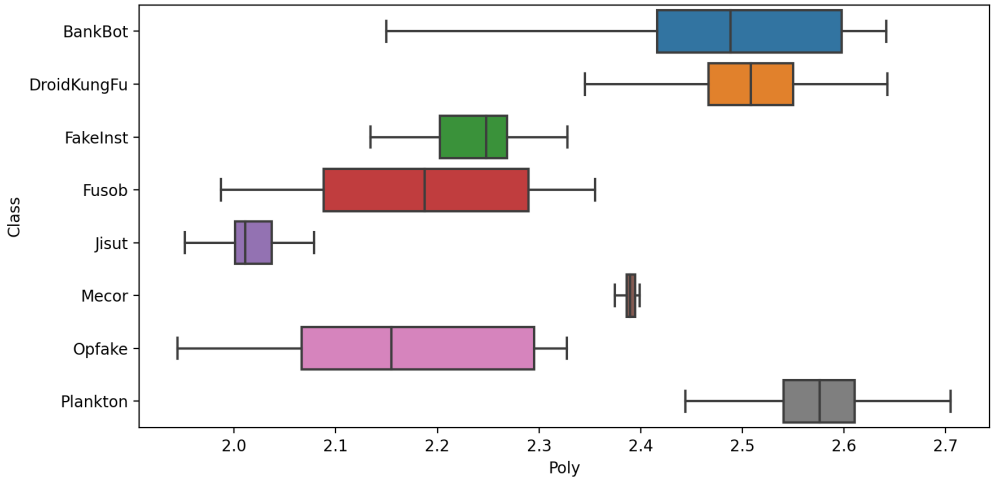


Figure 5.9. Poly feature distribution by families - box plot

Table 5.3. Classification results by feature selection algorithms and classifiers

	CFS-Subset	**Gain-Ratio**	**Infogain**	**ReliefF**	**All**
Random Forest	**0.952**	**0.961**	**0.961**	0.962	0.961
KNN	0.94	**0.961**	**0.961**	**0.966**	**0.962**
Logistic	0.671	0.823	0.834	0.832	0.89
C4.5	0.912	0.931	0.931	0.936	0.934
SMO	0.601	0.707	0.708	0.768	0.783

classifier for all feature selection methods is SMO. SMO had the highest success with 0.783 F-measure on the non-reduced dataset. However, even this value is relatively low compared to other classifiers. In classification experiments with data reduced by CFS-Subset, the best result was obtained with RF classifier with a score of 0.952. The

Table 5.4. Performance metrics using ReliefF and KNN

	TPR	FPR	Precision	Recall	F-Measure	MCC	ROC Area
BankBot	0.988	0.007	0.950	0.988	0.969	0.964	0.995
DroidKugFu	0.910	0.010	0.930	0.910	0.920	0,909	0,976
FakeInst	0.994	0.002	0.986	0.994	0.990	0.989	0.996
Fusob	0.998	0.001	0.994	0.998	0.996	0.995	0.999
Jisut	0.970	0.003	0.976	0.970	0.973	0.969	0.991
Mecor	0.998	0.002	0.984	0.998	0.991	0.990	0.998
Opfake	0.974	0.002	0.984	0.974	0.979	0.976	0.992
Plankton	0.898	0.011	0.924	0.898	0.911	0.898	0.970
Average	0.966	0.005	0.966	0.966	0.966	0.961	0.990

lowest results were obtained with SMO and Logistic, respectively. Although classification was made using only six features, quite acceptable classification performances were obtained with RF and KNN. The results obtained in classification experiments with data reduced by Gain-Ratio and Information gain are very close. Among both feature selection methods, the best classification results with 0.961 were obtained with KNN and Random Forest algorithms. Experiments with data reduced by ReliefF generally gave high results for all classifiers. RF, KNN, and C4.5 algorithms achieved higher results with the features selected with ReliefF compared to the classifications made using the whole data set.

The confusion matrix for the results obtained using ReliefF and KNN is shown in Figure 5.10. In Table 5.4, the performance metrics of the same configuration are shown.

When Table 5.4 is examined, it is seen that the Fusob and Mecor families have the best classification results and the lowest error rate, with one misclassification each. Only one sample of Fusob is classified as FakeInst. Similarly, a sample from the Mecor family is classified as BankBot. However, since the number of other classes classified as Fusob is less than those classified as Mecor, the precision of Fusob is higher than Mecor. One instance of FakeInst and two instances of Opfake from other families are classified as Fusob.

Table 5.5 compares Android malware family classification studies and the results obtained from this study. The survey on Android malware family detection did not consider an audio-based approach (Alswaina & Elleithy, 2020). As far as we examined, only two studies were found (Mercaldo & Santone, 2021; Casolare et al., 2021). When

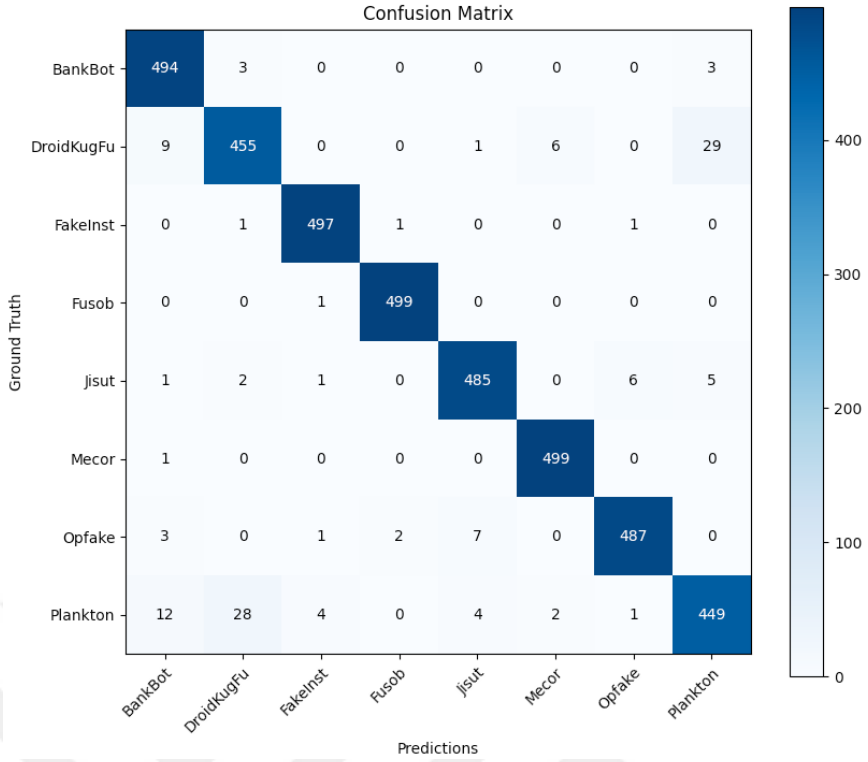


Figure 5.10. Confusion matrix of family classification results using ReliefF feature selection algorithm and KNN classifier

the results of both studies were examined, it was observed that the audio-based methods gave good results in malware family detection. Likewise, the results obtained from this study are remarkable. In addition, to the best of our knowledge, there has been no previous study that makes feature reduction in audio-based Android malware detection.

When the result of this study is compared with other studies, it was seen that good results were obtained with very few features. For example, while 3712 features were used in the study (Y. Zhang, Feng, Huang, Ye, & Weng, 2020), close results were obtained with only 16 features in this study. Also, an utterly balanced dataset using only six features with the CFS Subset method yielded more than 95% performance in this study. In the studies of the classification of Android malware according to their families, it has been seen that primarily unbalanced datasets are used, whereas a metric such as accuracy, which may be a problem in comparison, is preferred instead of a metric such as F-measure. In this study, a completely balanced and up-to-date dataset is handled, and results are given with the F-measure, which is fair to compare.

Table 5.5. Comparison of previous works

References	Year	Type	MD	FC	NC	NF	Precision	Recall	Result
(Mercaldo & Santone, 2021)	2021	Audio	Yes	Yes	71	28	0.911	0.913	0.922 Acc
(Casolare et al., 2021)	2021	Audio	No	Yes	10	29	0.907	0.907	0.988 Acc 0.907 F-Measure
(Y. Zhang et al., 2020)	2020	Manifest Features	No	Yes	10	3712	-	-	0.983 F-Measure
(Fang et al., 2020)	2020	Image Text Texture	No	Yes	15	64+	0.960	0.960	0.960 F-Measure
(Zhiwu, Ren, & Song, 2019)	2019	CFG and DFG Virtualization	No	Yes	20	313+	-	-	0.947 Acc
(Alswaina & Elleithy, 2018)	2018	Permissions	No	Yes	28	42	-	-	0.959 Acc
(Cavli & Sen, 2020)	2020	Hybrid Features	No	Yes	21	329	-	-	0.9804 Acc
Our	**2023**	**Audio**	**No**	**Yes**	**8**	**16**	**0.966**	**0.966**	**0.966 F-Measure**

MD: Malware Detection, **FC:** Family Classification, **NC:** Number of Class, **NF:** Number of Features, **Acc:** Accuracy

Following the Android malware family classification experiments, we conducted further tests to evaluate the impact of the selected features on malware detection. To assemble the dataset for experiments, we included 4038 benign samples from the CICMalDroid2020 dataset (Mahdavifar et al., 2022), along with the 4000 malicious samples previously used in the family detection phase. Experiments were conducted employing the selected features during the family classification stage. The results from these classification experiments are presented in Table 5.6.

Table 5.6. Binary classification results by feature selection algorithms and classifiers

	CFS-Subset	**Gain-Ratio**	**Infogain**	**ReliefF**	**All**
Random Forest	**0.953**	**0.958**	**0.959**	**0.960**	**0.961**
KNN	0.942	0.956	0.957	0.955	0.956
Logistic	0.921	0.930	0.930	0.928	0.943
C4.5	0.939	0.944	0.943	0.947	0.943
SMO	0.904	0.914	0.912	0.914	0.919

Upon examination of the results, it is observed that the highest results across all feature sets are achieved with RF. The peak performance in binary classification is an F-measure of 0.961, accomplished using all features. However, it should also be noted that when the selected features were used, results quite close to this value were also obtained. In the experiments where the RF classifier was used with the features selected by CFS Subset, Gain Ratio, Information Gain, and ReliefF, F-measure values of 0.953, 0.958, 0.959, and 0.960 were obtained, respectively. Considering the complexity and resource consumption brought by high dimensionality, the results obtained with the selected features are considered quite successful. In the evaluation of the classifiers, each demonstrates a performance exceeding a 0.900 F-measure. Notably, SMO and Logistic, which underperformed in family classification, show higher results in binary classification scenarios.

Detailed results for the RFF feature selection and RF classifier pair, which yielded

Table 5.7. Binary classification results using ReliefF feature selection algorithm and RF classifier

	TPR	**FPR**	**Precision**	**Recall**	**F-Measure**	**MCC**	**ROC Area**
Malware	0.970	0.050	0.951	0.970	0.960	0.920	0.993
Benign	0.950	0.030	0.969	0.950	0.960	0,920	0,993
Average	0.960	0.040	0.960	0.960	0.960	0.920	0.993

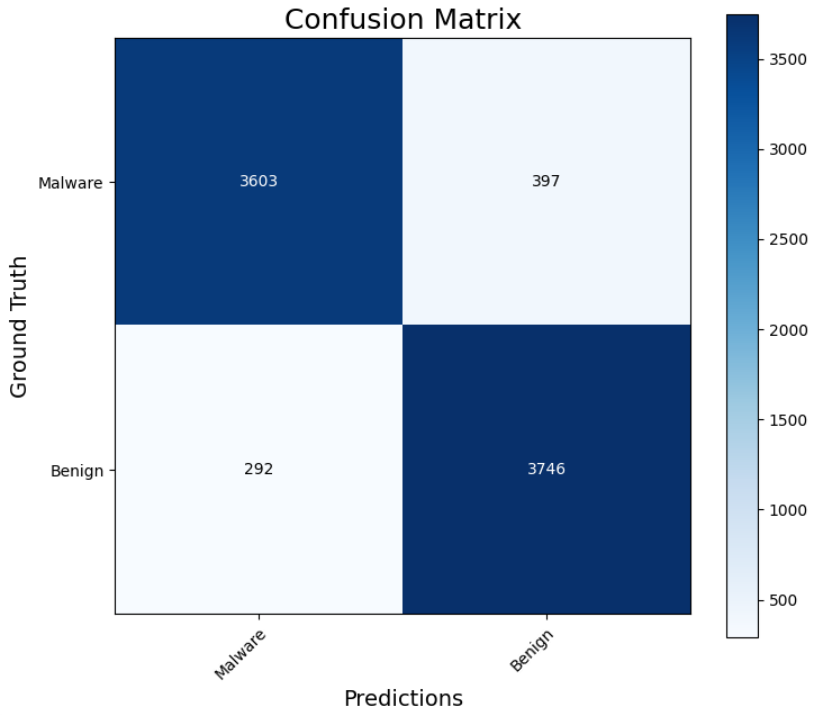


Figure 5.11. Confusion matrix of binary classification results using ReliefF feature selection algorithm and RF classifier

the highest performance in feature-selected results, are presented in Table 5.7 and Figure 5.11. Examination reveals precision values of 0.951 for the malware class and 0.969 for the benign class. These results demonstrate the audio-based approach's ability to perform highly in malware and family detection. Evaluating the obtained results, it becomes clear that the audio-based features can effectively represent the data with a limited number of features.

6 CONCLUSIONS

Android malware detection and family classification are among the widespread problems today. Although studies on permission-based approaches have brought detection approaches to a certain point, new approaches are needed against the developing malware threat every day. In this thesis, we conducted extensive research on Android malware detection and family detection, presenting a comprehensive account of the methodologies employed and the results obtained. Firstly, preliminary studies on permission-based Android malware detection are given to understand the Android malware detection process. Chapter 3 shares the studies and observations in this context. In Section 3.1 and Section 3.2, two studies that were covered within the scope of preliminary studies are detailed. Permission-based studies in Chapter 3 have led us to search for attribute sets other than permissions. Section 3.2, on the other hand, has led us to conclude that apart from classical approaches, image transformation and deep learning methods such as CNN produce effective results in malware detection. Observations derived from our preliminary studies have steered our research towards two directions in Android malware detection: investigating the impact of feature set combinations on classification performance and exploring the use of audio-image transformations.

In Chapter 4, we conducted research on the classification performance of various feature set combinations. The experiments revealed that the contribution of each feature set to the classification performance varied significantly. Among the four feature sets tested, it was observed that the API calls yielded the most favorable results. We found that the Command Signature and Intent Filter groups, which have fewer features, yield lower classification performances when used independently. Yet, when these feature groups are combined with ACS and MP, improvement in classification performance is observed. Notably, despite its lower feature count, the CS group demonstrated a higher contribution than the Intent group. This point underscores the potential efficacy of these lesser-used attribute sets when combined with others. Upon analyzing the classifiers, it was found that RF, MLP, and XGBoost demonstrated higher classification performances in both data sets. In the second part of the study, image representations

were generated from combinations of feature sets. A Convolutional Neural Network (CNN) model, including two convolutional layers, was employed to perform the training and testing processes. For each combination, the two datasets were cross-referenced, and their equivalents in the alternate dataset were utilized for testing following the training phase. The results suggest that our model is capable of achieving acceptable results even when the size of the training set is significantly smaller than that of the test set. This study performed extensive experiments with two data sets, four feature sets, ten classifiers, and a CNN. We believe these findings could serve as a reference point for researchers utilizing static features in Android malware detection and will guide them in choosing the optimal feature set - classification method combinations.

In Chapter 5, Android application files were evaluated within the audio domain. Application files were transformed into .wav audio files to perform Android malware family detection. Three additional features not included in similar studies were added to assess their impact on malware family detection. Four different feature selection methods were applied to determine the high-discriminative from low-discriminative features in Android malware detection, and the results were compared. The results indicated that one of the added features was selected by all methods, while at least three methods chose the remaining two features. This implies that these features exhibit a high discriminative capacity in Android malware family detection. Experiments with a dataset of 4000 samples divided into eight classes were conducted using five different classification algorithms. It was observed that the proposed method successfully detected malware families with an F-measure of 0.966 using only about half of the features. Even with only six features, an F-measure of 0.952 was achieved after feature selection.

Overall, it is essential to emphasize that carefully selecting the correct feature set(s) and classifier in Android malware detection studies is critical to achieving high classification performance while minimizing resource consumption. It is, therefore, a significant conclusion that researchers in this field should consider these factors due to their effects on time and resource usage. Furthermore, the application of innovative methods such as audio and image-based approaches have shown to contribute positively not only to Android malware detection but also to the detection of malware families. In light of these observations, our future research aims to develop a hybrid method for

malware and malware family detection, combining the image-based and audio-based approaches presented in this study. In addition to employing these two methods in a cascaded manner, research will also focus on visualizing the tabular data generated by the audio-based approach.

REFERENCES

AAPT2 — Android Studio — Android Developers. (2023). https://developer.android.com/tools/aapt2. ([Last access date: 12 June 2023])

Allix, K., Bissyandé, T. F., Klein, J., & Le Traon, Y. (2016). Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories* (pp. 468–471). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2901739.2903508 doi: 10.1145/2901739.2903508

Almomani, I., Alkhayer, A., & El-Shafai, W. (2022). An automated vision-based deep learning model for efficient detection of android malware attacks. *IEEE Access*, *10*, 2700–2720.

Alswaina, F., & Elleithy, K. (2018). Android malware permission-based multi-class classification using extremely randomized trees. *IEEE Access*, *6*, 76217–76227.

Alswaina, F., & Elleithy, K. (2020). Android malware family classification and analysis: Current status and future directions. *Electronics*, *9*(6), 942.

Alzubaidi, A. (2021). Recent advances in android mobile malware detection: a systematic literature review. *IEEE Access*, *9*, 146318–146349.

Androguard. (2023). https://github.com/androguard/androguard. ([Last access date: 12 June 2023])

APKPure Android Application Store. (2023). https://apkpure.com/tr/. ([Last access date: 30 June 2021])

Apktool. (2023). https://ibotpeaches.github.io/Apktool/. ([Last access date: 12 June 2023])

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss* (Vol. 14, pp. 23–26).

Arslan, R. S., & Tasyurek, M. (2022). Amd-cnn: Android malware detection via feature graph and convolutional neural networks. *Concurrency and Computation: Practice and Experience*, *34*(23), e7180.

Atacak, İ., Kılıç, K., & Doğru, İ. A. (2022). Android malware detection using hybrid anfis architecture with low computational cost convolutional layers. *PeerJ Computer Science*, *8*, e1092.

AV-ATLAS - Malware & PUA. (2023). https://portal.av-atlas.org/malware. ([Last access date: 12 June 2023])

Bakır, H., & Bakır, R. (2023). Droidencoder: Malware detection using auto-encoder based feature extractor and machine learning algorithms. *Computers and Electrical Engineering*, *110*, 108804.

Bakour, K., & Ünver, H. M. (2021). Deepvisdroid: android malware detection

by hybridizing image-based features with deep learning techniques. *Neural Computing and Applications*, *33*, 11499–11516.

Bartsch, M. A., & Wakefield, G. H. (2005). Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on multimedia*, *7*(1), 96–104.

Bijitha, C., & Nath, H. V. (2021). On the effectiveness of image processing based malware detection techniques. *Cybernetics and Systems*, 1–26.

Calik Bayazit, E., Koray Sahingoz, O., & Dogan, B. (2023). Deep learning based malware detection for android systems: A comparative analysis. *Tehnički vjesnik*, *30*(3), 787–796.

Casolare, R., Iadarola, G., Martinelli, F., Mercaldo, F., & Santone, A. (2021). Mobile family detection through audio signals classification. In *Secrypt* (pp. 479–486).

Cavli, O. F. T., & Sen, S. (2020). Familial classification of android malware using hybrid analysis. In *2020 international conference on information security and cryptology (iscturkey)* (pp. 62–67).

Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16–28.

Chen, Y., Jiang, H., Li, C., Jia, X., & Ghamisi, P. (2016). Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, *54*(10), 6232–6251.

ClassyShark: Android and Java Bytecode viewer. (2023). `https://github.com/google/android-classyshark`. ([Last access date: 12 June 2023])

Cuckoo Sandbox: Automated Malware Analysis. (2023). `https://cuckoosandbox.org/`. ([Last access date: 12 June 2023])

Damshenas, M., Dehghantanha, A., Choo, K.-K. R., & Mahmud, R. (2015). M0droid: An android behavioral-based malware detection model. *Journal of Information Privacy and Security*, *11*(3), 141–157.

Daoudi, N., Samhi, J., Kabore, A. K., Allix, K., Bissyandé, T. F., & Klein, J. (2021). Dexray: A simple, yet effective deep learning approach to android malware detection based on image representation of bytecode. In G. Wang, A. Ciptadi, & A. Ahmadzadeh (Eds.), *Deployable machine learning for security defense* (pp. 81–106). Cham: Springer International Publishing.

Darus, F. M., Ahmad, N. A., & Ariffin, A. F. M. (2019). Android malware classification using xgboost on data image pattern. In *2019 ieee international conference on internet of things and intelligence system (iotais)* (pp. 118–122).

DeGuard: Statistical Deobfuscation for Android. (2023). `http://apk-deguard.com/`. ([Last access date: 12 June 2023])

dex2jar: Tools to work with android .dex and java .class files. (2023). `https://github.com/pxb1988/dex2jar`. ([Last access date: 12 June 2023])

Ding, Y., Zhang, X., Hu, J., & Xu, W. (2020). Android malware detection method based on bytecode image. *Journal of Ambient Intelligence and Humanized Computing*, 1–10.

Ding, Y.-X., Zhao, W.-G., Wang, Z.-P., & Wang, L.-F. (2018). Automaticlly learning featurs of android apps using cnn. In *2018 international conference on machine learning and cybernetics (icmlc)* (Vol. 1, pp. 331–336).

Domeniconi, G., Moro, G., Pasolini, R., & Sartori, C. (2015). A study on term weighting for text categorization: A novel supervised variant of tf. idf. In *Data* (pp. 26–37).

Dubnov, S. (2004). Generalization of spectral flatness measure for non-gaussian linear processes. *IEEE Signal Processing Letters*, *11*(8), 698–701.

Elsersy, W. F., Feizollah, A., & Anuar, N. B. (2022). The rise of obfuscated android malware and impacts on detection methods. *PeerJ Computer Science*, *8*, e907.

Fang, Y., Gao, Y., Jing, F., & Zhang, L. (2020). Android malware familial classification based on dex file section features. *IEEE Access*, *8*, 10614–10627.

Farrokhmanesh, M., & Hamzeh, A. (2019, 06). Music classification as a new approach for malware detection. *Journal of Computer Virology and Hacking Techniques*, *15*, 1-20. doi: 10.1007/s11416-018-0321-2

Feng, J., Shen, L., Chen, Z., Wang, Y., & Li, H. (2020). A two-layer deep learning method for android malware detection using network traffic. *IEEE Access*, *8*, 125786–125796.

Galavotti, L., Sebastiani, F., & Simi, M. (2000). Experiments on the use of feature selection and negative evidence in automated text categorization. In *Research and advanced technology for digital libraries: 4th european conference, ecdl 2000 lisbon, portugal, september 18–20, 2000 proceedings 4* (pp. 59–68).

Ganesh, M., Pednekar, P., Prabhuswamy, P., Nair, D. S., Park, Y., & Jeon, H. (2017). Cnn-based android malware detection. In *2017 international conference on software security and assurance (icssa)* (pp. 60–65).

Gerardi, F., Iadarola, G., Martinelli, F., Santone, A., & Mercaldo, F. (2021, 09). Perturbation of image-based malware detection with smali level morphing techniques. In (p. 1651-1656). doi: 10.1109/ISPA-BDCloud-SocialCom -SustainCom52081.2021.00221

Giannakas, F., Kouliaridis, V., & Kambourakis, G. (2023). A closer look at machine learning effectiveness in android malware detection. *Information*, *14*(1), 2.

Giannakopoulos, T., & Pikrakis, A. (2014). *Introduction to audio analysis: a matlab® approach*. Academic Press.

Google play store. (2023). `https://play.google.com/store/apps`. ([Last access date: 12 June 2023])

Hall, M. A. (1999). *Correlation-based feature subset selection for machine learning* (Unpublished doctoral dissertation). Department of Computer Science, University of Waikato, Hamilton, New Zealand.

Hsien-De Huang, T., & Kao, H.-Y. (2018). R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections. In *2018 ieee international conference on big data (big data)* (pp. 2633–2642).

Hsu, C., & Lin, C. (2018). Cnn-based joint clustering and representation learning

with feature drift compensation for large-scale image data. *IEEE Transactions on Multimedia*, *20*(2), 421-429.

JADX: Dex to Java decompiler. (2023). `https://github.com/skylot/jadx`. ([Last access date: 12 June 2023])

Jiang, D.-N., Lu, L., Zhang, H.-J., Tao, J.-H., & Cai, L.-H. (2002). Music type classification by spectral contrast feature. In *Proceedings. ieee international conference on multimedia and expo* (Vol. 1, pp. 113–116).

Jusoh, R., Firdaus, A., Anwar, S., Osman, M. Z., Darmawan, M. F., & Ab Razak, M. F. (2021). Malware detection using static analysis in android: a review of feco (features, classification, and obfuscation). *PeerJ Computer Science*, *7*, e522.

Keras: Deep Learning for humans. (2023). `https://keras.io/`. ([Last access date: 12 June 2023])

Keyvanpour, M. R., Barani Shirzad, M., & Heydarian, F. (2023). Android malware detection applying feature selection techniques and machine learning. *Multimedia Tools and Applications*, *82*(6), 9517–9531.

Kinkead, M., Millar, S., McLaughlin, N., & O'Kane, P. (2021). Towards explainable cnns for android malware detection. *Procedia Computer Science*, *184*, 959–965.

Kira, K., & Rendell, e. a., Larry A. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Aaai* (Vol. 2, pp. 129–134).

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of relief. In F. Bergadano & L. De Raedt (Eds.), *Machine learning: Ecml-94* (pp. 171–182). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kouliaridis, V., Barmpatsalou, K., Kambourakis, G., & Chen, S. (2020). A survey on mobile malware detection techniques. *IEICE Transactions on Information and Systems*, *103*(2), 204–211.

Kumar, A., Sagar, K. P., Kuppusamy, K., & Aghila, G. (2016). Machine learning based malware classification for android applications using multimodal image representations. In *2016 10th international conference on intelligent systems and control (isco)* (pp. 1–6).

Kural, O. E., Kiliç, E., & Aksaç, C. (2023). Apk2audio4andmal: Audio based malware family detection framework. *IEEE Access*, *11*, 27527–27535.

Kural, O. E., Sahin, D. O., Akleylek, S., & Kılıç, E. (2019). Permission weighting approaches in permission based android malware detection. In *2019 4th international conference on computer science and engineering (ubmk)* (p. 134-139). doi: 10.1109/UBMK.2019.8907187

Kural, O. E., Sahin, D. O., Akleylek, S., Kılıç, E., & Ömüral, M. (2021). Apk2img4andmal: Android malware detection framework based on convolutional neural network. In *2021 6th international conference on computer science and engineering (ubmk)* (p. 731-734). doi: 10.1109/UBMK52708.2021.9558983

Kural, O. E., Sahin, D. O., & Kiliç, E. (2023). An extensive experimental study for android malware detection: Investigation of the effect of static feature groups on

classification performance. *Submitted article*.

Lan, M., Tan, C. L., Su, J., & Lu, Y. (2008). Supervised and traditional term weighting methods for automatic text categorization. *IEEE transactions on pattern analysis and machine intelligence*, *31*(4), 721–735.

Lashkari, A. H., Kadir, A. F. A., Taheri, L., & Ghorbani, A. A. (2018). Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 international carnahan conference on security technology (iccst)* (pp. 1–7).

Li, Q., Chen, G., Li, B., et al. (2023). Android malware detection based on program genes. *Security and Communication Networks*, *2023*.

Li, X., Tang, Y., Christo, M. S., Zhao, Z., & Li, Y. (2022). Android malware application detection method based on rgb image features in e-commerce. *Journal of Internet Technology*, *23*(6), 1343–1352.

Liu, P., Wang, W., Zhang, S., & Song, H. (2023, 04). Imagedroid: Using deep learning to efficiently detect android malware and automatically mark malicious features. *Security and Communication Networks*, *2023*, 1-11. doi: 10.1155/2023/5393890

Luo, J.-S., & Lo, D. C.-T. (2017). Binary malware image classification using machine learning with local binary pattern. In *2017 ieee international conference on big data (big data)* (pp. 4664–4667).

Mahdavifar, S., Alhadidi, D., & Ghorbani, A. A. (2022). Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. *Journal of network and systems management*, *30*, 1–34.

Maisonnave, M., Delbianco, F., Tohmé, F. A., & Maguitman, A. G. (2019). A flexible supervised term-weighting technique and its application to variable extraction and information retrieval. *Inteligencia Artificial*, *22*(63), 61–80.

Maniriho, P., Mahmood, A. N., & Chowdhury, M. J. M. (2022). A survey of recent advances in deep learning models for detecting malware in desktop and mobile platforms. *arXiv preprint arXiv:2209.03622*.

Marwaha, A., Malik, R. Q., Beram, S. M., Rizwan, A., Kishore, K. H., Thakur, D., ... Shabaz, M. (2023). Visualisation-based binary classification of android malware using vgg16. *IET Software*.

McAfee Mobile Threat Report. (2023). `https://www.mcafee.com/content/dam/global/infographics/McAfeeMobileThreatReport2021.pdf`. ([Last access date: 12 June 2023])

Meijin, L., Zhiyang, F., Junfeng, W., Luyu, C., Qi, Z., Tao, Y., ... Jiaxuan, G. (2022). A systematic overview of android malware detection. *Applied Artificial Intelligence*, *36*(1), 2007327. Retrieved from `https://doi.org/10.1080/08839514.2021.2007327` doi: 10.1080/08839514.2021.2007327

Memon, M., Unar, A. A., Ahmed, S. S., Daudpoto, G. H., & Jaffari, R. (2023). Feature-based semi-supervised learning approach to android malware detection. *Engineering Proceedings*, *32*(1), 6.

Mercaldo, F., & Santone, A. (2021). Audio signal processing for android malware

detection and family identification. *Journal of Computer Virology and Hacking Techniques*, *17*(2), 139–152.

Milosevic, N., Dehghantanha, A., & Choo, K.-K. R. (2017). Machine learning aided android malware classification. *Computers & Electrical Engineering*, *61*, 266–274.

MobSF: Mobile Security Framework. (2023). `https://github.com/MobSF/Mobile-Security-Framework-MobSF`. ([Last access date: 12 June 2023])

Mohamad Arif, J., Ab Razak, M. F., Awang, S., Tuan Mat, S. R., Ismail, N. S. N., & Firdaus, A. (2021). A static analysis approach for android permission-based malware detection systems. *PloS one*, *16*(9), e0257968.

Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011). Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security* (pp. 1–7).

Nataraj, L., Mohammed, T. M., Nanjundaswamy, T., Chikkagoudar, S., Chandrasekaran, S., & Manjunath, B. (2021). Omd: Orthogonal malware detection using audio, image, and static features. In *Milcom 2021-2021 ieee military communications conference (milcom)* (pp. 703–708).

Nissim, N., Moskovitch, R., BarAd, O., Rokach, L., & Elovici, Y. (2016). Aldroid: efficient update of android anti-virus software using designated active learning methods. *Knowledge and Information Systems*, *49*, 795–833.

Pan, Y., Ge, X., Fang, C., & Fan, Y. (2020). A systematic literature review of android malware detection using static analysis. *IEEE Access*, *8*, 116363–116379.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Dubourg, e. a., Vincent (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, *12*, 2825–2830.

Peiravian, N., & Zhu, X. (2013). Machine learning for android malware detection using permission and api calls. In *2013 ieee 25th international conference on tools with artificial intelligence* (pp. 300–305).

Rathore, H., Sahay, S. K., Nikam, P., & Sewak, M. (2021). Robust android malware detection system against adversarial attacks using q-learning. *Information Systems Frontiers*, *23*, 867–882.

Şahin, D. Ö., Kural, O. E., Akleylek, S., & Kılıç, E. (2021a). A novel android malware detection system: adaption of filter-based feature selection methods. *Journal of Ambient Intelligence and Humanized Computing*, 1–15.

Şahin, D. Ö., Kural, O. E., Akleylek, S., & Kılıç, E. (2021b). A novel permission-based android malware detection system using feature selection based on linear regression. *Neural Computing and Applications*, 1–16.

Şahin, D. Ö., Kural, O. E., Akleylek, S., & Kılıç, E. (2021c). Permission-based android malware analysis by using dimension reduction with pca and lda. *Journal of Information Security and Applications*, *63*, 102995.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, *24*(5), 513–523.

scikit-learn: machine learning in Python scikit-learn 1.2.2 documentation. (2023). https://scikit-learn.org/stable/. ([Last access date: 12 June 2023])

Shah, A., Kattel, M., Nepal, A., & Shrestha, D. (2019). *Chroma feature extraction.* Retrieved from https://www.researchgate.net/profile/Ayush-Shah-6/publication/363487456_Chroma_Feature_Extractionpdf/data/631f9a1770cc936cd301efc1/Chroma-Feature-Extraction.pdf

Shahid Alam, A. K. D. (2023). Mining android bytecodes through the eyes of gabor filters for detecting malware. *The International Arab Journal of Information Technology (IAJIT)*, *20*(02), 30 - 39. doi: 10.34028/iajit/20/2/4

Sharma, M., Chawla, M., & Gajrani, J. (2016). A survey of android malware detection strategy and techniques. In *Proceedings of international conference on ict for sustainable development: Ict4sd 2015 volume 2* (pp. 39–51).

Sharma, T., & Rattan, D. (2021). Malicious application detection in android—a systematic literature review. *Computer Science Review*, *40*, 100373.

Singh, A. K., Jaidhar, C., & Kumara, M. A. (2019). Experimental analysis of android malware detection based on combinations of permissions and api-calls. *Journal of Computer Virology and Hacking Techniques*, *15*, 209–218.

Singh, J., Thakur, D., Ali, F., Gera, T., & Kwak, K. S. (2020). Deep feature extraction and classification of android malware images. *Sensors*, *20*(24), 7013.

Singh, J., Thakur, D., Gera, T., Shah, B., Abuhmed, T., & Ali, F. (2021). Classification and analysis of android malware images using feature fusion technique. *IEEE Access*, *9*, 90102–90117.

Statista — Forecast number of mobile users. (2023). https://www.statista.com/statistics/218984/number-of-global-mobile-users-since-2010/. ([Last access date: 12 June 2023])

Statista — Global mobile os market share. (2023). https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. ([Last access date: 12 June 2023])

Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., & Blasco, J. (2014). Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications*, *41*(4, Part 1), 1104-1117. Retrieved from https://www.sciencedirect.com/science/article/pii/S0957417413006088 doi: https://doi.org/10.1016/j.eswa.2013.07.106

Tarwireyi, P., Terzoli, A., & Adigun, e. a., Matthew O. (2022). Barkdroid: Android malware detection using bark frequency cepstral coefficients. *Indonesian Journal of Information Systems*, *5*(1), 48–63.

Tarwireyi, P., Terzoli, A., & Adigun, M. O. (2023). Using multi-audio feature fusion for android malware detection. *Computers & Security*, 103282.

Tasyurek, M., & Arslan, R. S. (2023). Rt-droid: a novel approach for real-time android application analysis with transfer learning-based cnn models. *Journal of Real-Time Image Processing*, *20*(3), 1–17.

Tiwari, S. R., & Shukla, R. U. (2018). An android malware detection technique based on optimized permissions and api. In *2018 international conference on inventive research in computing applications (icirca)* (pp. 258–263).

Ünver, H. M., & Bakour, K. (2020). Android malware detection based on image-based features and machine learning techniques. *SN Applied Sciences*, *2*, 1–15.

Vasan, D., Alazab, M., Wassan, S., Safaei, B., & Zheng, Q. (2020). Image-based malware classification using ensemble of cnn architectures (imcec). *Computers Security*, *92*, 101748. Retrieved from `https://www.sciencedirect.com/science/article/pii/S016740482030033X` doi: https://doi.org/10.1016/j.cose.2020.101748

Vu, L. N., & Jung, S. (2021). Admat: A cnn-on-matrix approach to android malware detection and classification. *IEEE Access*, *9*, 39680–39694.

Wei, F., Li, Y., Roy, S., Ou, X., & Zhou, W. (2017). Deep ground truth analysis of current android malware. In *Detection of intrusions and malware, and vulnerability assessment: 14th international conference, dimva 2017, bonn, germany, july 6-7, 2017, proceedings 14* (pp. 252–276).

Wei, Y., Xia, W., Lin, M., Huang, J., Ni, B., Dong, J., ... Yan, S. (2016). Hcp: A flexible cnn framework for multi-label image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*(9), 1901-1907.

Witten, I. H., & Frank, E. (2002). Data mining: practical machine learning tools and techniques with java implementations. *Acm Sigmod Record*, *31*(1), 76–77.

Wu, B., Chen, S., Gao, C., Fan, L., Liu, Y., Wen, W., & Lyu, M. R. (2021). Why an android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *30*(2), 1–29.

Wu, Q., Zhu, X., & Liu, B. (2021). A survey of android malware static detection technology based on machine learning. *Mobile Information Systems*, *2021*, 1–18.

Yadav, P., Menon, N., Ravi, V., Vishvanathan, S., & Pham, T. (2022, 05). A two-stage deep learning framework for image-based android malware detection and variant classification. *Computational Intelligence*, *38*, 1748-1771. doi: 10.1111/coin.12532

Yan, P., & Yan, Z. (2018). A survey on dynamic mobile malware detection. *Software Quality Journal*, *26*(3), 891–919.

Yao, X., Li, Y., Shi, Z., Liu, K., & Du, X. (2023). Android malware detection based on sensitive features combination. *Concurrency and Computation: Practice and Experience*, *35*(6), 1–1.

Ye, G., Zhang, J., Li, H., Tang, Z., & Lv, T. (2022). Android malware detection technology based on lightweight convolutional neural networks. *Security and Communication Networks*, *2022*.

Yen, Y.-S., & Sun, H.-M. (2019). An android mutation malware detection based on deep learning using visualization of importance from codes. *Microelectronics Reliability*, *93*, 109–114.

Yerima, S. Y., & Sezer, S. (2018). Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE transactions on cybernetics*, *49*(2), 453–466.

Yilmaz, A. B., Taspinar, Y. S., & Koklu, M. (2022). Classification of malicious android applications using naive bayes and support vector machine algorithms. *International Journal of Intelligent Systems and Applications in Engineering*, *10*(2), 269–274.

Zhang, W., Luktarhan, N., Ding, C., & Lu, B. (2021). Android malware detection using tcn with bytecode image. *Symmetry*, *13*(7), 1107.

Zhang, Y., Feng, C., Huang, L., Ye, C., & Weng, L. (2020). Detection of android malicious family based on manifest information. In *2020 15th international conference on computer science & education (iccse)* (pp. 202–205).

Zhiwu, X., Ren, K., & Song, F. (2019). Android malware family classification and characterization using cfg and dfg. In *2019 international symposium on theoretical aspects of software engineering (tase)* (pp. 49–56).

Zhou, Y., & Jiang, X. (2012). Dissecting android malware: Characterization and evolution. In *2012 ieee symposium on security and privacy* (pp. 95–109).

Zhu, H., Wei, H., Wang, L., Xu, Z., & Sheng, V. S. (2023). An effective end-to-end android malware detection method. *Expert Systems with Applications*, *218*, 119593.

Zhu, H.-J., You, Z.-H., Zhu, Z.-X., Shi, W.-L., Chen, X., & Cheng, L. (2018). Droiddet: effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing*, *272*, 638–646.

Zou, K., Luo, X., Liu, P., Wang, W., & Wang, H. (2020). Bytedroid: android malware detection using deep learning on bytecode sequences. In *Trusted computing and information security: 13th chinese conference, ctcis 2019, shanghai, china, october 24–27, 2019, revised selected papers 13* (pp. 159–176).