

Mining Android Bytecodes through the Eyes of Gabor Filters for Detecting Malware

Shahid Alam

Department of Computer Engineering, Adana Alparslan
Turkes Science and Technology University, Turkey
salam@atu.edu.tr

Alper Kamil Demir

Department of Computer Engineering, Adana Alparslan
Turkes Science and Technology University, Turkey
akdemir@atu.edu.tr

Abstract: *One of the basic characteristics of a Gabor filter is that it provides useful information about specific frequencies in a localized region. Such information can be used in locating snippets of code, i.e., localized code, in a program when transformed into an image for finding embedded malicious patterns. Keeping this phenomenon, we propose a novel technique using a sliding window over Gabor filters for mining the Dalvik Executable (DEX) bytecodes of an Android application (APK) to find malicious patterns. We extract the structural and behavioral functionality and localized information of an APK through Gabor filtered images of the 2D grayscale image of the DEX bytecodes. A window is slid over these features and a weight is assigned based on its frequency of use. The selected windows whose weights are greater than a given threshold, are used for training a classifier to detect malware APKs. Our technique does not require any disassembly or execution of the malware program and hence is much safer and more accurate. To further improve feature selection, we apply a greedy optimization algorithm to find the best performing feature subset. The proposed technique, when tested using real malware and benign APKs, obtained a detection rate of 98.9% with 10-fold cross-validation.*

Keywords: *Android bytecode, malware analysis and detection, sliding window, gabor filters, gabor features, machine learning.*

Received February 14, 2021; accepted September 26, 2022
<https://doi.org/10.34028/iajit/20/2/4>

1. Introduction

Android Operating System (OS) became an almost indispensable choice for mobile devices since its first release in 2008 [34]. Like any other OS, Android is also susceptible to malware attacks. Just after 2 years in 2010, the first Android malware appeared in the scene. According to McAfee Mobile Threat Report, the total number of mobile malware programs increased by 24 million from 2018 to 2019 [24]. It is reported that 230,000 new malware samples are produced every day in 2019 [12]. The statistics show that there are a growing number of threats against Android OS. In 2019, companies spent an average of 2.4 million U.S Dollars in their defense against malware [12]. As a result, there are urgent needs to develop techniques and methods to diminish the malware attacks on Android OS. The main objective and purpose of this research is to identify malware using visual analytics for extracting important features that helps in classifying malware.

Visual data exploration was first introduced by Tuckey [30]. In 2001 and 2003 decomposition of an image was introduced to compute texture features [21, 29]. Consequently, visual analytics is also leveraged for malware detection in 2004 for the first time as far as we know [33]. It is easier for Malware writers to reuse the code and produce different variants of the original malware program. Although these variants

generate different signatures, there exist visual similarities among them. When they are visualized, the images exhibit the textual and structural similarities among malware variants.

Texture segmentation with Gabor filter is an old method in the literature [5]. Many types of research have been conducted to segment textures [3] or multi-textures [23] with Gabor filter. The research on using Gabor filters for texture classification shows that Gabor filters give localized frequency information. We believe that this is a very useful outcome to detect and analyze malware converted into images. This enables us to find embedded malicious code within a snippet of (localized) code in malware. As a result, the Gabor filter can guide us to find such information.

The main motivations of the research carried out in this paper are manifold.

1. Structural, and behavioral functionalities of an Android application need to be analyzed to detect if it is malicious or benign. The file *classes.dex*, Dalvik Executable (DEX) bytecodes of an Android APK, contains all the classes and APIs, and hence the structural and behavioral functionality of an Android application. Therefore, we first extract this information from the DEX bytecodes of an Android application and convert it to a grayscale image.
2. We apply Gabor filters to the grayscale image to generate filtered images. These filtered images of

different samples from a given malware family appear like those belonging to the same family but distinct from a benign sample, as shown in Figure 1. AnserverBot1 and AnserverBot2 malware programs (two variants of the AnserverBot malware program) are transformed into Gabor filtered images. The Figure shows that these two malware samples have similar texture features. However, the benign sample, CommissionPlus has different texture features.

- For finding similarities and to improve the feature selection process, we use a Window that slides over a 2D Gabor filtered image to help select important features for malware detection.

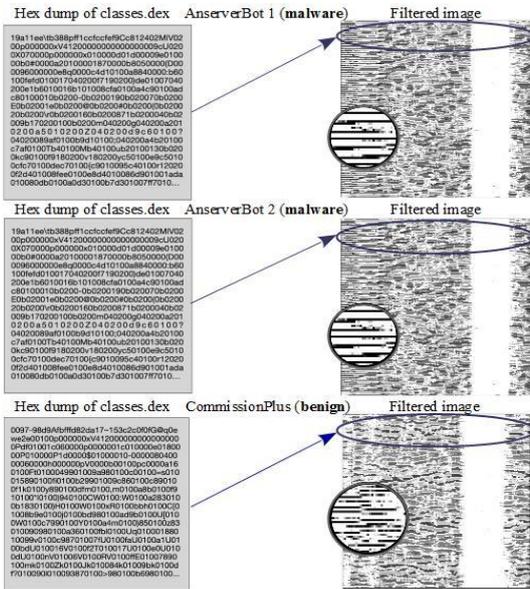


Figure 1. Similarity and differences in gabor filtered image of two malware and one benign android APKs.

Particularly, we present a static malware detection method based on the visual analytics approach. The major contributions of this work are:

- We propose a novel technique using a *sliding Window* over Gabor filtered images to detect malicious patterns in an Android APK.
- We perform static analysis on the grayscale image of the DEX bytecodes of an APK. Therefore, our analysis does not require any disassembly (which is never 100% correct) or execution (which is not safe) of the malware program. DEX bytecodes are extracted by decompressing the APK, and there is no disassembly required.
- We further improve feature selection by applying two other classic techniques. Firstly, by taking out the features (Windows) from malware that are common in malware and benign samples. Secondly, we apply a greedy optimization algorithm, called recursive feature elimination, which aims to find the best performing feature subset. This approach improved the accuracy by more than 5%.
- We evaluate our proposed methods by using two

different cross-validation techniques so that systematic and precise testing results are obtained; the overfitting problem is restricted; and the detection of new malware is achieved. Moreover, the evaluations are carried out by using different classifiers. Finally, our proposed method is compared with four other similar techniques. Our proposed technique, when tested with real malware and benign Android applications using 10-fold cross validation, obtained a detection rate of 98.9%.

The remainder of this paper is organized as follows. We discuss related works in section 2. We present a detailed overview of our approach, its design, and implementation in section 3. Section 4 presents the evaluation and comparison of our approach with four other such works. Section 5 finally concludes the paper and presents some future works.

2. Related Works

In this section we discuss the most relevant literature, similar to our work. Due to the popularity of Windows OS on Desktop and Android OS on smartphones, there are more malware programs written for these than other operating systems [1, 25]. Here, we describe some of the recent works that deal with malware detection on these two operating systems.

2.1. Android OS

Kumar *et al.* [11], a machine learning-based method is proposed to detect Android malware. Visual representation of APK file is transformed into Grayscale, Red, Green, and Blue (RGB), Cyan, Magenta, Yellow, and Key (CMYK), and Hue-Saturation-Lightness (HSL). At first, the Global Image Descriptor (GIST) feature extraction method [21], extract a particular set of features through Gabor filters, is applied to malware and benign image dataset. Then, different machine learning algorithms are used for classification. The highest accuracy achieved was 91%.

Chen *et al.* [2], the authors propose the XGBoost machine learning method to classify Android malware. In the beginning, the DEX file is extracted from Android malware. Then, the DEX file is converted into a binary sequence vector, and transformed into an 8-bit binary array. Each 8-bit binary number is considered as a byte corresponding to a 256-order grayscale. After that, it is converted into a pixel matrix. The GIST feature extraction is applied to the grayscale image. Neither the source nor the types of the samples used in the evaluation is mentioned in the paper. Therefore, we do not compare the technique proposed in [2] with ours.

Darus *et al.* [4], the authors applied the GIST features to visualized images of Android APKs, to classify malware. DEX bytecodes of an Android APK

is first translated to 8-bit grayscale image. The GIST features are extracted from the grayscale image for classification. They used different classifiers to evaluate their technique. They achieved an accuracy of 84.14%.

Naeem *et al.* [18], a cross-platform (Windows and Android operating systems) malware variant classification system is proposed. Here we only discuss their part that deals with Android applications. First, they decompile an Android Application (APK) to Java source code. The Java source code is then translated to grayscale image and GIST features are extracted. These features are then used for classifying Android APKs. The experiments carried out in the paper achieved promising results, obtaining an accuracy of 96.3% with Android applications.

Fang *et al.* [6], Android malware classification is investigated by the multiple kernel learning method to classify Android malware. At first, the DEX file is extracted from Android malware. Then, it is converted into an RGB image and plain text jointly. Textual features, such as variables, methods, and classes, are extracted from the DEX data section. Gabor features are extracted using the GIST descriptor from the RGB image of the DEX bytecode. Feature selection is performed by applying different kernels, such as linear and Gaussian. Finally, Support Vector Machine (SVM) is used for classification. The results indicate a precision and Detection Rate (DR) of 96%.

2.2. Windows OS

The work in [20] compares GIST based static texture analysis with dynamic analysis. It is found that static texture analysis can provide a comparable classification accuracy of 98% to that of dynamic techniques. Nataraj *et al.* [19] present grayscale malware images to classify using GIST feature extraction. The results show 98% accuracy.

Makandar and Patrot [14], at first, sub-band filtering was applied to the grayscale malware image. After that, the Gabor filter is applied to get second-order gradient features as static pattern recognition. Finally, the SVM classifier is applied for malware class recognition. 89.68% accuracy is obtained through experiments. Malware class identification is also studied in [13]. The Gabor wavelet is used for feature extraction. Along with Gabor, GIST and discrete wavelet transform and other features are considered. Finally, from an image processing-based perspective, SVM multi-class classifier is leveraged for malware classification. They got 98.88% accuracy.

Zhou and Jiang [34], visualized malware as a grayscale image. Then, the Gabor filter is used for extracting texture features. They used extremely randomized trees classifiers, and the results are compared with K-Nearest Neighbor (KNN), and Random Forest classifiers. The experimental results

revealed 96.19% accuracy and a 97.51% recall rate. Grayscale visualization technique is also studied in [10] for malware detection. Intensity-based and texture (Wavelet and Gabor) based features are used with the SVM classifier. Accuracy of 95% is achieved.

Naeem *et al.* [17], the authors present a lightweight malware classification system using GIST based features and hybrid local features. According to conducted experiments, 97.4% accuracy is achieved by the SVM classifier. GIST descriptors are applied to malware classification problem in [32]. After feature reduction by determining a minimal set of GIST features, deep learning techniques of Tensor Flow is applied for classification. Experimental results showed accuracy of between 90% and 100% in different experimental setups. Kumar *et al.* [11], present a malware classification work based on the grayscale visualization method. GIST, Histogram of Oriented Gradient, and Local Binary Pattern feature descriptors are used for the SVM classifier. The highest accuracy achieved is 98.73%.

Fang *et al.* [6], the authors exhibit another malware classification work where they use GIST along with entropy filter to compute texture features. 89% accuracy is achieved with the KNN classifier. A method for transforming a malware into a binary grayscale image to characterize malware variant is proposed in [16]. After transforming the malware into a grayscale image, the local and global malicious patterns feature descriptor is used to extract local and global image interest points. Then, feature selection methods are applied for key feature extraction. Finally, KNN, SVM, and Naive Bayes classifiers are applied for classification. An accuracy of up to 98.40% is achieved.

3. Overview of the System

Android applications are distributed in a packed (compressed) form called Android application packages (APKs). An APK contains files and directories, such as a manifest file, resources, classes.dex etc. The system proposed in this paper takes APK files as input and extracts DEX bytecodes from each APK; These bytecodes are converted to grayscale image and then Gabor features are extracted from these images; After preprocessing and feature selection, the selected Gabor features are used by a classifier for training to detect a sample either as malware or benign. Figure 2 provides an overview of the proposed system. The complete system is implemented as 2000+lines of Python code. The feature selection component is the most time-consuming part of the system. Therefore, to optimize we parallelized this part by using recently introduced Python's multiprocessing package. In the following sections, we explain and discuss in detail each of the components shown in Figure 2.

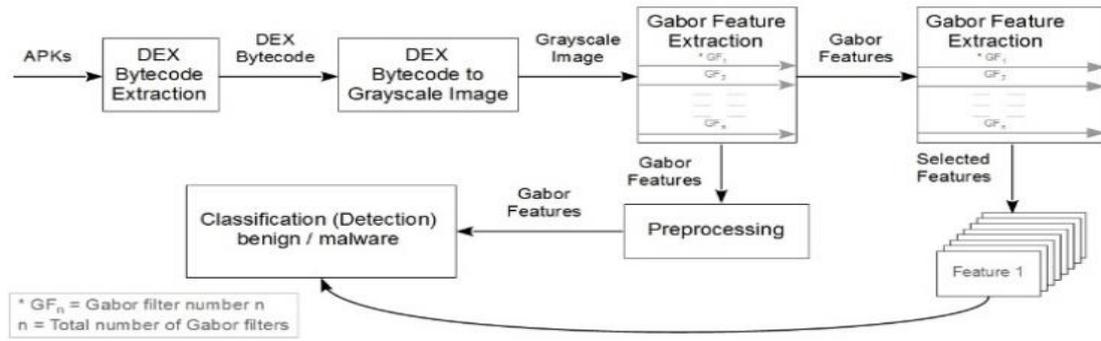


Figure 2. Overview of the system.

3.1. DEX Bytecode Extraction

Android programs are normally written in Java; compiled to Java bytecode; converted to Dalvik instructions and stored as Dalvik Executable Format (DEX) [27] bytecodes in a file *classes.dex*. These DEX bytecodes (i.e., *classes.dex*) are then combined with other resources and compressed to form an APK. The file *classes.dex* contains all the classes and APIs, and the structural and behavioral functionality of the application. Therefore, to analyse an APK, we decompress the APK file and extract *classes.dex* (DEX bytecodes) file.

3.2. DEX Bytecode to Grayscale Image

We store the extracted DEX bytecodes in a vector space of 8-bit unsigned integers. If the size of these bytecodes is more than 4 bytes (our experimentation reveals that ≤ 4 is of no importance), then we convert the bytecodes to a 2D grayscale image. Each pixel value (*row, col*) in this 2D grayscale image represents a DEX bytecode normalized in the range of 0–255. This value represents luminous (light) intensity [26] with 0 being the weakest and 255 the highest. As shown earlier in Figure 1, this intensity maps to the code in the application. For detecting malware, it is very useful to find more information about a snippet of (localized) code to search for malicious code embedded in an application. Applying Gabor filters [28] to this image facilitates us in finding such information. Gabor filters extract localized features from the image, i.e., snippets of code (patterns), to help find similarity with malicious code.

3.3. Gabor Feature Extraction

We apply Gabor filters to extract localized features (pattern of code) from an Android application's grayscale image. For this a set of filters is created as follows.

A 2D Gabor filter in the spatial domain is defined by the following function:

$$G(x, y; \lambda, \theta, \phi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \phi\right) \quad (1)$$

Where, $x' = x \cos \theta + y \sin \theta$ and $y' = -x \sin \theta +$

$y \cos \theta$ and (x, y) is the position/location of the pixel whose Gabor weight (co-efficient) is computed by the filter.

A convolution kernel is a square matrix that is slid across the image and multiplied with the input image to produce different filtered images. Three different convolution Gabor kernels of sizes 2, 25, and 30 were generated using Equation (1). These three kernel sizes gave us a better variance between the different Gabor filters. We observe that there was no effect of the size beyond 30 on the output (filtered) image. These kernels were then multiplied by the original image to produce different Gabor filtered images. In this way, we created 24 Gabor filtered images using different values of the above five parameters (λ , θ , ϕ , σ , and γ) and the three kernels. Table 1 shows the set of Gabor filters used to produce 24 filtered images of the original grayscale image of an APK's DEX bytecodes. These Gabor parameter values were computed empirically through a series of regression tests.

These filtered images are the coarse Gabor features of an APK. The extracted Gabor features after preprocessing and selection, are used by a classifier for training to detect a sample either as malware or benign. In the next section, we explain the preprocessing and feature selection performed on these extracted Gabor features.

3.4. Preprocessing and Feature Selection

Before preprocessing, we first formally define the Gabor features extracted in section 3.3 as follows. Let $Gfk = \{F_1, F_2, F_3, \dots, F_n\}$ denotes the extracted Gabor features for the k th APK; F_n is the filtered image n ; n is the total number of Gabor filtered images for the k th APK, and is same for all the APKs; i.e., each APK has n filtered images $\{F_1, F_2, F_3, \dots, F_n\}$; in our case $n=24$.

A filtered image is a 2D array of Gabor weights (computed by Gabor function in Equation (1)) for each pixel in the original grayscale image of an APK. These weights map to the coding intensity in the original DEX bytecodes of the APK. We are looking for finding similarities, i.e., part (Window) of a malicious code that is found in malware APKs. Therefore, we use the concept of a Window that slides over these Gabor

weights to find such similarities. Keeping this concept of a sliding Window, we define a filtered image as $f = \{W_1, W_2, W_3, \dots, W_n\}$, where p is the number of Windows in the filtered image, $W = \{W_1, W_2, W_3, \dots, W_s\}$, w is the Gabor weight and s is the size of the Window W . s is the same for all the Windows. Each of these Windows represents a specific Gabor feature.

Table 1. The set of Gabor filters used to produce 24 filtered images of the original grayscale image. For each of the 24 Gabor filters value of $\theta = 0$ and $\gamma = 0.05$.

	kernel size	θ	σ	λ
1	2	0	9	0.7
2	2	0	9	1.5
3	2	0	17	0.7
4	2	0	17	1.5
5	2	2	9	0.7
6	2	2	9	1.5
7	2	2	17	0.7
8	2	2	17	1.5
9	25	0	9	0.7
10	25	0	9	1.5
11	25	0	17	0.7
12	25	0	17	1.5
13	25	2	9	0.7
14	25	2	9	1.5
15	25	2	17	0.7
16	25	2	17	1.5
17	30	0	9	0.7
18	30	0	9	1.5
19	30	0	17	0.7
20	30	0	17	1.5
21	30	2	9	0.7
22	30	2	9	1.5
23	30	2	17	0.7
24	30	2	17	1.5

We remove a Window W from f , if the number of w 's (Gabor weights) with a value 0 (i.e., no intensity) in $W \geq 70\%$. It means, the intensity of code is very less in this Window and will not contain any useful code for comparison.

To select important Windows (Gabor features) out of this set of Windows that help us classify malware, we compute the frequencies Window Frequency (WF) and Document Frequency (DF) of a Window W_j in a filtered image F_i as follows:

$$WF_j = \frac{f_j}{p} \quad \text{and} \quad DF_j = 1 + \frac{M_i}{N}$$

Where, f_j is the number of times (frequency) W_j appears in a filtered image F_i ; and M_i is the number of all the filtered images F_i in the N APKs with W_j in it. After computing the WF_j and DF_j we assign a weight to window W_j as follows:

$$Weight_j = WF_j \times DF_j$$

We only keep W_j , if $Weight_j > 0.3$. This minimum value of $Weight_j$ is computed empirically. We carried out a small experiment with different values of $Weight_j$. We found a value > 0.3 as the optimal value, i.e., a W_j with the final weight of > 0.3 does not make any difference in the detection rate.

Similarly, we select specific Windows, satisfying Equation (2), from each filtered image of an APK. Let

a filtered image, after preprocessing and selection, $sf = \{W_1, W_2, W_3, \dots, W_q\}$, where q is the number of features (Windows) selected satisfying Equation (2) from the feature f . Then, $SGF_k = \{sf_1, sf_2, sf_3, \dots, sf_n\}$ is the set of selected features of the k th APK. We build (extract and select) these set of features separately for each set of benign and malware APKs in the dataset.

Common Features out-To further improve feature selection and classification accuracy, we take the features from malware out that are also present in benign. We define the set of features of all the benign and malware APKs in the dataset as $BF = \{SGF_1, SGF_2, SGF_3, \dots, SGF_a\}$ and $MF_{old} = \{SGF_1, SGF_2, SGF_3, \dots, SGF_b\}$ respectively. We build the new MF as follows.

$$MF = \{m \mid m \in MF_{old} \wedge m \notin BF\} \quad (2)$$

After taking the common features out, we build the set of final selected features for the complete dataset as follows.

$$SF = \{r \mid r \in MF \vee r \in BF\} \quad (3)$$

3.5. Classifications

Before performing classification, we further improve our model and apply Recursive Feature Elimination (RFE) to improve the accuracy of the classification. We used a greedy optimization algorithm which aims to find the set of best-performing features at each iteration. Our RFE technique iteratively

1. Trains the classifier.
2. Computes the rank for all the features.
3. Removes the feature having the smallest rank.

It repeats this process until the desired set (number) of features is eventually reached. To pick which classifier to use with RFE, we carried out a small experiment with 50 samples, out of which 25 were benign and 25 malware programs. These 50 samples were picked randomly from the dataset used in this paper. We build the set of features using Equation (3) for these 50 samples and the distribution is shown in Figure 3. For these 50 samples, 2455 features were selected. The number of selected features varies with the size of the dataset. The distribution of these selected features shown in Figure 3 reveals that the selection is almost balanced. It means, the dataset is balanced, same (25) number of samples in each class, and the feature distribution is also balanced, i.e., almost the same (~1250) number of features were selected (observed) for each class.

We can see from Figure 3 the benign and malware features can be separated by almost a straight line, and a linear classifier is best for such a distribution. In general, a Support Vector Machine (SVM) classifier is used with RFE. Therefore, we used a linear SVM classifier with 10-fold cross-validation to pick the best set, out of these features. The selected best features after applying RFE were used to build the model for

classification. We used three different classifiers (NaiveBayes, AdaBoost, and linear SVM) for training and testing to evaluate the accuracy of our model. The use of RFE improved the proposed model's classification accuracy by more than 5%.

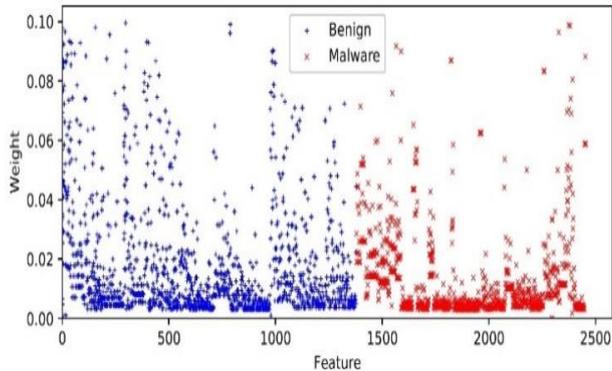


Figure 3. Distribution of the features built from 25 benign and 25 malware samples using equation 3. To save space and make the graph clearer, the weights are normalized.

4. Experimental Evaluation

We carried out an empirical study to analyze the correctness and efficiency of our approach. We performed two experiments, each using a different validation technique. In the first experiment, we used 80-20 train and test split validation and in the second experiment, we used k-fold cross-validation. Intel Core (TM) i-7-4510U CPU @ 2.00 GHz with 8 GB of RAM running Windows 8.1, was used to run all the experiments. In this section, we present the evaluation metrics, dataset, Window size computation, the empirical study (the two experiments), obtained results, and analysis.

4.1. Metrics

To evaluate the performance and accuracy of our model we use and define the following metrics. Detection Rate (DR), also called the true positive rate, is the percentage of samples correctly recognized as malware out of the total malware dataset. False Positive Rate (FPR) is the percentage of samples incorrectly recognized as malware out of the total benign dataset. Accuracy is the fraction of samples, including malware and benign, that is correctly detected as either malware or benign. Receiver Operating Characteristic (ROC) curve is a graphical plot used to depict the performance of a binary classifier. Area Under the ROC Curve (AUC) [7] is the probability that a detector/classifier will correctly classify a sample.

4.2. Dataset

Our dataset for the experiments consists of 1402 Android applications. Out of these, 701 are real Android malware programs collected from two

different resources [22, 34], and the other 701 are benign programs containing applications downloaded from Google Play, Android system programs, and shared libraries. Table 2 shows distribution of the 701 malware samples. Partitioning of the dataset for different experiments, including Window size computation, 80-20 validation and k-fold cross-validation is shown in Table 3. The dataset contains samples from a variety of malware families. Most of the Android byte code malware families are piggybacked applications (Geinimi, jSMShider, ARDR, Pjapps and all of the DroidKungFu families, DroidDreamLight, DroidDream). KMin and YZHC classes are standalone malware applications.

4.3. Window Size Computation

As mentioned in section 3.4, for selecting important features we use a Window that slides over the extracted Gabor filtered images and helps us classify malware. This Window contains the Gabor weight of pixels in the Gabor filtered image. In this section, we discuss the method used to select the size of this Window.

Table 2. Distribution of 701 Android malware samples.

Class/family	Number of samples
ARDR	19
AnserverBot	155
BaseBridge	24
DroidDream	13
DroidDreamLight	42
DroidKungFu1	23
DroidKungFu2	27
DroidKungFu3	214
DroidKungFu4	63
Geinimi	43
jSMShider	12
KMin	9
Pjapps	35
YZHC	22

To compute the Window size for our dataset, we randomly picked 200 samples (100 benign and 100 malware) from the training dataset. To optimize the results and pick the best Window size automatically we used a linear SVM classifier. During this testing, we perform 10 iterations, each time with a different 20 samples in the testing set and the rest of the 180 samples in the training set. Size of the Gabor filtered images for our dataset ranges from ~1K – ~3M bytes. Therefore, we repeated these 10 iterations 15 times, each time with a different Window size, ranging from 100–1500 with a step of 100. Our priority is the DR, therefore at the end of this experiment a Window size of 1000 was picked based on the best DR results. In the rest of the experiments, we used a Window size of 1000.

Table 3. Partitioning of the dataset (total 1402 samples \Rightarrow 701 benign and 701 malware) for different experiments.

Experiment	Total number of samples (benign/malware)	Training samples	Testing samples
Window size computation	200 ¹ (100/100)	180	20
80-20 validation	1402 (701/701)	1122	280
K-fold cross-validation	1402 (701/701)	1122	280 ²

¹The 200 samples were randomly picked from the training dataset.

²In this experiment we perform 10 ($K = \{5, 10\}$) iterations, each time with a different 280 samples in the testing set and the remaining 1122 samples in the training set.

4.4. Cross-Validation

We carried out two experiments, 80-20 train and test split validation, and stratified k-fold cross-validation.

For the first experiment, we chose the classic 80-20 train and test split validation. We divided the dataset randomly into two subsets. One (80%) for training and other (20%) for testing. The result of this validation with three different classifiers are shown in Table 4. Our dataset contains different types of malware. It is possible, with this type of validation, that one subset may not contain certain types of malware. This may result in overfitting. To avoid these and other problems we carried out the second experiment.

For the second experiment, we chose stratified k-fold cross-validation because of the following reasons: To include the complete dataset (not just 80% for training and 20% for testing) in training and testing; generalize the testing of the proposed predictive model to an independent dataset; limit the problems of overfitting and selection bias; and test the proposed technique's ability to predict new malware.

For this experiment, we divided the dataset randomly into k folds of equal size. With each fold we partitioned the dataset into two different complementary subsets, performing training on one subset and testing on the other subset. To make sure that each fold has the same proportion of samples from each class (benign and malware), we stratified the dataset. This process of cross-validation was repeated k times, with each of the k folds used exactly once for validation. We repeated this complete process twice, each time with a different k (5 and 10). The results of these 5-fold and 10-fold cross-validations with three different classifiers are shown in Table 4.

Table 4. Results of the proposed model using 80-20 train and test split validation, 5-fold and 10-fold cross validations with three different classifiers.

Classifier	DR	FPR	Accuracy	AUC
80-20 train and test split validation with RFE				
NB	99.80%	0	99.90%	0.99
AdaBoost	90.97%	0	95.37%	0.96
L-SVM	82.91%	0	90.39%	0.91
5-fold cross-validation with RFE				
NB	98.75%	0	99.36%	0.99
AdaBoost	92.21%	0	95.72%	0.95
L-SVM	86.88%	0	92.37%	0.92
10-fold cross-validation without RFE				
NB	93.65%	0	94.35%	0.94
AdaBoost	86.54%	0	87.40%	0.89
L-SVM	82.20%	0	85.34%	0.86
10-fold cross-validation with RFE				
NB	98.90%	0	99.43%	0.99
AdaBoost	92.33%	0	95.79%	0.95
L-SVM	88.49%	0	93.366%	0.93

NB=Naive Bayes, L-SVM=Linear SVM

As shown in Table 4, we almost got perfect results with the NaiveBayes (NB) classifier. NB is the simplest of all the Bayesian classifiers and assumes (naively) that all the features (attributes) of the samples are independent of each other given the context of the class [15]. Because of this assumption, NB learns the parameters for each feature separately, which greatly simplifies learning. This makes NB perform well when the number of features is very large. As in our case, the number of features increases as we increase the samples. For example, our model generated 2455 (shown in Figure 3), 7427 and 32027 features for 50, 200 and 1402 samples, respectively.

The classifier Decision Tree (DT) was used in the AdaBoost as the base estimator. A tree in a DT can be very complex and may not generalize well from the training data, i.e., a small change in the training data may result in a large change in the DT, and hence effects the final predictions [8].

SVM measures the complexity of a hypothesis based on the margin with which they separate the data [9]. Therefore, SVM works well if the data is separable with a wide margin. Our set of features is almost separable (as shown in Figure 3) but the margin of separation is not wide.

All the three classifiers used in this paper, perform well when the number of features is large with sparse instances. The set of features generated by the proposed model in this paper complies with these two properties, and hence our model gets an accuracy > 90% with all the three classifiers (with RFE).

5. Comparison with Other Works

Table 5 shows a comparison of our technique with four other similar Android malware detection techniques discussed in section 2. The reasons for including these works are:

1. All of them are using Gabor filters to detect malware.

2. Have used machine learning to improve the performance.
3. Have reported at least the DR and accuracy obtained.

Table 5. Comparison of our technique with four other similar Android malware detection techniques discussed in section 2.

Technique	Total number of samples (benign/malwre)	DR	FPR	Accuracy
Our technique ¹	1402 (701/701)	98.9%	0%	99.43%
Naeem <i>et al.</i> [18]	6000 (2000/4000)	97.5%	3%	96.30%
Fang <i>et al.</i> [6]	3000 (0/3000)	96.0%	NA	96.00%
Kumar <i>et al.</i> [11]	246 (108/138)	93.0%	10%	91.00%
Darus <i>et al.</i> [4]	600 (300/300)	69.1%	6.7%	84.14%

¹The results of our technique reported here are obtained with NaiveBayes classifier with RFE using 10-fold cross-validation.

Kumar *et al.* [11] and Fang *et al.* [6] used 10-fold and 5-fold cross-validation, respectively. The other two works [4, 18] did not mention about the validation technique used during their experimentation. All the techniques achieved an accuracy > 90%, except [4]. Fang *et al.* [6] used only malware samples for evaluation and did not use any benign samples, hence no FPR is reported in Table 5. Naeem *et al.* [18] decompiles an APK to get the Java source code, whereas we just decompress the APK to get the DEX bytecodes, for translating to grayscale image. This makes our technique much safer and accurate.

All the four techniques translate an APK (after preprocessing to extract either DEX bytecode or Java source code) to a grayscale image for applying Gabor filters. After translating, the GIST descriptor [21] is applied to extract a particular set of Gabor features. GIST descriptor also extracts features through Gabor filters but with a particular set of Gabor parameters. Whereas our technique does not use the GIST descriptor but extract different set of Gabor features. Therefore, the four techniques used almost similar (except few variations) set of Gabor parameters to extract features. This employ, that during training they were not able to fine tune the Gabor parameters and hence the selected features, as we did and were able to achieve a much higher accuracy.

Only two of the works [6, 18] have used a larger number of malware samples than used in this paper for evaluation. Naeem *et al.* [18] has not used any cross-validation to evaluate their technique, and [6] has not used any benign samples for validation. Whereas, to achieve systematic and precise testing, and to reduce any bias and unbalancing in the dataset, the evaluation carried out in this paper includes different cross-validation techniques with real malware and benign samples. Moreover, to further reduce the bias and unbalanced in the dataset, we use the same number of malware and benign samples.

5.1. Limitations

Our proposed approach is based on static analysis of a sample; therefore, it requires that the malicious code be available for static analysis. If an Android application requires to download malicious code upon initial execution (dynamic code loading), then the sample will not be correctly analyzed by the system. If an Android application dynamically (while executing) link a third-party library, which is not included in the application, then the library will not be processed. Our technique excels at detecting malware that has been previously known but will only detect an unknown (zero day) malware, if its structure is similar, up to a threshold, to an existing malware sample in the training database. Encryption and compression change the byte structure of a program, and hence our technique may not be able to classify such malware.

6. Conclusions

In this paper we have proposed a technique based on static analysis of a grayscale image of the Android APK. We apply different Gabor filters to this image for extracting features. A Window is slid over these to select important features for classifying the APK. To evaluate the performance of our technique we apply different cross-validations and compare it with four other similar techniques. Our technique when evaluated with real malware and benign samples using 10-fold cross-validation achieves an accuracy of 99.43%, outperforming the four other techniques.

One of the major challenges in securing IoTs (Internet of things) is the development of cross-architecture (x86, ARM, MIPS and PowerPC etc..) malware analysis and detection system. In future, we will explore the possibility of using similar image analysis technique (as used in this paper) for classifying IoT cross-platform malware applications. In this current work selected features are used for binary classification. We would like to improve our current feature selection method for multinomial classification, i.e, not only classifying malware programs as just malware but also classify them into their respective families. As mentioned in section 3 we parallelized feature selection component of our system. In future, to further optimize the system, we will also parallelize other components of the system, such as feature extraction etc.

References

- [1] Abuthawabeh M. and Mahmoud K., "Enhanced Android Malware Detection and Family Classification, Using Conversation-Level Network Traffic Features," *The International Arab Journal of Information Technology*, vol. 17, no. 4A, pp. 607-614, 2020.
- [2] Chen H., Du R., Liu Z., and Xu H., "Android

- Malware Classification Using Xgboost Based on Images Patterns,” in *Proceedings of 4th Information Technology and Mechatronics Engineering Conference*, Chongqing, pp. 1358-1362, 2018.
- [3] Clausi D. and Jernigan M., “Designing Gabor Filters for Optimal Texture Separability,” *Pattern Recognition*, vol. 33, no. 11, pp. 1835-1849, 2000.
- [4] Darus F., Ahmad S., and Ariffin A., “Android Malware Detection Using Machine Learning on Image Patterns,” *Cyber Resilience Conference*, Putrajaya, pp. 1-2, 2018.
- [5] Daugman J., “Image Analysis and Compact Coding by Oriented 2d Gabor Primitives,” in *Proceedings of Image Understanding and the Man-Machine Interface*, Los Angeles, pp. 19-30, 1987.
- [6] Fang Y., Gao Y., Jing F., and Zhang L., “Android Malware Familial Classification Based on Dex File Section Features,” *IEEE Access*, vol. 8, pp. 10614-10627, 2020.
- [7] Fawcett T., “An Introduction to ROC Analysis,” *Pattern Recogn.*, vol. 27, no. 8, pp. 861-874, 2006.
- [8] James G., Witten D., Hastie T., and Tibshirani R., *An Introduction to Statistical Learning*, Springer, 2013.
- [9] Joachims T., “Text Categorization with Support Vector Machines: Learning with Many Relevant Features,” in *Proceedings of European Conference on Machine Learning*, pp. 137-142, 1998.
- [10] Kancherla K. Mukkamala S., “Image Visualization Based Malware Detection,” in *Proceedings of Symposium on Computational Intelligence in Cyber Security*, Singapore, pp. 40-44, 2013.
- [11] Kumar A., Sagar K., Kuppusamy K., and Aghila G., “Machine Learning Based Malware Classification for Android Applications Using Multimodal Image Representations,” in *Proceedings of 10th International Conference on Intelligent Systems and Control*, Coimbatore, pp. 1-6, 2016.
- [12] List of Cyber Security Statistics for 2019. <https://purplesec.us/resources/cyber-security-statistics/>, Last Visited, 2020.
- [13] Makandar A. and Patrot A., “Malware Class Recognition Using Image Processing Techniques,” in *Proceedings of International Conference on Data Management, Analytics and Innovation*, Pune, pp. 76-80, 2017.
- [14] Makandar A. and Patrot A., “Malware Image Analysis And Classification Using Support Vector Machine,” *International Journal of Trends in Computer Science and Engineering*, vol. 4, no. 5, pp. 1-3, 2015.
- [15] McCallum A. and Nigam K., “A Comparison of Event Models for Naive Bayes Text Classification,” *AAAI-98 Workshop on Learning for Text Categorization*. vol. 752, pp. 41-48, 1998.
- [16] Naeem H., Guo B., Naeem M., Ullah F., Aldabbas H., and Javed M., “Identification of Malicious Code Variants Based on Image Visualization,” *Computers and Electrical Engineering*, vol. 76, pp. 225-237, 2019.
- [17] Naeem H., Guo B., and Naeem M., “A Light-Weight Malware Static Visual Analysis for Iot Infrastructure,” in *Proceedings of International Conference on Artificial Intelligence and Big Data*, Chengdu, pp. 240-244, 2018.
- [18] Naeem H., Guo B., Ullah F., and Naeem M., “A Cross-Platform Malware Variant Classification Based on Image Representation,” *KSII Transactions on Internet and Information Systems*, vol. 13, no. 7, pp. 3756-3777, 2019.
- [19] Nataraj L., Karthikeyan S., Jacob G., and Manjunath B., “Malware Images: Visualization and Automatic Classification,” in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, New York, pp. 1-7 2011.
- [20] Nataraj L., Yegneswaran V., Porras P., and Zhang J., “A Comparative Assessment of Malware Classification Using Binary Texture Analysis and Dynamic Analysis,” in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, New York, pp. 21-30, 2011.
- [21] Oliva A. and Torralba A., “Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope,” *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145-175, 2001.
- [22] Parkour M., “Mobile Malware Dump,” <http://contagiomindump.blogspot.com>, Last Visited, 2020.
- [23] Randen T. and Husoy J., “Optimal Filter-Bank Design for Multiple Texture Discrimination,” in *Proceedings of International Conference on Image Processing*, Santa Barbara, pp. 215-218, 1997.
- [24] Samani R., “McAfee Mobile Threat Report,” <https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf>, Last Visited, 2020.
- [25] Sharma S., Challa R., and Kumar R., “An Ensemble-Based Supervised Machine Learning Framework for Android Ransomware Detection,” *The International Arab Journal of Information Technology*, vol. 18, no. 3A, pp. 422-429, 2021.
- [26] Stimson A., *Photometry and Radiometry for Engineers*, Wiley-Interscience, 1974.
- [27] Team A., Android Dalvik Virtual Machine Opcodes,

- <http://developer.android.com/reference/dalvik/bytecode/Opcodes.html>, Last Visited, 2020.
- [28] Temes G. and Mitra S., *Modern Filter Theory and Design*, Wiley-Interscience, 1973.
- [29] Teuner A., Pichler O., and Hosticka B., "Unsupervised Texture Segmentation of Images Using Tuned Matched Gabor Filters," *IEEE Transactions on Image Processing*, vol. 4, no. 6, pp. 863-870, 1995.
- [30] Tukey J., *Exploratory Data Analysis*, Pearson, 1977.
- [31] Weldon T. and Higgins W., "Design of Multiple Gabor Filters for Texture Segmentation," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, Atlanta, pp. 2243-2246, 1996.
- [32] Yoo I., "Visualizing Windows Executable Viruses Using Self-Organizing Maps," in *Proceedings of the ACM Workshop on Visualization and Data Mining for Computer Security*, New York, pp. 82-89, 2004.
- [33] Zhou X., Pang J., and Liang G., "Image Classification For Malware Detection Using Extremely Randomized Trees," in *Proceedings of 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification*, Xiamen, pp. 54-59, 2017.
- [34] Zhou Y. and Jiang X., "Dissecting Android Malware: Characterization and Evolution," in *Proceedings of IEEE Symposium on Security and Privacy*, San Francisco, pp. 95-109, 2012.



Shahid Alam is currently working as an assistant professor in the department of Computer Engineering at Adana Alparslan Turkes Science and Technology University, Adana, Turkey. He received his PhD in Computer Science from University of Victoria, Canada in 2014. His research interests include software engineering, programming languages, computer security, and malware analysis and detection. He has published several journal and conference papers in these areas. Currently he is looking into applying compiler, binary analysis, and machine learning techniques to automate and optimize malware analysis and detection.



Alper Kamil Demir is an Assoc. Prof. at Computer Engineering Department of Adana Alparslan Turkes Science and Technology University since 2013. Between 2009 and 2013, he worked at Huawei Telecommunications Inc. as a Senior Software and Research Engineer. Between 2001 and 2009 he worked at Kocaeli University, Computer Engineering Department. He is interested in Security and Forensics, Computer Networks, Distributed Systems and Operating Systems in general. Currently, his research is focused on Internet of Things.