

Identification of Android APK malware through local and global feature extraction using meta classifier

Yoga Herawan, Imas Sukaesih Sitanggang, Shelvie Nidya Neyman

Department of Computer Science, Faculty of Mathematics and Natural Science, IPB University, Bogor, Indonesia

Article Info

Article history:

Received Mar 23, 2024

Revised Oct 18, 2024

Accepted Nov 20, 2024

Keywords:

APK Android visualization

Global feature extraction

Local feature extraction

Malware

Meta classifier

ABSTRACT

Android, the most widely used mobile operating system, is also the most vulnerable to malware due to its high popularity. This has significantly focused on Android malware detection in mobile security. While extensive research has been conducted using various methods, new malware's emergence underscores this field's dynamic nature and the need for continuous research. The motivation that drives malware developers to create Android malware constantly is the potential to access Android devices, thereby gaining access to sensitive user information. This study, which is a complex and in-depth exploration, aims to detect Android malware using a meta-classifier that combines the single-classifier light gradient boosting machine, support vector machine, and random forest. The process involves converting disassembled malware codes into grey images for global and local feature extraction. The classification accuracy is 97% at best on a malware dataset of 3,963 samples. The main contribution of this paper is to produce an Android APK malware detector model that works by combining multiple machine learning algorithms trained using the dataset resulting from local and global feature extraction algorithms.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Imas Sukaesih Sitanggang

Department of Computer Science, Faculty of Mathematics and Natural Science, IPB University

Wing 20 Level 5 Meranti Road, Dramaga District, Bogor 16680, Indonesia

Email: imas.sitanggang@apps.ipb.ac.id

1. INTRODUCTION

Android is the most widely used mobile operating system today. Based on information from StatCounter Global Stats, until June 2024, Android became the mobile operating system with the largest market share in the world, which amounted to 72.15%, beating iOS at 27.19%. The open-source Android operating system and free license make it more popular. The high popularity of Android is proportional to the increased risk of virus and malware attacks. Furthermore, the technology for framework-based application development is advancing rapidly, offering a convenient way to create Android-based applications. Those who wish to take advantage of this convenience often reverse-engineer completed applications and add program logic to create new applications quickly. However, this convenience also presents a risk, as some individuals may upload malicious scripts that can exploit the data on Android handsets. Nowadays, users frequently download apps from any location, leading to issues. Installing apps and permitting them to do anything they ask without understanding their purpose is what leads to security issues on smartphones, such as script intrusion. It can perform a security-violating operation, such as starting a covert data transfer, allowing attackers to obtain private information from devices. At some point, attackers will take control of the smartphone.

Software that seeks to obstruct regular computerized system operations by obtaining private information gaining illegal access to computer systems, and mostly harassing users is known as malware, short for malicious software [1]. The exponential growth of malware forces computer security researchers to develop new methods for protecting networks and systems. Malware identification is essential because of its exponential growth and ability to inflict extensive damage on a wide surface area [2]. Researchers have created methods for detecting malware as a result of the growth of malware [3]. There are three malware detection approaches: static, dynamic, and hybrid. Static analysis examines the alleged code without launching the program [4]. For feature extraction, the source code must be disassembled by analyzing permission usage [5], mining the code structures [6], analyzing the used components [7], and monitoring the application programming interface (API) invoked [7], [8]. It is a gateway within the Android architecture to access sensitive services [9]. Unlike static analysis, dynamic analysis examines the application's attributes and scrutinizes the events that transpire during its execution [4], [10], [11].

Harmful apps must be stripped of their intricate features and hidden structures to identify malware using new, effective detection techniques [12]. The advancement of the internet and technology will make the potential threat of Android APK malware even more significant, making it necessary to develop a machine learning-based malware detection system. The diverse anatomical structure of Android APK files makes the combination of local and global feature extraction techniques an opportunity to help produce accurate and fast models. Malware classification using machine learning (ML) is one of the technologies that artificial intelligence (AI) has made available for use in malware research. Data mining was first proposed in 2001 to identify malware on Windows operating systems [13]. The research employed RIPPER, naïve Bayes, and multi-classifier system as a machine learning method. The multi-naïve Bayes approach yields the best accuracy of 97.76% in testing.

In place of static and dynamic approaches, researchers have studied malware using visualization-based techniques. Large-scale malware detection can be achieved more effectively by visual analysis, which uses the structure of malware images rather than static or dynamic approaches. Computer malware classification method by visualizing binary malware in grayscale images [14]. This approach differs from malware experts' widespread static and dynamic detection approaches. These techniques are being introduced to provide a better alternative to the drawbacks of the static and dynamic techniques. Using the k-nearest neighbor (KNN) technique, 9458 samples were employed in his research, yielding an accuracy rate of 98%.

The executable file in malware was converted into a grayscale image for feature extraction using wavelet transform and Gabor filter [15]. The machine learning algorithm used is support vector machine (SVM), which has an accuracy of 95%. A lightweight malware detection method that makes it possible to detect malware from smartphones [7]. In their research, each feature is taken from the components of the *AndroidManifest.xml* file and the *classes.dex* file is collected into a space vector for an SVM approach with 94% accuracy. Android network flows and API calls were tested for malware detection with Android permissions and intent features [16]. In addition to providing accuracy results of 95.3% for static analysis and 83.3% for dynamic analysis on malware detection on the Android system, the study also offered the publicly accessible CICAndMal2017 dataset while performing dynamic analysis on actual smartphones. Regarding malware detection using APK image visualization, the random forest (RF) algorithm provides the best performance with an accuracy of 92.81% [17]. A malware detection framework based on a stack of ensemble learning algorithms called MFDroid [18]. The study uses seven feature extraction algorithms based on Permission, API Calls, and Opcode datasets and then produces an F1 Score of 96% using a meta-classifier.

A machine-learning approach based on feature extraction and grayscale picture categorization is used to identify Android APK malware [19]. This study turns most APK files—including multiDex, resources, certificates, and manifest files—into grayscale pictures to detect malware. Subsequently, local malware picture features are extracted using local feature extraction techniques. The accuracy of the model yields outcomes that are 96.86%. Conventional and ensemble machine learning algorithms are compared to determine the accuracy of the matrices comparison [20]. The Drebin dataset, which had 215 statically evaluated features, was employed in his research. The input data shows the attribute's availability in the dataset and is accessible in 0 and 1 formats. According to the investigation, the ensemble method performs better at predicting malware than conventional machine learning methods. Random forest yielded 99.1% accuracy, whereas light gradient boosting machine (LGBM) produced 99.5%.

Numerous research works on Android malware detection APKs have been published in academic journals. Earlier studies employed different APK components as datasets and then analyzed them statically or dynamically using machine-learning techniques. The primary motivation of this study is to know the impact of combining different machine-learning techniques on the outcomes of local and global extraction in the visualization of Android APKs. This study detects malware in Android APK files using a meta-classifier technique combining local and global dataset features, integrating the random forest, SVM, and LGBM methods. The remaining sections of the paper are structured as follows. The methodology used is explained in section 2. The analysis results and discussions are presented in section 3, and the conclusion in section 4.

2. THE PROPOSED METHOD

The main objectives of this study are to assess and analyze raw data APK as local and global features to improve the detection accuracy and performance of meta-classifiers. The study was conducted in three primary phases, each meticulously designed and executed: data preprocessing, machine learning, and model evaluation. Two methods for extracting features are available during the preprocessing stage: local and global. The learning process employs a single-classifier or meta-classifier at the machine-learning step. In order to construct a meta-classifier model, the single-classifier model's detection results will serve as a data source. Figure 1 is a big picture of creating a system to identify whether Android APK files are malware or benign. Model performance measurements are measured based on four primary criteria: accuracy, F1 score, recall, and precision. The time comparison procedure (measured in seconds) and the amount of random-access memory (RAM) used during the testing process are two additional complementing criteria in addition to the four primary measurement criteria for the model performance analysis.

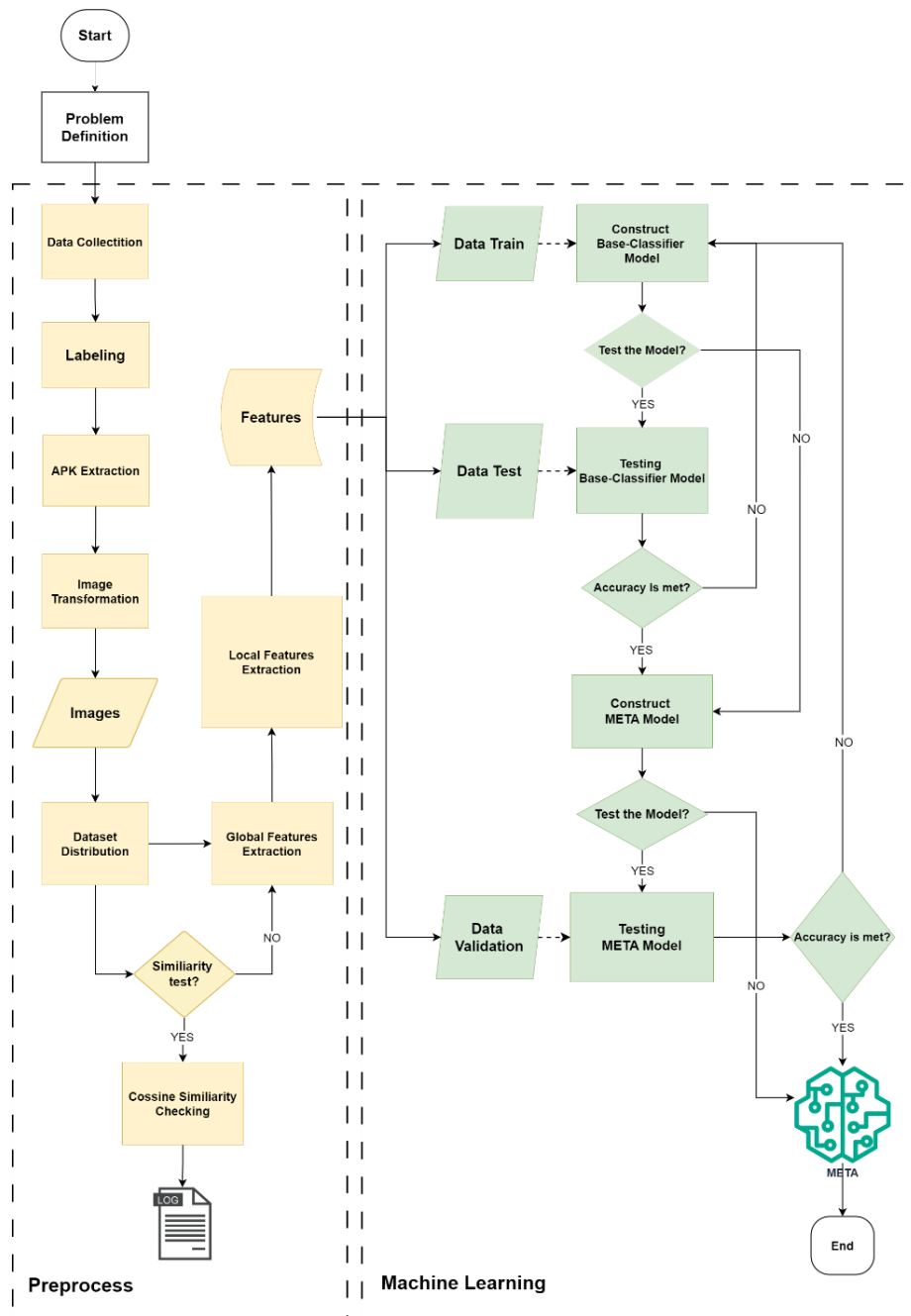


Figure 1. Steps of creating malware detection system

3. METHOD

3.1. Research scenario

A series of experiments need to be carried out to achieve the main objectives of this research. In addition, it can help obtain a model with the most optimal performance. Table 1 describes the experimental settings that were conducted in this research.

Table 1. Experimental settings

Scenario	Explanation
First experiment	Default experiment utilizing as-is datasets
Second experiment	The First experiment using the balanced dataset
Third experiment	Second experiment with grid search hyperparameter configuration applied to the base-classifier algorithm
Fourth experiment	Second experiment with random search hyperparameter configuration applied to the base-classifier algorithm

3.2. Data preprocessing

This research uses open datasets that can be freely downloaded [16]. The dataset is downloaded using the following link: <http://205.174.165.80/CICDataset/CICInvesAndMal2019/Dataset/APKs/> and <http://205.174.165.80/CICDataset/MalDroid-2020/Dataset/APKs/>. The 15.9 GB dataset used in this research includes 3,963 APKs classified as benign and malicious in ZIP Format. This analysis only handled the following categories of malware: adware, ransomware, SMS malware, and scareware. The collected data is then preprocessed to produce the .dex file type. Since all Java executable code is stored in files with the extension “.dex,” the DEX file shares many characteristics with an Android application [21]. Figure 2 shows the procedure for creating grayscale images from zip data sources.

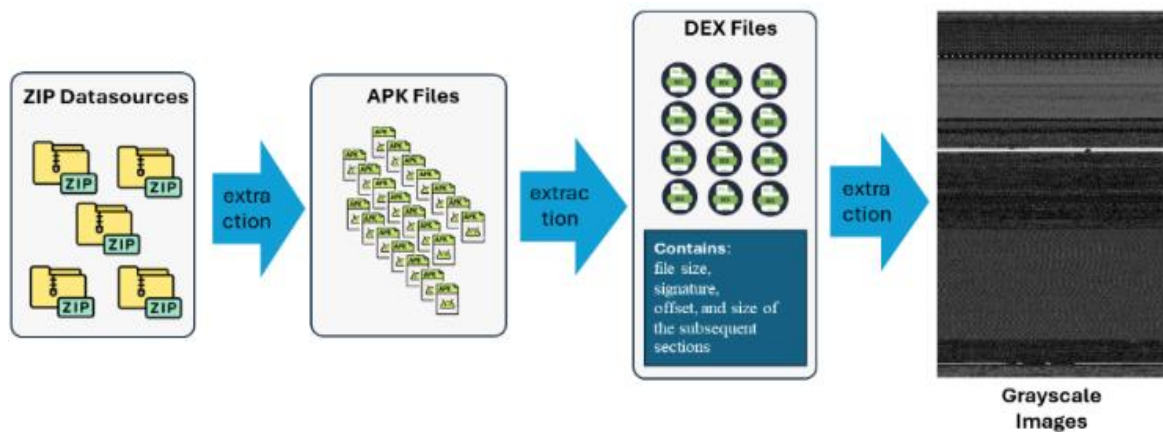


Figure 2. Procedure for creating grayscale images from zip data sources

The dataset distribution is split into training, testing, and validation, with the percentages being 72%, 20%, and 8%, respectively. The base-classifier model is constructed using the training dataset. The testing dataset, which accounts for 20% of the distribution, is crucial in training the meta-classifier model. The base-classifier model determines this model's accuracy. The validation dataset is then used to evaluate the meta-classifier model and assess its accuracy.

3.3. APK visualization

The header section, ids sections, *class_defs* sections, and data sections are among the sections that make up a DEX file. The header section contains top-level data, such as the subsequent sections' file size, signature, offset, and size. The system must gather the file size to determine the image's width before converting the data into images, using the references in Table 2. The data section is a crucial part of *classes.dex* contains class data and execution code, representing the application's behavior, whether it is classified as malware or benign application [22]. The desired information is kept in the *dex_header* of the *classes.dex* file may be retrieved by parsing the *dex_header* to obtain the *data_size* and *data_offset*.

Table 2. Image width recommendation [14]

File size range	Image's Width
< 10 kB	32
10 kB – 30 kB	64
30 kB – 60 kB	128
60 kB – 100 kB	256
100 kB – 200 kB	384
200 kB – 500 kB	512
500 kB – 1000 kB	768
>1000 kB	1024

The data size and offsets must be determined before section data can be extracted from the *classes.dex* file, are located at specific offsets. This information is obtained from the *classes.dex* file header. The header size of the *classes.dex* file is 112 bytes, and the data section's size and the data section's offset value are respectively located at offsets 104 and 108, as many as 4 bytes. Figure 3 provides a visual guide for digitally visualizing a raw APK file.

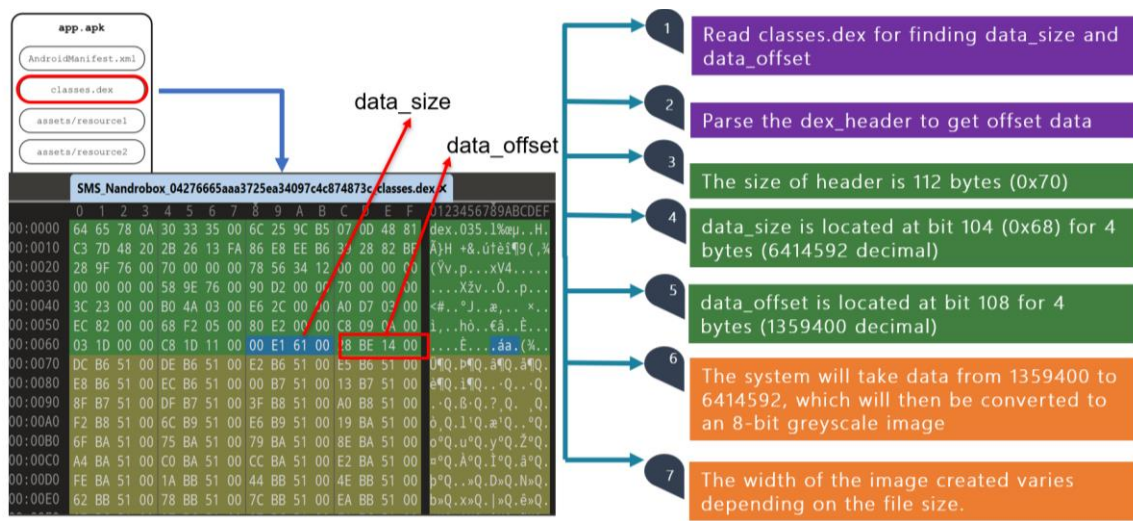


Figure 1. Procedures for turning an APK file into a digital picture

3.4. Extraction feature

Feature extraction is applied to the *.dex* file converted into a digital image to get the required information. This research uses two feature extraction methods: local and global. Local features characterize a narrow portion of the image, whereas global features describe the entire picture [23]. Meanwhile, the local feature method represents an image based only on a few prominent areas, which are constantly unaffected by viewpoint or changes in illusion [24].

3.4.1. Global feature extraction

The global features provide an overall description of the image to represent the complete object universally [25]. The GIST algorithm was used in this research to extract the global features of an APK visualization. A deficient dimensional representation of a given image called the spatial envelope is the foundation for the GIST image descriptor created by Torralba and Oliva [26]. The primary spatial structure of a scene can be represented using a series of perceptual dimensions, namely naturalness, openness, roughness, expansion, and ruggedness. The efficient estimation of these dimensions was demonstrated using coarsely localized and spectral information. This led to the creation of a multidimensional space for scenes. However, it is essential to note that while the GIST characteristics offered numerous benefits, there were also certain drawbacks, such as information loss in sparse grid computing [27]. The amplitudes of the output of K Gabor filters at various scales (S) and orientations (O) are combined to form the global descriptor, a significant aspect in the context of malware images. Each facial picture in the filter output is reduced to a size $N \times N$ block, yielding a vector with $S \times O \times N \times N$ dimensions to reduce the feature vector's size [28]. Figure 4 illustrates the operation of GIST on malware images.

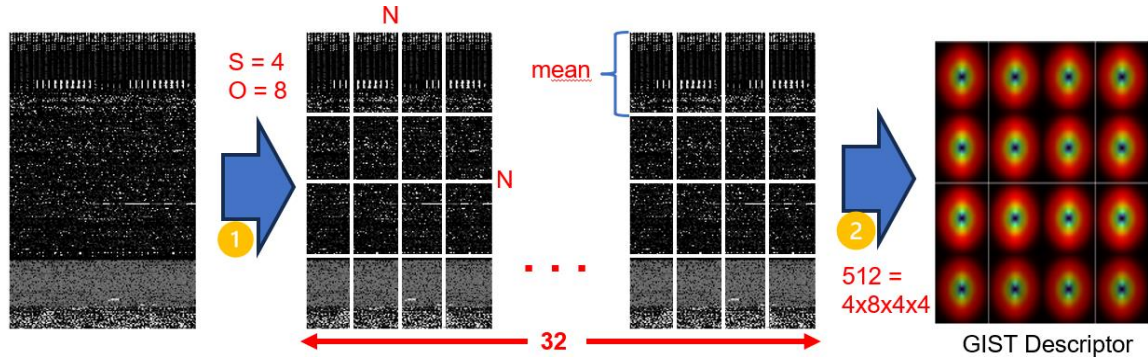


Figure 4. GIST feature extraction in malware image

3.4.2. Local feature extraction

The KAZE algorithm was used in this research to carry out local feature extraction. The image is described as batches of an object by the local features. These features are based on identifying numerous unique focal points within the picture, each described with a suitable and reliable descriptor [25]. In 2012, the non-linear scale-space KAZE algorithm was presented. The descriptor in KAZE is based on the local difference binary (LDB) descriptor, whereas the detector is based on the determinant of a Hessian matrix [29]. While lowering noise in the image, non-linear diffusion filtering preserves the borders of the regions at various sizes [30]. The KAZE algorithm uses the parameters in Table 3 to determine the descriptor values and crucial points.

Table 3. KAZE parameters used in the research

Parameter	Explanation	Values
Extended	To allow extended descriptor extraction (128-byte)	False
Upright	To allow the use of perpendicular (non-rotation-invariant) descriptors.	False
Threshold	The receiving point threshold for the detector response	0.001
nOctaves	Maximum octave evolution of the image	4
nOctaveLayers	Default number of sublevels per scale level	4
Diffusivity	Types of diffusivities. <i>DIFF_PM_G1</i> , <i>DIFF_PM_G2</i> , <i>DIFF_WEICKERT</i> or <i>DIFF_CHARBONNIER</i>	<i>DIFF_PM_G2</i>

An image's visual information can be found in the retrieved local features. A quantization approach such as k-means compresses the feature space to clusters for a compact picture representation. "Visual words" refer to the cluster centers [31]. The KAZE detection process produces a vector that contains a variety of data. However, the classifier algorithm only accepts n data and n-feature vectors as input. This means that the output from KAZE is not directly compatible with the classifier algorithm. The urgency of this problem necessitates a solution, which is to apply the bag of visual words (BOVW) to create a single feature vector from the many descriptors of local features [32].

Figure 5 explains the KAZE feature extraction process. BOVW employs a clustering algorithm to classify the vectors produced by the descriptor into the most suitable n-clusters. The algorithm predicts the classes of each descriptor data, and an essential step in the process is the formation of a histogram. This histogram is used to calculate the frequency of the class results indicated by the clustering algorithm. The final result is a normalized feature vector (local features) after applying BOVW, which can be used as a feature dataset for the classifier. The K-means clustering algorithm, a widely used method in unsupervised learning, is the algorithm of choice in this research.

3.5. Visualization-based machine learning model

In building a model, the base-classifier algorithm acts as the initial classifier. Next, the meta-classifier technique, a crucial step in the process, is used to construct the final predictions by utilizing the prediction results from the base-classifier method. Combining several classification models is then used as a new feature [33]. The results of the base classifier are combined and used to train the meta-classifier tasked with providing final prediction results [34]. SVM, RF, and LGBM were among the supervised learning algorithms employed in this research as base-classifiers. Moreover, the logistic regression technique, which serves as a meta-classifier, is employed in this study model.

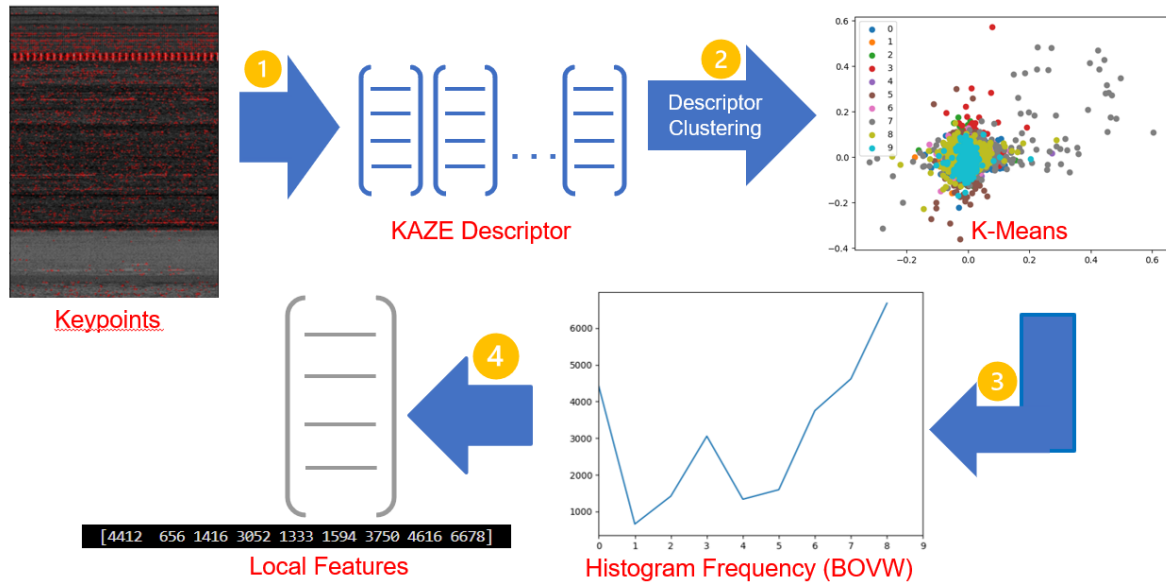


Figure 5. The KAZE features extraction process

3.5.1. Base-classifier

The dataset is processed using supervised learning techniques during the training and testing stages of creating the base-classifier model, which will produce each prediction model. The initial step in building the model is to run the training dataset for every algorithm to create a base-classifier model, a foundational model that will be used to make predictions. The third and fourth experiments tested hyperparameter tuning on the base-classifier model. The selection of appropriate hyperparameters can improve the accuracy and performance of the algorithm used [35]. Hyperparameter tuning determines the ideal value for the classifier hyperparameters to enhance the model's performance. Hyperparameters lower the total number of iterations, which can help to improve the mode's efficiency [36].

3.5.2. Meta-classifier

The performance of each base classifier produces varying levels of accuracy. Classifiers are integrated into one to increase the accuracy of the single classifier and improve its predicted performance [18]. Meta-classifiers combine decisions from several combination methods [37]. Logistic regression technique is utilized in the construction of the meta-classifier model. The meta-classifier of the logistic regression algorithm and a combination of base-classifiers have achieved the highest accuracy [38]. Figure 6 explains how the meta-classifier was implemented in this research to increase the accuracy of the base-classifier model.

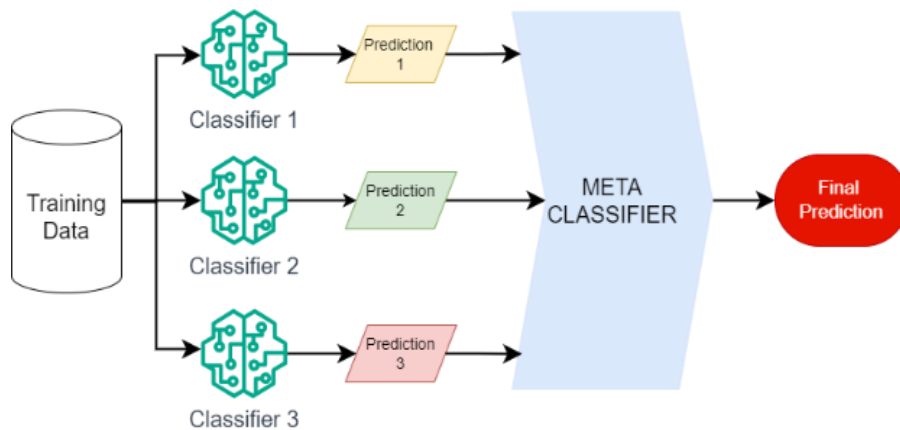


Figure 6. Flow diagram for the meta-classifier framework

4. RESULTS AND DISCUSSION

This section summarizes and discusses the main findings of the work. With a Ryzen 7 5800X CPU and 64 GB of RAM, IPB's computing server environment is used to run an experiment. This technical foundation, combined with the Python programming language and the main library packages for data science and mathematical computation, including Joblib, Matplotlib, NumPy, Scikit_learn, MLxtend, LightGBM, OpenCV-python, and Imbalanced-learn, allowed us to pre-process and visualize the raw Android APK, extracting local and global features. The machine-learning process led us to create the base-classifier and the meta-classifier models.

4.1. Image transformation

A hexadecimal reading on the trojan APK known as *SMS_Nandrobox* revealed that the *data_size* is 0061E100 bytes (reverse endian). The decimal representation of this value is 6414592. On the other hand, *data_offset* is 1359400 in decimal notation or 0014BE28 in hexadecimal. Data were taken from 6414592's offset 1359400 and processed by the system to create an 8-bit grayscale picture. The process of creating 8-bit grayscale images is as follows: a pixel is black if the data value per byte is 0x0000000 (0), white if the value is 0x11111111 (255), and grey if the value is between 0 and 255. Figure 7 shows the outcome of image transformation of adware family for *adware_youmi* in Figure 7(a) and *adware_shuanet* in Figure 7(b). Figure 8 shows the outcome of image transformation of SMS family for *SMS_fakenotify* in Figure 8(a) and *SMS_fakeinst* in Figure 8(b).

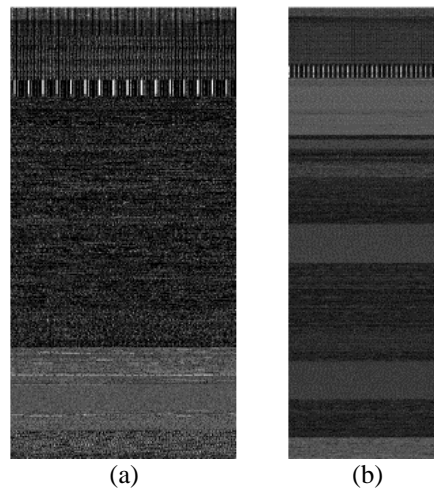


Figure 7. Image transformation results for malware (a) *adware_youmi* and (b) *adware_shuanet*

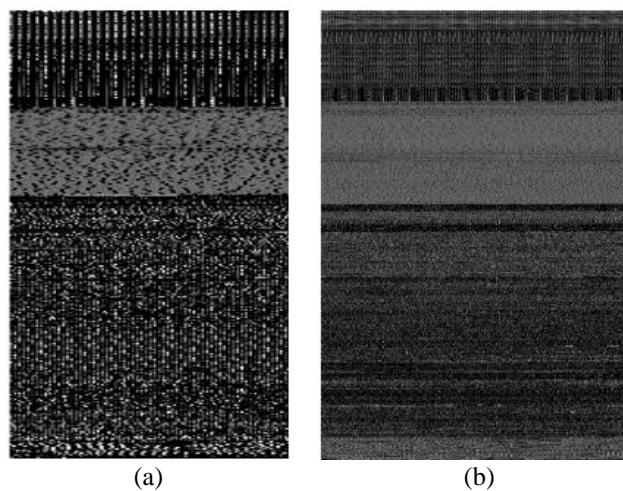


Figure 8. Image transformation results for malware (a) *SMS_Fakenotify* and (b) *SMS_FakeInst*

4.2. Global feature extraction

The GIST feature extraction technique extracts global features from a single grayscale image. This research processes up to 8 orientations using a four-scale Gabor filter, divides each feature map into 16 regions using a 4 × 4 grid, and then takes the average of each region's feature values. Concatenating the 16 averaged values of all 32 feature maps yields 4 × 8 × 4 × 4 = 512 GIST descriptors after determining each region's average feature values. Decimal numbers are kept in text files for every feature, as shown in Figure 9.

	0	1	2	3	4	5	6	7	8	9 ...	502	503	504	505	506	507	508	509	510	511	
0	0.101001	0.052823	0.098950	0.072584	0.099737	0.052043	0.102227	0.072075	0.101173	0.051900	...	0.014615	0.012201	0.021698	0.014214	0.014497	0.010978	0.019330	0.013088	0.015225	0.012184
1	0.065773	0.137039	0.066475	0.070932	0.063245	0.138565	0.065258	0.068877	0.061172	0.133038	...	0.015333	0.012129	0.014381	0.013285	0.014886	0.010516	0.014389	0.014821	0.013961	0.011854
2	0.067102	0.135522	0.067220	0.070119	0.062818	0.134869	0.066360	0.064529	0.063707	0.135926	...	0.015697	0.013094	0.015276	0.013268	0.015552	0.012982	0.013954	0.013304	0.014557	0.013003
3	0.064766	0.137204	0.064214	0.077412	0.060695	0.138474	0.062398	0.072702	0.061504	0.141859	...	0.014558	0.012571	0.013430	0.013855	0.015263	0.011795	0.015507	0.013551	0.015188	0.012085
4	0.061607	0.085295	0.065195	0.072921	0.062309	0.087724	0.065738	0.071586	0.061164	0.085251	...	0.011465	0.011809	0.010342	0.011269	0.012236	0.012167	0.009956	0.011037	0.011927	0.012490
...
836	0.050618	0.068241	0.050424	0.055856	0.051959	0.068477	0.047297	0.053585	0.051304	0.068000	...	0.010137	0.012674	0.007178	0.007890	0.010185	0.012311	0.007475	0.009617	0.011027	0.011634
837	0.057269	0.050216	0.081655	0.068958	0.054234	0.050467	0.080788	0.065542	0.054742	0.050523	...	0.009728	0.011165	0.007209	0.009015	0.010528	0.011261	0.007217	0.008626	0.010740	0.010811
838	0.055353	0.049436	0.075833	0.070267	0.056168	0.049773	0.072512	0.066570	0.057171	0.049795	...	0.009591	0.010720	0.006828	0.009610	0.009668	0.009670	0.006234	0.007830	0.010423	0.010576
839	0.047520	0.043529	0.078407	0.124843	0.048716	0.043772	0.079126	0.123262	0.049698	0.043892	...	0.008013	0.021578	0.005389	0.006975	0.008137	0.020930	0.006025	0.007099	0.007927	0.021065
840	0.060496	0.067587	0.077089	0.069098	0.060360	0.067520	0.076375	0.067112	0.058159	0.066030	...	0.011084	0.014400	0.010596	0.011463	0.011108	0.013189	0.010823	0.011460	0.011129	0.014986

841 rows × 512 columns

Figure 9. Grayscale image features processed against the GIST algorithm

4.3. Local feature extraction

The KAZE algorithm is instrumental in identifying the crucial points of the processed grayscale image. While not directly usable as feature selection outputs for an image, these identified critical spots produce the feature extraction value. This value is generated by constructing a codebook using the descriptor value to describe the identified critical point. The features created result from the bag of visual words (BOVW) construction procedure using the size ten dictionary (number of classes*5) chosen for this research.

Creating a codebook is a comprehensive process involving analyzing the grayscale image to identify any potential descriptors. As we find each characterizer, the number of records for that descriptor increases in the codebook. The culmination of this process is a histogram value that succinctly describes the unique characteristics of the grayscale image. Figure 10 visually represents the KAZE feature extraction, a process we execute using the powerful Python Dataframe package.

	0	1	2	3	4	5	6	7	8	9 ...	2843	2844	2845	2846	2847	2848	2849	2850	2851	2852	
0	[1039, 641, 744, 2448, 1925, 1972, 2976, 2072...]	[438, 430, 430, 666, 412, 464, 292, 517, 557, 893]	[951, 546, 666, 2477, 345, 1823, 1677, 1955, 495...]	[2286, 1010, 2073, 6565, 2073, 5361, 5322, 3655, 495...]	[1702, 1887, 1518, 1924, 1868, 1393, 3924, 3824, 190...]	[2384, 989, 8060, 8060, 2671, 8394, 6399, 5086...]	[748, 718, 738, 646, 727, 468, 1402, 655, 1619]	[479, 348, 433, 481, 363, 340, 591, 545, 1071]	[2420, 1522, 1549, 5054, 5054, 2884, 3878, 4658, 405...]	[2420, 1522, 1549, 5054, 5054, 2884, 3878, 4658, 405...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]	[3431, 2540, 2358, 5202, 4590, 4600, 7409, 435...]
1	Benign	Benign	Benign	Benign	Benign	Benign	Benign	Benign	Benign	Benign	...	Malware	Malware	Malware	Malware	Malware	Malware	Malware	Malware	Malware	

2 rows × 2853 columns

Figure 10. Grayscale image features processed against the KAZE algorithm

4.4. Hyperparameter tuning

Random search (RS) and grid search (GS) are hyperparameter search methodologies used in this research. The fourth experiment employed the random search technique, whereas the third used the grid search technique. In both cases, the model was trained on the training set, a crucial step that determines the model's performance, using optimal hyperparameters for each model. Optimal parameters identified using Grid Search and Random Search are presented in Table 4.

4.5. Machine-learning model

Four experimental schemes have been carried out using features from grayscale image datasets. This research is generally divided into three sections applied to four experimental schemes. The first section used local feature extraction (KAZE) results to train models for all experiments. Global feature extraction (GIST) results were used in the second section to train the model for all experiments. The third section combines the

classification results of three machine-learning algorithms, including building models using local, global, or a combination of features. The last section is called the META model. The results of the model performance comparison between KAZE and GIST algorithm features are shown in Table 4, demonstrating our confidence in our research process and results.

Table 4. The best parameters using grid search (GS) and random search (RS) techniques

Metode	Features extraction	Parameter	LGBM	RF	SVM
GS	GIST	Boosting_type	GOSS	-	-
		Learning_rate	0.2	-	-
		Max_depth	5	-	-
		Random_state	41	-	-
		Deterministic	True	-	-
		Force_col_wise	True	-	-
		Max_depth	-	6	-
		Max_features	-	None	-
		Max_lead_nodes	-	9	-
		N_estimators	-	25	-
		C	-	-	100
		Gamma	-	-	0.1
	kernel	-	-	Linear	
	KAZE	Boosting_type	GBDT	-	-
		Learning_rate	0.1	-	-
		Max_depth	-10	-	-
		Random_state	40	-	-
		Deterministic	True	-	-
		Force_col_wise	True	-	-
		Max_depth	-	6	-
		Max_features	-	None	-
		Max_lead_nodes	-	9	-
		N_estimators	-	50	-
		C	-	-	100
Gamma		-	-	0.0001	
kernel	-	-	RBF		
RS	GIST	Boosting_type	GBDT	-	-
		Learning_rate	0.1	-	-
		Max_depth	-5	-	-
		Random_state	42	-	-
		Deterministic	True	-	-
		Force_col_wise	True	-	-
		Max_depth	-	9	-
		Max_features	-	SQRT	-
		Max_lead_nodes	-	9	-
		N_estimators	-	150	-
		C	-	-	100
		Gamma	-	-	0.1
	kernel	-	-	Linear	
	KAZE	Boosting_type	GBDT	-	-
		Learning_rate	0.05	-	-
		Max_depth	-5	-	-
		Random_state	41	-	-
		Deterministic	True	-	-
		Force_col_wise	True	-	-
		Max_depth	-	9	-
		Max_features	-	None	-
		Max_lead_nodes	-	9	-
		N_estimators	-	100	-
		C	-	-	100
Gamma		-	-	0.0001	
kernel	-	-	RBF		

4.5.1. Model performance based on local feature extraction

This section uses a dataset from local feature extraction (KAZE) for building models. Based on data from Table 5, the best model performance based on KAZE feature data was obtained in the first experiment for the LGBM classifier. The LGBM model demonstrated exceptional performance with accuracy, F1 score, recall, and precision values of 95%. Figure 11 compares the execution time of the base classifier algorithm on KAZE feature data. Based on this data, the LGBM model requires the fastest execution time compared to other base-classifier algorithms. In addition, LGBM performed stably in all experiments.

A comparison of RAM resource usage needed by each base-classifier technique for processing KAZE feature data is shown in Figure 12. RF demands the most RAM when processing local feature data from KAZE feature extraction. The support vector machine model uses the least amount of RAM compared to other models.

Table 5. Model performance comparison using KAZE and GIST feature extraction

Measurement Aspect	FE KAZE			FE GIST			
	SVM	RF	LGBM	SVM	RF	LGBM	
1	Accuracy (%)	89	94	95	71	94	96
	Recall (%)	89	94	95	71	94	96
	Precision	88	94	95	50	94	96
	F1 Score (%)	88	94	95	58	94	96
	Time (second)	0.008	0.006	0.002	0.066	0.006	0.002
	RAM Usage (MB)	201.7	218.5	204.9	250.1	246.1	310.8
2	Accuracy (%)	88	94	93	82	93	93
	Recall (%)	88	94	93	82	93	93
	Precision	88	94	94	82	94	94
	F1 Score (%)	88	94	93	82	93	93
	Time (second)	0.017	0.007	0.002	0.246	0.007	0.002
	RAM Usage (MB)	208.2	228.4	214.7	257.6	266.8	317.9
3	Accuracy (%)	93	91	93	92	90	93
	Recall (%)	93	91	93	92	90	93
	Precision	93	91	94	92	90	94
	F1 Score (%)	93	91	93	92	90	93
	Time (second)	0.053	0.005	0.002	0.062	0.003	0.002
	RAM Usage (MB)	187.5	210.9	200.7	241.6	251.4	304.4
4	Accuracy (%)	94	90	92	91	88	93
	Recall (%)	94	90	92	91	88	93
	Precision	94	90	92	91	89	93
	F1 Score (%)	94	90	92	91	88	93
	Time (second)	0.054	0.009	0.002	0.065	0.014	0.002
	RAM Usage (MB)	187.6	211.3	201.2	241.7	251.2	303.7

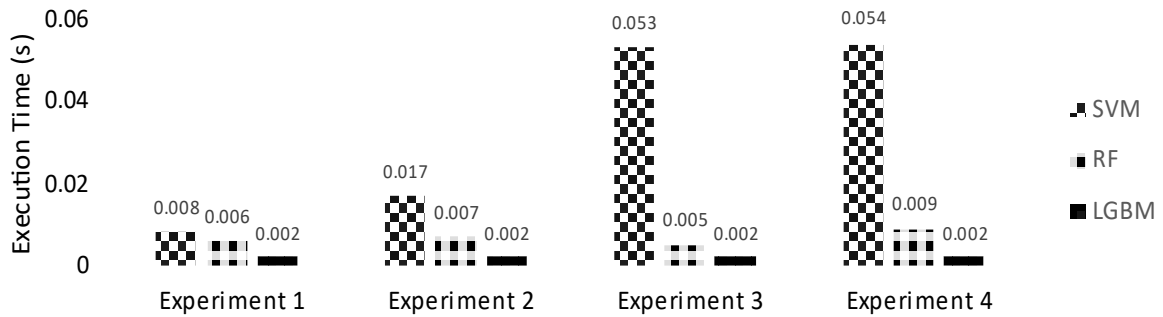


Figure 11. Comparison of execution time (s) for KAZE feature data

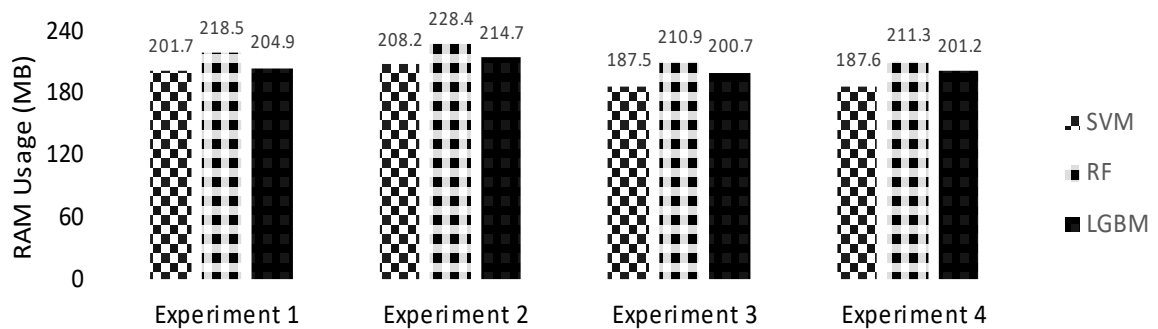


Figure 12. RAM resource comparison using KAZE feature data

4.5.2. Model performance based on global feature extraction

This section employs a global feature extraction (GIST) dataset for model building. The analysis, as shown in Table 5, was thorough, with the LGBM model demonstrating the best performance in the first trial across all main performance categories, including Accuracy, F1 Score, Recall, and Precision, achieving a 96% accuracy. Figure 13 provides a comprehensive comparison of the execution time of the base classifier algorithm in processing GIST feature data, further reinforcing the thoroughness of the analysis. The LGBM model's efficiency is highlighted, as it requires the fastest execution time compared to other base-classifier algorithms. In contrast, while effective, SVM requires the most extensive execution time for processing data from global feature extraction.

Figure 14 compares the RAM resource consumption by each base-classifier algorithm in processing global feature extraction data. The LGBM model requires the largest RAM resource allocation compared to other base-classifier models when processing GIST features. The highest peak RAM consumption was observed during the LGBM model's global data processing in the second experiment, which involved an imbalanced dataset handling technique. This significant increase in RAM consumption due to the data treatment technique underscores the influence of data on resource usage. Despite this, the performance of all base-classifier models remained stable throughout the experiments, indicating that the treatment of algorithms and data does not significantly influence RAM resource consumption.

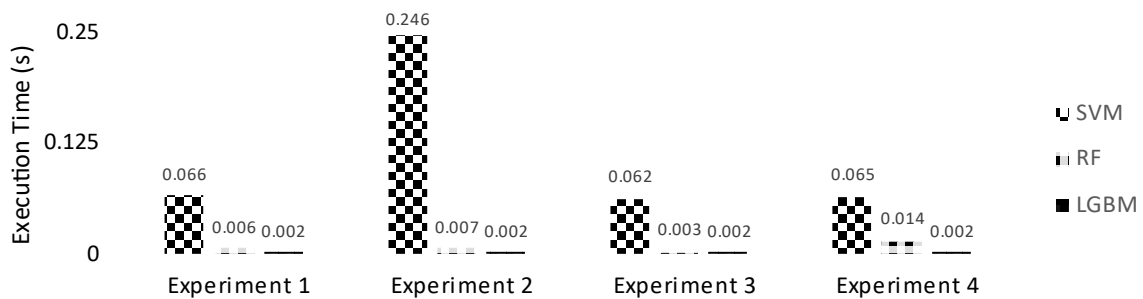


Figure 13. Comparison of execution time (s) for GIST feature data

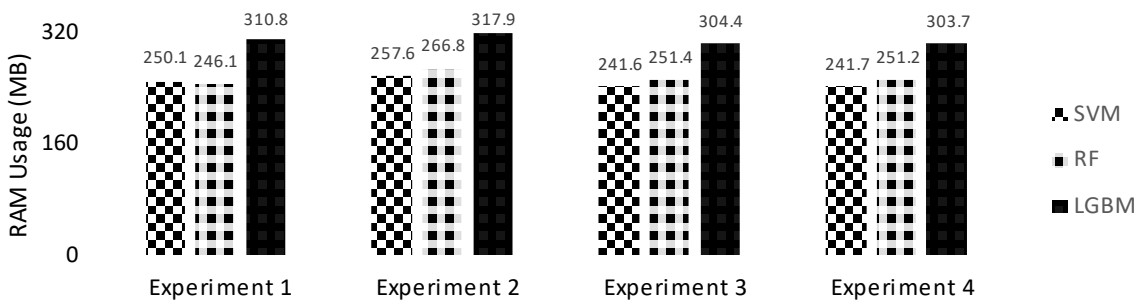


Figure 14. GIST feature data comparison of RAM resources

4.5.3. Meta classifier performance

The META model, which operates on either local or global data features or a combination of both, has performance calculation values explained in Table 6. In the fourth experiment, the LGBM model showcased its versatility by obtaining a score of 97% in all aspects of the main performance categories, namely accuracy, F1 score, recall, and precision. This adaptability is a crucial feature of the LGBM model, especially in processing combined feature data, combining local and global feature extraction results.

The execution times of the META algorithm on KAZE, GIST, and combined feature data are contrasted in Figure 15. This data indicates that in comparison to global and mixed feature extraction, local extraction (KAZE) contributes to the fastest META model execution time. A comparison of RAM resource consumption required by the META algorithm to process KAZE, GIST, and combined feature data is shown in Figure 16. The most minor RAM resources are required when the META model processes data from local feature extraction (KAZE). More considerable RAM resources are needed for the META model to process global feature data calculated by the GIST algorithm and combined features.

Table 6. Comparison of the performance of the META model on FE KAZE, GIST, and combination of both

Experiment	Measurement aspect	META		
		KAZE	GIST	COMBINATION
1	Accuracy (%)	93	95	96
	Recall (%)	93	95	96
	Precision	93	95	96
	F1 Score (%)	93	95	96
	Time (second)	0.545	0.611	0.606
	RAM Usage (MB)	176.4	284.2	287.9
2	Accuracy (%)	93	94	95
	Recall (%)	93	94	95
	Precision	94	94	95
	F1 Score (%)	93	94	95
	Time (second)	0.569	0.667	0.692
	RAM Usage (MB)	188.1	303.9	301.9
3	Accuracy (%)	93	94	96
	Recall (%)	93	94	96
	Precision	94	94	96
	F1 Score (%)	93	94	96
	Time (second)	0.542	0.676	0.661
	RAM Usage (MB)	171.5	288.1	294.5
4	Accuracy (%)	95	94	97
	Recall (%)	95	94	97
	Precision	95	94	97
	F1 Score (%)	95	94	97
	Time (second)	0.545	0.614	0.598
	RAM Usage (MB)	171.4	286.7	292.5

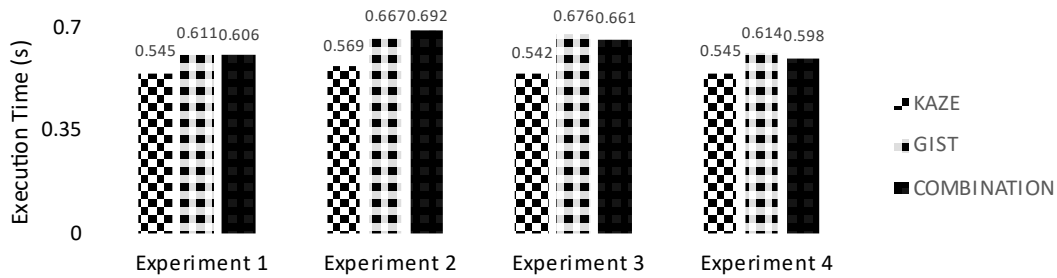


Figure 15. Comparison of META algorithms' execution time (s)

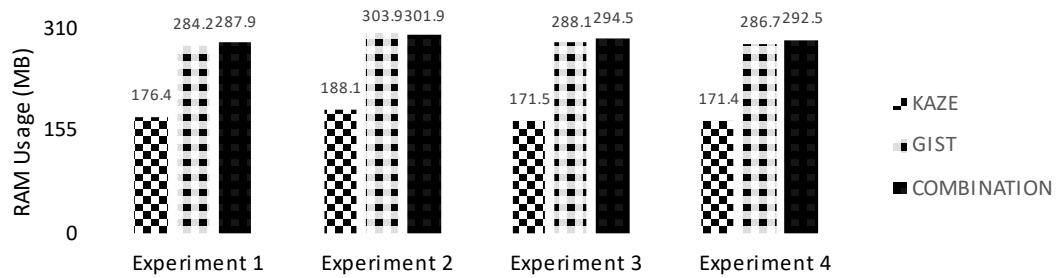


Figure 16. RAM resource comparison between META algorithms

4.6. Discussion

The Android APK, a crucial component in deploying and installing applications, is an archive file housing various data and resource files. These files contain critical information that can threaten the Android operating system. The diverse structures of APK files are a crucial focus in malware analysis. Understanding the significance of the Android APK structure is vital in comprehending the potential threats to the Android system. Current Android malware analysis approaches are categorized into static, dynamic, and hybrid, each with unique benefits and challenges. As a subset of artificial intelligence, various machine learning models can be employed to add intelligence to the implemented model. However, the literature is replete with machine learning-based solutions for Android malware detection.

The framework approach used in this research combines machine learning to extract local and global features on the APK that has been visualized. The process is that the *classes.dex* file in the Android APK is extracted for image transformation. The choice of *.dex* file is because the file contains program logic information, whereas most malware is not created from scratch but only changes the program logic slightly. This background underlies this research: by extracting local and global features in APK files, machine learning algorithms can efficiently and reliably recognize other malware variants. Apart from applying a single machine-learning model, this research combines several models to increase model accuracy.

The system design is meticulously divided into the preprocessing stage, machine learning model creation, and model evaluation. The first stage, data preprocessing, is a comprehensive process that converts a collection of APKs stored in a zip file into a pure APK file. This pure APK file is then used to extract the *classes.dex*, which is transformed into a digital image. The APK file, now a digital image, is extracted for its local and global features. The feature extraction results are divided into training, testing, and validation data to create machine learning models. The second stage involves creating a machine-learning model. In this research, a combination of several base-classifier algorithms is used, with parameters optimized using hyperparameter tuning techniques. The final result is a meta-classifier model that can be evaluated and its accuracy measured. The third stage is the model evaluation process, which thoroughly assesses the resulting machine learning model, including comparing accuracy, F1 score, RAM usage, and algorithm processing time.

Table 5 compares the performance of the base-classifier model on local and global features. The base classifier algorithms used are SVM, RF, and LGBM. The result is that the LGBM algorithm has the highest accuracy in local feature processing at 95% and global feature processing with an accuracy value of 96%. LGBM has the highest accuracy value in the first experiment, the default experiment, without any hyperparameter optimization or handling of an imbalanced dataset. Table 5 compares the performance of the meta-classifier model on local and global feature datasets. The meta-classifier algorithm used is logistic regression, which combines the prediction results from the base-classifier algorithm. The result is that the combination of local and global feature extraction significantly improves the performance of the meta-classifier model by 2% to 3%. This study obtained the highest meta-classifier accuracy in the fourth experiment, 97%. The fourth experiment used optimized parameters and a combined local and global dataset.

Based on the detailed explanation above, the objectives of this research have been successfully achieved. The combination of local and global features has significantly increased the model's accuracy, a key milestone in this study. Furthermore, the combination of several base-classifiers has been proven to enhance the accuracy of the base-classifier model, marking another successful outcome of this research.

5. CONCLUSION

A visualization-based model that recognizes an Android APK as benign or malware has been implemented. The model works by converting the *classes.dex* component into a grayscale image. Local and global feature extraction is implemented on the grayscale image, combined with the machine-learning algorithms: support vector machine, random forest, and LightGBM. Experimental results show that the proposed model has the best accuracy in the fourth experiment (META combination), which is 97% and requires 0.598 seconds of computation time. The main contribution of this paper is to produce an Android APK malware detector model that works by combining a combination of machine learning algorithms trained using the dataset resulting from the combination of local and global feature extraction algorithms. The study's conclusion is highly instructive and valuable for researchers in the various fields of machine learning. Several tasks, including malware identification and supervised classification based on dataset visualization, can be directly tackled using the developed technique.

REFERENCES




- [1] J. Aycocock, *Computer viruses and malware*, vol. 22. Springer US, 2006.
- [2] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," in *IKT 2013 - 2013 5th Conference on Information and Knowledge Technology*, May 2013, pp. 113–120, doi: 10.1109/IKT.2013.6620049.
- [3] Y. Zhang and B. Li, "Malicious code detection based on code semantic features," *IEEE Access*, vol. 8, pp. 176728–176737, 2020, doi: 10.1109/ACCESS.2020.3026052.
- [4] J. Singh, D. Thakur, F. Ali, T. Gera, and K. S. Kwak, "Deep feature extraction and classification of Android malware images," *Sensors*, vol. 20, no. 24, Dec. 2020, doi: 10.3390/s20247013.
- [5] Y. Zhang, M. Yang, Z. Yang, G. Gu, P. Ning, and B. Zang, "Permission use analysis for vetting undesirable behaviors in android apps," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1828–1842, Nov. 2014, doi: 10.1109/TIFS.2014.2347206.
- [6] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, "Dendroid: a text mining approach to analyzing and classifying code structures in Android malware families," *Expert Systems with Applications*, vol. 41, pp. 1104–1117, Mar. 2014, doi: 10.1016/j.eswa.2013.07.106.
- [7] D. Arp, M. Spreitzerbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," *Network and Distributed System Security (NDSS) Symposium*, 2014, doi: 10.14722/ndss.2014.23247.

- [8] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 127, Springer International Publishing, 2013, pp. 86–103.
- [9] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "AndroDialysis: analysis of Android intent effectiveness in malware detection," *Computers and Security*, vol. 65, pp. 121–134, Mar. 2017, doi: 10.1016/j.cose.2016.11.007.
- [10] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DynaLog: An automated dynamic analysis framework for characterizing android applications," in *2016 International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2016*, Jun. 2016, pp. 1–8, doi: 10.1109/CyberSecPODS.2016.7502337.
- [11] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, Jun. 2019, doi: 10.1109/TIFS.2018.2879302.
- [12] A. Alotaibi, "Identifying malicious software using deep residual long-short term memory," *IEEE Access*, vol. 7, pp. 163128–163137, 2019, doi: 10.1109/ACCESS.2019.2951751.
- [13] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 2001, pp. 38–49, doi: 10.1109/secpri.2001.924286.
- [14] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *ACM International Conference Proceeding Series*, Jul. 2011, pp. 1–7, doi: 10.1145/2016904.2016908.
- [15] K. Kancherla and S. Mukkamala, "Image visualization based malware detection," in *Proceedings of the 2013 IEEE Symposium on Computational Intelligence in Cyber Security, CICS 2013 - 2013 IEEE Symposium Series on Computational Intelligence, SSCI 2013*, Apr. 2013, pp. 40–44, doi: 10.1109/CICYBS.2013.6597204.
- [16] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible android malware detection and family classification using network-flows and API-calls," in *Proceedings - International Carnahan Conference on Security Technology*, Oct. 2019, pp. 1–8, doi: 10.1109/CCST.2019.8888430.
- [17] D. P. Anandia, "Android malware detection using image visualization and machine learning," (in Bahasa), M.S. thesis, Department of Electrical Engineering, Institut Teknologi Bandung, 2021.
- [18] X. Wang, L. Zhang, K. Zhao, X. Ding, and M. Yu, "MFDroid: A stacking ensemble learning framework for Android malware detection," *Sensors*, vol. 22, no. 7, Mar. 2022, doi: 10.3390/s22072597.
- [19] M. A. R. Khan, N. Kumar, and R. C. Tripathi, "Detection of Android malware app through feature extraction and classification of Android image," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 5, pp. 906–914, 2022, doi: 10.14569/IJACSA.2022.01305103.
- [20] N. Al Sarah, F. Y. Rifat, M. S. Hossain, and H. S. Narman, "An efficient Android malware prediction using ensemble machine learning algorithms," *Procedia Computer Science*, vol. 191, pp. 184–191, 2021, doi: 10.1016/j.procs.2021.07.023.
- [21] Y. Fang, Y. Gao, F. Jing, and L. Zhang, "Android malware familial classification based on DEX file section features," *IEEE Access*, vol. 8, pp. 10614–10627, 2020, doi: 10.1109/ACCESS.2020.2965646.
- [22] J. Jung, J. Choi, S. Cho, S. Han, M. Park, and Y. Hwang, "Android malware detection using convolutional neural networks and data section images," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, Oct. 2018, pp. 149–153, doi: 10.1145/3264746.3264780.
- [23] L. Kabbai, M. Abdellaoui, and A. Douik, "Image classification by combining local and global features," *Visual Computer*, vol. 35, no. 5, pp. 679–693, Apr. 2019, doi: 10.1007/s00371-018-1503-0.
- [24] M. Hassaballah, A. A. Abdelmgeid, and H. A. Alshazly, "Image features detection, description and matching," in *Studies in Computational Intelligence*, vol. 630, Springer International Publishing, 2016, pp. 11–45.
- [25] K. Bakour and H. M. Ünver, "VisDroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques," *Neural Computing and Applications*, vol. 33, no. 8, pp. 3133–3153, Jul. 2021, doi: 10.1007/s00521-020-05195-w.
- [26] A. Torralba and A. Oliva, "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.
- [27] B. Xie, J. Qin, X. Xiang, H. Li, and L. Pan, "An image retrieval algorithm based on GIST and SIFT features," *International Journal of Network Security*, vol. 20, 2018.
- [28] A. Vinay, B. Gagana, V. S. Shekhar, B. Anil, K. N. B. Murthy, and S. Natarajan, "A double filtered GIST descriptor for face recognition," *Procedia Computer Science*, vol. 79, pp. 533–542, 2016, doi: 10.1016/j.procs.2016.03.068.
- [29] P. Soleimani, D. W. Capson, and K. F. Li, "Real-time FPGA-based implementation of the AKAZE algorithm with nonlinear scale space generation using image partitioning," *Journal of Real-Time Image Processing*, vol. 18, no. 6, pp. 2123–2134, Mar. 2021, doi: 10.1007/s11554-021-01089-9.
- [30] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "KAZE features," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7577, Springer Berlin Heidelberg, 2012, pp. 214–227.
- [31] N. Ali *et al.*, "A novel image retrieval based on visual words integration of SIFT and SURF," *PLoS ONE*, vol. 11, no. 6, Jun. 2016, doi: 10.1371/journal.pone.0157428.
- [32] H. M. Ünver and K. Bakour, "Android malware detection based on image-based features and machine learning techniques," *SN Applied Sciences*, vol. 2, no. 7, Jun. 2020, doi: 10.1007/s42452-020-3132-2.
- [33] M. Ali, M. N. Haider, S. A. Lashari, W. Sharif, A. Khan, and D. A. Ramli, "Stacking classifier with random forest functioning as a meta-classifier for diabetes diseases classification," *Procedia Computer Science*, vol. 207, pp. 3453–3462, 2022, doi: 10.1016/j.procs.2022.09.404.
- [34] A. Khan, A. Khan, M. M. Khan, K. Farid, M. M. Alam, and M. B. M. Su'ud, "Cardiovascular and diabetes diseases classification using ensemble stacking classifiers with SVM as a meta-classifier," *Diagnostics*, vol. 12, no. 11, Oct. 2022, doi: 10.3390/diagnostics12112595.
- [35] S. Liang, J. Peng, Y. Xu, and H. Ye, "Passive fetal movement recognition approaches using hyperparameter tuned LightGBM model and Bayesian optimization," *Computational Intelligence and Neuroscience*, vol. 2021, no. 1, Jan. 2021, doi: 10.1155/2021/6252362.
- [36] N. Nayyer, N. Javaid, M. Akbar, A. Aldegheshem, N. Alrajeh, and M. Jamil, "A new framework for fraud detection in bitcoin transactions through ensemble stacking model in smart cities," *IEEE Access*, vol. 11, pp. 90916–90938, 2023, doi: 10.1109/ACCESS.2023.3308298.
- [37] G. L. Tsirogiannis, D. Frossyniotis, K. S. Nikita, and A. Stafylopatis, "A meta-classifier approach for medical diagnosis," in *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, vol. 3025, Springer Berlin Heidelberg, 2004, pp. 154–163.




- [38] N. A. M. Zaini and M. K. Awang, "Performance comparison between meta-classifier algorithms for heart disease classification," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 10, pp. 323–328, 2022, doi: 10.14569/IJACSA.2022.0131039.

BIOGRAPHIES OF AUTHORS






Yoga Herawan    received the Master of Computer Science from IPB University, Indonesia, in 2024. He currently holds the position of system application specialist at PT Pertamina Drilling Services Indonesia, Indonesia. His current research interests lie in machine learning, computer vision, and data analysis. He can be contacted at email: yoga.herawan@apps.ipb.ac.id.



Imas Sukaesih Sitanggang    received a Ph.D. in computer science from the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, in 2013. She is a lecturer in the Computer Science Department at IPB University, Indonesia. Her main research interests include spatial data mining and smart agriculture. She can be contacted at email: imas.sitanggang@apps.ipb.ac.id.



Shelvie Nidya Neyman    received a Ph.D. in electrical engineering and informatics from Bandung Institute of Technology, in 2015. She is a lecturer in the Computer Science Department at IPB University, Indonesia. Her main research interests include information security, cryptography, steganography, and watermarking. She can be contacted at email: shelvie@apps.ipb.ac.id.