Electronic Research Archive

http://www.aimspress.com/journal/era

ERA, 32(7): 4255–4290.
DOI: 10.3934/era.2024192
Received: 28 February 2024
Revised: 13 May 2024
Accepted: 24 June 2024
Published: 05 July 2024

Research article

Employing combined spatial and frequency domain image features for machine learning-based malware detection

Abul Bashar*

Department of Computer Engineering, Prince Mohammad Bin Fahd University, Khobar 31952, Saudi Arabia

* **Correspondence:** Email: abashar@pmu.edu.sa; Tel: +966138499382.

Abstract: The ubiquitous adoption of Android devices has unfortunately brought a surge in malware threats, compromising user data, privacy concerns, and financial and device integrity, to name a few. To combat this, numerous efforts have explored automated botnet detection mechanisms, with anomaly-based approaches leveraging machine learning (ML) gaining attraction due to their signature-agnostic nature. However, the problem lies in devising accurate ML models which capture the ever evolving landscape of malwares by effectively leveraging all the possible features from Android application packages (APKs).This paper delved into this domain by proposing, implementing, and evaluating an image-based Android malware detection (AMD) framework that harnessed the power of feature hybridization. The core idea of this framework was the conversion of text-based data extracted from Android APKs into grayscale images. The novelty aspect of this work lied in the unique image feature extraction strategies and their subsequent hybridization to achieve accurate malware classification using ML models. More specifically, four distinct feature extraction methodologies, namely, Texture and histogram of oriented gradients (HOG) from spatial domain, and discrete wavelet transform (DWT) and Gabor from the frequency domain were employed to hybridize the features for improved malware identification. To this end, three image-based datasets, namely, Dex, Manifest, and Composite, derived from the information security centre of excellence (ISCX) Android Malware dataset, were leveraged to evaluate the optimal data source for botnet classification. Popular ML classifiers, including naive Bayes (NB), multilayer perceptron (MLP), support vector machine (SVM), and random forest (RF), were employed for the classification task. The experimental results demonstrated the efficacy of the proposed framework, achieving a peak classification accuracy of 93.03% and recall of 97.1% for the RF classifier using the Manifest dataset and a combination of Texture and HOG features. These findings validate the proof-of-concept and provide valuable insights for researchers exploring ML/deep learning (DL) approaches in the domain of AMD.

Keywords: image-based data; spatial and frequency domain; malware identification; machine learning classifiers; feature extraction; feature hybridization

1. Introduction

In recent times, we have witnessed a rapid development and usage of mobile communication technology in our daily lives. The popularity of this technology has been due to the encompassing development of 5G wireless communication standards which provide high speed and reliable communication infrastructure. At the same time, there has been tremendous advancement in the communication, storage, and processing of data in the end-user devices in the form of smartphones, tablets, and various smart internet of things (IoT)-based systems. With a current estimate of about 6 billion existing smartphone users worldwide and a further increase of about 6% annually, it is but expected that the smartphone market is bound to dominate the technology arena in the near future. Statistics also suggest that nearly 70% of the smartphones have their operating system based on Android. Even though our study caters to the Android-based devices and their protection from malware, this approach can be easily adopted and applied to iphone operating system (iOS)-based systems as well.

The popularity of the Android operating system is due to its usage convenience, being open-source and high scalability. However, this widespread use of an open-source operating system also comes under the eyes of malicious hackers and cyber-criminals with evil intent of compromising user data and privacy. It is reported by Kapersky in their 2021 mobile malware report that there was a detected total of about 3.5 million malicious installation packages and about 17 thousand mobile ransomware Trojans [1]. Since most of the Android applications are downloaded and installed from the official Google Playstore, the cyber-criminals target it for hiding their malware in the disguise of clean, safe, and approved apps. It was found that out of a total of about 2.6 million apps on Google Playstore, 36% of them are classified as low quality, which are possible targets for hiding Android malware [2]. It is also reported by McAfee that there has been an increase of 38% in the Linux-based (Android) mobileware additions from the quater-four (Q4) of year 2020 to quater-one (Q1) of 2021 alone [3]. These malware mainly belong to the categories of resource consumption, privacy breach, remote control, inappropriate behavior, and malicious ransom. This results in financial costs to detect and recover damages to the infrastructure and also results in compromising the reputation of the business in the market and a disadvantage in comparison to the competitors.

Thus, it is imperative that efficient and timely detection of these malware are given due diligence and importance to avoid the risks and associated costs which may incur if these malware are successful in initiating a security breach. As a matter of fact, instead of reacting to amend the losses due to a security attack, there must be mechanisms to proactively predict an attack scenario and avert it in the best possible manner. Due to the prevalence of variety of AMD approaches in the literature, we provide a new taxonomy of AMD approaches in Figure 1 for easier understanding of this research domain.

One of the major categorization of the AMD approaches is based on the information contained in the malware, namely, signature-based and anomaly-based [4]. Signature-based approaches are fast, simple, widely available, and have shown good results in malware identification. However, the drawback is that they rely on the knowledge of signatures of known malware and hence fail to detect unknown or newly created malware. Also, creating malware signatures requires domain knowledge expertise and tedious human effort. On the other hand, anomaly-based approaches work on the concept of observing the behavior of all the applications over a period of time and classifying them as malign or benign. This historic data (of malware features) is then used as an input to an automated ML algorithm to learn a model based on the example observations made prior to training. However, this approach, too, requires generating malware features, which can be either done manually (handcrafted features [5]) or in an automated manner (using DL [6]).

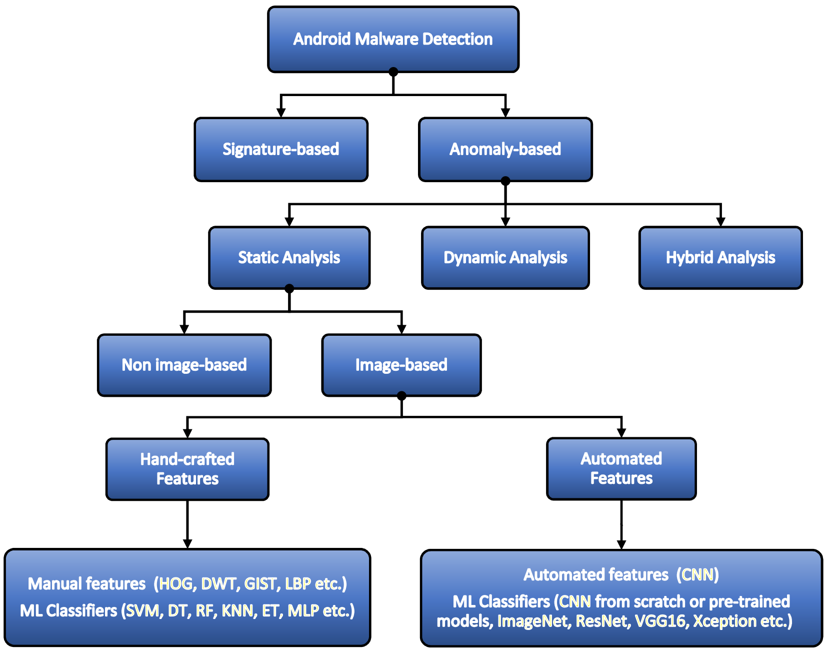

Android Malware Detection
Signature-based
Anomaly-based
Static Analysis
Dynamic Analysis
Hybrid Analysis
Non image-based
Image-based
Hand-crafted Features
Automated Features
Manual features (HOG, DWT, GIST, LBP etc.) ML Classifiers (SVM, DT, RF, KNN, ET, MLP etc.)
Automated features (CNN) ML Classifiers (CNN from scratch or pre-trained models, ImageNet, ResNet, VGG16, Xception etc.)

Figure 1. Taxonomy of the AMD approaches.

Another way of classifying the AMD process is the manner in which the applications are analyzed for generating the features, namely, static, dynamic, and hybrid. In the static analysis, the focus is on extracting features from the APKs without actually running the application. This involves using the de-compilation technology to perform reverse engineering on the Dalvik executable (DEX) file (classes.dex), which has the Dalvik byte code or/and the application's global configuration file (AndroidManifest.xml) [6]. The possible set of features extracted from the DEX executable can be based on Dalvik opcode, string, application programming interface (API) call, control flow graph, and data flow graph, whereas from the manifest file the features are usually the package names, components, permissions, environment, and intents. The benefit of static analysis is that it is faster and the processing overhead is lower however, it is unable to get relevant features (e.g., permissions or API access rights) if the hackers have used the obfuscation approach to malign the Android application.

In such a situation, dynamic analysis can be resorted to, where the Android application is actually run on a real device or in a sandboxed environment. The application's runtime behavior is observed over a period of time to identify any anomalous behavior and also record the features related to the log of system API execution states and collected network traffic. Even though dynamic analysis has the ability to detect malicious apps which hide using the obfuscation technique, it requires longer analysis time and data processing overheads. As the Android operating system (OS) keeps evolving in terms of its architecture and security requirements, the reverse-engineering features have a high possibility of becoming obsolete over a period of time and, hence, the feature extraction process needs to be repeated to keep up with the changes. To this end, image-based Android malware approaches have made some promising contributions in dealing with this challenge [7].

In an image-based AMD approach, the Dalvik byte code and the manifest files contents (which are binaries) are converted into images, where each 8 bits (byte code) is transformed into an red green blue (RGB) pixel. This gives a 2-D color image of the APKs which can be either malign or benign in nature. So, basically we are transforming the problem of identifying a malware from a traditional ML-based classification problem to an image classification problem. One of the main reasons for adopting this approach is to utilize the significantly accurate classification performance of image-based DL models based on the convolutional neural networks (CNN) [8] or to use handcrafted features with traditional ML classifiers. Another reason in terms of the general utility and long-term applicability of the said approach is that it can efficiently handle any small changes which hackers or malware authors incorporate in the APKs to hide their malicious code. Hence, we adopt this general direction of scope of the work and use it to build upon the proposed framework for identifying and classifying Android malware.

Research contributions: After the critical study of the existing AMD approaches, it was observed that the image-based approaches outnumber the nonimage based approaches and, hence, we conclude their popularity and applicability. Second, Drebin and information security centre of excellence (ISCX) datasets were most frequently used datasets for Android malware identification. Third, both automated and handcrafted features have been equally used in the research. Finally, both traditional ML and automated DL approaches have been used for AMD, each having their own merits and limitations. This led us to pose these research questions in the context of defining the scope of our work.

- What are the popular AMD approaches and their classification taxonomy?
- How can existing AMD approaches be improved by incorporating better feature information?
- What are the reliable image-based datasets for testing ML-based classification of Android botnets?
- What different categories of image features can be useful in making informed decisions by the AMD systems?
- Which ML classifiers are usually employed for detection and classification of Android malwares and how do they compare with each other?

Based on these research questions, the technical and empirical contributions are now clearly laid down for this research paper.

- To propose and elaborate on a new taxonomy of AMD approaches.
- To propose a novel hybridization of spatial and frequency domain features for accurate identification of Android malware using image-based datasets.
- To implement and test the proposed approach on the image-based datasets (Dex, Manifest and Composite) obtained from the ISCX Android Botnet dataset.
- To experiment and evaluate the proposed approach by hybridizing the spatial features (Texture and histogram of oriented gradients (HOG)) and frequency features (discrete wavelet transform (DWT) and Gabor).
- To compare the ML classifier (NB, SVM, MLP, and RF) results obtained from the proposed work with the existing approaches (traditional ML and automated DL) in the literature.

Paper organization: The rest of the paper is organized as follows. In Section 2, the background information about Android malware, their detection approaches, and the related work in this domain is presented to provide an overview of the problem domain. This is followed by Section 3, wherein architectural details of the proposed image-based framework using feature hybridization approach are elaborated for AMD. Section 4 provides the details of the dataset used, the experimental scenarios used, and the resources utilized for performing the experiments to prove the validity of the proposed methodology. This section also presents the results obtained, the related insights, and their comparison to other similar work in the extant literature. Finally, the paper is concluded in Section 5 by summarizing the key contributions of the research, identifying limitations, and suggesting ways for improvements in the form of future work.

2. Related work

This section presents a detailed review of the research literature in the domain of AMD. Even though this domain includes multitude of approaches, we choose to give attention to anomaly-based AMD approaches to limit our scope of work, as mentioned in Section 1. It is proposed to categorize existing related research work into two broad categories, which are based on the ML and DL models applied to the task of identifying Android malware. The first category (**nonimage**-based approach) is the one where the features from the APKs are extracted directly from the DEX and/or manifest files through reverse-engineering, and then an ML classifier is used for classifying the application as malign or benign. In the second category (termed as **image**-based approach), the DEX and/or manifest binary files are first transformed to images, followed by feature extraction from these images and finally using either an ML or DL classifier to classify the APKs as malign or benign. In order to comprehend the research presented in these two categories, we initially present a summary of three most recent review papers in the domain of AMD to grasp the basic concepts and terminologies.

2.1. Review papers

In the first review paper which we considered [9], the authors present a comprehensive summary of close to 300 research papers in the domain of AMD based on ML approaches. They provide details of the research status by touching upon key perspectives like data acquisition, data preprocessing, feature selection, ML models, and performance evaluation. Further, they classify the existing studies based on the type of detection, namely, static, dynamic, and hybrid. Finally, an intuitive classification of the research papers is done on the basis of four different categories of ML, namely, supervised, unsupervised, semi-supervised, and reinforcement learning. Even though the review has much detail and rigor, it is already two years old and, hence, another review paper is necessary to report the latest research approaches.

In the second review paper [6], the authors focus their literature search pertaining to the DL applications to AMD. They classify about 100 DL papers into five different categories, namely, static dynamic, hybrid analysis, image-based, and multi-class. Apart from summarizing the existing literature, they also provide generic architecture and functional flowchart of AMD using DL. Also, they describe the challenges related to datasets, feature selection, neural network training issues, and multi-class problem with possible recommendations.

The third review paper [4], is presenting a different perspective on AMD through a focus on approaches based on static analysis (manifest or code based), dynamic analysis, and hybrid analysis. The authors classify the papers with accuracy results of more than 90% and less than 90%. Also, apart from detecting malware, they also focus on research work related to ML applications to detect code vulnerabilities in APKs. Their other contribution is to study the datasets used, feature selection methods, and a concise comparison of various ML algorithms in regard to their advantages and disadvantages. One drawback of these two surveys is that they do not cater to image-based detection of Android malware with the use of DL methods.

2.2. Nonimage-based approaches

The first paper in this category [10] is where the authors performed static analysis of AMD on the data obtained from canadian institute of cybersecurity and malware (CICAndMal2017) dataset. They compare the performance of various ML classifiers such as NB, SVM, k-(nearest neighbor) NN, sequential minimal optimization (SMO), RF, MLP, and decision trees (DT). The handcrafted permission features were extracted from APK manifest files, which gave an accuracy of 88.9% for static analysis. Even though the approach was highly effective in terms of classification accuracy, the limitation was that only one dataset was used in the research. In another work, the authors demonstrated the effectiveness of combined usage of RF and principal component analysis (PCA) for identifying malwares using feature extraction approaches. They were able to achieve an accuracy of 90.33% by using handcrafted features from the CICAndMal2017 and Androzoo datasets focusing on only both static and dynamic analysis [11]. It was expected that they would compare the classifier performances with automated approaches.

Extensive study with 4 different ML classifiers was performed on datasets obtained from Google Play, AndroZoo and AppChina. The classifiers used in the study were RF, SVM, NB, and K-means [12]. The RF classifier gave an accuracy performance of 81.5% with feature-based classification involving manifest permissions. The remarkable feature of this research was that the authors experimented with multiple datasets, however, the results could be improved by considering other static analysis features like OpCode and API calls. In another research by [13], static analysis was performed on the AMD. Various ML models from the model driven engineering (MEG)Droid and Droidbit framework were used for the comparative performance study on identifying malwares based on the handcrafted static features. The MEGDroid-based algorithms provided an accuracy of 91.6%, proving it to be a versatile solution. However, one scope of improvement to the study is to incorporate features based on system calls. Malware classification framework (MalFCS) combines visualization, automated feature extraction, and classification using CNNs to extract family patterns from entropy graphs. It achieves an accuracy of 0.997 and 1 on Malimg and Microsoft datasets, respectively [14].

In a research which uses automated feature extraction as opposed to handcrafted features [15], CNN was utilized for automated feature extraction from the APK files. The code analysis for API calls, OpCode, and Manifest analysis for permissions were incorporated in the proposed solution. The CNN approach was applied on the Drebin and AMD datasets by providing a classification accuracy of 91 and 81%, respectively. The advantage of this approach is that it reduced over-fitting and, hence, could effectively detect new malwares. An improvement to this research can be to provide comparison of CNN with other ML/DL approaches. The most recent study by the researchers in [16] used a basic one-dimensional CNN to automatically learn manifest, intent, and API features from the APK repository files of a dataset consisting of Android applications from various de-facto standard datasets. In spite of using hybrid features for training their CNN model, they reported a low classification accuracy of 90%. Another noteworthy research achieves malware detection accuracy of 99% by employing the idea of federated Markov chains, which are also effective in preserving the privacy of end-user data in IoT environments [17].

2.3. Image-based approaches

In the image-based approaches using handcrafted features, the authors in [18], have used the HOG features from the image-transformed ISCX dataset to detect malign applications. They have used SVM, K-nearest neighbor (KNN), DT, RF, Extra Trees (ET), and extreme gradient boosting (XGBoost) as the ML classifiers to detect malware based on static analysis, and a classification accuracy of 92.4% was shown by the RF classifier. Since the image datasets are better suited for DL approaches, a comparison of the results in this direction was expected. In a similar research [5], the authors have augmented their feature set by combining the DEX files features with the manifest files features from the APKs and generating a single composite image. A validation accuracy of 87.1% was obtained with composite images. Even though novelty lies in working with composite images (which do not significantly enhance performance), an improvement of this work could be to test and compare the performance of existing ML classifiers with pretrained and fine-tuned CNN classifiers.

In the image-based category of research, there has been emphasis on automated feature extraction, as opposed to handcrafted approaches mentioned above. In this context, the authors in [19] have used the CNN model for just arriving at the static features in an automated manner using the images obtained from the Drebin dataset. However, for performing classification tasks they resorted to traditional ML classifiers like KNN, SVM, and RF. Out of these ML classifiers, the CNN-SVM combination achieved the highest classification accuracy of 92.59% with the grayscale images. However, it would have been insightful to further test their approach on the colored images. A similar study has also been performed by [20], on a self-compiled nonstandard dataset having images obtained from opcode features of the Android APK. The strength of their work lies in addressing the obfuscation issue of the Android malware, but only an accuracy of 89.96% has been achieved. Another work in this category is by [21], who has used the image-based ISCX Android Botnet dataset to extract features in an automated manner by applying popular pretrained CNN models with transfer learning. The models considered were MobileNetV2, RestNet101, visual geometry group (VGG)16, VGG19, InceptionRestNetV2, and DenseNet121. They achieved a maximum classification accuracy of 91% for Manifest dataset using the MobileNetV2 classifier. The strong aspect of this research is the variety of pretrained classifiers which were worked with, however, the classification accuracy could be further improved with data augmentation approaches. The most recent work in this category is from [22], who has used image-based data of the APK to retrain the deep CNN model such as ResNet to update them to ResNeXt+, by achieving a classification accuracy of 90.64%. The positive aspect of this approach is the testing on seven benchmarked datasets, however, comparison with handcrafted approaches is recommended.

2.4. *Basis for proposed methodology*

The critical and comparative literature survey on AMD presented above has been concisely summarized in Table 1. A few conclusions can be readily made from this literature review. First, not all ML/DL-based approaches have reported high classification accuracies (maximum of 92.59%). Even though there were other approaches which have higher accuracies, they are not included due to reasons like not being published in a reputable research database, nonstandard datasets, and highly unrealistic results. Second, most of the studies focus on only one type of approach, either automated features extraction with DNNs/CNNs or handcrafted features using traditional ML classifiers. Third, some of the researches have not used standard datasets in their experiments. Finally, a true and fair comparison among these existing and future studies is a challenging task and, hence, there is always a scope for more focused research efforts in the domain of AMD. Therefore, we propose and implement a unique feature hybridization approach which harnesses the combined merits of spatial-domain features (Texture and HOG) and frequency domain features (DWT and Gabor) for ML-based AMD, the detailed description of which is presented in the following section.

Table 1. A comparative summary of recent related work in ML/DL-based AMD.

Paper	Year	Approach	Feature extraction	Dataset	Accuracy	Pros	Cons
[10]	2023	k-NN, SMO, SVM, RF, DT, NB, MLP	Hand-crafted (non-image based)	CICAndMal2017 dataset	Static analysis accuracy of 88.9%	High effectiveness in terms of integrated features	Only one dataset was used
[11]	2023	RF and PCA	Hand-crafted (static analysis, non-image based)	CICAndMal2017 and Androzoo	Accuracy of 90.33%	Cascaded Deep Forest with PCA is the novelty	Hybrid analysis approach could be compared to automated approaches
[12]	2021	RF, SVM, NB, K-means	Hand-crafted (non-image based)	Google Play, AndroZoo, AppChina	RF accuracy 81.5%	Multiple datasets were experimented with	Other static features like OpCode, API calls could be considered
[13]	2021	ML algorithms of MEGDroid and Droidbot	Hand-crafted (non-image based)	AMD (Android Malware Dataset)	MEGDroid accuracy 91.6%	Very versatile analysis approach	System calls were not accounted for
[15]	2021	CNN	Automated (non-image based Manifest and Dex)	AMD (Android Malware Dataset) and Drebin	Accuracy of 91%	Reduced overfitting and detects new malwares effectively	Comparison with other ML/DL approaches
[16]	2022	One-dimensional CNN	Automated (non-image based Manifest and Intents)	Collection of Android apps from de-facto standard datasets	Accuracy of 90%	Combines intents and API features with Manifest features	Lower accuracy results
[22]	2024	ResNeXt+	Automated (image-based)	Seven benchmarked datasets	90.64% accuracy with ResNet models	Experimented with 7 benchmarked datasets	Comparison with hand-crafted approaches is recommended
[18]	2021	SVM, KNN, DT, RF, ET, XGBoost with HOG features	Hand-crafted (static analysis, image-based)	ISCX Android Botnet	Accuracy of 92.4% with RF	Use of ISCX dataset with various ML classifiers	A comparison with DL approach was expected
[5]	2022	SVM, KNN, DT, RF, ET, XGBoost with HOG features	Hand-crafted (image-based from dex and manifest)	ISCX Android Botnet	Validation accuracy of 87.1% with RF	Combining dex and manifest features in a single image	For comparison other features from images should be considered
[19]	2020	CNN, CNN-KNN, CNN-SVM, CNN-RF	Automated (image-based)	Drebin	Accuracy of CNN-SVM : 92.59%	Extracts the features using CNN and then classify with ML	Need to investigate the colored malware images in the study
[20]	2022	Pre-trained ResNet CNN	Automated (image-based Opcode features)	Self-generated dataset from various sources	Accuracy of 89.96%	Focus on identifying Obfuscated Android malware	Lower accuracy results
[21]	2022	Transfer Learning on pre-trained CNN models	Automated (image-based features)	ISCX Android Botnet	Accuracy of 91%	Used six popular pre-trained CNN models with Manifest images	Lower accuracy results

3. Proposed methodology

This section provides the details of the proposed image-based AMD approach, which is based on the existing related work and pertinent research objectives as identified before in Section 2. Figure 2 provides the major components of the proposed approach and further details of each of these components are described next.

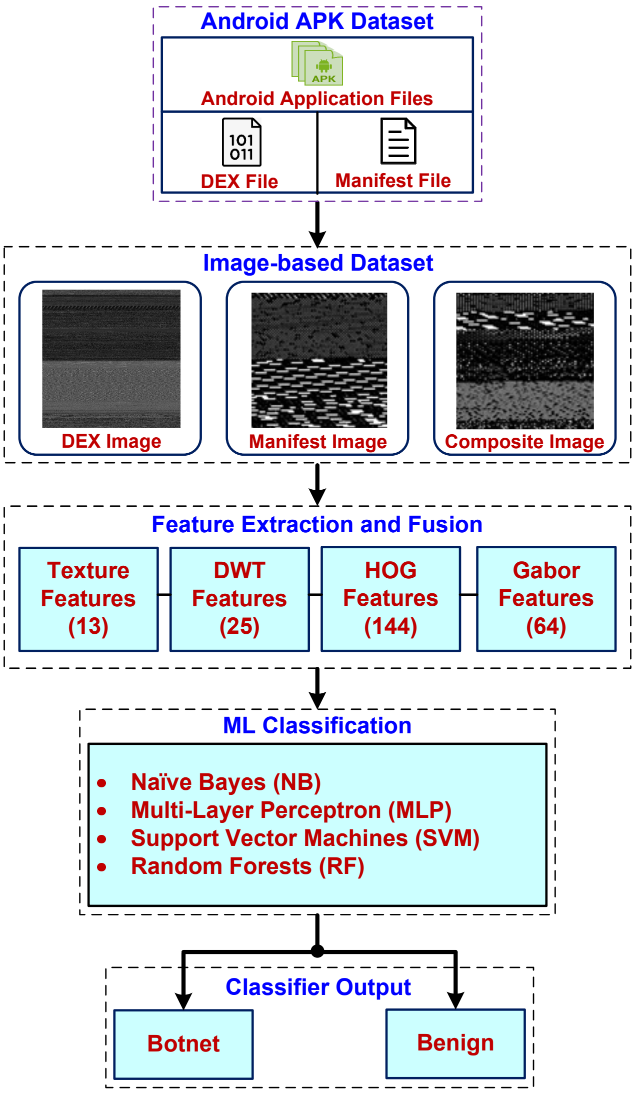


Figure 2. Proposed image-based Android malware detection approach.

3.1. Android system architecture

Android application is written using Java programming language and is compiled using the Android Software Development Kit (SDK). All the data and resource files are packed into a single compressed file as a package with an .apk extension. This .apk file is the one which is downloaded from the application repository (e.g. Google App store) and installed by the mobile phone users with the Android operating system. If the files inside this package are modified by the hacker, then the application is identified as a botnet. The following files which are part of the package are likely to be compromised leading to a botnet application [6].

- **classes.dex**: These files are the traditional java programs which are compiled into a class file as a byte code and can be executed by the Java virtual machine (JVM). For the Android system, an optimized version of the JVM is developed which is termed as the Dalvik virtual machine that executes the Dalvik byte-code instead of the Java byte-code. A dx tool is usually employed to convert the byte code to the Dalvik code during the packaging of the Android application.
- **AndroidManifest.xml**: This is a file (also called as configuration file) that describes the complete information about the Android application. These include registration information about the activity, service, provider, and receiver when developing an Android application. Also, this file contains information about the permissions declarations of the Android application along with the SDK version used for its development. This vital information is used to assess the trustworthiness of the application.
- **Other folders**: There are other folders in the .apk package which are assets (stores static files), lib (native library files), res (files related to layout and style), and meta information (META-INF) (signature information for apps integrity). In our proposed system, we are utilizing the classes.dex and AndroidManifest.xml contents to generate the dataset for training the ML classifiers.

3.2. Dataset description

The dataset used for the study in this paper is called as the "ISCX Android Botnet" dataset collected and prepared by the Canadian Institute for Cybersecurity at the University of New Brunswick [23]. The dataset consists of 1929 samples of botnets which are categorized in fourteen families and have appeared in the period of 2010 to 2014. Further details of these botnet names, the year first appeared (sorted with most recent), number of samples, and the Command and Control (C & C) type utilized for malware distribution are presented in Table 2. In order to distinguish between a botnet and benign application, we also collected 2500 clean apps from the Google Playstore, and these apps were tested with the VirusTotal tool and found to be non-malicious. In this dataset collection, the percentage of botnet samples is 44% and that of benign samples is 56%, and, therefore, this dataset is a balanced one. This observation is essential for validating the results of this study.

Figure 3 shows the process to convert the APK dataset into the image-based dataset. The first step is to unpack the APK repository into individual files (namely, AndroidManifest.xml, META.INF, assests, lib, res, and classes.dex), out of which only two files are of interest in our proposed approach (AndroidManifest.xml and classes.dex). A Python script is used to generate column vectors consisting of one byte of binary data in each array element (read from the two files (AndroidManifest.xml and classes.dex) individually and termed as Manifest byte vector and Dex byte vector, respectively). Then, equivalent decimal values of these bytes are calculated to get a pixel value in the range of 0 (corresponding to the binary pattern of 00000000) to 255 (corresponding to the binary pattern of 11111111). In the next step, these decimal values (between 0–255) are arranged in a square matrix form (named as the Pixel matrix) with equal number of rows and columns. The pixel values in the range of 0 to 255 have the grayscale colors starting from black (corresponding to 00000000) and ending in white (corresponding to 11111111). Finally, two-dimensional grayscale images (Manifest and DEX images) are obtained using the open computer vision (OpenCV) tool and stored in the .png format. Also, a composite image is obtained by concatenating the pixel values of the Manifest and DEX byte vectors (as obtained in the previous step, and this results in a larger column vector, where the resulting number of rows is actually the sum of the rows of the Manifest and DEX column vectors). The reason for choosing image representation of the data is twofolds, namely, to propose an approach which is independent of the Android OS updates/upgrades and to utilize the rich modeling and evaluation resources available for classifying images using ML-based computer vision tools.

Table 2. ISCX Android Botnet dataset details.

Botnet Family	Year	Samples	C&C Type
Notcompatible	2014	76	HTTP
Pletor	2014	85	SMS/HTTP
Sandroid	2014	44	SMS
Wroba	2014	100	SMS/HTTP
Misosms	2013	100	Email
Bmaster	2012	6	HTTP
Rootsmart	2012	28	HTTP
Tigerbot	2012	96	SMS
Anserverbot	2011	244	HTTP
Droiddream	2011	363	HTTP
Nickyspy	2011	199	SMS
Pjapps	2011	244	HTTP
Geinimi	2010	264	HTTP
Zitmo	2010	80	SMS

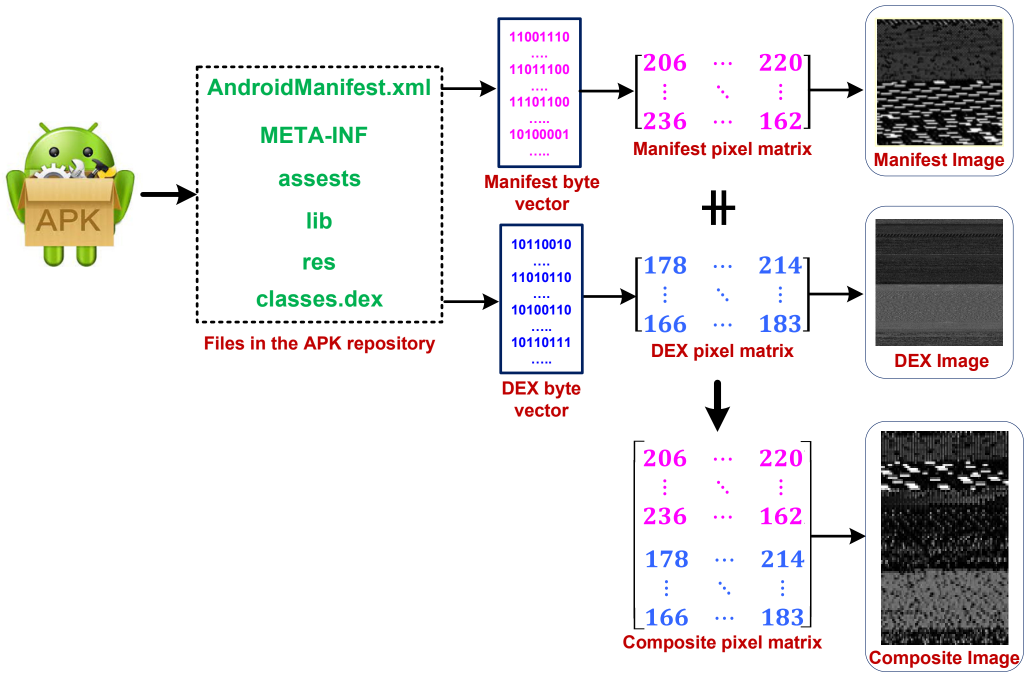


Figure 3. Process to convert the APK dataset to image-based dataset.

3.3. Feature extraction approaches

Feature extraction is a major component of the proposed framework where the image-based dataset is provided as an input and a set of image-related features are given at the output. These features are utilized to uniquely identify one image from another and, further, aid in classifying them in either of the two categories (botnet or benign). Feature extraction also helps in reducing the dimensionality of the dataset by limiting the number of unique features for image identification. However, extracting features from the images is a challenging task and requires the system designer to be aware of various options available. Apart from the types of features, it is required to know how to combine the varied features and also devise strategies for selecting the most relevant features [24]. Some of the popular spatial features extraction techniques include statistical features, HOG, local binary pattern (LBP), speed-up robust features (SURF), and those that extract features in the frequency domain are DWT, Gabor filters, and discrete Fourier transform (DFT).

In the presence of multitude of image feature extraction approaches, a detailed study was conducted to identify the choice of features. To this end, we focused on comparing the advantages and drawbacks of various methods. Even though there are seven broad classes of features extraction approaches, namely, statistical, structural, transform-based, model-based, graph-based, learning-based, and entropy-based, we narrowed down to two: statistical (or spatial domain) and transform-based (or frequency domain) based on the advantages they offer over other approaches [25]. From the spatial domain, we choose Texture and HOG features. Texture features have the advantages of easy implementation and less processing time, and also providing significant local spatial dependencies suitable for AMD application. The HOG features are also less computationally demanding and cater to rotation invariance and, hence, suitable for applications where data augmentation is required. From the frequency domain, we choose DWT and Gabor features. Even though DWT involves transformation of image to frequency domain, still it is less computationally complex compared to other approaches. Further, it can provide features at different scales by controlling the wavelet functions. For the Gabor approach, it can be very useful for extracting both low frequency and high frequency image components, and this makes them robust to resolution variations among images of a particular dataset. Even though no single choice of feature extraction methods can be a best choice, our main objective was to provide the feature hybridization concept and perform a proof of concept with the above chosen methods.

One of the objectives of this research is to investigate the effect of the number and type of image features on the performance of ML classifiers. More specifically, we would like to measure the ML classification accuracy by combining features from spatial and frequency domains. Hence, we propose to have a hybrid features approach in our system, which combines four different features, namely, Texture, HOG, DWT, and Gabor. It is noteworthy that in our framework there is a balanced mix of feature extraction techniques (Texture & HOG from the spatial domain & DWT and Gabor from the frequency domain) [26]. This leads us to a total of 246 features which we will be using in our approach with 13, 25, 144, and 64 features coming from Texture, HOG, DWT, and Gabor techniques, respectively. We have utilized the built-in toolboxes of Matlab R2022a, namely, image processing, computer vision, image filtering, and wavelet for extracting the abovementioned features. Features were extracted independently using different methods (Texture, HOG, DWT, and Gabor) and concatenated to result in 246 features for the purpose of classification. The best combination of these features is then meticulously selected after extensive experiments to achieve the objective of high classification accuracy of botnet and clean android APK samples.

3.3.1. Texture features

Texture features (TF) are those features which are obtained from the image by considering the relevant spatial domain information. In order to extract the spatial domain features, first a histogram is constructed based on the gray-level information present in each pixel of the image. Then, the histogram is used as a basis for the extraction of statistical information of first order from the images. The occurrence of the pixel intensity level is mathematically modeled as the probability density (PD) given by,

$$PD(i) = \frac{I(i)}{N}, \quad i = 1, 2, ..., L \tag{3.1}$$

where I is the intensity level present in the histogram, N is total pixels in the image, which is the product of number of horizontal and vertical pixels, $I(i)$ is the intensity of a specific grayscale level i, and L is the total number of gray levels in the image.

The PD forms the basis for six first order TFs which can be calculated for an image. These features include, TF_1 (mean) , TF_2 (variance), TF_3 (skewness), TF_4 (kurtosis), TF_5 (energy) and TF_6 (entropy) [27]. Here, *Mean* refers to the average intensity, *Variance* signifies the intensity variation about the mean value, *Skewness* measures the degree the asymmetry about the mean, *Kurtosis* provides an estimate on the number of outliers, *Energy* measures the histogram's uniformity, and *Entropy* measures the randomness in the histogram distribution. The formulae for these six features are mentioned below [28],

$$TF_1 = \sum_{i=1}^{L} iPD(i) \tag{3.2}$$

$$TF_2 = \sum_{i=1}^{L} (i - \mu)^2 PD(i) \tag{3.3}$$

$$TF_3 = \sigma^{-3} \sum_{i=1}^{L} (i - \mu)^3 PD(i) \tag{3.4}$$

$$TF_4 = \sigma^{-4} \sum_{i=1}^{L} (i - \mu)^4 PD(i) \tag{3.5}$$

$$TF_5 = \sum_{i=1}^{L} PD(i)^2 \tag{3.6}$$

$$TF_6 = -\sum_{i=1}^{L} PD(i) log_2 PD(i) \tag{3.7}$$

where $PD(i)$ is the probability density as given in Eq (3.1), $i = 1, 2, ..., L$, μ is defined as the mean given by TF_1, and σ is defined as the standard deviation estimated by $\sqrt{TF_2}$.

The first order features defined above are useful to describe the image characteristics, however they are a representative of localized information about the image pixels. In order to represent the image with more details with respect to relative positioning, spatial pixels characteristics need to be looked at. Therefore, now the second order statistical features are obtained from the spatial co-occurrence matrix $M(i, j)$, where we utilize the joint probability distribution of pixel pairs [29]. The joint probability distribution uses the distance measure d (usually 1 and 2) and phase measure θ (usually 0, 45, 90, and 135^o, which takes into account the relationship among four neighboring pixels). This results in seven second order statistical features termed as TF_7 (angular second moment energy), TF_8 (correlation), TF_9 (inertia), TF_{10} (absolute value), TF_{11} (inverse Difference), TF_{12} (entropy), and TF_{13} (maximum probability), which are mathematically defined as follows [28],

$$TF_7 = \sum_{i=1}^{L}\sum_{j=1}^{L}[PD(i,j)]^2 \tag{3.8}$$

$$TF_8 = \sum_{i=1}^{L}\sum_{j=1}^{L}\frac{i\, j\, PD(i,j) - \mu_x\mu_y}{\sigma_x\sigma_y} \tag{3.9}$$

$$TF_9 = \sum_{i=1}^{L}\sum_{j=1}^{L}(i-j)^2\, PD(i,j) \tag{3.10}$$

$$TF_{10} = \sum_{i=1}^{L}\sum_{j=1}^{L} \mid (i-j) \mid \; PD(i,j) \tag{3.11}$$

$$TF_{11} = \sum_{i=1}^{L}\sum_{j=1}^{L}\frac{PD(i,j)}{1+(i-j)^2} \tag{3.12}$$

$$TF_{12} = -\sum_{i=1}^{L}\sum_{j=1}^{L} PD(i,j)\, log_2\, PD(i,j) \tag{3.13}$$

$$TF_{13} = max(PD(i,j)) \tag{3.14}$$

Thereby, we are considering 13 features (first order features are 6 and second order features are 7) in our study.

3.3.2. HOG features

HOG is an image feature extraction very similar to Canny edge detector or scale-invariant feature transform (SIFT). It is popularly used in machine vision and image processing applications for objection identification and classification [5]. Given an image, the HOG algorithm calculates the presence of gradient orientations in a portion of an image called as a localized area. HOG is considered to be one of the good image descriptors as it calculates both the magnitude and phase of the gradient of each cell, and then it computes the features of the image. This is achieved by creating a histogram of the gradients using both the magnitude and phase information.

The HOG approach usually expects an image to be of size 128 × 64 pixels (height versus width). The gradient is calculated by combining the magnitude and phase information from the image for each pixel. Assuming a block of 3 × 3 pixels, first it calculates G_x and G_y for each pixel, based on the following equations.

$$G_x(r,c) = I(r,c+1) - I(r,c-1) \tag{3.15}$$

$$G_y(r,c) = I(r-1,c) - I(r+1,c) \tag{3.16}$$

where row and column are denoted as r and c for the pixels in an image. Then, G_x and G_y are used to calculate the magnitude and phase using the following equations,

$$G = \sqrt{G_x^2 + G_y^2} \tag{3.17}$$

$$\theta = tan^{-1}\left(\frac{G_x}{G_y}\right) \tag{3.18}$$

After calculating the gradient of each pixel, the gradient matrices (magnitude and phase matrix) are divided into 20×20 cells to form a block. In each block it creates a 9-point histogram by choosing a bin size of 9 resulting in an angle separation of $20^o = (180^o/9)$. In our proposed approach, we use a cell size of 20×20 and a block size of 4×4, which gives a HOG vector of 144 features ($9 \times 4 \times 4$).

3.3.3. DWT features

DWT is popularly applied for image feature extraction which takes into consideration both the temporal and frequency information present in images [30]. The process is to decompose any signal into basic functions called wavelets which have specific properties pertaining to the images that help in image processing. The coefficients of the wavelets are treated as feature vectors which are used to train the ML model. DWT is useful in effective removal of a noise from a noise corrupted signal and its implementation is computationally efficient. DWT in essence transforms a single variable function into a function of two variables, called translation and scale. It is a general practice to compute the wavelet coefficients at discrete levels having powers of two. DWT is obtained from continuous wavelet transform (CWT), which is given below where a signal $x(t)$ is scaled and shifted through a wavelet function ψ,

$$W(s,\tau) = \int_{-\infty}^{\infty} x(t)\frac{1}{|s|^{1/2}}\psi^*(\frac{t-\tau}{s})dt \tag{3.19}$$

where s and τ are scale and translation coefficients.

DWT is obtained from CWT where the s parameter is discretized on a log scale and the τ parameter is discretized with respect to scale parameter. This is done by making $s = 2^{-m}$ and $\tau = n \times 2^{-m}$, where m and n are some positive integers. Since m and n are nonnegative integers, they take the values as $m = 1, 2, 3, ...$ and $n = 1, 2, 3, ...$, which makes the scaling factors as a multiple of 2 and the translation coefficients as an integral of the scaling factor. By taking higher values of m and n results in finer representation of image features, however it leads to increased computation requirements. Thus, the family of wavelets is represented as [31]

$$\psi_{m,n}(t) = 2^{m/2}\ \psi(2^m t - n) \tag{3.20}$$

DWT decomposes a discrete signal $x[n]$ into low and high frequency components as given below,

$$x[n] = \sum_{k \in Z} c_{i,k}\, g_i[n - 2^i k] + \sum_{k \in Z} d_{i,k}\, h_i[n - 2^i k] \tag{3.21}$$

where $c_{i,k}$ are wavelet coefficients and $d_{i,k}$ are scaling coefficients for a one-dimensional signal.

$$c_{i,k} = \sum_n x[n]\, g_i^*[n - 2^i k] \tag{3.22}$$

$$d_{i,k} = \sum_n x[n]\, h_i^*[n - 2^i k] \tag{3.23}$$

where $g_i[n - 2^i k]$ are discrete wavelets and $h_i[n - 2^i k]$ are scaling sequences, and their conjugates provide the coefficient values.

In case of images, which represent two-dimensional data, we define 2-D DWT coefficients, where each dimension has its own set of coefficients. In a DWT feature extraction approach, N is defined as the number of levels up to which an image can be decomposed to extract features from each of the decomposed regions. In our proposed approach, the features from the images are extracted using $N = 3$ level DWT, resulting in a total of 25 features.

3.3.4. Gabor features

Gabor filter forms the basis for finding features in images by analyzing whether there exists any specific frequency component in a specific direction in a given localized region of observation. So, it essentially utilizes the frequency and orientation representation of an image to form a set of features for image analysis, which is very similar to the way the human visual system functions [32]. The impulse response of the Gabor filter is a result of the multiplication of two functions, namely, sinusoidal and Gaussian. Due to this product, we get a complex function which consists of two orthogonal components that are real and imaginary. The real component is given by

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) cos(2\pi \frac{x'}{\lambda} + \psi) \tag{3.24}$$

whereas the imaginary component is given by

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) sin(2\pi \frac{x'}{\lambda} + \psi) \tag{3.25}$$

where $x' = x\, cos(\theta) + y\, sin(\theta)$ and $y' = -x\, sin(\theta) + y\, cos(\theta)$.

In these equations, the wavelength of the sinusoidal function is given by λ, the orientation of the normal function is denoted as θ, the phase offset is ψ, the standard deviation of the Gaussian function is labeled as σ, and the spatial aspect ratio is defined as γ (a measure of ellipticity of Gabor function). In an image, which is a 2-D data, we require a Gabor filter in two-dimensions having different wavelengths and orientations in order to extract the image features. In our proposed approach, we are using $\lambda = 4$ pixels/cycle of the sinusoidal function and $\theta = 90^o$ that results in a total of 64 features (both magnitude and phase).

3.4. ML classifiers

A brief description of the ML classifiers being used in our framework is now presented. It identifies Android apps as either botnet or benign samples. In general terms, these algorithms take the features (labeled data) extracted from the image-based data of the APK files as input (as described in the previous sections) to learn a representative model. This trained model is then used for predicting the target variable of interest (botnet or benign in this case, which is the type of the Android application sample). These ML algorithms belong to the group of classifiers called supervised classifiers. In this paper, four prominent supervised classifiers have been applied for the classification task, which are NB, SVM, MLP, and RF. Even though there are numerous selection criteria for choosing an ML classifier, the criteria used in this study is based on simplicity of implementation (for NB), generality of the algorithm (for MLP), robustness toward data variance (for SVM) and high classification accuracy (for RF). It is to be noted that these four ML classifiers were practically experimented by using the Weka 3 software, which has Java-based user-friendly implementations.

3.4.1. Naive Bayes

NB classifier is a supervised ML approach which builds a model on the input data by utilizing the famous Bayes theorem [33]. NB is a popular choice for a classification problem as its mathematical modeling is simple to implement. It also exhibits reasonably good performance in terms of prediction accuracy. Generally speaking, NB devices a model based on conditional probabilities where a new data instance class to be predicted is represented as a set of features. Essentially, the task of the NB is to predict an output (target variable), which is dependent on the prior probability, the probability of observing the data instances given the hypothesis (conditional probability), and the probability of observing the current data instance itself. One major flaw of the NB is the assumption of statistical independence among the features of the input dataset (predictors and target), which more often is not the case. However, this assumption significantly simplifies the mathematical modeling of the NB, but at the same time has a negative effect on its prediction accuracy. Despite this drawback, NB is still a preferred modeling choice in scenarios where one is faced with scarcity of training data and it still performs reasonably well in terms of accuracy of classification.

3.4.2. Multilayer perceptron

MLP is a variation of the basic NN modeling concept which has a feed forward structure and consists of different layers of neurons [34]. It consists of a set of neurons (resembling the human brain architecture) arranged in three types of layers, namely, the input, the hidden, and the output. The input layer has the task of getting the data as an input and it basically implements a linear function. On the other hand, the hidden and output layers are designed in such a way that they implement a nonlinear function to model the data which is essentially nonlinearly separable. Therefore, MLP is a good choice for classifying data which cannot be modeled using linear functions, such as linear regression (LR) algorithm. Sigmoid function is a popular nonlinear activation function generally used by MLP models. Although more recently, researchers have also applied the ReLU (rectified linear unit) function in deep neural networks (DNNs). The key characteristic of MLP is the fully connected nature of the relationship between the adjacent hidden layers through weighted links. Initially, the MLP layers assign random weights to the links between the neurons during the training process. As the input layer keeps getting the input data instances (as a training set), these weights get iteratively updated. The "adjustment" of the MLP weights is done based on the amount of error between the predicted output and the actual output as implemented in the back propagation algorithm. MLPs have the ability to model data of various types, and as such have been applied in diverse areas such as machine vision, speech recognition, and natural language processing (NLP).

3.4.3. Support vector machines

SVM is an extension of NN which can model and classify both linear and nonlinear input data. SVM transforms the input training data into a multidimensional space through the use of hyperplanes in higher dimensions and then performs the classification task on the new data [35]. Another name for these hyperplanes is decision planes which "decide" the class the data belongs to. SVM searches for those vector points in the multidimensional space (support vectors) which define the decision boundary and also provide significant separation among the data classes. Kernel functions form the core component of the SVM model which implement the nonlinear mapping of the high-dimensional data. Some of the popularly used kernel functions in SVM are linear, Gaussian, Sigmoid, polynomial and radial basis, which have different mathematical complexity, efficiency, and accuracy performance. SVM generalizes better as compared to other classifiers and, therefore, is more suited for situations where the dataset consists of more features as compared to the number of training data examples.

3.4.4. Random forests

RF belongs to a class of classifiers called ensemble learning (EL), where multiple classifiers are generated and the classification results of the constituent classifiers are aggregated to provide an output. The resultant benefit of such a scheme is that it provides improved prediction accuracy performance as compared to the case where a single classifier is used to model the training data [36]. Generally DTs are used with the bagging and boosting methods to realize an ensemble classifier such as an RF. Boosting method recursively uses multiple DTs to iteratively improve the classification accuracy of the RF. However, in the bagging method, the constituent DTs make independent classification decisions and through majority voting, the final prediction accuracy performance result is obtained. In RFs, the bagging method involves an extra element of randomness where instead of choosing the predictor with the best split at each node, a predictor variable is chosen randomly from a subset of better predictors. This randomness in the prediction process makes the RF generalize better as compared to other classification algorithms and thus avoids the over-fitting problem to a greater extent.

4. Experimental results

In this section we will provide the details of the experimental setup used to devise various scenarios for testing our proposed approach. Also, we will define the performance metrics used for evaluating the ML algorithms for the classification tasks. Further, we will provide the detailed results and their discussions, comparison with existing approaches, and, finally, leading to conclusions obtained from our study.

4.1. Experimental setup

In order to generate the results for the proposed approach of image-based Android malware classification, we devised and implemented a set of experiments. There were three sets of parameters in our study, namely, the ML classifiers, feature extraction approaches, and the dataset types. As already mentioned, the four ML classifiers we proposed to test were NB, MLP, SVM, and RF. The parameter settings of these ML algorithms, which we used during the experiments, are enlisted in Table 3. Default parameter settings of Weka tool were used for the ML algorithms, those of which are not specified in Table 3. We also have four feature extraction techniques, namely, Texture, DWT, HOG, and Gabor features, which were abbreviated as *Tex*, *Dwt*, *Hog*, and *Gabor*, respectively, for the purpose of concise presentation of results. Finally, we have three different image-based datasets, namely, Dex, Manifest, and Composite (Dex and Manifest features combined). To provide a comprehensive analysis and testing of our proposed approach, we performed the following experiments.

Table 3. Experimental parameter settings of ML algorithms.

ML classifier	Parameter	Value
MLP	Batch size	100
	Hidden layers	124
	Learning rate	0.3
SVM	Batch size	100
	Kernel	Radial Basis Function (RBF)
	Learning rate	0.001
	Loss	0.1
	Optimizer	Sequential Minimal Optimizer (SMO)
	Soft margin (C)	1
RF	Batch size	100
	Epochs	100
	Tree depth	20
NB	Batch size	100
	Distribution	Normal
	Discretization	False

- *Dataset effect*: In this experiment, we fixed the dataset and implemented all the combinations of the feature extraction techniques and classification with all the classifiers. Since there were four feature extraction techniques, we got 15 hybrid combinations of them, namely, 4 single (Tex, Dwt, Hog, and Gabor), 6 pairs (TexDwt, TexHog, TexGabor, DwtHog, DwtGabor, and HogGabor), 4 triplets (TexDwtHog, DwtHogGabor, TexHogGabor, and TexDwtGabor) and 1 quadruplet (TexDwtHogGabor). For each of the datasets, we got three sets of results, as depicted in Figures 4–6, for Dex, Manifest, and Composite datasets, respectively.
- *Features effect*: In this experiment, we identified the best hybrid feature combination and obtained Accuracy, Recall, Precision, and F1 score metrics for each of the three datasets. The performance metrics of the four classifiers were compared to find the best classifier. These results are presented in Tables 4–6, for *Tex*, *TexHog*, and *TexDwt* feature combinations, respectively.

- *Cross Validation*: In this experiment, we performed 10-fold cross validation to get an estimate on the suitability of the ML classifier models for classifying the unseen image data of the Android malware. These results are presented in Table 7, where only the best feature combination (*TexHog*) is studied.

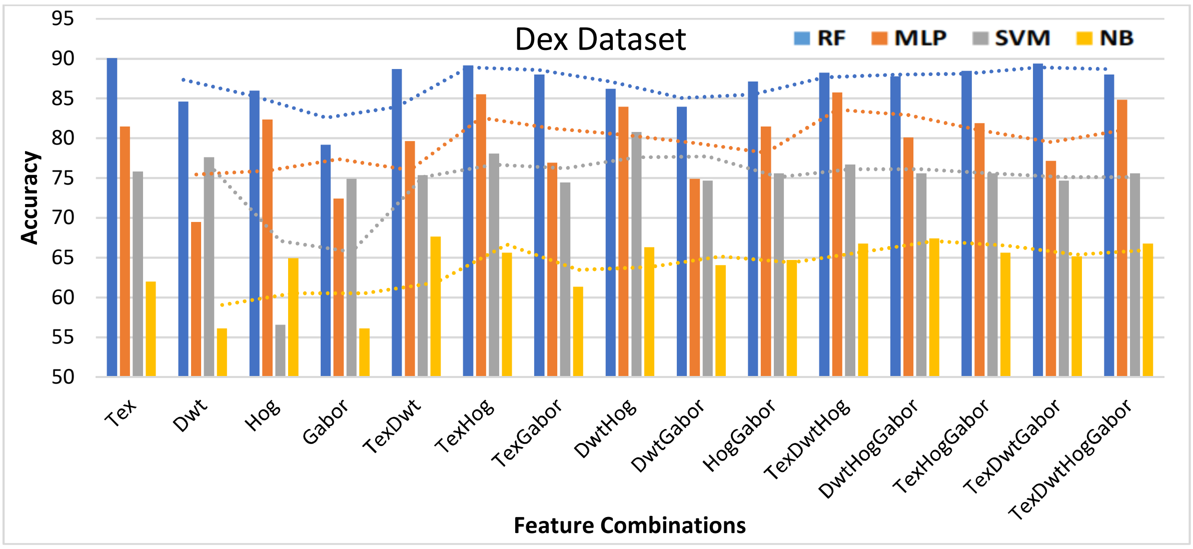


Figure 4. Classifier accuracies from Dex dataset for all feature combinations.

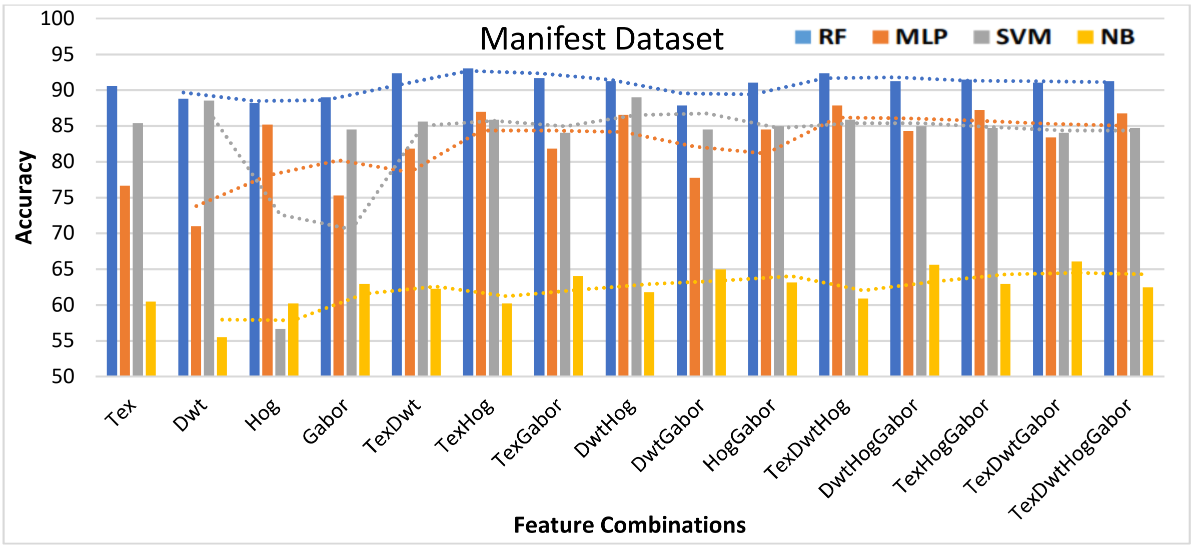


Figure 5. Classifier accuracies from Manifest dataset for all feature combinations.

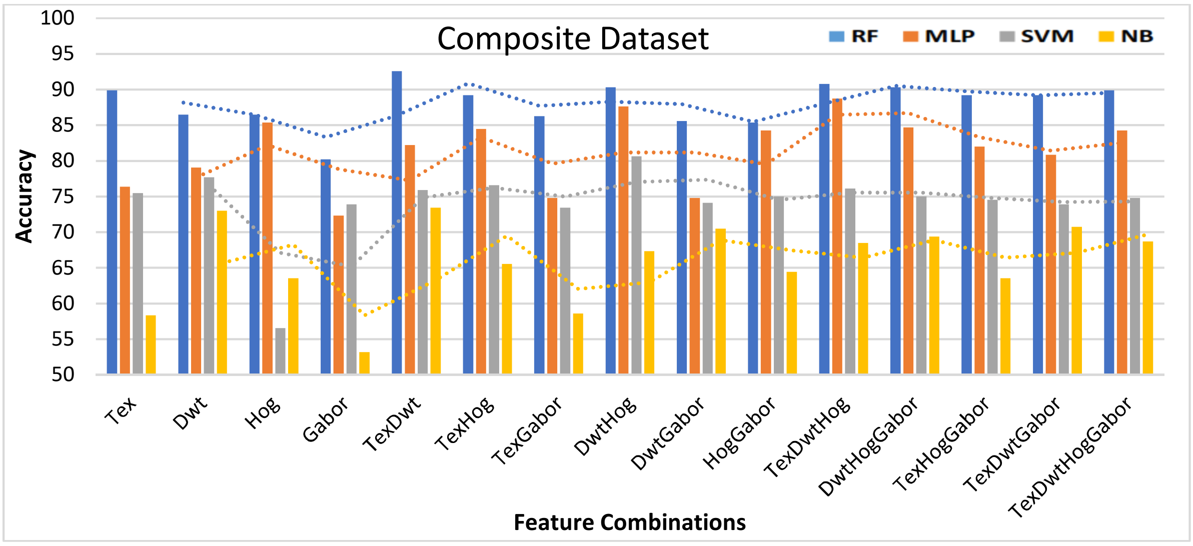


Figure 6. Classifier accuracies from Composite dataset for all feature combinations.

4.2. *Performance metrics*

The following performance metrics were used to evaluate the ML classifier models.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{4.1}$$

$$Precision = \frac{TP}{(TP + FP)} \tag{4.2}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{4.3}$$

$$F1\ score = \frac{(2 \times Recall \times Precision)}{(Recall + Precision)} \tag{4.4}$$

where TP, FP, FN, and TN are true positives, false positives, false negatives and true negatives, respectively.

4.3. *Results & discussion*

We now begin our results presentation and discussion along the same lines as described in the experimental setup above.

4.3.1. Dataset effect

The objective of this experiment is to find how the ML classifiers perform across different datasets. As we mentioned earlier, we have three image datasets obtained from the Dex file, Manifest file, and the Composite file and four feature extraction techniques.

- *Dex dataset:* In Figure 4, we show the accuracy results for the Dex dataset for 15 hybrid combinations of the feature types for four ML classifiers. The following observations can be readily made from this figure. First, RF classifier performance is the best as compared to MLP, SVM, and NB. For example, in the case of TFs, the accuracies of RF, MLP, SVM, and NB are 90.05, 81.45, 75.79, and 61.99%, respectively. Similar trends are observed even when the features are combined as a pair, triplets, or quadruplets. Second, the NB classifier is the one which has the least classification performance across all feature combinations. Third, the hybrid combinations of features do not seem to have a positive effect on the performance, at least in the Dex dataset scenario. Fourth, the maximum accuracy of 90.05% is obtained in the Dex dataset for the RF classifier with the TFs alone.
- *Manifest dataset:* Moving on to the Manifest dataset results, we observe that there is an increase in the performance results. As can be seen from Figure 5 for the TFs, we obtain accuracies of 90.56, 76.63, 85.39, and 60.45% for RF, MLP, SVM, and NB, respectively. However, the increase in the performance is for RF and SVM classifiers only. One important conclusion which we can make is that the Manifest dataset must be a preferred choice (as compared to Dex dataset) for classifying Android apps as Botnet or Benign. Another observation which can be made is that in this case, the hybrid feature combination is providing a better performance as compared to a single feature set. For example, while using RF with TFs alone, we get an accuracy of 90.56%, but by combining the TFs with DWT features, the accuracy improves to 92.36% (an increase of 1.8%). The maximum accuracy of 93.03% and a recall of 97.1% is obtained for RF classifier with the combination of Texture and HOG features.
- *Composite dataset:* Further, we observe the performance of the ML classifiers on the Composite dataset for all the feature combinations. It was found that this dataset resulted in reduced accuracies as compared to the manifest dataset. As can be seen from Fig 6, for the TFs we obtain accuracies of 89.86, 76.35, 75.45, and 58.33% for RF, MLP, SVM, and NB, respectively. The decrease in the performance is for all the classifiers. One important conclusion which we can make is that the Composite dataset is not to be preferred when (as compared to Manifest dataset) classifying Android apps as Botnet or Benign. Another observation from these results is that the hybrid feature combination is still providing a better performance as compared to single feature set. For example, while using RF, we get average accuracies as 85.75, 88.21, 89.86, and 89.86% for the single, pairs, triplets, and quadruplet feature combinations, respectively. The maximum accuracy of 92.57% is obtained for the RF classifier with the combination of Texture and DWT features.
- *Summary of dataset effect:* It was interesting to look at the trends for the ML classifiers across the datasets, which is depicted in Figure 7. It can be observed that the Dex dataset provides a lower performance as compared to the Manifest dataset. However, when we use the Composite dataset, the performance decreases. In general, all the ML classifiers perform the best for the Manifest dataset (RF is 91%, MLP is 82.46%, and SVM is 83.55%), with an exception of the NB classifier. It should be noted that these accuracies are averaged over all feature combinations. Again, it can be seen that RF is the best performer and NB is the least one. So, the conclusion is to use Manifest dataset for identifying and classifying Android malware as botnet or benign.

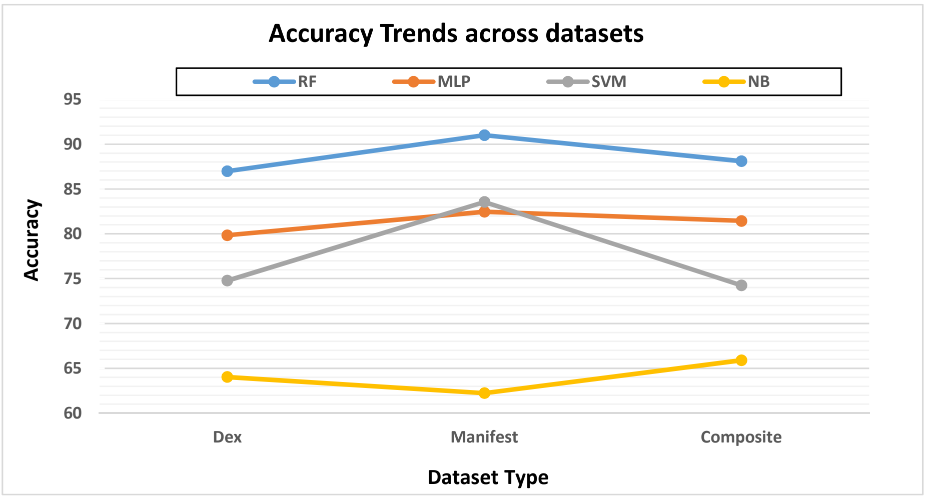


Figure 7. Effect of datasets on accuracies of ML classifiers.

4.3.2. Features effect

The objective of this experiment is to find how the ML classifiers perform based on the type and combinations of image-based data features. As we mentioned earlier, we have four feature extraction approaches, namely, *Tex*, *Dwt*, *Hog*, and *Gabor*, and three datasets, namely, Dex, Manifest, and Composite. Due to the combination of the features, we propose the term hybridization levels, which can take values of *Single*, *Pairs*, *Triplets*, and *Quadruplet*. We now present our observations and discussion related to the effect of features on the classifier performance.

- *Hybridization effect*: In order to find the effect of hybridization levels on the ML performance, we first considered the Manifest dataset results. The reason for choosing this dataset for the study was based on the fact that in the previous results, we came to the conclusion that the Manifest dataset was the best option (as compared to the Dex and Composite datasets) for classifying the Android malware. Since there are multiple combinations in each hybridization level, we calculated the average accuracies for all the four ML classifiers, and the results are depicted in Figure 8.
  We can conclude from the results that there is, in general, an increasing trend in the accuracies as the hybridization level increases. However, this conclusion is surely true for MLP (accuracy values of 77.02% for *Single*, 83.2% for *Pairs*, 85.67% for *Triplets*, and 86.7% for *Quadruplet*). For the other classifiers (RF, SVM, and NB), an interesting phenomenon is observed for the *Quadruplet* case, where the accuracy almost saturates or even decreases. These results also clearly depict the overall accuracy results across the four ML classifiers, where the best one is the RF and the worst performer is the NB. SVM and MLP are average performers with similar accuracy results. From these observations, we conclude that RF is the best performer in terms of accuracy and the variation in its performance is the least, and as such it is a stable ML classifier. On the other hand, due to an increase in the features, the MLP classifier shows an increased accuracy performance.
- *Hybridization level-wise performance*: The next objective was to find the hybridization level-wise best ML performer. These results are for the RF classifier as it was found to be the best among the four classifiers. However, we present the accuracies which are averaged over the three datasets. Due to the space limitations, we only present the final average results. For the *Single* level case, the

best performer was the *Tex* features with an accuracy of 90.56% (see Table 4). In the *Pairs* level, the best feature combination was *TexHog* features with an accuracy of 93.03%. As we increase the level to *Triplets*, 92.36% was the best accuracy given by the combination of *TexDwtHog*. Finally, in the *Quadruplet* case, the accuracy was found to be 91.24%. The conclusion of this study is that for the RF classifier (the best classifier), the best combination of features is *TexHog*, which means a hybridization level of *Pairs* is sufficient to give the best performance. There is no further gain achieved in the case of higher hybridization levels.

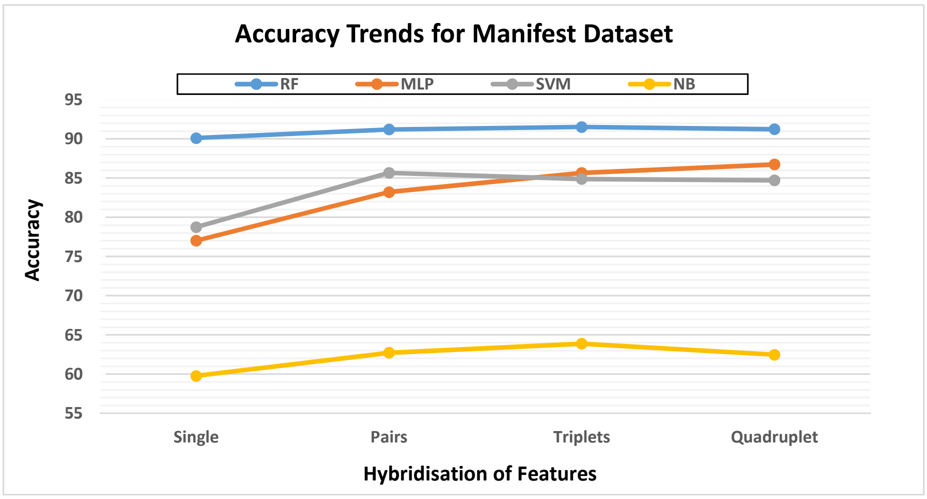


Figure 8. Effect of hybridization level on accuracies for the Manifest dataset.

Table 4. Testing results for three datasets using only Texture features.

Dataset	Classifier	Accuracy	Recall	Precision	F1 score
Dex	RF	**90.05**	0.926	0.880	0.839
	MLP	81.45	0.902	0.858	0.818
	SVM	75.79	0.848	0.828	0.808
	NB	61.99	0.636	0.678	0.725
Manifest	RF	**90.56**	0.949	0.908	0.870
	MLP	76.63	0.836	0.865	0.896
	SVM	85.39	1.000	0.805	0.674
	NB	60.45	0.575	0.614	0.658
Composite	RF	**89.86**	0.937	0.888	0.845
	MLP	76.35	0.882	0.868	0.855
	SVM	75.45	1.000	0.653	0.485
	NB	58.33	0.679	0.682	0.686

- *Highest performing features*: To find the top three feature set performers, we focused only on the maximum accuracies for the RF classifier for the Manifest dataset. It was found that the best accuracy of 93.03% was obtained for the TexHog, followed by 92.36% for *TexDwt* and *TexDwtHog* (tied together). These results are also verified from the previous conclusion obtained during the hybridization level-wise performance above. This study concludes another interesting observation that out of the four feature extraction approaches, only *Tex*, *Dwt*, and *Hog* features are found among the top performers, and the *Gabor* approach is not contributing much to the classification performance. Now, in Table 5 (for *TexHog* case) and Table 6 (for *TexDwt* case), we present all the four performance metrics (Accuracy, Recall, Precision, and F1 score) to get an overall behavior of the classifiers. Also, we present the results across all the three datasets. The conclusion is that the best performer is the RF classifier for the Manifest dataset using the *TexHog* feature combination by providing a classification accuracy of 93.03% and a recall of 97.1%. However, if someone is interested in using a single feature only, then *Tex* feature can give a maximum accuracy of 90.56% with the Manifest dataset. The results of *TexDwtHog* were not presented since it had similar performance as compared to the *TexDwt*, which could be achieved by combining only two types of features.

Table 5. Testing results for three datasets using Texture and HOG features.

Dataset	Classifier	Accuracy	Recall	Precision	F1 score
Dex	RF	**89.39**	0.888	0.877	0.865
	MLP	84.02	0.844	0.857	0.870
	SVM	80.14	1.000	0.705	0.544
	NB	70.95	0.660	0.672	0.684
Manifest	RF	**93.03**	**0.971**	0.915	0.865
	MLP	86.97	0.850	0.850	0.850
	SVM	84.91	1.000	0.790	0.653
	NB	62.92	0.560	0.612	0.674
Composite	RF	**89.19**	0.934	0.867	0.808
	MLP	84.46	0.823	0.821	0.819
	SVM	77.70	1.000	0.657	0.490
	NB	68.69	0.650	0.631	0.613

Table 6. Testing results for three datasets using Texture and DWT features.

Dataset	Classifier	Accuracy	Recall	Precision	F1 score
Dex	RF	**88.71**	0.882	0.868	0.855
	MLP	79.73	0.767	0.767	0.767
	SVM	77.88	1.000	0.660	0.492
	NB	67.65	0.609	0.657	0.714
Manifest	RF	**92.36**	0.954	0.908	0.865
	MLP	81.80	0.783	0.793	0.803
	SVM	85.62	1.000	0.801	0.668
	NB	68.31	0.638	0.630	0.622
Composite	RF	**92.57**	0.935	0.912	0.891
	MLP	83.11	0.812	0.805	0.799
	SVM	75.90	1.000	0.616	0.446
	NB	73.42	0.717	0.678	0.642

- *Summary of features effect:* The experiments conducted for a total 15 features combinations involving 4 distinct feature extraction approaches (namely, Tex, Hog, Dwt, and Gabor) provided significant insights related to the proposed feature hybridization concept. However, it is worth stating the time overhead involved in feature extraction and fusion process. For a total of 4429 images (2500 clean and 1929 botnet), it took a processing time of 173.84, 177.54, 177.16, and 171.75s for Tex, Hog, Dwt, and Gabor features, respectively. It was observed that the time overheads are very close to each other due to the fact that significant time in the process was involved in loading the images (using Matlab R2021a functions) and the actual feature extraction time was significantly lower. The specifications of the machine used had an Intel CPU core i7 11th gen with a clock frequency of 3.0 GHz and a RAM of 16 GB running on Windows 11.

4.3.3. Cross validation results

The ML classifiers which have been developed using the training images dataset need to be checked for their generalization ability on the unseen dataset. To verify this, we performed a 10-fold cross validation of the ML classifiers using the validation dataset. The average validation accuracies have been depicted in Table 7 for all the four classifiers. In this table we present the best performing feature combination of *TexHog* with a hybridization level of *Pairs*. It should be recalled that the best test classification accuracy of 93.03% was achieved with RF classifier for the Manifest dataset. However, for the sake of comparison, we present the Accuracy, Recall, Precision, and F1 score for all the three datasets. It can be observed from Table 7 that the accuracy of RF is the highest for the Manifest dataset (89.92%), followed by Dex dataset (87.87%) and Composite dataset (87.04%). Similar patterns were observed for the test accuracies earlier (refer to Table 5). Also, it should be noted that the other performance metrics follow the same pattern with RF as the best classifier, followed by MLP, SVM, and NB. These validation results verify that the ML models are valid and can be used for testing unknown Android apps for being botnet or benign with a high level of accuracy. One point, however, to be observed is that the validation accuracies are lower as compared to the test accuracies and it gives an indication that an increase in the dataset sizes are warranted.

Table 7. 10-fold cross validation results for three datasets using Texture and HOG features.

Dataset	Classifier	Accuracy	Recall	Precision	F1 score
Dex	RF	**87.87**	0.917	0.850	0.793
	MLP	83.31	0.810	0.807	0.805
	SVM	77.38	0.999	0.648	0.480
	NB	66.67	0.612	0.625	0.639
Manifest	RF	**89.92**	0.946	0.875	0.814
	MLP	86.56	0.848	0.844	0.842
	SVM	84.51	1.000	0.782	0.643
	NB	60.05	0.533	0.582	0.642
Composite	RF	**87.04**	0.898	0.842	0.794
	MLP	83.46	0.810	0.810	0.811
	SVM	75.56	1.000	0.610	0.439
	NB	66.40	0.609	0.625	0.641

4.3.4. Results from DL approaches

One of the experiments was very important from the point of view of comparing our manual feature extraction based approach with automated feature extraction from DL approaches. For this purpose, we considered six most prominent pretrained CNN models, namely, MobileNetV2, InceptionResNetV2, ResNet101, VGG16, VGG19, and DenseNet121. Both the ML and DL approaches were tested on the DEX and Manifest datasets, which were images-based datasets from ISCX Android Botnet repository. Since for both the approaches the inputs were image-based data, it was fair to compare their performance in terms of classification accuracy, precision, recall, and F1 score. These results for all the six pretrained CNN models are presented in Table 8. It is to be noted that these results have also been reported in our work in [21], but have been included here as well for the ease of readability and comparison purposes.

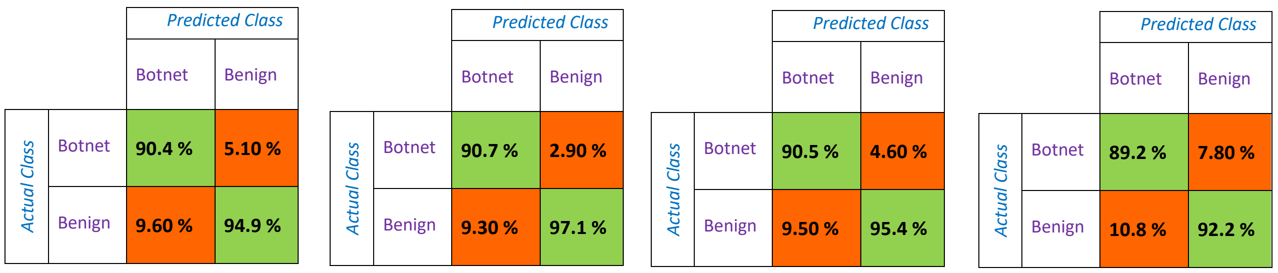
Table 8. Testing results of six pretrained DL models on DEX and Manifest datasets.

	DEX				**Manifest**			
Model	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*Accuracy*	*Precision*	*Recall*	*F1 Score*
MobileNetV2	88.0	86.3	86.8	86.6	**91.0**	**89.2**	**89.9**	**89.5**
InceptionResNetV2	85.0	83.1	83.8	83.5	87.0	85.2	85.8	85.6
ResNet101	**90.0**	**88.1**	**88.7**	**88.4**	**91.0**	**89.2**	**89.8**	**89.5**
VGG16	88.0	86.2	86.8	86.5	89.0	87.3	87.8	87.4
VGG19	89.0	87.2	87.8	87.5	89.0	87.1	87.9	87.6
DenseNet121	84.0	82.3	82.9	82.7	90.0	88.3	88.8	88.5

The results with DL models are also exhibiting similar trends as observed with our results in the ML-based approach. In general, the classification accuracy was better from the Manifest dataset as compared to the DEX dataset. For example, the MobileNetV2 had a better accuracy (e.g. 91%) for the Manifest dataset as compared to the Dex dataset (e.g. 88%). Also, the MobileNetV2 was the best performer in the Manifest dataset category (accuracy of 91%), with ResNet101 having very similar performance. However, since MobileNetV2 has a simpler model structure and is less computationally demanding, it is considered the best performer both in terms of accuracy and lesser complexity. The summary of this comparison with our approach is presented subsequently.

4.3.5. Confusion matrices

Finally, we look at the figures from the confusion matrices to correlate with the accuracy performance results obtained earlier. Figure 9 presents the confusion matrices for three best feature combinations for the RF classifier obtained from the test classification results, namely, *Tex*, *TexHog*, and *TexDwt* cases for the Manifest dataset. The fourth confusion matrix is for the case of DL approach (specifically, MobileNetV2) for the purpose of comparison with our approach. First, on the average, the benign class prediction percentages (94.9% for *Tex*, 97.1% for *TexHog*, and 95.4% for *TexDwt*) are better than the botnet class percentages (90.4% for *Tex*, 90.7% for *TexHog*, and 90.5% for *TexDwt*). The reason for this is that we have more training examples of benign class (2500 instances) as compared to the botnet class (1929 instances). Second, the errors in predicting the botnet as benign (5.1% for *Tex*, 2.9% for *TexHog*, and 4.6% for *TexDwt*) are less than the errors in predicting benign as botnet (9.6% for *Tex*, 9.3% for *TexHog*, and 9.5% for *TexDwt*). This is a desirable characteristic of our model as it is fine to get more false alarms (predicting benign as botnet), than to miss detect a botnet and treat it as benign. Third, the best prediction accuracy (90.7%) of the botnet class is achieved by the *TexHog* case, which was also the best scenario in the case of the test and validation results. Finally, it is observed that the DL approach has lower values for the two metrics, namely, benign class prediction (92.2%), botnet class prediction (89.2%), and higher for two metrics, namely, predicting botnet as benign (7.8%) and benign as botnet (10.8%). These values from the confusion matrices clearly prove that our proposed approach (feature hybridization with classification using ML classifiers) performs better than the DL approaches (automated feature extraction and classification).



Actual Class	Predicted Class: Botnet	Predicted Class: Benign
Botnet	90.4 %	5.10 %
Benign	9.60 %	94.9 %

(a) Texture Features only

Actual Class	Predicted Class: Botnet	Predicted Class: Benign
Botnet	90.7 %	2.90 %
Benign	9.30 %	97.1 %

(b) Texture + HOG Features

Actual Class	Predicted Class: Botnet	Predicted Class: Benign
Botnet	90.5 %	4.60 %
Benign	9.50 %	95.4 %

(c) Texture + DWT Features

Actual Class	Predicted Class: Botnet	Predicted Class: Benign
Botnet	89.2 %	7.80 %
Benign	10.8 %	92.2 %

(d) MobileNetV2 Model

Figure 9. Confusion matrices for the best combination of features for Manifest dataset.

4.3.6. Comparison with existing research

In order to put our research work in context of the existing results available in the literature, we now provide a comparative study. For this purpose, first we looked at some existing solutions, which have extensively studied classification performance and their robustness has been validated [39]. Even though they performed very well, the datasets used by these approaches are different than the ISCX dataset, these solutions cater to dynamic AMD and are focused on obfuscation studies, and ours is a static approach. As such, a direct comparison could not be made in the context and scope of the proposed work. However, we have included a comparative study table with nonimage-based AMD approaches (see Table 9). This table shows that our approach outperforms some of the approaches in the nonimage-based category, which are both using handcrafted features and automated features, in spite of being applied to varied datasets. To the best of our knowledge the ISCX dataset was not used for nonimage-based study.

Then, we explored the comparison with image-based approaches a summary of which is presented in Table 10. It is to be noted that the comparison cannot be done on a one-to-one basis, as the experimental setup, conditions, and model parameters are bound to be different. The factors which we have tried to make common for comparison is the dataset (ISCX), dataset type (manifest features), classifier (RF), and the feature extraction technique (HOG features). First, we compare our *validation accuracy* results with the paper by Yerima et. al [5] who has also used the Manifest dataset, and in this case we get about 1% increased accuracy of 88.2% as compared to theirs (which was 87.1%). However, they get better performance when using the XGBoost classifier, which cannot be compared with RF classifier performance as in our case. Second, for the *testing recall* results, the approach in [18] achieves a performance of 93% and our approach reaches 97.1%. We have achieved a significant performance gain of about 4%, however they have used auto-encoders along with HOG, while we have used TFs with HOG. It is well-known that auto-encoders [37], which belong to the DL category of models, also require high computational resources. It is to be noted that the *testing accuracy* results are comparable for both the approaches. So, in conclusion, we achieve better results with reduced model training costs and computational complexity. A quantitative comparison in terms of computational requirements (e.g., central processing unit (CPU)/ graphics processing unit (GPU) cycles, memory capacity, and execution times) is envisioned as a future extension to the research presented in this paper.

Finally, the last comparison is with the approaches in the automated feature extraction category using DL approaches. Here, we demonstrate a very important insight, namely, handcrafted features-based approaches (our approach) can perform better or at par with DL-based approaches. In one of the recent works [21], the authors have used the same image-based dataset (ISCX Android Botnet) and experimented with the Transfer Learning feature of popular pretrained CNN models, such as MobileNetV2, RestNet101, InceptionRestNetV2, VGG16, VGG19, and DenseNet121. They achieved a maximum classification *testing accuracy* of 91% for the Manifest dataset using the MobileNetV2 classifier. This concludes that DL (or CNN) approaches fail to reach the classification testing accuracy of 93.03% (falls short by about 2%, which is a significant accuracy loss), which we have achieved in the current paper by leveraging the handcrafted feature extraction. In another work by Ganesh et. al [38], they used the CNN model that automatically extracts the features (from a different dataset called as Drebin) as compared to our handcrafted features. While they have used complex classification model (CNNs) with higher computational costs, we have used manually-extracted features which are not computationally demanding, and the results are the same (93% testing classification accuracy). In conclusion, our study achieves either better performance or comparable to the ones in the extant literature, with reduced computational complexity.

Table 9. Comparison with existing research on nonimage-based AMD approaches.

Paper	Approach	Dataset	Results	Pros	Cons
Zhang et. al (2023) [11]	RF with PCA	CICAndMal2017	90.33% (test accuracy)	PCA with cascaded RF	Comparison with automated feature extraction is missing
Hasan et. al (2021) [13]	Various traditional ML approaches	Android Malware Dataset	91.6% accuracy	Robustness analysis done	Could have included system calls information
Millar et. al (2021) [15]	Basic CNN models	Android Malware Dataset	91.00% (testing accuracy)	Caters to new malware detection	Basic CNN to be compared with pre-trained DL models
Mohammed et. al (2022) [16]	1-D CNN models	Various datasets combined	90.00% accuracy	Uses Manifest and Intents information	Lower classification accuracy
Proposed approach (this paper)	(HOG + Texture) with RF	ISCX Android Botnet (manifest)	**93.03%** (testing accuracy)	Feature hybridization (TF, DWT, HOG, Gabor) and comparison with DL approaches	Comparison with dynamic approaches

Table 10. Comparative study with existing research on image-based AMD approaches.

Paper	Approach	Dataset	Results	Pros	Cons
Yerima et. al (2022) [5]	HOG with RF	ISCX Android Botnet (manifest)	87.1% (validation accuracy)	Composite image of DEX and manifest features	Comparison with DNN is missing
Proposed approach (this paper)	HOG with RF	ISCX Android Botnet (manifest)	**88.2%** (validation accuracy)	Choice and control over features, Feature hybridization (TF, DWT, HOG, Gabor)	Comparison with DNNs in terms of computational requirements
Yerima et. al (2021) [18]	(HOG + Autoencoders) with RF	ISCX Android Botnet (manifest)	93% (testing recall)	Six different ML classifiers compared	Comparison with DL was expected
Proposed approach (this paper)	(HOG + Texture) with RF	ISCX Android Botnet (manifest)	**97.1%** (testing recall)	Choice and control over features, Feature hybridization (TF, DWT, HOG, Gabor)	Comparison with DNNs in terms of computational requirements
Mohammed et. al (2022) [21]	Transfer Learning on pre-trained CNN models	ISCX Android Botnet (manifest)	91.00% (testing accuracy)	Uses six popular pre-trained CNN models with Manifest images	The testing accuracy obtained is 2% less than our approach
Proposed approach (this paper)	(HOG + Texture) with RF	ISCX Android Botnet (manifest)	**93.03%** (testing accuracy)	Choice and control over features, Feature hybridization (TF, DWT, HOG, Gabor)	Comparison with DNNs in terms of computational requirements

4.3.7. Insights summary

Based on the experiments we conducted and the results achieved therein, we now present meaningful and valuable insights to the researchers in this domain in a concise manner below:

- **Best ML Classifier:** RF was the best ML classifier in terms of classification accuracy across all datasets and feature space (see Table 5).
- **Consistent ML Classifier:** MLP is the only classifier which provided a consistent improvement in the classification accuracy with an increase in the number of hybridization levels (see Figure 8).
- **Spatial v/s Frequency domain features:** One of the very important insight was the consistently better performance of spatial domain features as compared to frequency domain features. More specifically, HOG and Texture features gave the highest classification accuracy with various ML classifiers (see Table 5).
- **Pairs hybridization is the best:** It was found with experimentation that going from single to a pair of features resulted in an improved performance in the case of RF, NB, and SVM. A further increase in the hybridization levels (triplets and quadruplets) did not achieve any further gain (see Figure 8).
- **Feature agnostic ML classifier:** It was found with experimentation that going from single to a pair of features resulted in an improved performance in the case of RF, NB, and SVM. A further increase in the hybridization levels (triplets and quadruplets) did not achieve any further gain (see Figure 8).

- **Best dataset:** The Manifest dataset outperformed the DEX and composite datasets. This led us to reason that the metadata present in the Manifest file is a much more valuable and contributing factor in the classification process as compared to the actual data present in the DEX file (see Figure 7).
- **ML v/s DL:** Based on the experimental results, it was concluded that for the image-based ISCX Android Botnet dataset, the use of ML classifiers on the handcrafted features achieved better performance (by about 2% improved classification accuracy) as compared to DL/CNN classifiers using automated feature extraction (see Table 10).
- **Comparison criteria:** We have observed that most of the research work in the literature focuses only on the classification accuracy as a criteria for performance comparison. However, we believe that other criteria such as computational requirements (e.g., CPU/GPU cycles, memory capacity, and execution times), control over feature space and model choice, versatility, and explainability should also be taken into account (see Tables 10, 9 and 1).

4.3.8. System optimization, evolution, and sustainability

There have been numerous AMD solutions presented in the past and will also be presented in the future. The proposed solution in this paper is one among such multitude of solutions. The AMD solution presented in this paper could only be evaluated on a certain limited set of data, ML classifiers, and feature extraction techniques due to the predefined scope of work (feature hybridization being the core contribution and novelty) and constraints of time. In order to critically evaluate our proposed solution and find means for improvements, we consider the following three crucial issues of system optimization, evolution, and sustainability.

- **Optimization:** There have been efforts in other security research domains with respect to efficiency improvements and safety considerations, by borrowing concepts from network theory and control theory. Heterogeniety in the Android Malware samples can be addressed by the use of heterogenous network representational learning to mine implicit patterns for advanced malware identification [40]. Further, to cater for the need to identify new/recent malwares, it is recommended to incorporate online learning mechanisms as opposed to routine offline learning from static data. The system would benefit from a feedback control loop from the output (e.g., reduction in classification accuracy) to the input (e.g., to increase the dataset samples or the learning rate) for optimized performance [41].
- **Evolution:** One of the important considerations in the performance measurement of a system is its variability with the passage of time. An AMD solution implemented today will have a degraded performance tomorrow as it will not be able to identify new malwares. This requires that the system is studied over an extended period of time [42]. In order to capture the time-varying system behavior, provisions need to be made for retraining the models at regular intervals [43]. Therefore, the results presented earlier lose their significance as new data becomes available for fine-tuning the ML models. Thus, further work can focus on incorporating and modeling metrics like malware data evolution rate, labeled data availability, resource constraints, and performance accuracy targets.
- **Sustainability:** There has been significant attention to making AMD systems as sustainable as possible [44]. This led to the concept of self-evolving AMD systems which stand the test of time and can identify even the most recent malwares. As time passes, the APK evolves based on the

user needs and requirements, and this also results in the increase in the type and severity of Android malwares. In order to be abreast of such evolutions, AMD systems need to be assessed at regular intervals and over a long span of time. Solutions such as Droidspan [45] have incorporated the metric of sustainability, which gives us an indication of how frequently the ML model requires updates if the features become deteriorated. As far as a qualitative comparison is concerned, our proposed work tries to reasonably match the performance of exisitng approaches, as we are using a dataset which spans over 5 years as compared to other approaches which cover more time span (e.g., 8 years). It falls short in accuracy and model updatability, but scores better in terms of feature engineering due to the hybridized nature of feature modeling.

5. Conclusions and future work

To address the need for an automated and efficient botnet detection system, this research paper proposed, implemented, and evaluated an image-based AMD framework using the feature hybridization concept for identifying and classifying botnet infected Android applications. To achieve this objective, the framework consisted of Android malware data collection, conversion from file-based data to images, feature extraction, and applying supervised ML classifiers. The ISCX Android Botnet dataset was converted to images and categorized as Dex, Manifest, and Composite datasets. Feature extraction from images was performed using Texture, HOG, DWT, and Gabor approaches and all of their fifteen combinations were used to test the classifier performances. The ML classifiers used to identify the Android botnets were NB, SVM, MLP, and RF. It was found from the experimental results that the best ML classifier was RF, followed by MLP, SVM, and NB, due to the fact that RF ensembles the prediction results of multiple classifiers for improved classification accuracy. The dataset which resulted in the best prediction was the Manifest dataset, since the metadata present in the manifest file contains comprehensive, valuable, and distinguishable information about specific Android application, which aided in assessing the trustworthiness of the application.

In the single features case, the Texture features provided the best performance because they extracted both the first and second order spatial features from the images, which significantly distinguished a botnet image from a benign image. The best performing hybridization level was a pairing of Texture and HOG features, which confirmed the fact that an ensemble of features from the same domain (spatial) provided enhanced information to make an informed classification decision. The best classification testing accuracy of 93.03% and a recall of 97.1% was achieved by the RF classifier for the Manifest dataset with the hybridized features coming from Texture and HOG approaches. Some of the areas for further improvement in this research are to perform multi-class classification to identify the fourteen different families of Android malware from the ISCX Android Botnet dataset. Another interesting study could involve the comparison (in terms of computational requirements) of handcrafted feature extraction approaches to automated feature extraction using DL models such as CNN, you only look once (YOLO), or pretrained DNNs (e.g., GoogleNet, AlexNet, or VGG19).

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The author would like to acknowledge the support of College of Computer Engineering and Science at Prince Mohammad bin Fahd University (PMU) to perform this research work.

Conflict of interest

The authors declare there is no conflict of interest.

References

1. T. Shishkova, A. Kivva, *Mobile Malware Evolution 2021*, 2021. Available from: https://securelist.com/mobile-malware-evolution-2021/105876.
2. AppBrain, *Number of Android Apps on Google Play*, 2024. Available from: https://www.appbrain.com/stats/number-of-android-apps.
3. McAfee, *McAfee Mobile Threat Report*, 2021. Available from: https://www.mcafee.com/content/dam/global/infographics/McAfeeMobileThreatReport2021.pdf.
4. J. Senanayake, H. Kalutarage, M. O. Al-Kadri, Android mobile malware detection using machine learning: A systematic review, *Electronics*, **10** (2021), 1606. https://doi.org/10.3390/electronics10131606
5. S. Y. Yerima, A. Bashar, A novel android botnet detection system using image-based and manifest file features, *Electronics*, **11** (2022), 486. https://doi.org/10.3390/electronics11030486
6. Z. Wang, Q. Liu, Y. Chi, Review of android malware detection based on deep learning, *IEEE Access*, **8** (2020), 181102–181126. https://doi.org/10.1109/ACCESS.2020.3028370
7. I. Almomani, A. Alkhayer, W. El-Shafai, An automated vision-based deep learning model for efficient detection of android malware attacks, *IEEE Access*, **10** (2022), 2700–2720. https://doi.org/10.1109/ACCESS.2022.3140341
8. D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, Q. Zheng, Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture, *Comput. Netw.*, **171** (2020), 107138. https://doi.org/10.1016/j.comnet.2020.107138
9. K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, H. Liu, A review of android malware detection approaches based on machine learning, *IEEE Access*, **8** (2020), 124579–124607. https://doi.org/10.1109/ACCESS.2020.3006143
10. F. Taher, O. AlFandi, M. Al-kfairy, H. Al Hamadi, S. Alrabaee, Droiddetectmw: A hybrid intelligent model for android malware detection, *Appl. Sci.*, **13** (2023), 7720. https://doi.org/10.3390/app13137720
11. X. Zhang, J. Wang, J. Xu, C. Gu, Detection of android malware based on deep forest and feature enhancement, *IEEE Access*, **11** (2023), 29344–29359. https://doi.org/10.1109/ACCESS.2023.3260977

12. N. Herron, W. B. Glisson, J. Todd McDonald, R. K. Benton, Machine learning-based android malware detection using manifest permissions, in *Proceedings of the 54th Hawaii International Conference on System Sciences*, (2021), 6976.

13. H. Hasan, B. T. Ladani, B. Zamani, Megdroid: A model-driven event generation framework for dynamic android malware analysis, *Inform. Software Tech.*, **135** (2021), 106569. https://doi.org/10.1016/j.infsof.2021.106569

14. G. Xiao, J. Li, Y. Chen, K. Li, Malfcs: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks, *J. Parallel Distr. Com.*, **141** (2020), 49–58. https://doi.org/10.1016/j.jpdc.2020.03.012

15. S. Millar, N. McLaughlin, J. M. del Rincon, P. Miller, Multi-view deep learning for zero-day android malware detection, *J. Inf. Secur. Appl.*, **58** (2021), 102718. https://doi.org/10.1016/j.jisa.2020.102718

16. A. T. Kabakus, Droidmalwaredetector: A novel android malware detection framework based on convolutional neural network, *Expert Syst. Appl.*, **206** (2022), 117833. https://doi.org/10.1016/j.eswa.2022.117833

17. G. D'Angelo, E. Farsimadan, M. Ficco, F. Palmieri, A. Robustelli, Privacy-preserving malware detection in android-based iot devices through federated markov chains, *Future Gener. Comp. Sy.*, **148** (2023), 93–105. https://doi.org/10.1016/j.future.2023.05.021

18. S. Y. Yerima, A. Bashar, Bot-img: A framework for image-based detection of android botnets using machine learning, in *2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA)*, (2021), 1–7. https://doi.org/10.1109/AICCSA53542.2021.9686850

19. J. Singh, D. Thakur, F. Ali, T. Gera, K. S. Kwak, Deep feature extraction and classification of android malware images, *Sensors*, **20** (2020), 7013. https://doi.org/10.3390/s20247013

20. J. Tang, R. Li, Y. Jiang, X. Gu, Y. Li, Android malware obfuscation variants detection method based on multi-granularity opcode features, *Future Gener. Comp. Sy.*, **129** (2022), 141–151. https://doi.org/10.1016/j.future.2021.11.005

21. A. S. Mohammed, S. Seher, S. Y. Yerima, A. Bashar, A deep learning based approach to android botnet detection using transfer learning, in *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, (2022), 543–548. https://doi.org/10.1109/CICN56167.2022.10008334

22. Y. He, X. Kang, Q. Yan, E. Li, Resnext+: Attention mechanisms based on resnext for malware detection and classification, *IEEE T. Inf. Foren. Sec.*, **19** (2024), 1142–1155. https://doi.org/10.1109/TIFS.2023.3328431

23. A. F. Abdul Kadir, N. Stakhanova, A. A. Ghorbani, Android botnets: What urls are telling us, in *Network and System Security* , Springer, Cham, (2015), 78–91. https://doi.org/10.1007/978-3-319-25645-0_6

24. M. A. Al-Asadi, S. Tasdemir, Empirical comparisons for combining balancing and feature selection strategies for characterizing football players using fifa video game system, *IEEE Access*, **9** (2021), 149266–149286. https://doi.org/10.1109/ACCESS.2021.3124931

25. A. Humeau-Heurtier, Texture feature extraction methods: A survey, *IEEE Access*, **7** (2019), 8975–9000. https://doi.org/10.1109/ACCESS.2018.2890743

26. A. Latif, A. Rasheed, U. Sajid, J. Ahmed, N. Ali, N. I. Ratyal, et al., Content-based image retrieval and feature extraction: A comprehensive review, *Math. Probl. Eng.*, **2019** (2019), 9658350. https://doi.org/10.1155/2019/9658350

27. V. Verma, S. K. Muttoo, V. B. Singh, Multiclass malware classification via first-and second-order texture statistics, *Comput. Secur.*, **97** (2020), 101895. https://doi.org/10.1016/j.cose.2020.101895

28. G. N. Srinivasan, G. Shobha, Statistical texture analysis, in *Proceedings of World Academy of Science, Engineering and Technology*, **36** (2008), 1264–1269.

29. A. Ramola, A. K. Shakya, D. Van Pham, Study of statistical methods for texture analysis and their modern evolutions, *Eng. Rep.*, **2** (2020), e12149. https://doi.org/10.1002/eng2.12149

30. N. Chawla, H. Kumar, S. Mukhopadhyay, Machine learning in wavelet domain for electromagnetic emission based malware analysis, *IEEE T. Inf. Foren. Sec.*, **16** (2021), 3426–3441. https://doi.org/10.1109/TIFS.2021.3080510

31. N. Aggarwal, R. K. Agrawal, First and second order statistics features for classification of magnetic resonance brain images, *J. Signal Inf. Process.*, **3** (2012), 146–153. https://doi.org/10.4236/jsip.2012.32019

32. A. Pinhero, M. L. Anupama, P. Vinod, C. A. Visaggio, N. Aneesh, S. Abhijith, et al., Malware detection employed by visualization and deep neural network, *Comput. Secur.*, **105** (2021), 102247. https://doi.org/10.1016/j.cose.2021.102247

33. F. Shang, Y. Li, X. Deng, D. He, Android malware detection method based on naive bayes and permission correlation algorithm, *Cluster Comput.*, **21** (2018), 955–966. https://doi.org/10.1007/s10586-017-0981-6

34. O. S. Jannath Nisha, S. Mary Saira Bhanu, Permission-based android malware application detection using multi-layer perceptron, in *Intelligent Systems Design and Applications*, Springer, Cham, (2018), 362–371. https://doi.org/10.1007/978-3-030-16660-1_36

35. M. Wadkar, F. Di Troia, M. Stamp, Detecting malware evolution using support vector machines, *Expert Syst. Appl.*, **143** (2020), 113022. https://doi.org/10.1016/j.eswa.2019.113022

36. H. J. Zhu, T. H. Jiang, B. Ma, Z. H. You, W. L. Shi, L. Cheng, Hemd: a highly efficient random forest-based malware detection framework for android, *Neural Comput. Appl.*, **30** (2018), 3353–3361. https://doi.org/10.1007/s00521-017-2914-y

37. R. Ali, A. Ali, F. Iqbal, M. Hussain, F. Ullah, Deep learning methods for malware and intrusion detection: A systematic literature review, *Secur. Commun. Netw.*, **2022** (2022), 2959222. https://doi.org/10.1155/2022/2959222

38. M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, H. Jeon, Cnn-based android malware detection, in *2017 international conference on software security and assurance (ICSSA)*, (2017), 60–65. https://doi.org/10.1109/ICSSA.2017.18

39. H. Cai, N. Meng, B. Ryder, D. Yao, Droidcat: Effective android malware detection and categorization via app-level profiling, *IEEE T. Inf. Foren. Sec.*, **14** (2018), 1455–1470. https://doi.org/10.1109/TIFS.2018.2879302

40. Y. Wang, Z. Liu, J. Xu, W. Yan, Heterogeneous network representation learning approach for ethereum identity identification, *IEEE T. Comput. Soc. Sy.*, **10** (2023), 890–899. https://doi.org/10.1109/TCSS.2022.3164719

41. J. Zhao, Y. Lv, Q. Zeng, L. Wan, Online policy learning based output-feedback optimal control of continuous-time systems, *IEEE T. Circuits-II*, **71** (2022), 652–656. https://doi.org/10.1109/TCSII.2022.3211832

42. Haipeng Cai. Embracing mobile app evolution via continuous ecosystem mining and characterization, in *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, (2020), 31–35. https://doi.org/10.1145/3387905.3388612

43. H. Cai, B. Ryder, A longitudinal study of application structure and behaviors in android, *IEEE T. Software Eng.*, **47** (2020),2934–2955. https://doi.org/10.1109/TSE.2020.2975176

44. H. Cai, J. Jenkins, Towards sustainable android malware detection, in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, (2018), 350–351. https://doi.org/10.1145/3183440.3195004

45. H. Cai, Assessing and improving malware detection sustainability through app evolution studies, *ACM T. Softw. Eng. Meth.*, **29** (2020), 1–28. https://doi.org/10.1145/3371924