Computers, Materials & Continua
DOI: 10.32604/cmc.2023.028316
Article

Tech Science Press

Android Malware Detection Using ResNet-50 Stacking

Lojain Nahhas[1], Marwan Albahar[1],*, Abdullah Alammari[2] and Anca Jurcut[3]

[1]Department of Computer Science, Umm Al Qura University, P.O. Box 715, Mecca, Saudi Arabia
[2]Curriculums and Teaching Department, Faculty of Education, Umm Al Qura University, P.O. Box 715, Mecca, Saudi Arabia
[3]School of Computer Science, University College Dublin, Belfield, Dublin, Ireland
*Corresponding Author: Marwan Albahar. Email: mabahar@uqu.edu.sa

Received: 06 February 2022; Accepted: 05 May 2022

Abstract: There has been an increase in attacks on mobile devices, such as smartphones and tablets, due to their growing popularity. Mobile malware is one of the most dangerous threats, causing both security breaches and financial losses. Mobile malware is likely to continue to evolve and proliferate to carry out a variety of cybercrimes on mobile devices. Mobile malware specifically targets Android operating system as it has grown in popularity. The rapid proliferation of Android malware apps poses a significant security risk to users, making static and manual analysis of malicious files difficult. Therefore, efficient identification and classification of Android malicious files is crucial. Several Convolutional Neural Network (CNN) based methods have been proposed in this regard; however, there is still room for performance improvement. In this work, we propose a transfer learning and stacking approach to efficiently detect the Android malware files by utilizing two well-known machine learning models, ResNet-50 and Support Vector Machine (SVM). The proposed model is trained on the DREBIN dataset by transforming malicious APK files into grayscale images. Our model yields higher performance measures than state-of-the-art works on the DREBIN dataset, where the reported measures are accuracy, recall, precision, and F1 measures of 97.8%, 95.8%, 95.7%, and 95.7%, respectively.

Keywords: Android malware; convolutional neural network; malware analysis; malware classification; image classification; support vector machine

1 Introduction

Malware is defined as any program that has a malicious intent (malicious software). They are designed to disrupt normal device operation, display unwanted advertising, steal sensitive data, or remotely take control of the user's device. Viruses, worms, Trojan horses, ransomware, rootkits, and botnets are all types of malwares [1]. Malware systems have evolved to become more sophisticated over the years. Malware uses polymorphic and metamorphic techniques to evade detection by conventional malware detection techniques [2–5]. Extensive static analysis is required to detect newly developed malware that is too sophisticated to evade detection by emulators. Malware can also be propagated through metamorphosis techniques such as multi-packing, registry modification, encryption, anti-debugging, code transformation, virtual machines, and instruction permutation.

The evasion form of malware is intelligent, as it changes its infection routines to include an initial step that carefully inspects the environment in which it runs, allowing it to select the most appropriate time to execute the payload [4,6–9]. By utilizing reusable development modules and automated development, new variants of malware can be created [10–12]. For the purpose of creating new malware variants, malware developers frequently alter small sections of the original source code [11,13,14]. As a result, it is a challenging task to distinguish between different variants of the same family of malware [15,16]. Android apps that are malicious in nature, on the other hand, can infiltrate smartphones and mine cryptocurrency without the user's knowledge. Mobile malware is expected to continue to grow and proliferate, allowing criminals to commit a wide range of cybercrimes on mobile devices. There was a total of 35 million malware attacks, but according to the mobile threat report [17] published by McAfee in the first quarter of 2020, there were approximately 800,000 new malware attacks that were detected in the fourth quarter of 2019, surpassing the total number of malware attacks. Recently, LeifAccess emerged as a detritus malware with capability of creating and posting fake reviews on Google Play Store by exploiting the OAuth leveraging accessibility advantage. After the installation, it operates in the background without displaying a shortcut or an icon. The emerging of new malware families and increasing number of mobile malwares posed a serious threat to the Android ecosystem's security. Several studies shaded the light on combating the security concerns faced Android ecosystem by conducing number of research related to the detection and classification of Android malware samples [18,19]. Despite the progress that made in Android malware detection and classification, a new challenge arisen related to the changes on malware malicious behavior over time. Thus, it makes it challengeable task to the machine learning-based malware classifiers to avoid performance deteriorations.

Static analysis and dynamic analysis are the techniques that are considered most important for identifying malicious software. Malware can be identified through a combination of signature and behavioral techniques. Static analysis can be done in situations where a piece of source code is under suspicion without running the program. The source code must be disassembled to extract features [20–22]. This technique is not resistant to obscure code and loading dynamic code [23–26]. On the other hand, the dynamic analysis examines the characteristics and traces of suspect use during the implementation [27–32]. Dynamic analysis is a promising technique, but it is time-and resource-intensive [23,33]. Intelligent malware uses dynamic analysis with anti-emulation technology [34–36]. In addition, it takes a lot of manual effort or human intervention to use static and dynamic techniques on such files. This requires knowledge of the domain to analyze the application or to reverse engineer it [37–41]. For Android malware family classification, the time required to create features manually throughout the Android Application Package (APK) structure is significantly high [4,6,42–45]. These safety mechanisms require high computer resources, and it is challenging to deploy them in a restricted smartphone environment [34]. Android malware traces are being investigated using Classes.dex, resources, manifests, and Android application certificate files [46,47].

Android malware detection has received a great deal of attention in the academic and commercial communities due to the prevalence of attacks against the Android mobile operating system. There is, however, a significant discrepancy between the amount of work that has been done and the number of malicious applications that are published daily.

Recent efforts have been made to reduce the number of malware attacks by testing deep learning and optimization algorithms. The proposed work converts Android malware applications to malware images. A CNN was developed to automatically extract the rich features of malware images using a proven and widely accepted method for classifying images by the research community. As a result, these characteristics were used to classify malicious applications according to their families. This methodology recommends that binary data extracted from Android files be converted into images. Analysts can see through malware binary images without executing them using visualization-based techniques. The performance of feature representation can be accomplished by deep learning algorithms without the need for any assumptions or configuration of parameters. In addition, deep learning models can learn complex patterns and solve the dimensionality problem with little guidance. The following is a list of the primary contributions:

1. We propose a stacked architecture consisting of ResNet-50 and non-linear Support Vector Machine (SVM) for identification of malware Android APK converted to graylevel images.
2. We examine which elements of the malware sample are more useful for classifying Android malware families using an expandable analysis for our stacked architecture classification network. To retrieve high-quality information, non-intuitive features are also transformed into fingerprint images.
3. The proposed model was tested and validated using the DREBIN dataset. There are 5560 applications in this dataset, belonging to 179 different malware families.

This paper is organized as follows. Section 2 reviews the methods used to detect Android malware. Section 3 describes the use of the dataset for Android malware detection and Android package conversion to images. The proposed approach is presented in Section 4. The detailed results are presented and discussed in Sections 5 and 6 respectively. Finally, conclusions are drawn, and future directions are discussed in Section 7.

2 Related Work

Researchers have performed visualization-based malware analyses [10,48–50]. The structure of malware images is directly affected by visualization techniques [11,51–53]. In contrast to static and dynamic approaches, visualization-based analyses allow for faster malware sample classification since no disassembly or execution is needed. Thus, the task was to provide a method for classifying many malware samples that was superior to that achieved with conventional technologies. In [54], APK files were converted using Hue–Saturation–Lightness (HSL), Red–Green–Blue (RGB), grayscale, and Cyan–Magenta–Yellow–Black (CMYK). To decide whether an application was benign or malicious, three machine learning classifiers (decision trees, random forest, and k-nearest neighbors) were trained on each image representation using Global Image Descriptors (GISTs). On the grey image, the random forest classification had a high accuracy of 91 percent. A visualization tool was used to fine-grained classify Portable Executable (PE) files [11]. The malware was disguised as an RGB image file. The dataset included 7,087 malware samples from 15 families. By merging global and local malware classification functions, the model was developed. The data and code portions of the file were processed to create feature vectors for local attributes. The RGB image's global characteristics were deleted. To train the model, random tree, support vector, and k-nearest neighbor classifications were used to train the model. According to the malware classification trial results, the random forest classification had a 97.47% accuracy score. The approach did not function with non-PE files due to the complexity of generating RGB-colored images and extracting valid local features for these malicious codes.

In [55], only the coding for the APK files was considered. As a result, the DEX files were converted into Java Achive (JAR) files using the dex2jar package. A Java Application Descriptor (JAD) tool was used to translate other JAR files into Java files. Each APK file was supplemented with code from different text files. The Word Frequency-Inverse Document Frequency (TF–IDF) technique was used to differentiate keywords in the text records. The TF–IDF weight is a mathematical measure that simplifies evaluating a word's value in many text files. The cumulative number of words in a text is multiplied by the number of times the word gets the normalized TF. The IDF decides the worth of a word. The number of documents in the corpus is divided by the logarithm of the number of documents in the corpus. This increases the scarcity of the words while reducing their frequency. These files were grouped after the related words were extracted from the text files. The SimHash [56] and the djb2 [57] algorithms were used to process the classes into images. A convolutional neural network was used in the analysis, and it was graded with 92 percent accuracy. META-INF, resources, and AndroidManifest. XML, among other APK file building blocks, was overlooked. Reference [58] included a total of 12,000 malware images from 32 malware families. A vector sustainable machine and the efficiency of customizers including Convolutional Neural Network (CNN) and k-nearest neighbors, Local Binary Patterns [LBPs] and GIST). 93.92% of the chosen dataset had high precision using the six-layer conversion-trained neural network model with LBP functionality. Grayscale and Red–Green–Alpha (RGBA) graphics were used to represent the malware. The LBP-trained CNN model's productivity was investigated. According to the report, malware may cause essential features of a color image to be lost. In essential malware detection, the decision on which subset of features to include is difficult in machine learning. The design of the correct feature set is required to produce an effective malware detection or analysis model. Reference [59] developed C language visualization tools to investigate the internal configuration (anomalies or patterns) of malware executable files. They mapped the.dex bytes to the image pixels to reveal a set of features for malware classification. Legitimate Android developers use a variety of obfuscation tools to protect their intellectual property. Malware authors exploit and abuse these tools and techniques to make malware versions for Android more resistant. The authors used the visualization-based methodology of [60] to fingerprint the obfuscation tools used in the Android application development period. An image is used to display the malware binary. They used an image to quantify two types of statistical properties, which were then combined to retrieve data from the application developer's obfuscation software. The accuracy rates for fingerprinting the obfuscation method and classifying the obfuscated and original implementations, respectively, were calculated to be 73 percent and 86 percent, according to the researchers. While the review of the literature establishes that an APK file is a sequence of bits and thus a binary image, there is no clear consensus among researchers on the type of analysis and prominent APK parameters appropriate for malware classification. Traditional malware classification methods rely on the extraction of dynamic and static features. These approaches typically employ code analysis to address the problem of malware classification. Existing malware classification techniques make use of both signature-based and feature-based classification. Unfortunately, these systems have several drawbacks, including excessive resource usage, code obfuscation, and code disassembly. Furthermore, researchers discovered that these methods require a lot of time and space. The era of deep learning-infused visualization approaches is beginning in Android security. The proposed methodology solves the multiclass malware classification problem by combining the power of visualization and deep learning approaches. A deep learning architecture reduces the requirement to gather features such as meta-data information, permissions, API calls, and other dynamic variables to produce a high-quality malware classification model. Recent research in security and privacy has demonstrated the value of solutions combining deep learning and visualization-based analysis [61,62]. For Windows malware, the majority of solutions [10,11,16] had high classification accuracy [10,16].

As a result of only being able to conduct their experiments on Windows systems, the researchers had to use PE files. Windows's hardware architecture differs greatly from that of mobile devices based on the lightweight Android operating system, which is why Windows is the most popular desktop operating system. Consequently, it is not suitable to classify Android malware families using tools designed for Windows platform applications. For Android malware detection, the reference [63] suggested Integrating Neural Architecture and Visualization Technologies (SARVOTAM). The machine creates fingerprint images from the malware's non-intuitive features in order to collect high-quality data. The study's precision was 92.59% when using the DREBIN dataset. As reported in [64], 17 different datasets are used to detect Android malware. It is challenging to perform benchmark tests due to the lack of standard benchmark datasets. Thus, machine learning model performance may fluctuate on some datasets. Furthermore, when obfuscation attacks are used, some static machine learning methods fail to function properly. There is no guarantee that a classifier model based on different datasets is still effective for new malicious applications. The authors of [65] created a model based on feature fusion. In which features derived from deep levels of CNN layers were combined with handmade features such as LBP, GIST and Gray Level Co-occurrence Matrix (GLCM) to create a feature fusion for the classification of Android malware images. They achieved 93.24% accuracy using the malware image combination of CR + AM using the feature fusion-SVM classifier.

In this study, we utilized the transfer learning ability of a complex architecture, ResNet-50, in combination with a non-linear SVM, to identify malicious files. ResNet-50 is a 50-layer architecture that can learn complex patterns in data more easily than other CNN-based architectures. Furthermore, the softmax layer was replaced with a non-linear SVM, which serves as a supplement to the identification task. The proposed stacked architecture is trained and tested on the DEBRIN datasets consisting of malicious Android APKs after converting them to gray-level images. Different combinations were used to test the model's performance, and results associated with different groups were reported.

3 Materials and Methods

3.1 Dataset

The DREBIN dataset was utilized to evaluate our experiments. The dataset contains a total of 5,560 files representing 179 different malware families (See Fig. 1). The DREBIN dataset has been utilized as the benchmark for malware research in most publications. GingerMaster [23], GoldDream [24], and Aslan [25] were among the malware families in the dataset. Each file and folder are contained within an APK's ZIP archive. These files are combined to create an application. Instead of focusing on the malware itself, the study's main goal was to test the proposed malware identification method.

3.2 Malware Android Package Conversion to Images

The fundamental files that are taken into consideration for visualization in an APK are the classes, dex, resources, manifest, and certificates. In this research, malware images are created by making use of the aforementioned four categories of malicious APK files. The binary data is first transformed into 8-bit vectors, and then the resulting vectors are used to create grayscale images. A malware sub-string is initially composed of a series of many sub-strings, each of which is 8 bits in length and is referred to as a pixel. In the next stage, the 8-bit substring is converted into a decimal number that can take on values between 0 and 255. In addition, each of the malware substrings was first converted into a one-dimensional vector, and then transformed into a two-dimensional matrix with a width that was specified. The grayscale image is presented here as a matrix that has two dimensions. The procedure for converting APK files to grayscale images is illustrated in Fig. 2.

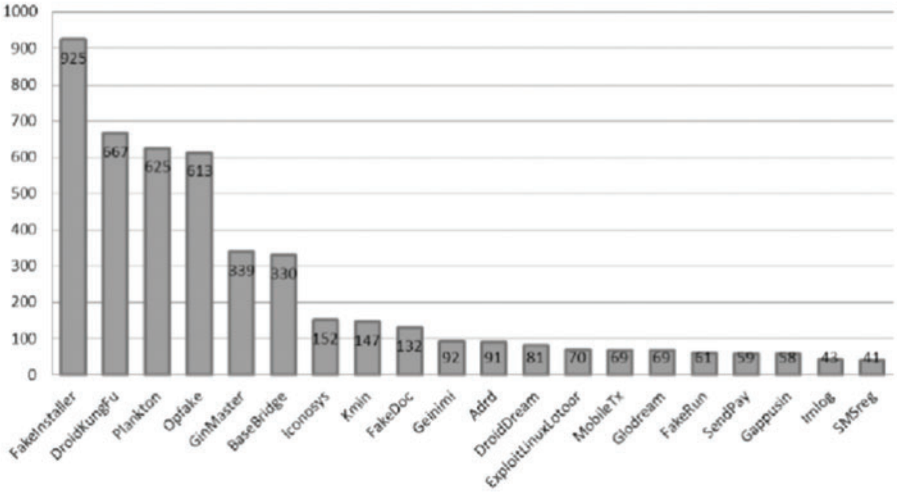

Figure 1: Top 20 malware families in DREBIN dataset

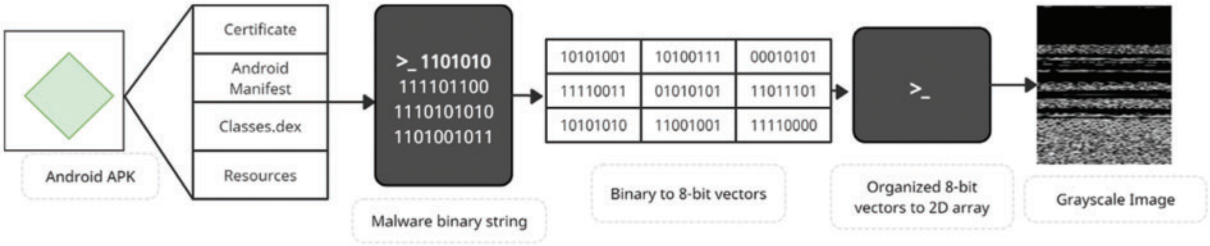

Figure 2: Conversion process of Android APK to 2D gray level image

The dimensions of the APK files that are shown in Table 1 served as the basis for determining the width of the images. As a direct consequence of this, the size of the file also influences the height. Inputs need to have the same shape for CNN-based models to work properly. So, rather than attempting to find the optimal size for an APK file through trial and error, we make use of the dimensions suggested by [63]. The primary goal in selecting the sizes was to retain as much information as required while still maintaining a compact format. This work follows the procedure suggested empirically by [16,63] to eliminate the need for the method of finding the correct sizes through trial and error. Grayscale images can be used to represent an entire application package (APK). The images of the DREBIN Android malware were created by combining fifteen different file structures, each of which contained at least one image belonging to a different family of malware. This was done to create the images of the DREBIN Android malware. Fig. 3 presents the images that were made from the files after they were processed. These files included resources (RS), Android Manifest (AM), Classes.dex (CL), and certificate (CR). In this study, different combinations of these images have been tested, as shown in Table 2.

Table 1: Fixed image width by file size

File size	Width
<50 KB	64
50–100 KB	128
100–200 KB	256
200–500 KB	512
500–1,000 KB	1,024

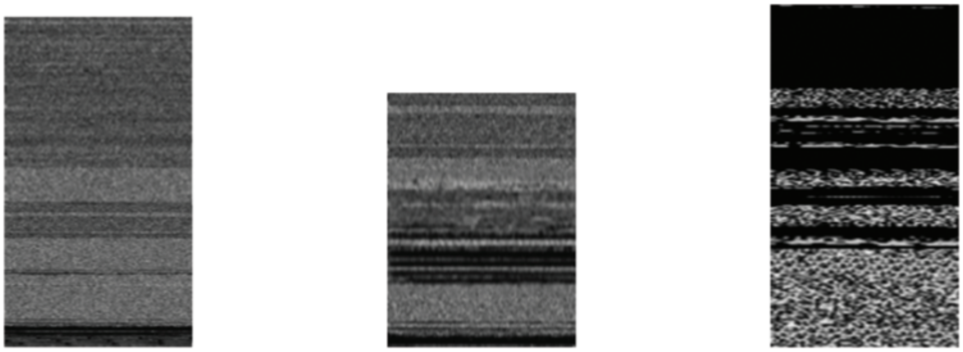

Figure 3: Illustration of some of the malware images using the files section of Android manifest (AM), classes.dex (CL), and Certificate (CR)

Table 2: Different combinations and their corresponding instances are included in our experiment

Combination	CR	AM	RS	CL	CR+ AM	CR+ RS	CR+ CL	AM+ RS	AM+ CL	RS+ CL	CR+ AM+ RS	CR+ AM+ CL	CR+ RS+ CL	AM+ RS+ CL	CR+ AM+ RS +CL
No. of instances	1826	4659	4659	4660	4659	4659	4660	4659	4660	4660	4659	4660	4660	4660	4660

4 Proposed Model

Transfer learning uses the previous pre-trained model with some or no modification for another problem. A machine uses the knowledge gained from a previous task to enhance its generalization ability for a subsequent task. This way, it enables us to build a more robust architecture in the most cost-effective way instead of training and fine-tuning from scratch. Various models can be used for transfer learning, such as AlexNet, GooLeNet, and VGG. They stacked many convolutional layers, leading to difficulty optimizing the networks, vanishing gradient problems, and degradation problems. For Android malware detection, various CNN models are used, which require end-to-end training. In this work, the pre-trained ResNet-50 model is utilized instead of end-to-end training. This reduces the execution time and improves the classification results. ResNet-50 is a widely accepted architecture that is beneficial for solving complicated tasks and improving detection performance. It tries to solve the optimization, vanishing gradient, and degradation problems found in other networks by incorporating the data fed to previous layers into the following layers [66]. In this work, we incorporated the previous preprocessing method described in [63], where the malware's non-intuitive features were converted to fingerprint images in order to extract useful information. For identification, the ResNet-50 is fine-tuned where the softmax layer is replaced with the SVM model with a Gaussian kernel. The ResNet-50 architecture consists of five stages, each consisting of a convolution block and an identity block, as illustrated in Fig. 4.

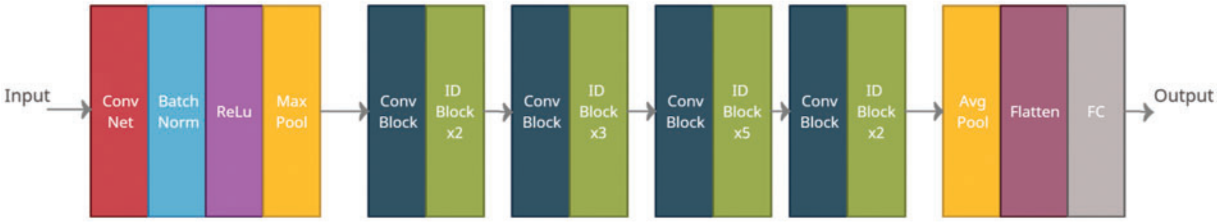

Figure 4: ResNet-50 architecture having 5 stages consists of Convolution blocks and ID blocks

Each convolution block consists of 3 connected convolution, batch normalization and ReLU units as shown in Fig. 5.

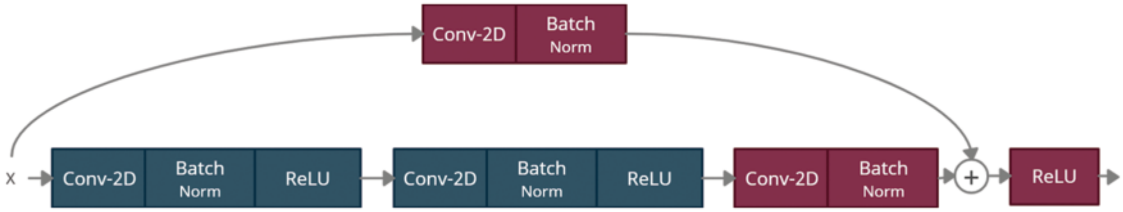

Figure 5: Convolution blocks of ResNet-50 consists of three convolution, batch norm and ReLU units

Nevertheless, the identity block is reduced to two units, as depicted in Fig. 6. The softmax layer is replaced with an SVM integrated with a Gaussian kernel. This architecture has over 23 million trainable parameters. Various numbers of layers in the back-end of the model have been tested and evaluated, and it has been concluded that employing a smaller number of layers reduces the classification measures. Nonetheless, the increment in the number of layers has no significant effect on performance and only increases the time to train and execute the model. Furthermore, the classification performance of the proposed model is better than all other state-of-the-art models. However, a major limitation is the complexity of the model during training, which increases the Multiply-Accumulate (MAC) and Floating Point Operations (FLOPs).

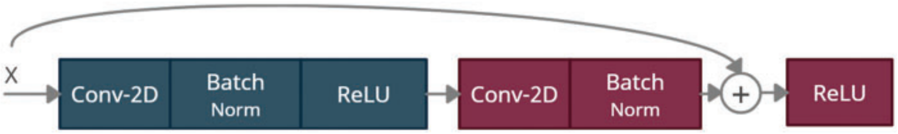

Figure 6: Identity blocks of ResNet-50 consists of two convolution, batch norm and ReLU units

As is illustrated in Table 3, the architecture of the proposed model consists of the following elements: The first layer contains a convolution with a kernel size of 7 × 7 and 64 kernels with a stride of size 2. Next, conv2 is a 1 × 1 convolution, 64 kernels following 3 × 3, 64 kernels, and a 1 × 1, 256 kernels. These three layers are repeated three times for a total of nine layers. In the conv3 convolution block, there are 1 × 1, 128 kernels. Next is a kernel of 3 × 3, 128 kernels following 1 × 1, 512 kernel convolutions, repeated four times for 12 layers. Similarly, the next convolution block, conv5, consists of 1 × 1, 256 kernels following 3 × 3, 256 kernels, leading up to 1 × 1, 1,024 kernels. This process was repeated six times for a total of 18 layers. In the last convolution block, there were 1 × 1, 512 kernels followed by 3 × 3, 512 kernels and 1 × 1, 2,048 kernels. This was repeated three times for a total of nine layers in this block. The last layer, which consists of 1,000 nodes with softmax entropy, is replaced with nonlinear kernels to identify malicious images. The simulations were run on a system with 20 GB of RAM and an Intel® CoreTM i3 processor, as well as an NVIDIA GeForce GTX 1080ti graphics processing unit with a frame buffer of 11 GB.

Table 3: Architecture of the proposed stacked model consisting front-end of ResNet-50 and SVM

Layer name	Output size	Layers statistics
conv1	112 × 112	7 × 7, stride = 2
pooling	-	3 × 3 max pool, stride = 2
conv2	56 × 56	1 × 1, 64 3 × 3, 64 × 3 1 × 1, 256
conv3	28 × 28	1 × 1, 128 3 × 3, 128 × 4 1 × 1, 512
conv4	14 × 14	1 × 1, 256 3 × 3, 256 × 6 1 × 1, 1024
conv5	7 × 7	1 × 1, 512 3 × 3, 512 × 3 1 × 1, 2048
1 × 1 average pool, the output is fed to SVM for identification		
FLOPS		$\approx 4 \times 10^9$

5 Experimental Results

To demonstrate the detection performance of the proposed model, the confusion matrix is generated, as illustrated in Table 4. Furthermore, we computed precision, recall, accuracy, and F1 measures from the information presented in Table 5.

Accuracy is computed by dividing the total number of true positives (TP) and true negatives (TN) by the total number of predictions, i.e., all entries in the confusion matrix. Recall, which is also known as the true positive rate (TPR), is calculated by dividing the total number of TP by the sum of TP and TN. It shows the number of instances correctly classified by the model. Similarly, precision (PR) is another measure that considers the number of instances that were considered malicious, but they were non-malicious, also known as false positives (FP). All these quantitative measures are shown in Table 6, which demonstrates a 97% accuracy, while the other measures are at 95%. The performance of the proposed model is higher on malware families such as certificates and Android manifests. This represents that the model learned and perceived the actual behaviors of these files.

Table 4: Confusion matrix for the twenty different classes

Family name	Fake Installer	Dro id KungFu	Plan kton	Op fake	Gin Master	Base Bridge	Icon osys	Kmin	Fake Doc	Gei nimi	Adrd	Droid Dream	Exploit Linux Lotoor	Glodr eam	Mob ileTx	Fake Run	Send Pay	Gap pusin	Im log	SMS reg
Fake Installer	300	1	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	1	0	0
Droid Kung Fu	0	208	2	1	1	3	0	0	0	0	1	0	1	0	2	0	0	1	0	0
Plankton	0	0	203	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
Opfake	0	0	0	202	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GinMaster	0	2	1	0	107	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
BaseBridge	1	1	2	0	0	104	0	0	0	1	0	0	1	0	0	0	0	0	0	0
Iconosys	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0
Kmin	0	0	0	0	0	0	0	49	0	0	0	0	0	0	0	0	0	0	0	0
FakeDoc	0	0	0	0	0	0	0	0	44	0	0	0	0	0	0	0	0	0	0	0
Geinimi	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0
Adrd	0	1	0	0	0	0	0	0	0	0	27	0	0	0	2	0	0	0	0	0
DroidDream	0	0	0	1	0	1	0	0	0	0	0	25	0	0	0	0	0	0	0	0
ExploitLinux Lotoor	0	0	1	0	1	0	0	0	0	0	0	1	20	0	0	0	0	0	0	0
Goldream	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0	0	0	0
MobileTx	0	1	0	0	1	0	0	0	0	0	1	0	0	0	20	0	0	0	0	0
FakeRun	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0
SendPay	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0
Gappusin	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	17	0	0
Imlog	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0
SMSreg	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13

Table 5: Performance measures obtained via the combination of AM and CR files

Measures	Results (%)
Accuracy	97%
Precision	95.7%
Recall	95.8%
F1	95.7%

Table 6: Individual recall, precision and F1 score of each class

Type	Recall	Precision	F1
FakeInstaller	98.36	99.34	98.85
DroidKungFu	94.55	96.74	95.63
Plankton	98.54	97.13	97.83
Opfake	100	98.54	99.26
GinMaster	95.54	95.54	95.54
BaseBridge	95.41	95.41	95.41
Iconosys	100	96.15	98.04
Kmin	100	100	100
FakeDoc	100	100	100
Geinimi	100	100	100
Adrd	90	93.1	91.53
DroidDream	92.59	96.15	94.34
ExploitLinuxLotoor	86.96	83.33	85.11
Goldream	100	100	100
MobileTx	86.96	83.33	85.11
FakeRun	95	90.48	92.68
SendPay	100	100	100
Gappusin	89.47	89.47	89.47
Imlog	92.86	100	96.3
SMSreg	100	100	100

However, the performance is lower on ExploitLinuxLootor, MobileTx, Gappusin, and BadeBridge due to fewer samples relative to other malware types. Furthermore, the performance is the lowest of the DroidDream, MobileTx, Gappusin, and ExploitLinuxLooter malware families. These malware families exhibit a significantly lower number of samples in the training dataset, and it seems to affect rooted Android devices. There is a high certainty that these types alter their signatures after getting root access to the device. This statement can be analyzed in the future, which opens a new dimension of research to evaluate existing algorithms on rooted and non-rooted devices. It can also be seen from the results that DroidDream, Imlog, and DroidKungFu have a lower recall rate. However, their precision is high. Such contrasting results illustrate that the nature of these files learned by the model is different from other families. In Fig. 7, we show the number of instances of each class misclassified by the proposed model. During training, the classification models try to extract discriminative features. If the input data contains a lot of information and relevant features, the classification task of the model becomes more accurate. As shown in this study, various combinations of the images were used in classification to increase the accuracy. We have seen that the combination of CR and AM has produced the maximum classification results. As a result, time and effort can be saved in inspecting the entire APK structure for Android malware classification. Misclassification occurs as a result of similarity between images of different classes. In order to resolve this issue in future work, we need to propose a common framework based on frequency transformation, which could be distinguished in similar images.

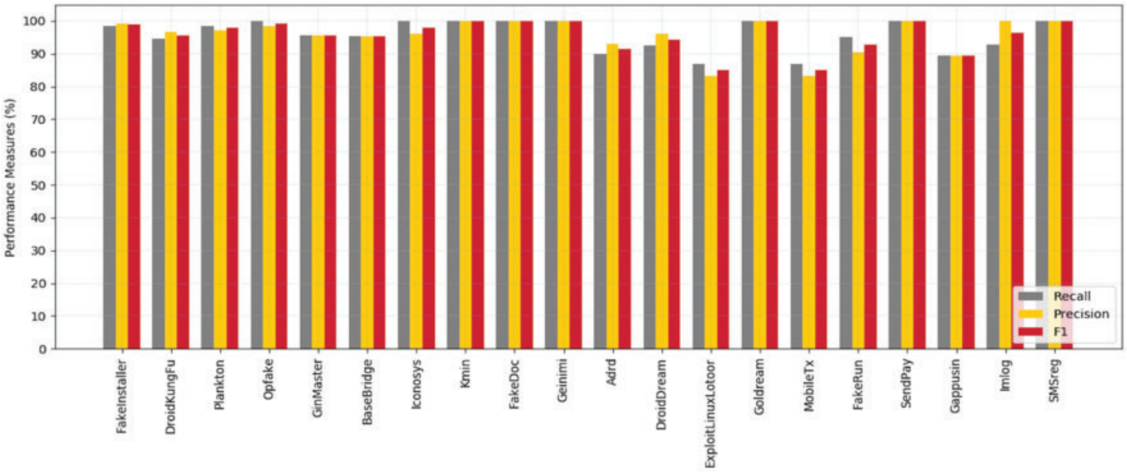

Figure 7: Recall, precision and F1 score of top 20 malware families in DREBIN dataset for the proposed model

6 Discussion

In our line of work, the proposed model performed admirably for approximately 100 epochs. The simulation results were recorded for the DREBIN dataset after converting malicious Android apps into fingerprint images and utilizing AM and CR images. These results were recorded after using AM and CR images. Our model's performance was evaluated in comparison to that of several other models considered to be state-of-the-art, and the results showed that our model's performance was superior. As can be seen in Fig. 8, the evaluation is compared for a number of different possible combinations of the image types. Table 7 shows the highest level of accuracy achieved using the combination of AM and CR.

Both the observations and the results of the simulations indicate that both files contain the maximum amount of relevant information about different types of malicious software, which results in satisfactory classification performance. In addition to measures of classification, Table 8 provides a time-based comparison for each combination that was utilized in the research, as well as the number of images that were processed per second and belonged to the appropriate class. Consequently, once the model has been trained with high-quality classification metrics, it can be utilized for testing in a wide variety of different applications. Table 8 shows the average processing time required to process a single image, which is comparable to that of earlier studies [63]. Therefore, once the model is integrated and utilized in software systems, its execution performance will be identical to that of the state-of-the-art method.

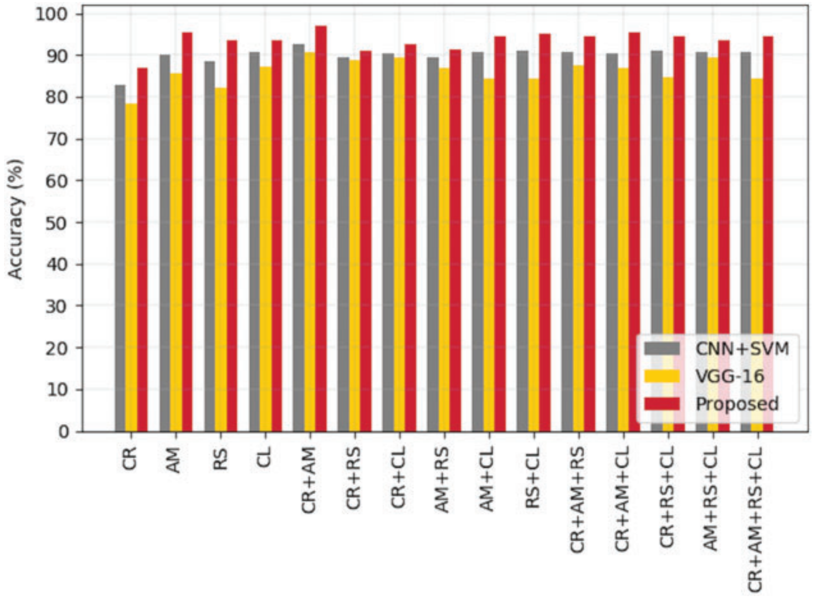

Figure 8: Accuracy score of each combination of the images considered in simulations

Table 7: Accuracy of the proposed model compared with other state-of-the-art works for various combinations of images [63]. The highest accuracy for each model is shown in bold

S/No.	Image combination (%)	CNN	CNN-RF	CNN-SVM	VGG-16	Proposed (%)
1	CR	83.5	83.4	82.9	78.2	86.8
2	AM	89.7	84.8	90.1	85.7	95.4
3	RS	86.8	84.5	88.5	82.1	93.6
4	CL	89.4	87.5	90.5	87.2	93.4
5	**CR + AM**	**91.4**	87.5	**92.5**	**90.5**	**97.0**
6	CR + RS	87.1	85.8	89.4	88.9	91.0
7	CR + CL	89.3	88.4	90.2	89.3	92.7
8	AM + RS	88.2	84.9	89.4	86.7	91.3
9	AM + CL	89.3	88.6	90.8	84.4	94.5
10	RS + CL	88.4	87.5	90.9	84.3	95.2
11	CR + AM + RS	89.4	85.5	90.7	87.6	94.6
12	CR + AM + CL	89.3	**88.8**	90.5	86.8	95.5
13	CR + RS + CL	89.5	88.1	90.9	84.5	94.6
14	AM + RS + CL	88.5	87.9	90.7	89.2	93.6
15	CR + AM + RS + CL	89.3	87.8	90.7	84.3	94.5

Table 8: A comparison of execution time and images processed per second by the proposed model

S/No.	Combination	Execution time (s)	Images processed/second
1	CR	241.2	7.57
2	AM	763.8	6.1
3	RS	887.4	5.25
4	CL	1103.1	4.22
5	CR + AM	890.2	5.23
6	CR + RS	1004.4	4.64
7	CR + CL	1109.7	4.2
8	AM + RS	870.5	5.35
9	AM + CL	1130.4	4.12
10	RS + CL	1093.3	4.26
11	CR + AM + RS	924.7	5.04
12	CR + AM + CL	1139.4	4.09
13	CR + RS + CL	1233.5	3.78
14	AM + RS + CL	1207.9	3.86
15	CR + AM + RS + CL	1513.7	3.08

7 Conclusion

In this study, we proposed a classification model for Android malware that uses ResNet-50 and SVM. The ResNet-50 was used because it has transferable learning abilities. The first step was to use substrings from many binary malware files in the DREBIN dataset to generate vectors in the 8-bit range. The next step is to convert these vectors into grayscale images. The ResNet-50 model's softmax layer for classification is replaced by a support vector machine (SVM), which uses a non-linear kernel to improve detection performance. In addition, various combinations of the images were used to fine-tune the model in search of the files that had the greatest impact on the model. From simulation results, it can be concluded that the certificate and Android manifest (CR + AM) are the most suitable features for identifying and classifying malware, as they contain sufficient information. Using the CR and AM images, we reported the highest accuracy, recall, precision, and F1 measures. When using the DREBIN dataset, the highest level of accuracy achieved was 97%. In the future, we intend to extend the evaluation to include malware on additional platforms to evaluate the effectiveness of our model.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. Qamar, A. Karim and V. Chang, "Mobile malware attacks: Review, taxonomy and future directions," *Future Generation Computer Systems*, vol. 97, pp. 887–909, 2019.

[2] S. Dong, M. Li, W. Diao, X. Liu, J. Liu *et al.,* "Understanding android obfuscation techniques: A large-scale investigation in the wild," in *Proc. of the Int. Conf. on Security and Privacy in Communication Systems*, Singapore, pp. 172–192, 2018.

[3] D. Maiorca, D. Ariu, I. Corona, M. Aresu and G. Giacinto, "Stealth attacks: An extended insight into the obfuscation effects on android malware," *Computers & Security*, vol. 51, pp. 16–31, 2015.

[4] G. Suarez-Tangil, "DroidSieve: Fast and accurate classification of obfuscated android malware," in *Proc. of the Seventh ACM on Conf. on Data and Application Security and Privacy*, Scottsdale, AZ, USA, pp. 309–320, 2017.

[5] K. Bakour, H. M. ünver and R. A. Ghanem, "Deep camouflage: Evaluating android's anti-malware systems robustness against hybridization of obfuscation techniques with injection attacks," *Arab Journal for Science and Engineering*, vol. 44, pp. 9333–9347, 2019.

[6] J. Garcia, M. Hammad and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of android malware," *ACM Transactions on Software Engineering and Methodology*, vol. 26, pp. 1–29, 2018.

[7] V. Rastogi, Y. Chen and X. Jiang, "Catch me if you can: Evaluating android anti-malware against transformation attacks," *IEEE Transactions on Information Forensics and Security*, vol. 9, pp. 99–108, 2014.

[8] O. Mirzaei, J. de Fuentes, J. Tapiador and L. Gonzalez-Manzano, "AndrODet: An adaptive android obfuscation detector," *Future Generation Computer Systems*, vol. 90, pp. 240–261, 2019.

[9] V. Balachandran, S. Sufatrio, D. J. J. Tan and V. L. L. Thing, "Control flow obfuscation for android applications," *Computers & Security*, vol. 61, pp. 72–93, 2016.

[10] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717–46738, 2019.

[11] J. Fu, J. Xue, Y. Wang, Z. Liu and C. Shan, "Malware visualization for fine-grained classification," *IEEE Access*, vol. 6, pp. 14510–14523, 2018.

[12] F. Wei, Y. Li, S. Roy, X. Ou and W. Zhou, "Deep ground truth analysis of current android malware," in *Proc. of the Int. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, Bonn, Germany, Switzerland, pp. 252–276, 2017.

[13] N. Xie, X. Wang, W. Wang and J. Liu, "Fingerprinting android malware families," *Frontiers of Computer Science*, vol. 13, pp. 637–646, 2019.

[14] S. Ni, Q. Qian and R. Zhang, "Malware identification using visualization images and deep learning," *Computers & Security*, vol. 77, pp. 871–885, 2018.

[15] S. Türker and A. B. Can, "AndMFC: Android malware family classification framework," in *Proc. of the 2019 IEEE 30th Int. Symp. on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, Istanbul, Turkey, pp. 1–6, 2019.

[16] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei *et al.,* "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, pp. 107138, 2020.

[17] McAfee, "McAfee mobile threat report Q1," 2020. [Online]. Available: https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf, Accessed on: Jan. 2, 2022.

[18] M. A. Albahar, M. S. ElSayed and A. Jurcut, "A Modified ResNeXt for Android Malware Identification and Classification," *Computational Intelligence and Neuroscience*, Hindawi Limited, vol. 2022, pp. 1–20, 2022.

[19] S. Y. Yerima, M. K. Alzaylaee, A. Shajan and P. V., "Deep learning techniques for android botnet detection," *Electronics*, vol. 10, no. 519, 2021.

[20] L. Li, "Iccta: Detecting inter-component privacy leaks in android apps," in *Proc. of the 2015 IEEE/ACM 37th IEEE Int. Conf. on Software Engineering*, Florence, Italy, 16–24, pp. 280–291, 2015.

[21] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil and S. Furnell, "AndroDialysis: Analysis of android intent effectiveness in malware detection," *Computers & Security*, vol. 65, pp. 121–134, 2017.

[22] A. Martín, H. D. Menéndez and D. Camacho, "MOCDroid: Multi-objective evolutionary classifier for android malware detection," *Soft Computing*, vol. 21, no. 24, Springer Science and Business Media LLC, pp. 7405–7415, 2016.

[23] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian *et al.,* "Constructing features for detecting android malicious applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[24] A. Naway and Y. Li, "A review on the use of deep learning in android malware detection," *arXiv2018*, arXiv:1812.10360, 2018.

[25] O. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[26] S. Venkatraman, M. Alazab and R. Vinayakumar, "A hybrid deep learning image-based analysis for effective malware detection," *Journal of Information Security and Applications*, vol. 47, pp. 377–389, 2019.

[27] H. Cai, N. Meng, B. Ryder and D. Yao, "DroidCat: Effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 1455–1470, 2019.

[28] A. Martín, V. Rodríguez-Fernández and D. Camacho, "CANDYMAN: Classifying android malware families by modelling dynamic traces with markov chains," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 121–133, 2018.

[29] W. You, B. Liang, W. Shi, P. Wang and X. Zhang, "TaintMan: An ART-compatible dynamic taint analysis framework on unmodified and non-rooted android devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, pp. 209–222, 2017.

[30] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino *et al.,* "Risk analysis of android applications: A user-centric solution," *Future Generation Computer Systems*, vol. 80, pp. 505–518, 2018.

[31] P. Teufl, M. Ferk, A. Fitzek, D. Hein, S. Kraxberger *et al.,* "Malware detection by applying knowledge discovery processes to application metadata on the android market (Google play)," *Security and Communication Networks*, vol. 9, pp. 389–419, 2016.

[32] M. K. Alzaylaee, S. Y. Yerima and S. Sezer, "DynaLog: An automated dynamic analysis framework for characterizing android applications," in *Proc. of the 2016 Int. Conf. on Cyber Security and Protection of Digital Services (Cyber Security)*, London, UK, pp. 1–8, 2016.

[33] A. Sadeghi, H. Bagheri, J. Garcia and S. Malek, "A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software," *IEEE Transactions on Software Engineering*, vol. 43, pp. 492–530, 2017.

[34] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur *et al.,* "Android security: A survey of issues, malware penetration, and defenses," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 998–1022, 2015.

[35] M. K. Alzaylaee, S. Y. Yerima and S. Sezer, "Emulator *vs.* real phone: Android malware detection using machine learning," in *Proc. of the 3rd ACM on Int. Workshop on Security and Privacy Analytics*, Scottsdale, AZ, USA, pp. 65–72, 2017.

[36] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection," in *Proc. of the 9th ACM Symp. on Information, Computer and Communications Security*, Kyoto, Japan, pp. 447–458, 2014.

[37] H. Gascon, F. Yamaguchi, D. Arp and K. Rieck, "Structural detection of android malware using embedded callgraphs," in *Proc. of the 2013 ACM Workshop on Artificial Intelligence and Security*, Berlin, Germany, pp. 45–54, 2013.

[38] D. Su, J. Liu, X. Wang and W. Wang, "Detecting android locker-ransomware on Chinese social networks," *IEEE Access*, vol. 7, pp. 20381–20393, 2018.

[39] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen and Y. Rahulamathavan, "PIndroid: A novel android malware detection system using ensemble learning methods," *Computers & Security*, vol. 68, pp. 36–46, 2017.

[40] B. Jung, T. Kim and E. G. Im, "Malware classification using byte sequence information," in *Proc. of the 2018 Conf. on Research in Adaptive and Convergent Systems*, Honolulu, HI, USA, pp. 143–148, 2018.

[41] S. Wu, P. Wang, X. Li and Y. Zhang, "Effective detection of android malware based on the usage of data flow APIs and machine learning," *Information and Software Technology*, vol. 75, pp. 17–25, 2016.

[42] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez and J. Blasco, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," *Expert Systems with Applications*, vol. 41, pp. 1104–1117, 2014.

[43] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi *et al.,* "DroidScribe: Classifying android malware based on runtime behavior," in *Proc. of the 2016 IEEE Security and Privacy Workshops (SPW)*, San Jose, CA, USA, pp. 252–261, 2016.

[44] C. Yang, Z. Xu, G. Gu, V. Yegneswaran and P. Porras, "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in android applications," in *Computer Security-ESORICS 2014*, vol. 8712, pp. 163–182, Switzerland: Springer International Publishing, 2014.

[45] M. Hanif, R. Naqvi, S. Abbas, M. A. Khan and N. Iqbal, "A novel and efficient 3D multiple images encryption scheme based on chaotic systems and swapping operations," *IEEE Access*, vol. 8, pp. 123536–123555, 2020.

[46] R. A. Naqvi, M. Arsalan, A. Rehman, A. U. Rehman, W. -K. Loh *et al.,* "Deep learning-based drivers emotion classification system in time series data for remote applications," *Remote Sensing*, vol. 12, pp. 587, 2020.

[47] Arp, M. Spreitzenbarth, M. Hubner, H. Gascon and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proc. of the 2014 Network and Distributed System Security (NDSS) Symp.*, San Diego, CA, USA, pp. 23–26, 2014.

[48] L. Nataraj, D. Kirat, B. S. Manjunath and G. Vigna, "Sarvam: Search and retrieval of malware," in *Proc. of the Annual Computer Security Conf. (ACSAC) Workshop on Next Generation Malware Attacks and Defense (NGMAD)*, New Orleans, LA, USA, 10 December 2013.

[49] L. Nataraj, V. Yegneswaran, P. Porras and J. A. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proc. of the 4th ACM Workshop on Security and Artificial Intelligence*, Chicago, IL, USA, pp. 21–30, 2011.

[50] M. Farrokhmanesh and A. Hamzeh, "A novel method for malware detection using audio signal processing techniques," in *Proc. of the 2016 Artificial Intelligence and Robotics (IRANOPEN)*, Qazvin, Iran, pp. 85–91, 2016.

[51] J. Zhang, Z. Qin, H. Yin, L. Ou, S. Xiao *et al.,* "Malware variant detection using opcode image recognition with small training sets," in *Proc. of the 2016 25th Int. Conf. on Computer Communication and Networks (ICCCN)*, Waikoloa, HI, USA, pp. 1–9, 2016.

[52] K. Han, B. Kang and E. G. Im, "Malware analysis using visualized images and entropy graphs," *International Journal of Information Security*, vol. 14, pp. 1–15, 2015.

[53] K. Han, B. Kang and E. G. Im, "Malware analysis using visualized image matrices," *The Scientific World Journal*, vol. 2014, pp. 1–15, 2014.

[54] A. Kumar, K. P. Sagar, K. S. Kuppusamy and G. Aghila, "Machine learning based malware classification for android applications using multimodal image representations," in *Proc. of the 2016 10th Int. Conf. on Intelligent Systems and Control (ISCO)*, Coimbatore, Tamil Nadu, India, pp. 1–6, 2016.

[55] Y. S. Yen and H. M. Sun, "An android mutation malware detection based on deep learning using visualization of importance from codes," *Microelectronics Reliability*, vol. 93, pp. 109–114, 2019.

[56] Y. Li, F. Liu, Z. Du and D. Zhang, "A simhash-based integrative features extraction algorithm for malware detection," *Algorithms*, vol. 11, pp. 124, 2018.

[57] Y. Li, J. Jang, X. Hu and X. Ou, "Android malware clustering through malicious payload mining," in *Proc. of the Int. Symp. on Research in Attacks, Intrusions, and Defenses*, Atlanta, GA, USA, pp. 192–214, 2017.

[58] J. S. Luo and D. C. T. Lo, "Binary malware image classification using machine learning with local binary pattern," in *Proc. of the 2017 IEEE Int. Conf. on Big Data (Big Data)*, Boston, MA, USA, pp. 4664–4667, 2017.

[59] A. Jain, H. Gonzalez and N. Stakhanova, "Enriching reverse engineering through visual exploration of android binaries," in *Proc. of the 5th Program Protection and Reverse Engineering Workshop*, Los Angeles, CA, USA, pp. 1–9, 2015.

[60] Y. Ning, "Fingerprinting android obfuscation tools using visualization," Ph.D. dissertation, Dept. Comput. Sci., New Brunswick Univ., Fredericton, NB, Canada, 2017.

[61] C. Ieracitano, A. Adeel, F. C. Morabito and A. Hussain, "A novel statistical analysis and autoencoder driven intelligent intrusion detection approach," *Neurocomputing*, vol. 387, pp. 51–62, 2020.

[62] S. M. Kasongo and Y. Sun, "A deep learning method with wrapper-based feature extraction for wireless intrusion detection system," *Computers & Security*, vol. 92, pp. 10172, 2020.

[63] J. Singh, D. Thakur, F. Ali, T. Gera and K. S. Kwak, "Deep feature extraction and classification of android malware images," *Sensors*, vol. 20, no. 24, pp. 7013, 2020.

[64] Q. Wu, X. Zhu and B. Liu, "A survey of android malware static detection technology based on machine learning," *Mobile Information Systems*, vol. 2021, pp. 1–18, Mar. 2021.

[65] J. Singh, D. Thakur, T. Gera, B. Shah, T. Abuhmed *et al.,* "Classification and analysis of android malware images using feature fusion technique," *IEEE Access*, vol. 9, pp. 90102–90117, 2021.

[66] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Appendix

List of abbreviations used throughout the paper.

Abbreviation	Meaning
SVM	Support Vector Machine
APK	Android Application Package
HSL	Hue–Saturation–Lightness
RGB	Red–Green–Blue
CMYK	Cyan–Magenta–Yellow–Black
GIST	Global Image Descriptor
PE	Portable Executable
JAR	Java Archive
JAD	Java Application Descriptor
TF–IDF	Word Frequency-Inverse Document Frequency
CNN	Convolutional Neural Network
LBP	Local Binary Pattern
RGBA	Grayscale and Red–Green–Alpha
SARVOTAM	Suggested Integrating Neural Architecture and Visualization Technologies
GLCM	Gray Level Co-occurrence Matrix
CL	Classes.dex
AM	Android Manifest
CR	Certificate
RS	Resources
MAC	Multiply-Accumulate
FLOP	Floating Point Operation