Detection of Android Malware App through Feature Extraction and Classification of Android Image

Mohd Abdul Rahim Khan[1,2]
Department of Computer Science[1],
College of Computer and Information Sciences,
Majmaah University, Majmaah, 11952, Saudi Arabia
Department of Computer Science and Engineering[2],
Lingaya's Vidyapeeth, Faridabad,
Haryana, India

Prof. Nand Kumar
Department of Computer Science and Engineering,
Lingaya's Vidyapeeth, Faridabad,
Haryana India

R C Tripathi
Department of Computer Science and Engineering,Teerthanker Mahaveer University, Moradabad

***Abstract*—Android apps have security risks due to rapid development in android devices. In the Android ecosystem, there are many challenges to detecting Android malware. Traditional techniques such as static, dynamic, and hybrid approach, most of the existing approaches require a high rate of human intervention to detect Android malware. Most of the current techniques have the most significant security challenges to detect Android malware, the inspection of Android Package Kit(APK) file structures, increased complexity, high processing power, more storage space, and much human intervention. This paper proposed Machine Learning(ML)based algorithms to detect Android malware apps through feature extraction and classification of grayscale images. In our proposed approach, convert most of the files of APK such multiDex, resources, certificate, and manifest files transform into a grayscale image, using the image algorithm to extract the local feature of the image. In the paper used different ML models to classify the local features with the help of multiple images of malware families. This approach deals with the obfuscation attack.it can hide in any files of APK. The proposed approach enhanced accuracy reached up to 96.86%, and computation time did not increase more than the existing techniques. The quality of that proposed worked; it has a high classification accuracy and less complexity validation loss.**

***Keywords*—Android malware; obfuscation attack machine learning; android application package (APK); android malware app; grayscale images**

I. Introduction

Android operating system (OS) is the most popular OS in the smart device ecosystem. Due to intelligence devices, every android user is very close to and dependent on Android Application Package (APK). In the present scenario, the android users sharing sensitive information, banking operation, e-shopping, locations information, the identity of the users, and privacy of data are also involved. In Android, device security is the biggest challenge and severe issue. A survey report of GDATA in 2019 [1] showed that 1,852,170 Android malware samples were detected in the first half of 2019. Here, data showed an android malware is detected every 8 seconds. The statistical report represents eight mobile infected by malicious out of ten Android devices [2]. One more research report, Google detected 86% of the total Android devices market in 2017. The most popular GlobalStats website showed that 73% of Android-based devices counted sales of total devices of Android in 2019 [3]. Due to the popularity of Android devices, Android app becomes more targeting apps compared to other kinds of apps. As per one evolution report of mobile malware, 5,321,142 apps were installed on devices, 151,359 mobile apps were detected as Trojans, 60,176 were detected as mobile ransomware by Kaspersky 2018 [4].

Android users threaten by different types of malware families; some are distributed by Google Play stores, some type apps such as downloader, banker, and hidden ads [5]. Most of the extensive attacks pointing the Android OS. The hackers mainly focused on attacking games, banking, academics, e-shopping domain. However, this domain published many malicious apps, which have gaps between the app development and the number of works. Third-party stores have untrusted apps; most gaming apps have adware due to the repackaging technique [6]; with the help of repackaging tools, reassemble the original app and add the malicious code with the original code and then assemble again, upload on third party store. Here, the main challenging task to identify malicious apps is the most severe issue. Most existing techniques are static and dynamic; most techniques used behavior and signature base to identify the android malware.

Static techniques do not require running apps; they disassemble the code and extract the feature of apps to identify. The dynamic approach always needs to run the application and identify the android malware through behavior and signature base. These techniques have significant drawbacks; it requires more computing power, resources, and space [7]. The dynamic analysis was evaded by some powerful and intelligent malware [8]. Moreover, existing dynamic and static techniques used the manual intervention of humans. It also needs domain expertise to identify reverse engineering [9]. The existing approach used single classes.dex file, but in the current scenario, we have multiclass files, or multiDex files [10], which have not been converted into a grayscale image to detect malware.

Our proposed work takes care of all essential files such as multiDex (MD), resources.ARSC(RS), Manifest.xml(MX), and certificate (CR) files of APK to detect Android malware. The existing approaches need human intervention to separate the dex, meianfest.xml resource.ARSC(RS) and certificate files convert into the grayscale image [11-12].

META-INF: This file very essential in Android apps, which the information about the signature and information

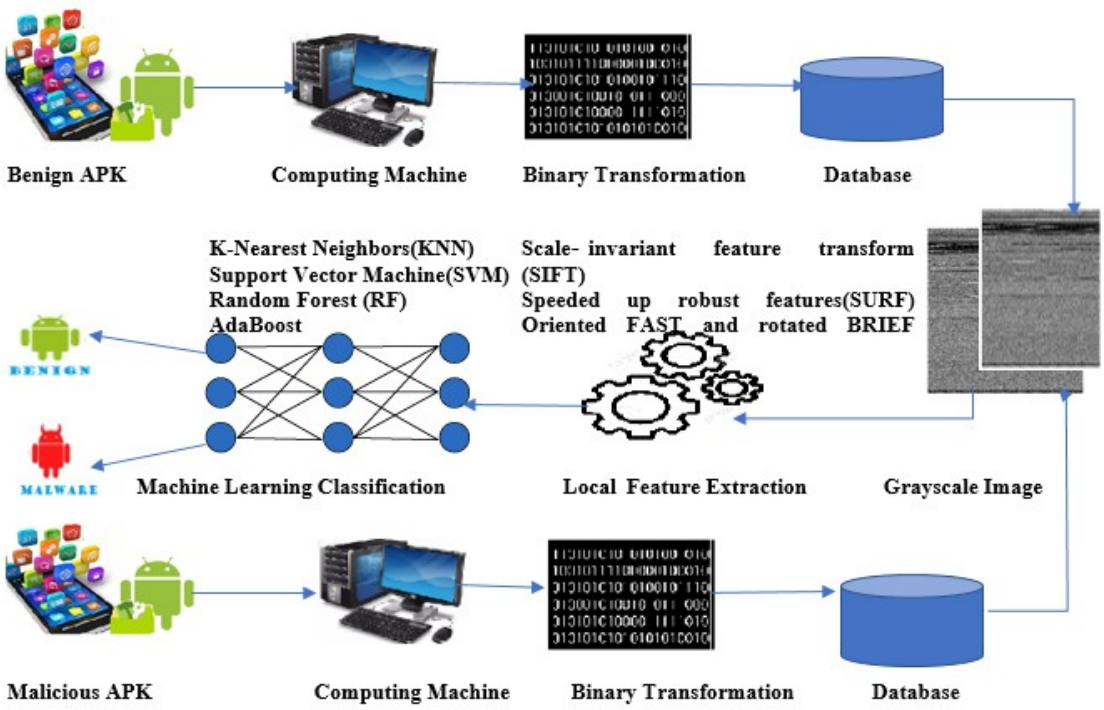

Fig. 1. Proposed Methodology to Detect the Android Malware

about the resource list.

Lib: lib file is used to run the specific device architectures of the native library, such as armeabiv7a and x86.

Res: to Keep the resources such as images. Which is not compiled with resources.arsc
Assets: Raw information about resources

AndroidManifest.xml: Meta information about the apps such as version, content, and name of APK files.

Multiple classes.dex: Main and necessary file of apps, which run java class methods on the devices.

Resources.arsc: Compiled all resources on the devices which is used by the apps.

Android apps development using java.class files. By the DX tools convert multiple java.class into the DEX files. DEX and manifest are essential files in APK, and DEX consists of the data structure; the interpreter used the different data types that belong to the data structure. All static reverse engineering tools used the DEX files to reassemble the apps for reengineering. Multiple methods are proposed to protect the DEX files. Our Proposed approach does not require any human intervention, does not require separate files, and does not need reverse engineering to find different types of files. Our proposed approach takes less computation power to detect the android malware because it takes less time complexity because it worked without any reverse engineering operation. Our proposed approach used DEBIAN and AMD datasets containing 10560 apps (5000 benign and 5560 malicious apps). The grayscale image datasets, each containing 10560 samples (5000, 5560 benign, malicious samples, respectively), were constructed based on diverse files from the contents of the APK collections. Firstly, all the benign and malicious APK convert into Grayscale images, a block diagram depicted in Fig. 2. Secondly, extract the local features from images using image-based feature extraction techniques such as SIFT, SURF, and ORB. Thirdly, apply the BOVW approach to convert multiple local feature descriptor vectors into a single feature vector to feed into ML classifiers. Finally, extract the global and local features and apply the different ML classifier techniques such as AdaBoost K-Nearest Neighbors(KNN), Support Vector Machine(SVM), and Random Forest(RF). The Proposed approach worked on the raw bytes of grayscale images; the main advantage of this approach does not require any reengineering operation and making different types of datasets. The existing approaches have the main disadvantage, approaches that require human intervention. Our approach proposed safe from human intervention and reengineering operation. Many ML algorithms are developed for the detection of malware apps. The most common challenge in Android malware detection is obfuscation attacks. Malicious code can be hidden in any files of APK, which is very dangerous to android malware app detection. Our proposed works have a novelty that now no needs to do reverse engineering to obtain all files of APK. Directly conversion of Entire APK files structure converts into a grayscale image. Most of the existing techniques used separate files to transform into grayscale scale images to analyze the image-based android malware detection. All existing methods do not care about multiple DEX, Share Object (SO), Meta-Inf, lib files, etc., just observation of manifest, single DEX files, resource files only. In the meantime, the author should explain the functions of multiDEX (MD). Resources, ARSC (RS), Manifest.xml (MX), and certificate (CR) files of APK separately because they are used to detect Android malware. Then, the proposed methodology to detect the android malware

is well represented in Fig. 1.

II. Related Work

Many researchers worked in the domain of Android malware detection; some are listed below in this section. An approach designed to analyze the suspicious behaviors and detection of resources abuse [13]. The major drawback of this approach is the need to decompile the app and embedded hook code; this approach used runtime events to track and monitor the logging. The SafeDroid static framework approached, which statically analyzed the DEX (Dalvik Executable). By this approach, extract the binary feature vectors to train various ML classifiers [14]. Moreover, the multiple features are system calls, app permission, system events [15]. Those features train RF classifiers to analyze whether Apps are affected by malicious or not. Some approaches differentiate whether the app is a malicious or normal app based on patterns permission [16], the required permissions extracted statically. Most popular permissions are registered into class [17] to define whether the permission is benign or malicious. The permissions of a class determine the benign and malicious app. Moreover, a data mining technique made the constructive pattern of permission to determine whether the android app is malicious or benign. Here, the authors applied the bi-clustering method to used permissions. Also, the authors used the information of the Android app package and permissions to train of KNN, Linear Discrimination(LD) function, and Radial Basis Function (RBF) network. Moreover, Application Programming Interface(API) system calls integrated with permissions [18] are used as features to train the RF classifier of android's apps classification. It is a very lightweight method for detecting Android malware through ML and dataflow-related API system calls used in this approach [19]. In [20], the proposed approach used the n-gram series to extract the features from the opcode of malicious and benign apps. This approach used a limited number of features to train RF and Support Vector Machines (SVM) classifiers. The proposed approach [21] installed the Android application(APK) on Android devices to extract dynamic features such as networks behavior, memory consumption, computation power, time-space, battery, and binder; these features are used to classify malware. This dynamic approach [22] captured network traffic behaviors of running Android applications(APK) from different android devices. This traffic correlates with malware URLs and with DNS service network traffic for the detection of malware. An approach [23] used to fog computing reduces the load and dynamically enhances the computation power to detect Android malware. Another approach [24] used the API system calls and network behaviors, collectively applied to detect Android malware. In [25] this paper, the authors showed the multiple network behavior and emulator-based dynamic experiments to analyze android malware. Android operating system embedded by an extension kit has been proposed [26] to deals with confused delegate attacks [a genuine APK is manipulated for communicating with the trusted application for Inter-Process Communication(IPC)]. To enhance permissions-based policy [27]at runtime tracking and communication link analysis by pre-defined policy to prevent malicious behavior. Moreover, the signature set is constructed by network log and correlated with the permissions-based methods [28] for android apps classification. The recent approach [29] uses reverse engineering techniques to decompile the APK, extract the source code, and convert it into a grayscale image. The constructed dataset of images is used to train a convolutional neural network (CNN) to detect the malicious app. API system calls and semantic information is used to train the Short-Term Long Memory (LSTM)[30] model to classify the android malware. Moreover, a hybrid approach includes CNN and deep autoencoder (DAE) [31] to detect Android malware. In [32], this proposed approach used the hybrid scheme; it extracts dynamic and static behavior features used to train the deep learning model. Also, in [33] approach extracted the four features, such as permissions, rate of permission, system events, APIs system calls used to train the collective RF classifier. DREBIN [34] is a static analysis approach; this approach used similar malicious apps as per experiment works (5,560 malicious apps). This method used as many possible features of apps and was added with joint vector space. Due maximum number of features and determination increased the complexity level. This paper [35] proposed the classification of the dependency graph. The features extracted from the dependency graph make the semantic feature set from the weighted contextual API of the graph. The metric of the homogeneous app determines same the application behaviors The sensitive and important API call allocated the weight according to the Android malware family [36]. Every app implemented the function call graph (FCG), and each FCG construct the sensitive API call-related graph (SARG). The SARG has the parent and sensitive API call nodes. Here, train multiple machine learning approaches to classify the common behavior of the malware family. Moreover, from source code is extracted from hexadecimal representation and converted into RGB images [37]. The color RGB dataset is used to train a CNN classifier to classify Android malware. Furthermore, the Android (APK) application converted to grayscale images, then extract the feature of grayscale images for training the RF classifier for classification in [38]. Also, in [39], extracted the feature from 2D of Opcode Sequences and assigned the weight based on their occurrence. The weight value is converted into grayscale images. The image detection approach is very limited to detecting Android malware domains. Local and global feature extraction of the entire APK is more effective than existing approaches. Our paper has mainly converted the image into grayscale without any reverse engineering tools. It does not require separating the files of APK such as resource, Multidex, manifest, and certificate. Moreover, it does not require human intervention; most existing techniques have a common issue of human intervention and extracting the source code from reverse engineering tools.

III. Methodology

This section discussed the full detail of the proposed model. The first subsection briefly describes constructed dataset, the other section described the brief details of extracted features, and the last section briefly describes the training Machine Learning (ML) classification.

1) Dataset: this dataset. Our experiment setup used the 5560 files of android malware and 5000 benign apps from AMD, which have 179 different Android malware families. In the investigation of research of android malware from 2012 to 2020, most of the researchers used the DREBIN dataset. The DREBIN dataset has the most famous malware such as

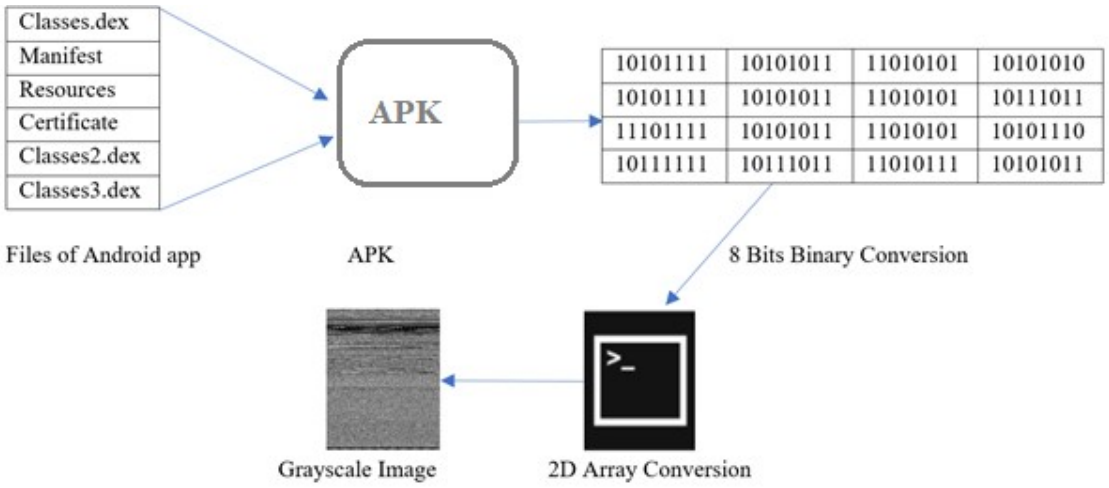

Fig. 2. APK to Image Conversion Process

DroidKungFu, GingerMaster, GoldDream, and Fake Installer. The primary objective of our proposed approach is to detect Android malware. The DREBIN dataset has the most famous malware such as DroidKungFu, GingerMaster, GoldDream, and Fake Installer. The primary objective of our proposed approach is to detect Android malware. Many researchers used the DREBIN dataset to analyze the android malware, and various institutions utilized this dataset to investigate Android malware. Our Proposed models used the DREBIN dataset because it has 179 different android malware families, appropriate for any investigation dataset.

A. Transformation APK into Grayscale Images

The Android APK files convert into grayscale images [40]. In this proposed article, the authors construct the malware images using files of the Android app from malware APK. The APK is transformed into 8-bit vectors, and then the 8-bit vector transforms into a grayscale image. Every substring has an 8-bit value as a pixel converted into a decimal value between 0-255, shown in Fig. 2. Any digital file on the memory device is stored as a stream of a bit of '0' and' 1'. In the model read every APK file as a binary stream, group every eight bits, and store them in a new file with the image file extension.

B. Local Features Extraction

The local feature is a defined image object (basically, in the image, a cluster of pixels or small blobs) [41]. The local feature of images is the most stimulating point in the image, which defines the image descriptor vectors(DV) or feature vectors. The set of feature vectors is described by different types of algorithms. Our proposed approach used the four different algorithm types to extract the local features as Scale invariant feature transform(SIFT), Speeded up robust features (SURF), Oriented FAST, and Rotated BRIEF(ORB). Those methods are very famous in the malware domain for better accuracy.

1) SIFT: The Scale invariant feature transform (SIFT) method is applied to extract the local feature key points. This method computes the Laplacian of Gaussian on the multiple scale level to provide a better result. This SIFT algorithm obtains the local minima and local maxima of stimulating points with the help of LoG at different scale levels. The several Laplacian of Gaussian on different scale levels (ρ), by the scale, obtain the local maxima, minima of every single pixel in the image. The Laplacian of Gaussian (LoG) calculation is costly for feature points to more or less extent. The Laplacian of Gaussian(LoG) approximately is determined by Eq.1.

$$\rho\nabla^2 L = \frac{\partial L}{\partial \rho} = \frac{L(x, y, k\rho) - L(x, y, \rho)}{k\rho - \rho} \quad (1)$$

where L(x,y,ρ) is the Laplacian Gaussian on the position (x, y) at scale ρ. L(x,y,kσ) is the Laplacian of Gaussian(LoG) on the position (x, y) at scale kρ, and the kρ is a scale a little more than ρ. The SIFT methods identified the stimulating points at the level of 128-bit descriptors in Eq.1. The extracted feature from the input images through the SIFT matched each feature of k nearest neighbors. The main objective of SIFT is to object recognition techniques to panorama stitching. As a result, the system is insensitive to the images' ordering, positioning, scale, and illumination. Two-Dimension isotropic measure by the Laplacian to the second spatial derivative of an image. The Laplacian Gaussian approach highlights areas of speedy intensity change and is often used for zero-crossing edge detectors. In our system, the Gaussian smoothing filter reduces its sensitivity to noise for smoothing with something approximating.

2) SURF: The algorithm that Speeds up robust features(SURF) [42] is the faster algorithm, and it can be the replacement for SIFT. This algorithm is faster and more robust for similarity comparison and similarity invariant of images. SURF algorithm plays a vital role in the real type of tracking and recognition of the object. The main merit of this algorithm is box filters approximation and calculation of the integral images. Additionally, it has the location and scale-based determinant of the Hessian matrix. The Hessian matrix has good performance to obtain the image key points, and it has good accuracy. In the SURF algorithm filtered by Gaussian

kernel, with location X=(x,y), and scale ρ in Eq.2.

$$H(x,\ y) = \begin{pmatrix} Sxx\ (x,\ \rho) & Sxy\ (x,\ \rho) \\ Sxy\ (x,\rho) & Syy\ (x,\rho) \end{pmatrix} \quad (2)$$

where, Sxx (x,ρ) has a Gaussian kernel derivative on the point of x in the image, and similarly for Sxy(x,ρ) and Syy (x,ρ). Haar-wavelet responses determine horizontal and vertical paths to the neighborhood of size six and used the 64 Bit Descriptor. Within interest point neighborhood, distribution of Haar-wavelet responses obtained from descriptor description. We deed integral images to speed up the system. Additionally, using the 64 Bit Descriptor dimensions to improve the system's performance for feature computation increases robustness and matching. In the invariant to rotation, we recognized the reproducible orientation for the interest points. For this reason, we obtained the Haar-wavelet responses in the vertical and horizontal directions. The circular neighborhood of radius 6s around the interest points, with s the scale that the interesting point detected. Therefore, our proposed approach uses integral images for fast filtering again. Only six actions are needed to SURF: Speeded Up Robust Features, the seventh determines the feedback in the vertical and horizontal directions at any scale.

3) ORB: The feature vector Oriented FAST and rotated BRIEF (ORB) is a high-speed keypoint detector [43]; in BRIEF, descriptors have much modification to improve the algorithm performance. The ORB algorithm detects the keypoint in images by using the FAST algorithm. Also used the Harris corner to detect the key point. Moreover, it used the multiscale feature with 32 bits BRIEF-based descriptor

$$(S;x,\ y) = \begin{Bmatrix} 1 : S(x)\ <\ S(y) \\ 0 : S(x) \geq S(y) \end{Bmatrix} \quad (3)$$

where S is the flattened spot in the image, and S(x) is the intensity in Eq. 3. In the implementation of the FAST algorithm, we extract the kernel windows from single line buffers. In the approach, the center pixel is subtracted from each circle pixels. The result is measured with the minContrast value whenever the obligatory number of consecutive pixels exceeds the threshold level; the center is marked as the corner. For the circle region, evaluate the sum-of-absolute-difference (SAD) metric. Only the differences that exceed the minimum contrast threshold level are involved in the metric. This calculation means that the algorithm detects a light center pixel surrounded by dark pixels or a dark center pixel surrounded by light pixels as corners with high metrics. The Harris algorithm used five image filters, and three circular windows and evaluated the two gradients. The design of the calculation of the eigenvalue of the Harris matrix practices three multipliers and three adders and is pipelined to optimize performance.

C. Machine Learning (ML) Classification

Our proposed models used four types of Machine Learning models such as Adaboost, K-Nearest Neighbors(KNN), Support Vector Machine(SVM), and Random Forest(RF) to classify the extracted local features from Grayscale images.

1) K-Nearest Neighbors (KNN): K-Nearest Neighbors(KNN) is a supervised ML models, which is used for the classification of input data. It recognizes data points classified into multiple classes and calculates the class label for the new input data point. This method is famous for classifying the object into the train closest feature space. The nearest neighbors are signified by K in KNN, and the maximum unknown data points classify near to K neighbors. The primary benefit of the KNN algorithm uses the minimum distance to search the nearest neighbors. The selection of the number of nearest neighbors is essential to obtain the augmented KNN model. The selection of the number of nearest neighbors is essential to get the augmented KNN model.

2) Support Vector Machine (SVM): Support Vector Machine(SVM) also is a supervised ML algorithm. In this model, take the past input data and predict the feature output. The primary purpose of SVM is classification, but it is also used for regression statements. The SVM algorithm chooses the support vectors in the dataset at the extreme points. It selects the maximum distance between the support vector and hyperplane as much as possible. A class in support vectors has the maximum distance from the hyperplane. The distance margin defines as the distance between different support vector classes. The sum of D+ and D- is calculated as distance margin, where D-, hyperplane has the minimum distance from the closest negative point and D+, hyperplane has the minimum distance from the closest positive point. The main aim of SVM is to find the maximum distance margin, which gives the optimal hyperplane. The optimal hyperplane always gives excellent classification. In the case of non-linear, which produces low and no distance margin, SVM showed misclassification. In that scenario, SVM used the kernel functions to convert the non-linear data into 2D or 3 D dimension arrays. The minor dimensional feature is converted into high dimensional feature space by the kernel functions.

3) Random Forest (RF): Random Forest (RF) is one of the most common and powerful supervised ML algorithms. RF executes efficiently massive datasets and predicts accurate results. This algorithm support both types of functionality, such as classification and regression—the decision tree support RF to enhance the accuracy and flexibility. In general, with more trees in the forest, the output would be more predictable. The more trees in the RF reduce the risk of when a statistical model fits exactly against its training data. RF can obtain good accuracy in case of missing a large proportion. According to attributes, the new object classifies, and the decision tree gives the classification output per the ruleset.

4) AdaBoost: The first boosting algorithm is AdaBoost, which solved multiple problems. The AdaBoost constructs a robust classifier from multiple week classifiers. This algorithm keeps a single split of the decisions tree with the week stump, known as the decision stump. AdaBoost always keeps more load on tough to classify, easy to handle the problem, and do less. This algorithm has solved both types of problems, such as classification and regression. Multiple APK's are repackaged, which steal code by reverse engineering methods and reassemble with another name by adding adware or small scripts of malicious code into repacked APK. Here the APK has very slightly changed, so the dataset has slight noise in

data We found that in the case of less noisy data, only a few hyperparameters need to be tuned to improve the Adaboost performance. In the case of the small number of input variables, KNN models provide excellent performance. Whenever we increase more number of input variables, the performance of KNN degrades. In our dataset, we used multiple DEX files based on APKs. All files structure of APK were converted into grayscale images, which increased the number of input variables and memory size and complexity of the KNN model.

IV. Proposed Models

In our work, we proposed an image-based detection of android malware using machine learning classification. In this process, Android APK converts into a grayscale image, extracts the image feature using image processing techniques, and trains the machine learning classification to detect malicious or benign apps depicted in Fig. 1. The novelty of this approach the entire files of APK transforms into images to deal with the obfuscation attack. Most of the existing techniques used only three files of APK to transform into the image. The main disadvantage of the other techniques requires decompiling the APK and separating the files such as DEX (MD), ARSC(RS), Manifest.xml(MX), and certificate files. Moreover, the disadvantage is that it does not take care of the mutltiDEX files. If APK has more than 6500 methods in the app, it needs to create the multiDex files [40]. If the malicious code is embedded with second or third classes.dex files, no existing algorithm detects the Android malware app from multiDex class files. The primary source of the malicious code is embedded into classes.dex files. Our models used three algorithms (SURF, ORB, and SIFT) to extract local features (LF) descriptors from the grayscale image dataset. One by one, local features (Extracted from each image) train to multiple machine learning algorithms (RF, KNN, DT, and AdaBoost). The multiple descriptors represent an image. Above mentioned machine learning algorithm gave the multiple vectors as outputs, which cannot be direct as inputs for any machine learning algorithm. This model used the Bag of Visual Words(BOVW) to create one feature vector with multiple local feature descriptors [41]. The BOVW uses any clustering techniques to fragment the extracted descriptors vectors into multiple clusters. Then the cluster is predicted by the clustering algorithm.

A. Accuracy Assessment

The accuracy metrics for multiple Machine Learning models were determined based on the precision, recall, f1-score, and accuracy in Eq. 4, 5, 6 , 7 respectively , and precision in fraction of data entries of malicious activity are categorized as truly Android malware.

$$\text{Precision} = \frac{True\ Possitive}{True\ Possitive + False\ Possitive} \quad (4)$$

The recall is the fraction of malicious apps data of correctly classified malicious families.

$$\text{Recall} = \frac{True\ Possitive}{True\ Possitive + False\ Negative} \quad (5)$$

TABLE I. Obtained Accuracy, Precision, Recall and F1-Score from Multiple Machine Learning Models using Multiple Local Features Extractor Methods

		Machine Learning Methods			
Performance Evaluator	Feature Vectors Methods	K-Nearest Neighbors	Support Vector Machine	Random Forest	AdaBoost
Accuracy	SIFT	92.42%	94.06%	94.65%	93.16%
	SURF	94.69%	95.37%	96.33%	96.86%
	ORB	89.41%	89.83%	91.42%	92.83%
Precision	SIFT	94.48%	91.00%	95.11%	95.71%
	SURF	95.79%	94.63%	97.44%	97.41%
	ORB	90.46%	89.13%	92.48%	93.36%
Recall	SIFT	91.31%	94.97%	93.42%	92.66%
	SURF	93.47%	96.29%	95.09%	96.35%
	ORB	88.33%	90.70%	90.24%	92.34%
F1-Score	SIFT	92.40%	94.14%	94.57%	93.18%
	SURF	94.71%	95.45%	96.25%	96.88%
	ORB	89.39%	89.91%	91.34%	92.85%

f1- score is the harmonic mean between sensitivity and precision.

$$\text{f1} - \text{score} = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (6)$$

The accuracy or complete classification accuracy is the portion of all suitably classified negative and positive records with the losses.

$$\text{Accuracy} = \frac{TN + TP}{TN + FN + TP + FP} \quad (7)$$

$$\text{Cross-EntropyLoss} = -\frac{1}{N} \sum_{o=1}^{n} log\ p_{\ model}(y_o \epsilon\ Cy_o) \quad (8)$$

V. Experiment

The experiment has been designed on Intel core™ i-7 10700 CPU @ 3.8 GHz with 16 GB RAM. The experiment used the BOVW algorithm, which needs a size 120 codewords vocabulary. The K-means technique collected all key points from created datasets, and it has the codeword vocabulary size 120. Moreover, the proposed model used Opencv, Sklearn python libraries for the implementation of laboratory works.

The performance of different Machine Learning models has been achieved in terms of the whole percentage of true positive, true negative, false positive, and false-negative decisions. Our local features extractor models, such as SIFT, SURF, and ORB, extract key points from the image dataset. The extracted local feature passed to train four renowned machine learning models, i.e., K-Nearest Neighbors(KNN), Support Vector Machine (SVM), Random forest, and AdaBoost. The complete result with multiple ML models and local feature extractor models is presented in Table I and shown in Fig. 3. The validation set of the accuracy and losses in our proposed works proves that the results are correct, not overfitting problems depicted in Fig. 4. The high accuracies, precision, recall, and Fi-score from different machine learning models are displayed in Table II and Fig. 3. if the losses decrement and accuracy developments of both groups are like the same, then the process aborted changed the modeling parameters to remove the overfitting problem. Last, the AdaBoost model accuracy touched 96.86%. The traditional machine-learning algorithm shows the performance of each algorithm in Table III and is

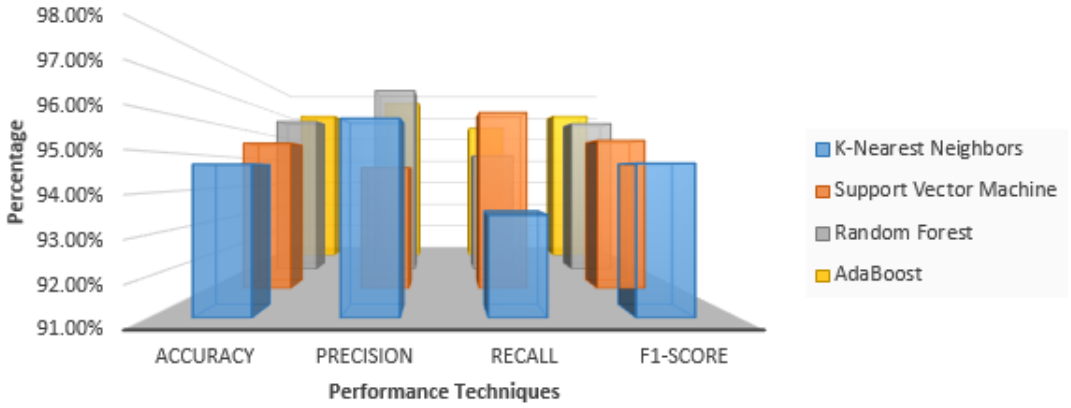

Fig. 3. Performance of Multiple Machine Learning Models

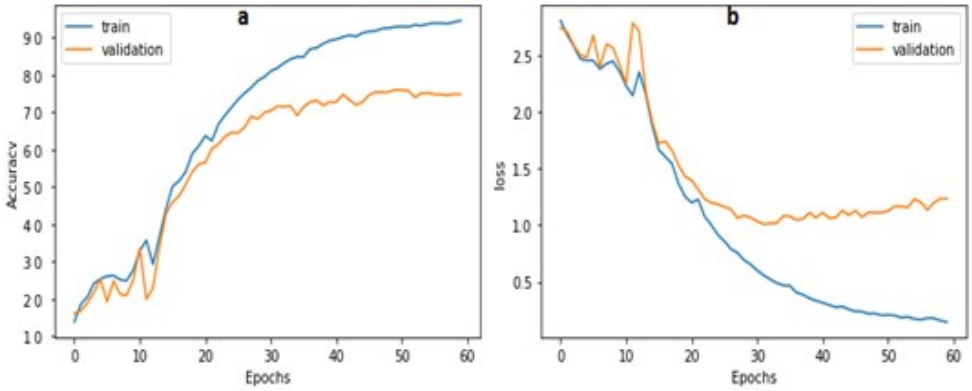

Fig. 4. (a): Train and Validation of AdaBoost Accuracy; (b): Train and Validation Loss of Model using SURF Local Feature

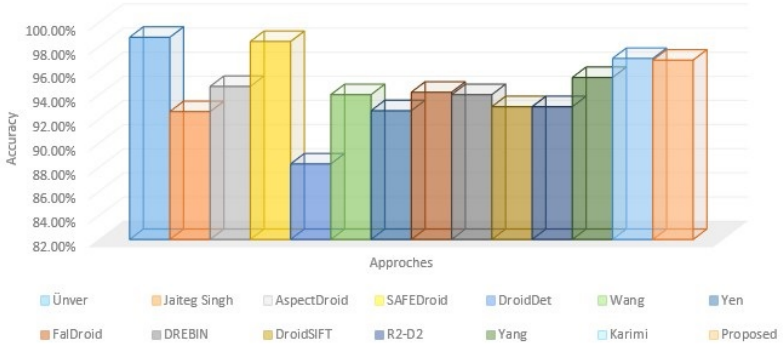

Fig. 5. Comparison of different Existing Approaches

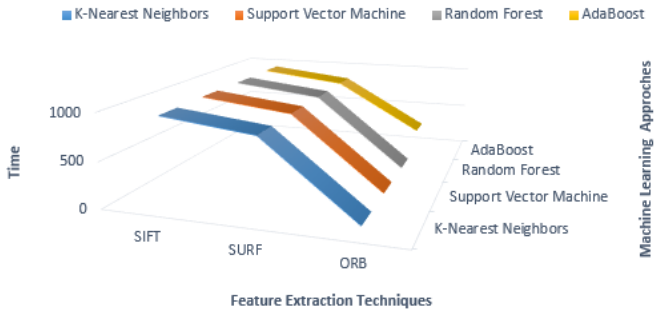

Fig. 6. Execution Time of the Proposed Model with Feature Extraction

TABLE II. PROPOSED MULTIPLE MACHINE LEARNING MODELS WITH ACCURACY, PRECISION, RECALL AND F1-SCORE

ML Method	Accuracy	Precision	Recall	F1-Score
K-Nearest Neighbors	94.69%	95.79%	93.47%	94.71%
Support Vector Machine	95.37%	94.63%	96.29%	95.45%
Random Forest	96.33%	97.44%	95.09%	96.25%
AdaBoost	96.86%	97.41%	96.35%	96.86%

TABLE III. COMPARISON OF DIFFERENT EXISTING APPROACHES

Model	Accuracy	Types
Ünver [11]	98.75%	Image-based
Jaiteg Singh [12].	92.59%	Image-based
AspectDroid[13]	94.68%	Hybrid analysis
SAFEDroid[14]	98.40%	Static analysis
DroidDet[15]	88.26%	Static analysis
Wang[16]	94.00%	Static analysis
Yen[23]	92.67%	Image-based
FalDroid[33]	94.20%	Static analysis
DREBIN[34]	94.00%	Static analysis
DroidSIFT[35]	93.00%	Static analysis
R2-D2[36]	93.00%	Image-based
Yang[37]	95.42%	Image-based
Karimi[38]	97.00%	Image-based
Proposed	**96.86%**	**Image-based**

depicted in Fig. 4. The accuracy value maximum 96.88% in the AdaBoost model, with losses, using SURF local feature extraction models in Fig. 4. In our work, computation time is scanned in each step of all ML models, inclusive of the feature extraction process, training, and testing of the model. The computation time from different ML models, the feature extraction process, training, and validation are presented in Table IV and Fig. 6.

VI. CONCLUSION

Our proposed model uses an image-based framework to classify the android app, whether malicious or benign app. Most image-based detection techniques do not care about the multiDex files of APK, using only a single class.DEX file for image conversion. In existing techniques of image-based malware detection found [11-12,23,36,37] the maximum detection probability of malware in classes.DEX file, not in another file such as Resources.ARSC, Manifest.xml, certificate files, if the hackers hide malicious code into second or third classes.DEX files, there is no chance to detect the malware in previous approaches. In our experimental works, transform

TABLE IV. THE EXECUTION TIME OF THE PROPOSED MODEL

Feature Vectors Methods	Machine Learning Models			
	K-Nearest Neighbors	Support Vector Machine	Random Forest	AdaBoost
SIFT	941.63	945.87	941.03	943.91
SURF	827.00	830.64	826.72	828.91
ORB	43.36	44.87	44.10	266.43

all classes.DEX APK file's contents into grayscale images. We used the image processing techniques to extract the local feature of images, including SIFT, SURF, and ORB models. The Local features are classified using machine learning models (KNN, SVM, RF, and AdaBoost) to detect Android malware. The achieved results exhibited that the proposed approach overtakes the existing techniques in classification accuracy and computational time. Our work showed that the AdaBoost detection rate reached up to 96.86 %, shown in Fig. 5, and run time did not exceed 0.0195 s on average for each sample. In the future, we will try to use the local and global features of images on multiDEX files to classify the Android malware to improve accuracy.

ACKNOWLEDGMENT

Mohd Abdul Rahim Khan would like to thank the Deanship of Scientific Research at Majmaah University for supporting this work under Project No. R-2022-162

REFERENCES

[1] G Data, Bochum, Germany, Tech. Rep. [Online] Access date 5/5/2021. Available: https://www.gdatasoftware.com/mobile-internet-security-android.

[2] Gartner (2018) Gartner says worldwide sales of smartphones recorded firstever decline during the fourth quarter of 2017.https ://www.gartner.com/en/newsr oom/press -relea ses/2018- 02-22-gartn er-says-world wide-sales -of-smart phone s-recor dedfirst-ever-decli ne-durin g-the-fourt h-quart er-of-2017. Accessed 27 Oct 2019.

[3] StatcounterGlobalStats (2020) Mobile operating system market share worldwide. Mobile Operating System Market Share Worldwide https: //gs.statco unte r.com/os-market-share /mobile /worldwide. Accessed 09 Mar 2020.

[4] SecureList (2018) Mobile malware evolution 2018. https ://securelist .com/mobil e-malwa re-evolu tion-2018/89689 /. Accessed 27 Oct-2019.

[5] DoctorWeb (2019) Doctor Web's overview of malware detected on mobile devices in September 2019." https ://news.drweb.com/show/revie w/?i=13446 . Accessed 27 Oct 2019.

[6] M.R. Khan, R.C. Tripathi, and A. Kumar, Repacked android application detection using image similarity, *Nexo Revista Científica*, vol 33, no.1, pp.190-199, june, 2020.

[7] W. Wang, M. Zhao, Z. Gao, Xu, G., H. Xian, Li, Y. and X. Zhang, Constructing features for detecting android malicious applications: issues, taxonomy and directions. *IEEE acces*s, vol.7, no. 2019, pp.67602-67631, June, 2019.

[8] M.K.Alzaylaee, S.Y. Yerima, and S. Sezer, Emulator vs real phone: Android malware detection using machine learning. *In Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics*, Scottsdale, Arizona USA , 2017, pp. 65-72.

[9] B. Jung, T.Kim, and E.G Im, October. Malware classification using byte sequence information., *In Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, Honolulu ,Hawaii ,2018, pp. 143-148.

[10] M.R. Khan, and M.K Jain, Protection android app with multiDEX and SO files from reverse engineering, *Materials Today: Proceedings*, vol. 2021, no. 1, pp. 1-9, January ,2021.

[11] H.M. Ünver, and K. Bakour, Android malware detection based on image-based features and machine learning techniques, *SN Applied Sciences*, vol 2, no.7, pp.1-15, june, 2020.

[12] Singh, J., Thakur, D., Ali, F., Gera, T., & Kwak, K. S. Deep feature extraction and classification of android malware images. Sensors, vol 20, no. 24, p.7013, Jan, 2020.

[13] A.I. Ali-Gombe, B. Saltaformaggio, D. Xu, and Richard III, G.G., Toward a more dependable hybrid analysis of android malware using aspect-oriented programming, computers *& security*, vol. 73, no. 2018, pp.235-248, Mar, 2018.

[14] R. Goyal, A. Spognardi, N. Dragoniand and M.Argyriou, SafeDroid: a distributed malware detection service for android. *In 2016 IEEE 9th international conference on service-oriented computing and application*s (SOCA), Macau, China 2016, pp. 59-66.

[15] H. J . Zhu, Z. H . You, Z. X. Zhu, W. L. Shi, X Chen, & , L. Cheng , DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model, *Neurocomputing*, vol 272,no. 2018, pp. 638-646, 2018.

[16] C.Wang, Q. Xu, , X.Lin, , & S. Liu, Research on data mining of permissions mode for Android malware detection, *Cluster Computing*, vol. 22 , no. 6, pp. 13337-13350, Nov, 2019.

[17] V. Moonsamy, J. Rong, S. Liu , Mining permission patterns for contrasting clean and malicious android applications, *Future Generation Computer Systems*, vol 36, pp. 122–132, Jul, 2014.

[18] G.Tao, Z. Zheng, , Z. Guo, , & M. R. Lyu, MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs, *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 355-369,Dec, 2017.

[19] M. Turner, B.Kitchenham, , P. Brereton, , S. Charters, , & D. Budgen, Does the technology acceptance model predict actual use? A systematic literature review, *Information and software technology*, vol 52, no 5, pp. 463-479, May ,2010.

[20] G.Canfora, A. De Lorenzo , E. Medvet, F. Mercaldo, & C. A. Visaggio, Effectiveness of opcode ngrams for detection of multi family android malware. *In 2015 10th International Conference on Availability, Reliability and Security (IEEE),* Toulouse, France, 2015, pp. 333-340.

[21] H. Papadopoulos, , N. Georgiou, , C. Eliades, , & A. Konstantinidis, Android malware detection with unbiased confidence guarantees, *Neurocomputing*, vol. 280,no. 2018, pp.3-12, Mar ,2018.

[22] O. Somarriba, and U. Zurutuza, A collaborative framework for android malware detection using DNS & dynamic analysis. *In 2017 IEEE 37th Central America and Panama Convention (CONCAPAN XXXVII)(IEEE)* , Managua, Nicaragua , 2017,pp. 1-6.

[23] M.R. Khan, S. Dubey, and R.C. Tripathi, Network Traffic Based Detection of Repackaged Android Apps via Mobile Fog Computing, *International Journal of Future Generation Communication and Networking*, vol 14, no. 1, pp.2824-2838, May , 2021.

[24] F. Tong, and Z. Yan, A hybrid approach of mobile malware detection in Android, Journal *of Parallel and Distributed computing*, vol. 103, no. 2017, pp.22-31, May ,2017.

[25] M.K. Alzaylaee, S.Y. Yerima, and S. Sezer, Emulator vs real phone: Android malware detection using machine learning. *In Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics*, Scottsdale, Arizona USA ,2017, pp. 65-72.

[26] M. Dietz, , S. Shekhar, Y.Pisetsky, A. Shu, and D.S. Wallach, Quire: Lightweight provenance for smart phone operating systems. *In USENIX security symposium*, Vol. 31, no. 2011, p. 3, August , 2011.

[27] S. Bugiel, L.Davi, A. Dmitrienko, T. Fischer, and A.R. Sadeghi, Xmandroid: A new android evolution to mitigate privilege escalation attacks. Technische Universität Darmstadt, *Technical Report* , TR-2011-04, Apr, 2011.

[28] A.T. Kabakus, and I.A. Dogru, an in-depth analysis of Android malware using hybrid techniques, *Digital Investigation*, vol. 24, no. 2018, pp.25-33, Mar,2018.

[29] M.R. Khan, and M.K. Jain, A novel technique for detecting repacked android applications using constant key point selection-based hashing and limited binary pattern texture feature extraction, *Journal of Ambient Intelligence and Humanized Computing*,vol 12, no 2021, pp.1-12, Mar, 2021.

[30] Z. Yuan, Y. Lu, and Y. Xue, Droiddetector: android malware characterization and detection using deep learning, *Tsinghua Science and Technology*, vol. 21 no. 1, pp.114-123, Feb, 2016.

[31] W. Wang, M. Zhao, and J. Wang, Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network, *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp.3035-3043, Aug, 2019.

[32] H.J. Zhu, T.H., Jiang, B. Ma, Z.H. You, W.L. Shi, and L. Cheng, HEMD: a highly efficient random forest-based malware detection framework for Android, *Neural Computing and Applications*, vol 30 no. 11, pp.3353-3361, Dec, 2018.

[33] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, Android malware familial classification and representative sample selection via frequent subgraph analysis, *IEEE Transactions on Information Forensics and Security*, vol 13, no. 8, pp.1890-1905, Feb, 2018.

[34] D. Arp, M.Spreitzenbarth,M. Hubner, H. Gascon, K. Rieck, and C.E.R.T. Siemens, Drebin: Effective and explainable detection of android malware in your pocket. *In Ndss*, Vol. 14,no. 2014, pp. 23-26, Feb, 2014.

[35] M. Zhang, Y.Duan, H. Yin, and Z. Zhao, Semantics-aware android malware classification using weighted contextual api dependency graphs. *In Proceedings of the 2014 ACM SIGSAC conference on computer and communications securit*y, Scottsdale, Arizona USA,2014, pp. 1105-1116.

[36] T. Hsien-De Huang, and H.Y. Kao, R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections, *In 2018 IEEE International Conference on Big Data (Big Data)*, New York, USA, 2018, pp. 2633-2642.

[37] M. Yang, and Q. Wen, detecting android malware by applying classification techniques on images patterns, *In 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA*), Chengdu, China ,2017, pp. 344-347.

[38] A. Karimi, and M.H. Moattar, Android ransomware detection using reduced opcode sequence and image similarity, In *2017 7th International Conference on Computer and Knowledge Engineering (ICCKE)* , Mashhad, Iran., 2017, pp. 229-234.

[39] E. Salahat, and M. Qasaimeh, Recent advances in features extraction and description algorithms: A comprehensive survey. In 2017 IEEE international conference on industrial technology (ICIT) , Toronto, ON, Canada,2017 , pp. 1059-1063.

[40] H. Bay, T. Tuytelaars, and L. Van Gool, Surf: Speeded up robust features. *In European conference on computer vision*, Springer, Berlin, Heidelberg, 2006, pp. 404-417.

[41] E. Rosten, and T. Drummond, Machine learning for high-speed corner detection. In European conference on computer vision , Springer, Berlin, Heidelberg, 2006 , pp. 430-443.

[42] K. Lim, N.Y Kim, Y. Jeong, S.J. Cho, S. Han, and M. Park, Protecting Android Applications with Multiple DEX Files Against Static Reverse Engineering Attacks, *Intelligent Automation and Soft Computing*, vol. 25 , no. 1, pp.143-154, Mar, 2019.

[43] N. Ali, K.B. Bajwa, R. Sablatnig, S.A. Chatzichristofis, , Z. Iqbal, , M. Rashid, and H.A. Habib, A novel image retrieval based on visual words integration of SIFT and SURF, *PloS one*, vol. 11, no. 6, p. e0157428, Jun, 2016.