




Research Article

Android malware detection based on image-based features and machine learning techniques

Halil Murat Ünver¹ · Khaled Bakour¹ 

Received: 23 November 2019 / Accepted: 23 June 2020 / Published online: 29 June 2020
© Springer Nature Switzerland AG 2020

Abstract

In this paper, a malware classification model has been proposed for detecting malware samples in the Android environment. The proposed model is based on converting some files from the source of the Android applications into grayscale images. Some image-based local features and global features, including four different types of local features and three different types of global features, have been extracted from the constructed grayscale image datasets and used for training the proposed model. To the best of our knowledge, this type of features is used for the first time in the Android malware detection domain. Moreover, the bag of visual words algorithm has been used to construct one feature vector from the descriptors of the local feature extracted from each image. The extracted local and global features have been used for training multiple machine learning classifiers including Random forest, k-nearest neighbors, Decision Tree, Bagging, AdaBoost and Gradient Boost. The proposed method obtained a very high classification accuracy reached 98.75% with a typical computational time does not exceed 0.018 s for each sample. The results of the proposed model outperformed the results of all compared state-of-art models in term of both classification accuracy and computational time.

Keywords Android malware · Image local feature · Image global feature · Malware visualization

1 Introduction

The proliferation of mobile phones has made it an indispensable part of our daily lives. Most people are using these devices to access important services such as accessing banking applications or making e-shopping. So, naturally, these devices contain very sensitive data that the developers of malicious applications aspire to access. The statistical studies estimate that there are at least eight out of ten used mobile devices are based on the Android operating system. For example, due to Gartner reports [1], Google's Android extended its lead by capturing 86 per cent of the total mobile phones market in 2017. Also due to Statcounter GlobalStats statistical web site, the percentage of Android-based smartphones devices' sales

accounted for more than 73% of the total sales of smartphones for 2019 [2]. Therefore, naturally, the Android operating system became the most important target for the developers of malicious applications targeting smartphones. GData Internet security centre stated that 2,040,293 new malware samples were recorded in the first half of 2018 (almost 1.2 million of it have been discovered in the second quarter alone) with a 31 per cent increase over the second half of 2017 [3]. Also, according to SecureLIST's 2018 Mobile malware evolution report, 5,321,142 new malicious installation packages, 151,359 new mobile banking trojans, 60,176 new mobile ransomware Trojans have been detected by Kaspersky Lab products and technologies in 2018 [4]. Moreover, according to Doctor Web's overview about the smartphones'

Halit Bakır: Khaled Bakour's name can be written in two different ways due to his dual citizenship.

✉ Khaled Bakour, khaledbakour@kku.edu.tr; Halil Murat Ünver, unver@kku.edu.tr | ¹Department of Computer Engineering, Kırıkkale University, Kırıkkale, Turkey.



SN Applied Sciences (2020) 2:1299 | <https://doi.org/10.1007/s42452-020-3132-2>

malware detected in September 2019 [5], Android users were threatened by various malware, many of which were distributed via Google Play such as Android.DownLoader, Android.Banker and Android.HiddenAds.

Due to the widespread of the attacks targeting the Android mobile operating system the android's malware detection domain got big attention in the academic and commercial domains. But, although the big number of works conducted on this domain there is a gap between the achieved works and the huge number of malicious applications published on a daily base, which it has made even Google play store untrusted to some extent.

The main contributions of this paper can be concluded as follows:

- Three grayscale image datasets each of which contains 9700 samples (4850 benign samples and 4850 malware samples) have been constructed based on different files from the contents of the APK archives. The first dataset has been constructed by converting the Manifest.xml file of each android application into a grayscale image. The second image dataset has been constructed by converting the DEX byte code files of each android application into a grayscale image. The final dataset images have been constructed by converting the Manifest.xml, DEX, and Resource.ARSC files of each Android application into a grayscale image.
- Four different image-based local features, including SIFT, SURF, KAZE and ORB, and three different image-based global features, including Colour Histogram, Haralick Texture and Hu Moments, have been extracted and used to train multiple machines learning classifiers. To the best of our knowledge, this type of features is used for the first time in android malware detection domain.
- The bag of visual words (BOVW) algorithm has been used to obtain one feature vector from multiple local feature's descriptor vectors so that it can be fed to the machine learning classifiers.
- The extracted local and global features have been used to train six different machine learning classifiers including Random forest, K-nearest neighbors, Decision Tree, Bagging, AdaBoost and Gradient Boost.
- Moreover, the proposed method is generic, where any type of apps can be converted to images and used to train the proposed model.

2 Related works

As mentioned before, a big number of academic works have been conducted in the android malware detection domain, some of which will be listed in this section.

A hybrid instrumentation engine is designed by [6] in order to achieve dataflow analysis, detection of resource abuse, and analysing of suspicious behaviours. The proposed model depends on decompiling the app, injecting some hooks code sections, recompiling the app and dynamically executing it for tracking and logging the runtime events. SafeDroid, a static analysis based framework has been proposed in [7]. The proposed framework depends on analysing the DEX (Dalvik Executable) code statically to extract binary feature vectors which have been used for training multiple machine learning classifiers. Moreover, multiple features including permissions, sensitive APIs (Application program interfaces), system events and permission-rate have been used by [8] to train a random forest classifier to detect whether an Android App is malicious or not. A combination of the required and used permissions has been used in [9] to generate permission patterns in order to differentiate between normal and malicious apps. In [10], the required and used permissions have been extracted and analysed statistically. Then, the most popular permissions in each apps' class (whether benign or malware) have been determined and used for constructing patterns that have been used to distinguish between malicious and benign apps. Moreover, a bi-clustering method has been used for visualizing the required and used permissions, and a mining method has been proposed to produce contrastive permission patterns that can separate non-harmful and harmful apps. Also, the used permissions and app's package information have been used in [11] to train each of KNN (K-Nearest Neighbour), Linear Discrimination function and RBF Network. Moreover, the API calls have been correlated with the permissions by [12] and used as features for training and testing a random forests classifier used in android apps' classification. Furthermore, a lightweight method for Android malware detection based on machine learning and dataflow-related APIs has been proposed in [13]. In [14], the n-gram sequences have been extracted from the opcode of both benign and malware Android applications to generate reduced feature vectors used in training Support Vector Machines (SVM) and Random Forest (RF) classifiers. In [15], the APK files have been installed on a real Android smartphone and multiple dynamic features including Binder, Battery, Memory, CPU and Network behaviour have been collected and used in malware classification. A dynamic analysis framework has been proposed by [16] for classifying android apps based on monitoring the apps' runtime behaviour and malicious URLs and correlating them with DNS service network traffic in order to detect the malicious behaviour at the network level. In [17], the system calls and app's network access behaviour have been collected for constructing patterns set used in Android malware detection. Multiple experiments have been conducted

in [18] in order to compare between emulator-based and phone-based dynamic malware analysis. An extension kit for the Android operating system has been proposed by [19] to handle confused deputy attacks [i.e. a trusted application is manipulated by a malicious application through IPC (Inter-process Communication)]. The permissions framework of android system has been extended in [20] to perform a runtime monitoring and communication links analysing to prevent malicious behaviour based on a pre-defined policy. Moreover, the permissions have been used with network log data for constructing a signatures set used in android apps classification in [21]. Also, in [22], some static features have been used with some dynamic behaviour features to construct binary feature vectors used as input for training a deep learning model. Furthermore, in [23], the APK archives have been decompiled and the source code has been parsed for collecting and digitizing the importance of each word within it. After that, the digitized word values in each APK have been converted to an image. The constructed image dataset has been used for training a convolutional neural network used in android malware detection. Some semantic information from system call sequences has been considered by [24] to train a Long Short-Term Memory (LSTM) language model. A deep autoencoder (DAE) and convolutional neural network (CNN) based hybrid android malware detection model has been proposed by [25]. Also, four groups of feature including sensitive APIs, system events, permissions, and permission rate have been extracted by [26] and used to train an ensemble random forest classifier. In [27], each sensitive API call used in the android app has been assigned a weight to differentiate their corresponding importance according to the malware family, and a function call graph FCG has been constructed to represent each app. The constructed FCGs have been simplified into a sensitive API call-related graph (SARG) which contain only sensitive API call nodes and their parent nodes. Then, the common malicious behaviours shared by malware within the same family have been generated and used to train multiple machine learning classifiers. Moreover, the source code of android apps have been extracted by [28] and the hexadecimal representation of the instructions has been used for constructing the RGB image's three colour channels' values. The constructed RGB image dataset has been used to train a convolution neural network used as android apps classification model. Furthermore, some android APK archive contents have been converted to grayscale images and the GIST image features have been extracted from the images for training Random forests classifier in [29]. Also, in [30], the 2-g of Opcode Sequences have been weighted based on their frequency in android apps dataset. The constructed weight values have been converted to one grayscale image such that each value

has been converted to one pixel in the constructed image. The best pixel values have been selected for constructing a feature vector which has been used as a signature for ransomware detection.

The image processing-based malware detection techniques have been used limitedly to some extent in the android malware detection domain. And since image local and global features have been proven their efficiency in the image classification domain, in this work, it has been suggested testing this type of features in the android malware detection domain. Thus, this paper aims to test the effectiveness of image processing techniques in classifying android applications.

3 Materials and methodologies

In this section, the used materials and methodologies in addition to the proposed model will be discussed. At first, the constructed datasets, used features and other used techniques will be described briefly, then, the proposed model will be discussed in detail.

3.1 The constructed image datasets

In this work, it has been proposed constructing grayscale image datasets from different APK archive's contents in order to test the image processing techniques' efficiency in the android malware detection domain. To this end, the desired file has been read as a binary bitstream and stored as a sequence of bytes. Since the grayscale image has been used in this work, and since the grayscale image pixel values are ranged from 0 to 255, so the read byte sequence can be converted to pixels in the image. On other words, each byte in the bytes stream represents a pixel in the final image.

In this work, it has been constructed three images dataset each of which contains 9700 samples includes 4850 benign images and 4850 malicious images. The malware samples contain 4850 samples selected randomly from Drebin and Malgenom well-known android malware datasets, in addition to 4850 samples have been selected randomly from AMD android malware dataset. The Drebin android malware dataset [31] contains 5,560 Android malicious apps from 179 different families, and the Malgenom android malware dataset [32] contains 1260 malicious apps from 49 different families. The AMD (Android Malware Dataset) [33] contains 24,553 samples, categorized in 135 varieties among 71 malware families. On the other hand, the benign apps have been downloaded from Google app store using APKPure free online downloader. A Python script has been written to ensure that the downloaded apps are benign based on scanning

them using the Virus Total online API. The app has been accepted as benign only if it could not be detected by any antivirus engine out of more than 60 engines available at the Virus Total service. The algorithm used for downloading and scanning the benign dataset is shown in Fig. 1. The constructed three image datasets will be described in the following sections.

3.1.1 Manifest file-based image dataset

The Manifest.xml file is one of the most important files in the android application. The Manifest file is the first file read by the system when executing any android app. This file is considered as a road map specifying how the application will be executed and how its behaviour will be. So, it has suggested converting this file to grayscale images in order to test its efficiency in android apps' classification.

Due to the limitation in the Manifest file size, the width of the image constructed using this file is 64 pixels and its height is variable in accordance with the file size. It is worth mentioning, that the image width 64 has been selected experimentally after testing multiple widths including 256, 128 and 64. The big similarity between the Manifest files-based images of the Android applications from the same malware family can be seen in Fig. 2.

3.1.2 DEX code-based image dataset

Generally, Android application code is written using Java programming language and compiled to DEX code. DEX (Dalvik executable) code is an optimized byte code used to initialize and execute Android applications. Since Dex code contains the actual code of the app, these files are very important to specify the app's behaviour. So, it has

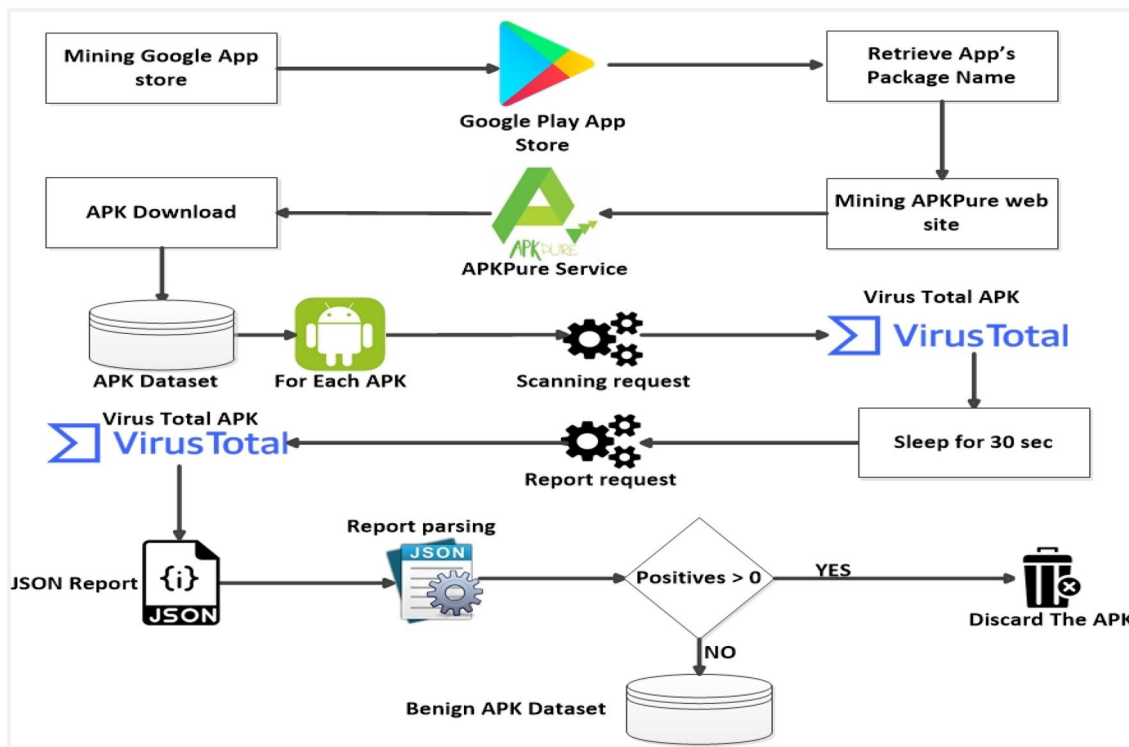
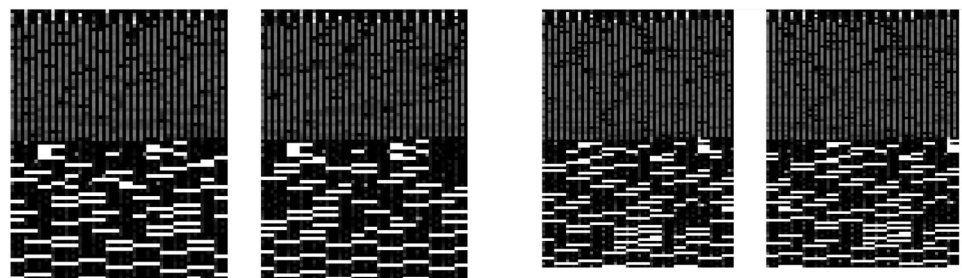


Fig. 1 The algorithm used for constructing the used benign apps dataset

Fig. 2 a Manifest files of two malware samples. b Manifest files of two benign samples



suggested converting Dex code files to a grayscale image in this work to collect as much as possible information about the app's behaviour.

3.1.3 Manifest-DEX-ARSC image dataset

In the third dataset, it has been suggested constructing the images using three files from APK archive contents, i.e. Manifest, DEX and Resource.arsc files. The Resource.arsc file contains the app's compiled resources in a binary format. Also, the Resource.arsc file contains references to the application's resources such as UI layouts and string values.

It worth mentioning that the width of the constructed images in the second and third image dataset is 256 pixels while their height varies in accordance with the size of the used files.

3.2 Image global features

This type of features describes the image as a whole [34], such that the whole image is described using a one feature vector. In this work, three different global features have been used in order to test the efficiency of this type of features in the malware detection domain. The first used global feature is Colour Histogram, in which a histogram is calculated based on the distribution of colours in the image's pixels (pixels intensity). Since the grayscale image representation has been used in this work the constructed histogram contains 255 bins (because that the grayscale image pixels take intensity values between 0 and 255). The second used global feature is Hu Moments. The image moments are a weighted average of the image pixels' intensity [35]. The Hu moments are a moment type composed of seven number calculated using central moments. The central moment can be calculated using Eq. 1

$$\mu_{ij} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (x - \bar{x})^i \cdot (y - \bar{y})^j \cdot f(x, y) dx \cdot dy \quad (1)$$

where,

$$i, j = 1, 2, 3 \dots \quad \bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

Hu moments are invariant to translation, scale, reflection and rotation. The third global feature that has been used in this paper is Haralick Texture. Texture descriptor provides measures of image properties such as smoothness, coarseness, and regularity [36]. Haralick textures are one of the most famous texture features used in image classification. Haralick texture features are calculated based on the co-occurrence matrix and composed of 13 features.

3.3 Image local features

The image local features describe the image's objects (it is based on describing a small group of pixels or small blobs in the image) [37]. So, by using this type of algorithms the image is described as a set of feature vectors (descriptor vectors). On other words, local features aim to detect the interesting points in the image and describe them as a set of descriptors. In this work, four of the most used local features i.e. SIFT, SURF, KAZE and ORB have been extracted to test the efficiency of this type of features in the malware detection domain.

3.3.1 Scale-invariant feature transform (SIFT)

The SIFT algorithm is based on calculating Laplacian of Gaussian LOG at different scale values and selecting the scale which gives the best results. In the SIFT algorithm, the interesting points are the local maxima/local minima obtained using the Laplacian of Gaussian's scale space. On other words, in this algorithm, a multiple Laplacian of Gaussian is applied at a different scale level (σ values) and the responses are plotted. After that, the scale that gives a local maxima/local minima for each pixel in the image is selected. Since the LOG's computation is costly to some extent, the SIFT algorithm uses approximate Laplacian of Gaussian which is calculated using Eq. 2.

$$\sigma \Delta^2 G = \frac{\partial G}{\partial \sigma} = \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \quad (2)$$

where, $G(x, y, \sigma)$ is LoG in the location (x, y) for the scale σ . $G(x, y, k\sigma)$ is LoG in the location (x, y) for scale $k\sigma$, $k\sigma$: is a scale slightly larger than σ .

The SIFT algorithm represents the detected key points (the interesting points) using 128-bit descriptors.

3.3.2 Speeded up robust features (SURF)

This algorithm is introduced by [38] as a fast algorithm that can be an alternative of SIFT algorithm. SURF is a fast and robust algorithm for similarity invariant representation and comparison of images. This algorithm characterized by its fast computation using box filters (It uses box filters for LOG approximation), thus it can be used in real-time applications such as tracking and object recognition. The advantage of using box filters approximation in this algorithm is that the convolution with a box filter can be easily calculated using integral images. Moreover, SURF algorithm is based on the determinant of Hessian matrix for both scale and location. The SURF algorithm uses Hessian matrix to detect the key points in the image, which has a good performance in term of computation time and

accuracy. To this end, the image is filtered by a Gaussian kernel, so that for a given point $X=(x, y)$ the Hessian matrix at scale σ is defined as in Eq. 3.

$$\mathcal{H}(x, y) = \begin{pmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{pmatrix} \quad (3)$$

where $L_{xx}(x, \sigma)$ is the convolution of the Gaussian second order derivative for the image I in the point x , and similarly for $L_{xy}(x, \sigma)$ and $L_{yy}(x, \sigma)$.

After detecting the key points SURF descriptors are created with two steps. In the first step, the Haar-wavelet responses are calculated in the horizontal and vertical direction for a neighbourhood of size $6s$. Then, a square region aligned to the selected orientation is constructed and the SURF descriptor (64-bit vector) is extracted from it.

3.3.3 KAZE feature

This algorithm has been introduced in 2012 by [39] as a method for features detection and description in nonlinear scale-spaces. This algorithm works based on the original image resolution, without performing any downsampling at each new octave as done in the SIFT algorithm. The KAZE detector is based on normalized determinant of the Hessian Matrix at multiple scale levels. The local minima/maxima (extrema) is selected from a rectangular window of size $\sigma_i \times \sigma_i$ on the current, upper and lower scales. Like in the SURF algorithm the dominant orientation is found in a circular area (has a $6\sigma_i$ radius) around each detected interesting point in order to obtain rotation invariant descriptors. The descriptor vector size in this algorithm is 64-bit.

3.3.4 Oriented FAST and rotated BRIEF (ORB)

Basically the ORB feature is a fusion of FAST key point detector [40] and BRIEF descriptor [41] with many modifications to enhance the performance. The ORB algorithm uses the FAST algorithm to detect the key points in the image, then it uses Harris corner to find the top N among the detected key points. Moreover, the ORB uses a pyramid to produce multiscale features. The ORB features use 32-bit BRIEF-based descriptor which is calculated using a set of binary intensity test like in Eq. 4.

$$\tau(P; x, y) = \begin{cases} 1 & : P(x) < p(y) \\ 0 & : P(x) \geq p(y) \end{cases} \quad (4)$$

where, P is a smoothed image patch.
 $P(x)$ is the intensity of p at a point x .

3.4 The proposed model

In this work, a new image-based android malware classification model has been suggested. To this end, some files from the source of the android application have been converted into grayscale images. Then, the image processing techniques and machine learning algorithms have been used to classify the android apps as benign or malicious as illustrated in Fig. 3. The novelty of the proposed model is based on that the used image features have not been used before in the malware detection domain. Particularly, in the first step of the proposed model, each sample in the benign and malware apps dataset has been extracted to use its source for constructing the grayscale image datasets. The proposed model composed of two sub-models, each of which has been trained using a different group of features. In the first sub-model, the APK samples have been extracted and their sources (i.e. Manifest, Dex or Manifest-Dex-Resources.arsc) have been converted to grayscale images. The global features that have been mentioned before i.e. Colour Histogram, Hu Moments and Haralick Texture have been extracted from each image in the constructed image datasets. The extracted three global features have been stacked in one feature vector. The obtained feature vector has been normalized and used as input to train multiple machine learning classifiers. Six well-known machine learning classifiers i.e. Random forest, K-nearest neighbors, Decision Tree, Bagging, AdaBoost, Gradient Boost have been adopted and trained using the constructed global features-based vectors. In the second sub-model, four different local feature algorithms (i.e. SIFT, SURF, ORB and KAZE) have been used to extract local features descriptors from the constructed image datasets. The extracted local features have been used one by one to train the mentioned machine learning classifiers. Since the local features algorithms use multiple descriptors to represent each image, the output of these algorithms is multiple vectors. Since almost all machine learning algorithms accept n -samples n -feature vectors as an input, the output of the local features cannot be used directly as input for the machine learning classifiers. So, some techniques such as Bag of Visual Words (which has been used in this work) are used to construct one feature vector from multiple local feature descriptors [42]. The Bag of Visual Words is based on using any clustering algorithm for splitting the extracted descriptors vectors to multiple clusters. Then, the clustering algorithm is used to predict the cluster of each vector in the descriptors dataset.

So, first of all, the local descriptors (SIFT, SURF, KAZE or ORB descriptors) have been extracted from one of the constructed image datasets' samples and the local features descriptor dataset has been constructed. Then, the K-means algorithm has been used for clustering the

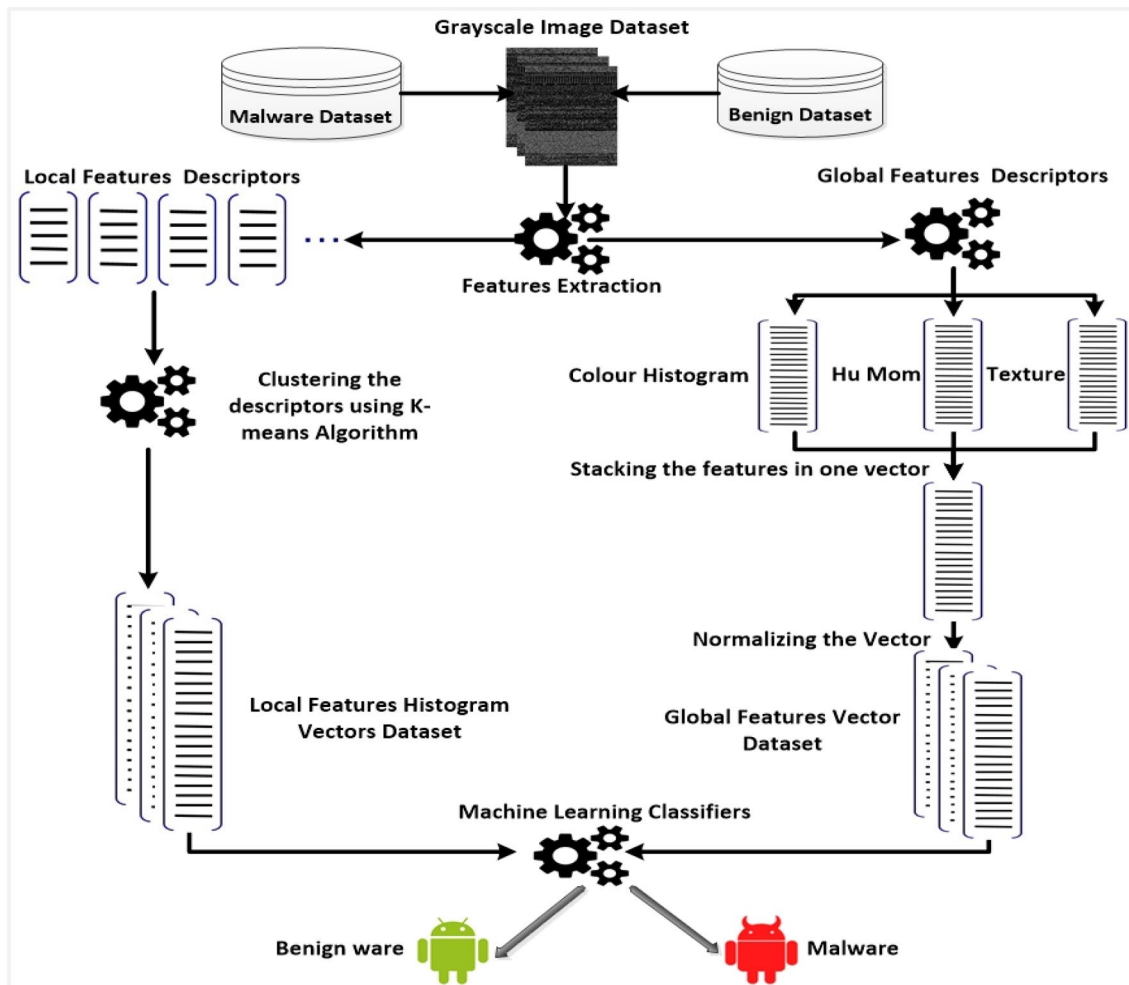


Fig. 3 The proposed malware classification model

constructed descriptors dataset into multiple clusters. After that, the K-mean algorithm has been used to predict the class of each descriptor in the image datasets. A histogram has been constructed based on the frequency of the predicted descriptors' classes using Eq. 5.

$$BOVW(i) = \{q[1], q[2], \dots, q[j]\} \tag{5}$$

where, $BOVW(i)$ is the Bag of Visual Words vector (classes histogram) for the image i . $q[j]$ the frequency of the descriptors belonging to class j in the image i .

The histogram values have been normalized during its calculation using Eq. 6.

$$q[j] = q[j] + \frac{1}{No.OfKeypointsintheimage} \tag{6}$$

After applying Bag of Visual Words to the descriptors extracted from each image, it has been obtained a one normalized feature vector. So, at the end of this stage, a dataset composed of n samples n feature vectors have been constructed. The constructed feature dataset has been used to train the machine learning classifiers that mentioned above. Algorithm 1 illustrates the proposed and developed model pseudo code algorithm.

```

Input: Benign application Dataset:  $APK_{\beta} = \{apk_1, apk_2, \dots, apk_i\}$ 
          Malware application Dataset:  $APK_{\mu} = \{apk_1, apk_2, \dots, apk_j\}$ 
Output: The label of the applications
# Image dataset construction stage:
1. For all  $apk_i \in APK_{\beta}$  do
    Extract the contents of the  $apk_i$ 
    Construct the desired grayscale images
    End for
2. For all  $apk_j \in APK_{\mu}$  do
    Extract the contents of the  $apk_j$ 
    Construct the desired grayscale images
    End for
# First sub-model (Global feature-based classification model)
3. For all  $img_i \in images$  do {Where images: the applications' image dataset}
     $\mathcal{H} \leftarrow$  extracting the colour histogram feature
     $\mu \leftarrow$  extracting the Hu Moments feature
    Texture  $\leftarrow$  extracting the haralik texture feature
     $GFV_i \leftarrow$  stacking  $\mathcal{H}$ ,  $\mu$  and Texture in one vector {Where  $GFV_i$  the global feature vector}
     $GFV_i \leftarrow$  Normalizing the global feature vector
     $X \leftarrow$  Add the constructed the  $GFV_i$  vector to the global features dataset
     $Label_i \leftarrow$  0 or 1 {Where  $Label_i$ : the label of the application}
    End for
# Second sub-model (Local feature-based classification model)
1. For all  $img_i \in images$  do {Where images: the applications' image dataset}
     $Desc_i, KP_i \leftarrow$  extract SIFT, SURF, KAZE or ORB local feature
    { $Desc_i, KP_i$ : the extracted feature's Descriptors and key points list}
     $Label_i \leftarrow$  0 or 1 {Where  $Label_i$ : the label of the application}
    End for
2. Bag of Visual Words Algorithm (vocabulary construction)
    $i \leftarrow$  0
   For all  $desc_i \in extracted\_image\_descriptors\_dataset$  do
     Histogram  $\leftarrow$  construct a zeros vector with a length of the number of clusters
      $KP_{No} \leftarrow$  the number of the detected key points in each image
     For all vector  $\in desc_i$  do
       clusterId  $\leftarrow$  Decide the cluster of the vector using K – means algorithm
       Histogram [clusterId]  $\leftarrow$  Histogram [clusterId] +  $\frac{1}{KP_{No}}$ 
     End for
      $i \leftarrow i + 1$ 
      $X \leftarrow$  Add the calculated histogram vector to the local features dataset
   End for
# Splitting the dataset into testing and training dataset
1. Split the the extracted features based on 10 – fold Cross – Validation
2. Training and testing the machine learning classifiers
Return The application label
  
```

Algorithm 1. the pseudocode of the proposed and developed model's algorithm.

4 Experiments results

All the experiments have been applied using an Intel Xeon E5-2680 8 cores and 64 GB of RAM. A codeword vocabulary of size 120 (i.e. No of class * 10) has been chosen empirically for applying the BOVW algorithm. With other words, all the key points that gathered from the constructed datasets have been clustered using K-means algorithm (i.e. MiniBatchKMeans algorithm from Sklearn library) to a vocabulary of size = 120. Also, the k-fold cross-validation with $k = 10$ has been used so that 90% of the dataset has been used for training the proposed model and 10% for testing the proposed model. Moreover, some python libraries such as Opencv, Sklearn, etc. have been used in the proposed model's development. All the performance (classification accuracy) has been calculated in terms of the overall percentage of true/false (positive/negative) decisions. The suggested two sub-models have been trained using the local and global features extracted from the three constructed image datasets. The extracted local and global features have been used to train six well-known machine learning classifiers i.e. Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost. The obtained results have been divided into the local features-based results and the global features-based results, each of which will be discussed in detail in the following sections.

4.1 The local features-based results

The extracted four local features have been used one by one to train the second sub-model of the proposed model. In the following sections, the results obtained using each local feature extracted from each one of the constructed grayscale image datasets will be discussed. Four different experiments have been applied to each of the constructed image datasets. In each experiment, one of the used local

features i.e. SIFT, SURF, ORB or KAZE has been extracted, converted to one feature vector using BOVW algorithm and used to train the machine learning classifiers in the second sub-model. Table 1 illustrates the results obtained using the image local features.

4.1.1 The Manifest dataset based local features results

Four different experiments have been conducted over this image dataset so that in each experiment different local feature has been used to train the model. In the first experiment, the SIFT local feature's descriptors have been extracted from each grayscale image in the Manifest file-based dataset. After training the machine learning classifiers using the SIFT's descriptors based BOVW vectors the classification accuracies of each of Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost were 93.59, 90.69, 90.69, 89.89, 93.89 and 89.19% respectively.

In the second experiment, the SURF local features' descriptors have been extracted to construct the BOVW vectors used for training the second sub-model in the proposed model. The obtained results showed that the accuracies of the above-mentioned classifiers reached 91.87, 87.45, 84.94, 88.25, 92.57 and 84.34% respectively.

In the third experiment, the ORB local features have been extracted from Manifest file-based dataset and used to train the proposed model. The classification accuracies of the used classifiers were 72.08, 65.16, 69.58, 67.97, 72.49 and 68.17% respectively.

In the last experiment, the KAZE features have been used to train the second sub-model in the proposed model. The results showed that the classification accuracies were 91.87, 86.55, 86.75, 86.14, 92.37 and 83.83% for Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost respectively.

Table 1 The classification accuracies obtained using the image local features

Feature	Dataset	RF	K-NN	DT	Bagging	AdaBoost	Gboost
SIFT	Manifest	93.59	90.69	90.69	89.88	93.89	89.18
	DEX	95.56	94.12	92.04	94.32	95.87	93.61
	Manifest-DEX-ARSC	96.5	94.81	93.83	95.46	97.01	95.79
SURF	Manifest	91.87	87.45	84.94	88.25	92.57	84.34
	DEX	96.90	96.29	95.88	95.15	97.22	95.88
	Manifest-DEX-ARSC	97.81	96.73	94.78	96.81	97.87	97.43
KAZE	Manifest	93.59	90.69	90.69	89.89	93.89	89.19
	DEX	97.42	96.80	94.64	96.49	97.22	96.19
	Manifest-DEX-ARSC	98.31	97.37	95.49	96.93	98.16	97.50
ORB	Manifest	72.09	65.16	69.58	67.97	72.49	68.17
	DEX	90.21	87.42	86.49	87.52	90.72	87.11
	Manifest-DEX-ARSC	93.21	91.72	87.67	90.90	93.56	88.77

Bold values indicate the best classification accuracy obtained using each local feature

4.1.2 The DEX dataset based local features results

Four experiments have been conducted over this dataset to test the effectiveness of the local features extracted from the DEX code-based images in classifying the android applications.

In the first experiment, the SIFT features have been extracted from each image in the DEX files-based image dataset. The BOVW algorithm has been used to construct a feature vector from the SIFT descriptors extracted from each image in the dataset. The accuracies of the Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost were 95.56, 94.12, 92.04, 94.33, 95.88 and 93.61% respectively.

In the second experiment, the SURF descriptors-based BOVW vectors have been used to train the machine learning classifiers in the second sub-model. The results showed that the accuracies of Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost were 96.90, 96.29, 95.88, 95.15, 97.22 and 95.88% respectively.

In the third experiment, the ORB features have been extracted from the DEX code-based images to construct the BOVW vectors used in training the machine learning classifiers. The results showed that the classification accuracies were 90.21, 87.42, 86.49, 87.53, 90.72 and 87.11% respectively.

In the last experiment that conducted over this dataset, the KAZE feature has been extracted from the DEX code-based images. When the constructed BOVW vectors have been used to train the desired classifiers, the obtained classification accuracies were 97.42, 96.80, 94.64, 96.49, 97.22 and 96.18% respectively.

4.1.3 The Manifest-DEX-ARSC image dataset based local features results

Four experiments have been conducted using this dataset too. In each experiment different local feature's descriptors have been used for constructing the BOVW vectors used for training the machine learning classifiers.

In the first experiment, the SIFT feature has been extracted from Manifest-DEX-ARSC image dataset. The extracted SIFT descriptors have been used to construct one feature vector using the BOVW algorithm. When the constructed BOVW vectors have been used to train the machine learning classifiers, the classification accuracies of Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost classifiers were 96.5, 94.8, 93.8, 95.4, 97 and 95.7% respectively.

In the second experiment, the SURF feature has been used for training the machine learning classifiers. The classification accuracies of Random forest, K-nearest

neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost machine learning classifiers were 97.8, 96.7, 94.7, 96.8, 97.8 and 97.4% respectively.

In the third experiment, the ORB features have been extracted from the Manifest-DEX-ARSC image dataset. The BOVW algorithm has been used for constructing the feature vectors from the extracted features' descriptors. The results showed that the classification accuracies were 93.23, 91.71, 87.65, 90.92, 93.51 and 88.73% respectively.

In the last experiment, the KAZE features have been extracted from each image in the Manifest-DEX-ARSC image dataset. Then, the BOVW algorithm has been used to construct feature vectors from the extracted KAZE descriptors. The constructed BOVW vectors have been used for training the machine learning classifiers in the proposed second sub-model. The obtained results showed that the classification accuracies were 98.32, 97.29, 95.41, 96.89, 98.12 and 97.55% respectively.

4.1.4 The local features-based results' discussion

It has been noted that the results obtained using the local features was worse when the Manifest-based image dataset has been used. Also, it has been observed that when the Manifest-based image dataset has been used the best results have been obtained using the SIFT algorithm, where its classification accuracy was ranging from 89.89 (using Bagging classifier) to 93.53% (using Random forest classifier). On the other hand, the worst accuracy has been obtained using the ORB feature (ranging from 65.16 to 72.49%). Moreover, when the DEX code-based image dataset has been used, the best accuracies have been obtained using the KAZE feature (ranging from 94.63 to 97.42%). The SURF algorithm's classification accuracy ranked second with an accuracy ranging from 95.87 to 96.91%, followed by the SIFT algorithm (its classification accuracy was ranging from 92.04 to 95.88%), and the ORB features gave the worst results, where its classification accuracy was ranging from 87.11 to 90.21%. Moreover, when the Manifest-DEX-ARSC image dataset has been used to train the classifiers, the best results have been obtained using the KAZE features (ranging from 95.41 to 98.35%). The SURF algorithm obtained the second-best classification accuracies (ranging from 94.73 to 97.89%), while the classification accuracies obtained using the SIFT algorithm was ranging from 93.82 to 97%. The worst classification accuracy obtained using this dataset also has been gotten using the ORB algorithm (ranging from 87.64 to 93.21%).

Table 2 The classification accuracies obtained using the image global features

Dataset	RF	K-NN	DT	Bagging	AdaBoost	Gboost
Manifest	98.29	94.78	96.18	94.98	98.49	97.09
DEX	98.14	98.25	97.22	97.63	98.35	97.83
Manifest-DEX-ARSC	98.49	98.31	97.89	98.1	98.75	98.19

Bold values indicate the best classification accuracy obtained using each local feature

4.2 The global features-based results

When the global features mentioned before have been extracted and used to train the machine learning classifiers the results were as in Table 2:

4.2.1 The Manifest dataset-based global features' results

In the first experiment, the global features have been extracted from the Manifest image dataset and used for training the first sub-model. The obtained results showed that the classification accuracies of Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost classifiers were 98.19, 95.78, 95.78, 98.59, 97.69 and 98.09% respectively.

4.2.2 The DEX dataset-based global features' results

In the second experiment, the global features have been extracted from the DEX files-based image dataset and used for training the first sub-model in the proposed model. The classification accuracies of Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost classifiers were 98.45, 98.25, 97.22, 97.53, 98.55 and 98.25% respectively.

4.2.3 The Manifest-DEX-ARSC dataset-based global features' results

In the third experiment, the global features have been extracted from the image dataset constructed using the Manifest, DEX and ARSC files. When the extracted global features have been used for training the machine learning classifiers, the classification accuracies of Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and

Gradient Boost were 98.75, 98.3, 98.1, 98.2, 98.7 and 98% respectively.

4.2.4 The global features' results discussion

The classification accuracies obtained using global features were ranging from 95.78 to 98.75% based on the nature of the image dataset and the trained classifier. It has been noted that the results obtained using the Manifest image-based global features were worse than that obtained using the DEX image dataset and the Manifest-DEX-ARSC image dataset. Also, it has been noted that the best results have been obtained using the Random forest, AdaBoost and Gradient Boost classifiers. Furthermore, it is noted that the results obtained using the Decision tree and Bagging classifiers were worse than that obtained using the other classifiers to some extent.

4.3 Testing the proposed model using other datasets

The proposed model has been tested using AMD dataset (Android Malware Dataset) to prove its efficiency in detecting any android malware dataset. The AMD android dataset is one of the largest android malware datasets contains more than 24000 samples related to 71 families. 4850 malware samples have been selected randomly from the AMD dataset, and three malware image datasets have been constructed. After that, the proposed model has been tested using the constructed AMD based image datasets. Since the results obtained using the global features were better than that obtained using the local features, only the classification accuracy obtained using the global features extracted from this dataset has been tested. The obtained results (illustrated in Table 3) showed that the classification accuracy reached more than 98% when the proposed model has been trained using the global features

Table 3 The results of testing the proposed model using AMD malware dataset

Dataset	RF	KNN	DT	Bagging	AdaBoost	GBoost
Manifest dataset	97.61	93.99	95.19	94.29	97.40	96.61
DEX dataset	97.11	96.71	97.02	95.99	97.12	96.40
Manifest-DEX-ARSC	98.36	96.81	97.1	96.71	98.36	96.81

Bold values indicate the best classification accuracy obtained using each local feature

Table 4 The computational time of the proposed model

	RF	KNN	DT	Bagging	AdaBoost	G-boost
SIFT	868.84	864.94	864.39	865.61	867.04	867.75
SURF	762.99	759.65	759.39	759.99	761.4	762.18
KAZE	49.36	48.31	47.77	48.95	262.87	48.66
ORB	41.22	39.83	39.36	40.51	244.73	40.19
Global features	172.49	166.14	165.17	168.07	168.78	176.37

Table 5 Comparison of the proposed model results with the results of some previous works

Model name	Methodology	Time/Sec	Accuracy (%)
AspectDroid [6]	Hybrid analysis	–	94.68
SAFEDroid [7]	Static analysis	–	98.4
DroidDet [8]	Static analysis	–	88.26
Wang.et al. [9]	Static analysis	–	94
FalDroid [27]	Static analysis	4.6	94.2
DREBIN [31]	Static analysis	0.75	94
DroidSIFT [43]	Static analysis	0.06	93
R2-D2 [28]	Image-based	0.5	93
Yang.et al. [29]	Image-based	–	95.42
Karimi.et al. [30]	Image-based	–	97
Yen.et al. [23]	Image-based	–	92.67
Proposed Model	Image-based	0.018	98.75

extracted from the AMD-based Manifest-ARSC-DEX image dataset.

4.4 The computational time analysing study

The computational cost has been analysed for each stage in the proposed model including the feature extraction, training and testing the model. The computational time study has been conducted using the Manifest file-based image dataset’s experiments, and the results were as in Table 4. The computational time analysis study for the proposed model showed that the SIFT features needs the highest run-time overhead for training the model, where its computational time reached 866.43 s on average including the time of features extraction and training the model. In contrast, it has been observed the ORB and KAZE features need the lowest average total run-time overhead (74.31 and 84.32 s respectively) for features extraction and model’s training and testing.

5 The results comparison

In this section, the obtained results have been compared with the results of some state-of-the-art works. Table 5 illustrates the comparison that has been conducted between this work results and some other works’ results in term of classification accuracy and computational time. The compared works have been selected carefully so that they include static analysis, dynamic analysis, hybrid analysis and image-based analysis frameworks. It has been concluded that the proposed model outperformed the other frameworks with classification accuracy reached 98.75% with a typical computational time does not exceed 0.018 s for each sample.

6 Discussion and decision

In this work, a malware visualisation method has been proposed for detecting Android malware based on grayscale image representation and machine learning techniques. Two types of image-based features have been extracted from the constructed malware image datasets and used to train six machine learning classifiers in multiple scenarios. It has been observed that the global features can give better classification accuracy than that obtained using the local features almost in all experiments. Particularly, the classification accuracy reached more than 98% when the global features extracted from each of Manifest, DEX and Manifest-DEX-ARSC image dataset have been used to train the AdaBoost classifier. Also, the classification accuracy reached more than 98% when the local features extracted from the Manifest-DEX-ARSC image dataset have been used to train the AdaBoost classifier. With other words, the best classification accuracies in this work have been obtained using the AdaBoost classifier. In general, the local features extracted from the Manifest image dataset gave classification accuracies less than that obtained using the local features extracted from the DEX or Manifest-DEX-ARSC image datasets. Also, it has been noted that the ORB local features gave the worst classification accuracies in all experiments where its classification accuracy was ranging from 65.16% to 93.56% based on the used image

dataset and the trained classifier. According to the results of the conducted time complexity analysis study, the ORB features need the smallest total computational time for extracting the features, training the model and detecting the malware samples. Particularly, the average of the total time needed for training the model using the ORB features was 74.31 s; which means 0.008 s for each malware sample on average. But in contrast, the ORB features gave the worst classification accuracy results in all conducted experiments. Furthermore, it has been concluded that the SIFT features need the highest cost time for extracting the features and training the model (the worst case), where its average computational time was 866.43 s, which means 0.091 s on average for each sample.

It is worth mentioning that the KAZE features gave the best classification accuracy (reached more 98%) which has been obtained using the local features with an acceptable run-time overhead close to that needed for the ORB feature (where the KAZE feature's average total computational time was 84.32 s; which means 0.009 s for each sample). On the other hand, the average total time needed for extracting the global features and training the model was 169.50 s; which means 0.018 s for each sample. With other words, the model needs 0.018 s (which considered a very efficient computational time) on average for each sample to give a classification accuracy reached 98.75%. The proposed model outperformed all of the compared previous works in term of the classification accuracy and computational time, where its classification accuracy reached 98.75% and its computational time was 0.018 s for each sample.

7 Limitations

The proposed model's performance may be affected by the code obfuscation and code manipulation techniques which is a drawback of almost all static analysis detection techniques. This can be overcome by integrating the image-based features with some robust code-based semantic features, which has been left to the future works. Also, the proposed method cannot detect the injection attacks proposed in [44]. It will be tried to bypass these types of attack using image-based object detection techniques in the future works.

8 Conclusions

In this paper, a visualization-based framework has been proposed for classifying the android applications as benign ware or malware. The proposed model is based on converting some APK archive's contents into

grayscale images and using image processing techniques and machine learning algorithms for android apps classification. To this end, three different grayscale image datasets have been constructed. In the first dataset, the APK archives have been decompiled and the Manifest.xml files have been converted to grayscale images. In the second dataset, the DEX code files of each application have been converted into a grayscale image. In the third dataset, each of Manifest, DEX and Resources.ARSC files from each application have been converted into a grayscale image. Two types of image-based features (i.e. Global features and Local features) have been extracted for training multiple machine learning classifiers (i.e. Random forest, K-nearest neighbors, Decision tree, Bagging, AdaBoost and Gradient Boost). Three global features including Colour Histogram, Hu Moments and Haralick Texture have been extracted, normalized and stacked in one feature vector and used for training the above-mentioned machine learning classifiers. On the other hand, four image local features including SIFT, SURF, KAZE, ORB have been extracted and the bag of visual words (BOVW) algorithm has been used for constructing one feature vector from the extracted local feature's descriptors. The obtained results showed that the proposed model outperforms the previous conducted works in term of the classification accuracy and computational time, where its classification accuracy reached more than 98% with a typical run-time overhead did not exceed 0.018 s on average for each sample.

In the future works, the proposed model will be expanded for detecting camouflaged malware samples (such as the obfuscation techniques-based attacks mentioned in Sect. 7) using image-based object detection techniques.

Acknowledgements The authors thank Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu and Haojin Zhu for sharing their KuafuDet malware Dataset. Also, the authors thank Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon and Konrad Rieck for sharing their Drebin Android malware dataset. Moreover, the authors thank the authors of ADM Android malware dataset for sharing their dataset.

Author contributions The manuscript was written by Khaled Bakour under the supervision of H.Murat Ünver. The modelling, analysis, and software development have been conducted by Khaled Bakour under the supervision of H.Murat Ünver.

Funding This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

- Gartner (2018) Gartner says worldwide sales of smartphones recorded first ever decline during the fourth quarter of 2017. <https://www.gartner.com/en/newsroom/press-releases/2018-02-22-gartner-says-worldwide-sales-of-smartphones-recorded-first-ever-decline-during-the-fourth-quarter-of-2017>. Accessed 27 Oct 2019
- StatcounterGlobalStats (2020) Mobile operating system market share worldwide. Mobile Operating System Market Share Worldwide <https://gs.statcounter.com/os-market-share/mobile/worldwide>. Accessed 09 Mar 2020
- G-DATA (2018) Malware figures for Android rise rapidly. <https://www.gdatasoftware.com/blog/2018/07/30937-malware-figures-for-android-rise-rapidly>. Accessed 27 Oct 2019
- SecureList (2018) Mobile malware evolution 2018. <https://securelist.com/mobile-malware-evolution-2018/89689/>. Accessed 27 Oct-2019
- DoctorWeb (2019) Doctor Web's overview of malware detected on mobile devices in September 2019." <https://news.drweb.com/show/review/?i=13446>. Accessed 27 Oct 2019
- Ali-Gombe AI et al (2018) Toward a more dependable hybrid analysis of android malware using aspect-oriented programming. *Comput Secur* 73:235–248. <https://doi.org/10.1016/j.cose.2017.11.006>
- Goyal R, et al (2016) SafeDroid: a distributed malware detection service for Android. In: 2016 IEEE 9th international conference on service-oriented computing and applications (SOCA). 2016. IEEE. <https://doi.org/10.1109/soca.2016.14>
- Zhu H-J et al (2018) DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing* 272:638–646. <https://doi.org/10.1016/j.neucom.2017.07.030>
- Wang C et al (2018) Research on data mining of permissions mode for Android malware detection. *Clust Comput*. <https://doi.org/10.1007/s10586-018-1904-x>
- Moonsamy V, Rong J, Liu S (2014) Mining permission patterns for contrasting clean and malicious android applications. *Future Gener Comput Syst* 36:122–132. <https://doi.org/10.1016/j.future.2013.09.014>
- Xiaoyan Z, Juan F, Xiujuan W (2014) Android malware detection based on permissions. In: 2014 International conference on information and communications technologies (ICT 2014). <https://doi.org/10.1049/cp.2014.0605>
- Tao G et al (2018) MalPat: mining patterns of malicious and benign android apps via permission-related APIs. *IEEE Trans Reliab* 67(1):355–369. <https://doi.org/10.1109/tr.2017.2778147>
- Wu S et al (2016) Effective detection of android malware based on the usage of data flow APIs and machine learning. *Inf Softw Technol* 75:17–25. <https://doi.org/10.1016/j.infsof.2016.03.004>
- Canfora G, et al (2015) Effectiveness of opcode ngrams for detection of multi family android malware. In: 2015 10th International conference on availability, reliability and security. IEEE
- Papadopoulos H et al (2018) Android malware detection with unbiased confidence guarantees. *Neurocomputing* 280:3–12. <https://doi.org/10.1016/j.neucom.2017.08.072>
- Somarriba O, Zurutuza U (2017) A collaborative framework for android malware detection using DNS & dynamic analysis. In: 2017 IEEE 37th Central America and Panama Convention (CON-CAPAN XXXVII). <https://doi.org/10.1109/concapan.2017.8278529>
- Tong F, Yan Z (2017) A hybrid approach of mobile malware detection in Android. *J Parallel Distrib Comput* 103:22–31. <https://doi.org/10.1016/j.jpdc.2016.10.012>
- Alzaylaee MK, Yerima SY, Sezer S (2017) Emulator versus real phone: Android malware detection using machine learning. In: Proceedings of the 3rd ACM on international workshop on security and privacy analytics. ACM. <https://doi.org/10.1145/3041010.08.3041010>
- Dietz M, et al (2011) Quire: lightweight provenance for smart phone operating systems. In: USENIX security symposium. San Francisco, CA
- Bugiel S, et al (2011) XManDroid: a new Android evolution to mitigate privilege escalation attacks. Technische Universit at Darmstadt, Technical Report TR-2011-04
- Kabakus AT, Dogru IA (2018) An in-depth analysis of Android malware using hybrid techniques. *Digit Investig* 24:25–33. <https://doi.org/10.1016/j.diin.2018.01.001>
- Yuan Z, Lu Y, Xue Y (2016) Droiddetector: android malware characterization and detection using deep learning. *Tsinghua Sci Technol* 21(1):114–123. <https://doi.org/10.1109/TST.2016.7399288>
- Yen Y-S, Sun H-M (2019) An Android mutation malware detection based on deep learning using visualization of importance from codes. *Microelectron Reliab* 93:109–114. <https://doi.org/10.1016/j.microrel.2019.01.007>
- Xiao X et al (2019) Android malware detection based on system call sequences and LSTM. *Multimedia Tools Appl* 78(4):3979–3999. <https://doi.org/10.1007/s11042-017-5104-0>
- Wang W, Zhao M, Wang J (2019) Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *J Ambient Intell Humaniz Comput* 10(8):3035–3043. <https://doi.org/10.1007/s12652-018-0803-6>
- Zhu H-J et al (2018) HEMD: a highly efficient random forest-based malware detection framework for Android. *Neural Comput Appl* 30(11):3353–3361. <https://doi.org/10.1007/s00521-017-2914-y>
- Fan M et al (2018) Android Malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Trans Inf Forensics Secur* 13(8):1890–1905. <https://doi.org/10.1109/tifs.2018.2806891>
- Huang TH, Kao H (2018) R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based Android Malware Detections. In: 2018 IEEE international conference on big data (big data). <https://doi.org/10.1109/bigdata.2018.8622324>
- Yang M, Wen Q (2017) Detecting android malware by applying classification techniques on images patterns. In: 2017 IEEE 2nd international conference on cloud computing and big data analysis (ICCCBDA). IEEE. <https://doi.org/10.1109/icccbda.2017.7951936>
- Karimi A, Moattar MH (2017) Android ransomware detection using reduced opcode sequence and image similarity. In: 2017 7th International conference on computer and knowledge engineering (ICCKE). <https://doi.org/10.1109/iccke.2017.8167881>
- Arp D, et al (2014) Drebin: effective and explainable detection of android malware in your pocket. in Ndss
- Zhou Y, Jiang X (2012) Dissecting Android Malware: characterization and evolution. In: 2012 IEEE symposium on security and privacy. <https://doi.org/10.1109/sp.2012.16>
- Wei F, et al (2017) Deep ground truth analysis of current android malware. In: International conference on detection of intrusions and malware, and vulnerability assessment. Springer, Berlin
- Hassaballah M, Awad AI (2016) Detection and description of image features: an introduction. In: Awad AI, Hassaballah M (eds) Image feature detectors and descriptors : foundations and applications. Springer, Cham, pp 1–8. https://doi.org/10.1007/978-3-319-28854-3_1
- Zhihu H, Jinsong L (2010) Analysis of Hu's moment invariants on image scaling and rotation. In: 2010 2nd International

- conference on computer engineering and technology. <https://doi.org/10.1109/iccet.2010.5485542>
36. Kumar RM, Sreekumar K (2014) A survey on image feature descriptors. *Int J Comput Sci Inf Technol* 5:7668–7673
 37. Ehab Salahat MQ (2017) Recent advances in features extraction and description algorithms: a comprehensive survey. In: *IEEE international conference on industrial technology (ICIT)*. 2017 of Conference. Toronto. <https://doi.org/10.1109/icit.2017.7915508>
 38. Bay H, Tuytelaars T, Van Gool L (2006) Surf: speeded up robust features. In: *European conference on computer vision*. Springer, Berlin
 39. Alcantarilla PF, Bartoli A, Davison AJ (2012) KAZE features. In: *European conference on computer vision*. Springer, Berlin
 40. Rosten E, Drummond T (2006) Machine learning for high-speed corner detection. In: *European conference on computer vision*. Springer, Berlin
 41. Calonder M, et al (2010) Brief: binary robust independent elementary features. In: *European conference on computer vision*. Springer, Berlin
 42. Ali N, Bajwa KB, Sablatnig R, Chatzichristofis SA, Iqbal Z, Rashid M et al (2016) A novel image retrieval based on visual words integration of SIFT and SURF. *PloS one* 11(6):e0157428. <https://doi.org/10.1371/journal.pone.0157428>
 43. Zhang M, et al (2014) Semantics-aware android malware classification using weighted contextual API dependency graphs. In: *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 2014 of conference. Scottsdale, Arizona, USA: Association for Computing Machinery. <https://doi.org/10.1145/2660267.2660359>
 44. Bakour K, Ünver HM, Ghanem R (2019) A deep camouflage: evaluating Android's anti-malware systems robustness against hybridization of obfuscation techniques with injection attacks. *Arab J Sci Eng* 44(11):9333–9347. <https://doi.org/10.1007/s13369-019-04081-5>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.