IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 October 2023, accepted 6 November 2023, date of publication 9 November 2023,
date of current version 17 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3331739

RESEARCH ARTICLE

Detection of Android Malware Using Machine Learning and Siamese Shot Learning Technique for Security

FAHDAH A. ALMARSHAD[1], MOHAMMED ZAKARIAH[2], (Member, IEEE), GHADA ABDALAZIZ GASHGARI[3], EMAN ABDULLAH ALDAKHEEL[4], AND ABDULLAH I. A. ALZAHRANI[5]

[1]Department of Information Systems, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia
[2]Department of Computer Science, College of Computer and Information Science, King Saud University, Riyadh 11633, Saudi Arabia
[3]Department of Cybersecurity, College of Computer Science and Engineering, University of Jeddah, Jeddah 23445, Saudi Arabia
[4]Department of Computer Science, College of Computer and Information Science, Princess Nourah bint Abdulrahman University, Riyadh 11671, Saudi Arabia
[5]Department of Computer Science, College of Science and Humanities-Al Quwaiiyah, Shaqra University, Shaqra 11961, Saudi Arabia

Corresponding author: Eman Abdullah Aldakheel (eaaldakheel@pnu.edu.sa)

This work was supported by Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia, through Researchers Supporting, under Project PNURSP2023R409.

ABSTRACT Android malware security tools that can swiftly identify and categorize various malware classes to create rapid response strategies have been trendy in recent years. Although many application fields have demonstrated the usefulness of implementing Machine Learning and deep learning methods to provide automation and self-learning services, the scarcity of data for malware samples has been cited as a hurdle in creating efficient deep learning-based solutions. In this paper, a one-shot learning-based Siamese neural network is proposed to overcome this issue, as it can both identify malware assaults and categorize malware into multiple categories. The Drebin dataset, which is divided into benign and harmful components, is used in our suggested methodology. The efficiency of the suggested strategy is evaluated through a dataset made up of 9476 goodware applications and 5560 Android malware apps. The five critical phases of its implementation are pre-processing, data partitioning, model architecture, training, and assessment. In both the training and testing phases, Siamese networks are trained to rank sample similarity, and the accuracy is determined using N-way one-shot tasks. According to the experiment's findings, our Siamese Shot model fared better than the other standard approaches, obtaining an accuracy of 98.9%. Additionally, the most well-liked platforms are Keras and TensorFlow.

INDEX TERMS Android malware, security tools, machine learning, deep learning, one-shot learning, Siamese neural network, Drebin dataset, efficiency, N-way one-shot tasks, TensorFlow.

I. INTRODUCTION

The 'easy to use' feature and the effectiveness of various apps, as well as the ongoing improvements in smart devices' hardware and software, are driving a rapid increase in smartphone usage and related applications in this technological era [1]. Studies conducted in this field indicate that by 2024, 4.5 billion people are anticipated to own smartphones [2]. Among these smartphone devices, the most popular smartphone operating system is Android [3]. A 75.5% market share is held by Android [4]. As a result of its widespread adoption, Android is more vulnerable to malware and viruses which makes it an appealing target for attackers. However, studies indicate that these threats and assaults can be tackled using Machine learning (ML) as ML algorithms can create a classifier from a set of training instances [5], [6]. Thus, utilizing examples while developing malware detectors avoids the need to specify identifiers explicitly.

Moreover, Machine learning-based malware detection studies are becoming increasingly common to achieve a high degree of detection accuracy [7], [8]. ML algorithms, which can make decisions after learning from data templates, have

The associate editor coordinating the review of this manuscript and approving it for publication was Barbara Masucci.

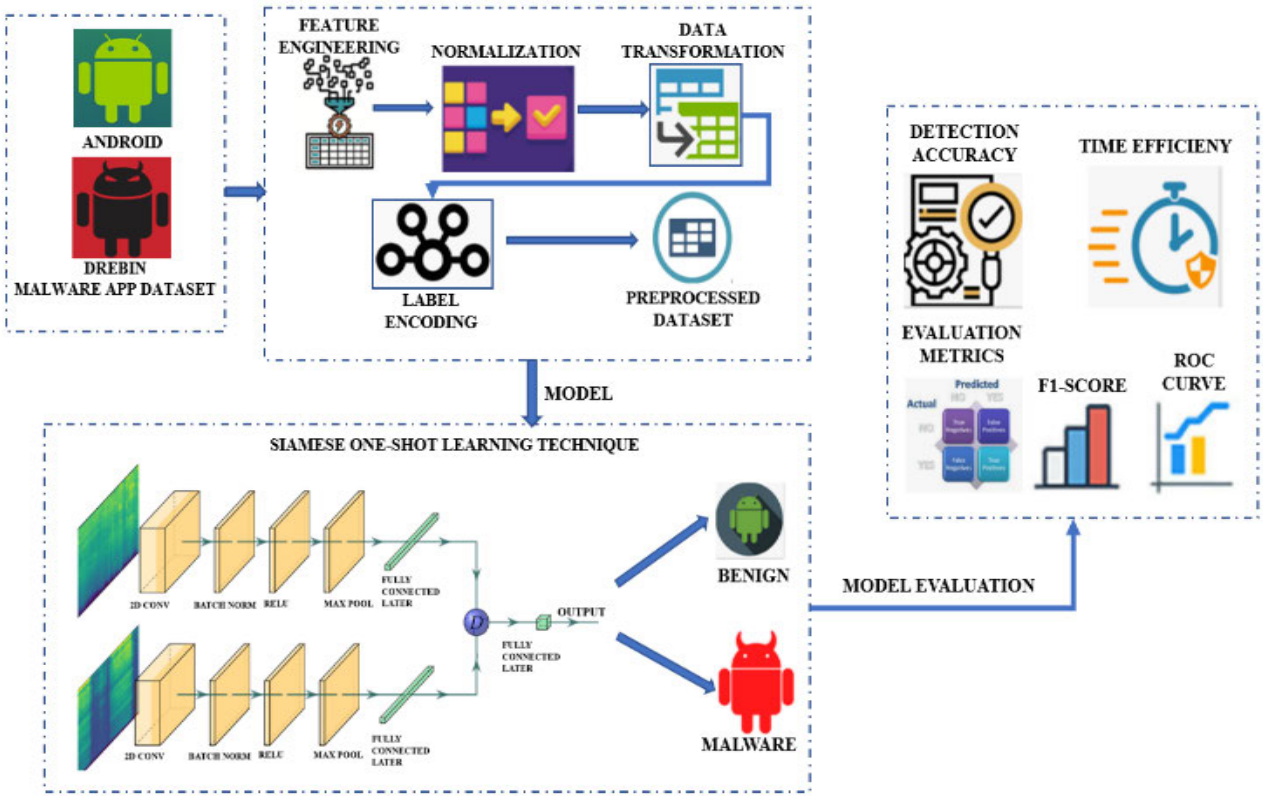


FIGURE 1. Overview of the proposed model for Android Malware detection using the Siamese one-shot learning technique.

been used in past studies. Also, ML aims to keep humans out of computer systems as much as possible [9]. Decisions are predicted by machine learning using experience or prior data as well as computer learning approaches. Supervised and unsupervised learning approaches may examine the characteristics and monitor the model [10]. The training is continued until the model masters accurately predict every sample [11]. Furthermore, the development of malware detection systems has made use of a variety of ML methods, including support vector machines (SVM) [12], K-nearest neighbor (KNN) [13], Bayesian estimation [9], genetic algorithms [14], etc. These ML algorithms are trained to discriminate between malicious and benign samples using unsupervised learning techniques that supply the inputs without goals [15]. However, the supervised and unsupervised learning approaches were combined in several experiments to detect any malicious activity.

Additionally, malware detection is a crucial information security area closely related to businesses' economic, legal, and reputational problems. It has presented a severe threat to individuals and businesses due to the rapid multiplication of malware varieties. A viable strategy to address several issues with malware detection is, to use deep learning as a tool for creating and improving detection algorithms [16]. Three complex and symmetric techniques are provided to influence performance: the dense layer model, the LSTM model, and correlation-based feature selection. The malware variants updated from earlier versions will avoid detection through various signatures by utilizing highly specialized replication techniques [17]. But when it comes to deep learning, there are a lot of challenging factors to be kept in mind while considering detection algorithms. In addition, Machine learning-based detection techniques have received much attention recently to match detection with the rate of malware growth. The strong performance of learning outcomes utilizing either supervised machine learning models or deep learning models has been demonstrated in several articles [18]. However, they need many training samples and a considerable learning period in the training process. Given the early introduction of new malware types, it is impossible to gather enough samples. Training on an extensive range of malware samples is essential for developing sustainable learning models [19]. If any new malware surfaces, the model must be retrained on the full sizable dataset.

Another type that trains models with limited data is known as 'One-shot learning' and holds the ability to effectively prevent overfitting. Humans can learn things from a limited number of instances, which is how this notion is created. If trained on a small number of examples, the traditional machine-learning approach will incur considerable overfitting [20]. Our method, however, used Siamese networks, which use twin convolutional neural networks to form the

architecture and share the same parameters. According to empirical data, utilizing the Drebin Dataset, Siamese convolutional networks achieved a test accuracy of 92.0% when used for one-shot image recognition [21]. In our approach, the malware samples underwent pre-processing procedures to become picture representations before being fed into the Siamese convolutional neural networks. The output sigmoid layer's similarity score represents the malware family determination, which is the fundamental idea behind using Siamese networks to identify Android malware.

In this paper, a one-shot learning-based Siamese neural network is proposed to overcome this issue, as it can both identify malware assaults and categorize malware into multiple categories. Siamese One-Shot Learning technique is used to propose a model for Android malware detection as shown in the Fig. 1.

The following are the contributions made by our suggested model:

- The study develops a robust and dynamic model by combining Siamese one-shot learning with the outcomes of machine learning-based algorithms. In particular, this combination enables the model to deal with a lack of training samples during the early stages of fresh malware development and yet achieve good classification performance.
- Limited training samples are overcome via one-shot learning utilizing Siamese networks. When it comes to malware detection, traditional machine-learning techniques may need a significant amount of labeled data for training, which might be challenging to collect. The suggested method effectively manages the lack of data by training on a single sample from each class and learning to generalize from it.
- To find patterns and connections among the features in the Drebin Dataset, the study uses visualization analysis. The researchers can identify the most crucial factors that affect application categorization by visually representing the relationships between distinct attributes.
- The proposed model performs better than comparable research and has a resilient and dynamic structure. The model outperforms other conventional methods for malware identification by achieving an accuracy of 98.9% by utilizing Siamese one-shot learning and visualization techniques.

The research under study is divided into 7 sections: the 1 section is the introduction part that has already been discussed. The 2 section is the literature review part which presents a quick synopsis of current research on the issue. The dataset, the suggested model and its characteristics, ML-based techniques, and performance measures to be used in the experimental evaluation are all discussed in Sections III and IV. Whereas Sections V and VI explain the experiments' results and discussion. A summary and recommendations for more research on the subject are included in the last section, which is 7.

II. LITERATURE REVIEW

The designed software used for malevolent intent is called malware. Malware producers are concentrating on this area due to the extensive usage of mobile devices and the rising user base [5], [7], [9]. This software may be used to achieve several objectives. Some of these malicious goals include: interfering with the regular operation of the Android operating system it is working on, obtaining the user's personal and sensitive information illegally/without the user's consent, seizing the user's device, obtaining information for ransom, or displaying unwanted advertising content to the user. A significant amount of research has been done in the area of malware detection to prevent malware makers from reaching these aims [9]. Static, dynamic, and hybrid analyses are the three primary types of studies done in this setting.

The static analysis methods look at an application's source code without executing it [10], [11] to find malicious activity in the suspect program. Both permission-based [11] and these systems primarily use signature-based [12] mechanisms. The signature-based techniques help in detecting malware by comparing the application code to known harmful code fragments which are stored in a database. However, with permission-based techniques, the application's requests for permission are compared to the kinds of permissions that malicious programs typically make. While static analysis techniques are quicker than dynamic analysis techniques [16], they are less effective at revealing information about an application's behavior. At the same time, they are in use because they need to identify the application code that will dynamically load. In contrast, it is being used [17].

Another method is the 'One-shot learning' technique which uses past information and a small sample of images to get a general idea of the database. This technique has been widely used in some applications, including the classification and identification of images, voice recognition, Siamese neural networks, and prototype networks. Through the display of harmful code, Sharma et al.'s static approach of assembly language operation code analysis was suggested [20]. A Siamese neural network-based end-to-end framework was recently created in this study [21] to identify malware. Fuzzy class memberships were used in this research [22] to try the modification of raw data, which was subsequently put into the Siamese neural network to prevent intrusion assault. Even though multiple feature extraction techniques were employed in this study to obtain competitive results, only semantic feature embedding can be learned. Understanding how these were employed for feature embedding was challenging because the semantic content of the raw binary files they utilized, needed to be clearly described. A model learned for a general characteristic from uncommon categories was unreliable for capturing the typical features of malicious code. A high-level malware class feature with a meta-learner was presented in this study [23], which assessed the effectiveness of their concept in identifying malware with distinctive properties. The issue with accurately capturing

TABLE 1. Review of the literature including the dataset used, technique used, and the findings.

Ref	Datasets	Methodology	Results
[1]	- There are 5600 malware samples in the DREBIN datasets, representing 176 distinct families. The experiment employed the remaining 5,500 samples after we eliminated 63 samples that lacked a manifest file or structural flaws. When it came to malware samples, we disregarded any rights that were not in use. The remaining 93 permissions (85 built-in and 8 bespoke) were used as the features.	- Machine Learning, KNN, DT, RF, SVM, Evaluation Metrics, DNN, LightGBM.	- Accuracy and - F1-Score of 0.9621 and 0.9315, respectively.
[5]	- Around 10 studies' data were extracted for answering RQ1. - Twenty-three studies' data were extracted for answering RQ2. - Around 19 and 37 studies data were extracted for answering RQ3 and RQ4.	- Static, Hybrid, Dynamic Analysis, MLP Model, KNN, Linear Regression, Naïve Bayes, SVM, Random Forest, Machine Learning.	- Precision: 0.979. - Accuracy of 97.75%. - For Android vulnerability Detection, model accuracy is below 90%.
[7]	- The collection included 5560 Android malware applications and 9467 Android goodware apps.	- Machine learning, Random Forest, Confusion Matrix, SVM, KNN, DT, GB, AdaBoost.	- This model outperformed other algorithms with an accuracy of 95.24%.
[9]	- The first dataset consists of thirty-five attributes and around one hundred thousand records. - The second dataset comprises fifteen thousand Android apps with 5400 malicious and 9500 benign.	- Deep Learning, LSTM, ReLU Activation function, SVM, DT, RF.	- They achieved 84%, 66%, 60%, and 71% accuracy for the feature selected on the Malware dataset.
[26]	- The DREBIN collection contains 125,000 benign apps from the Play Store and 5,700 malware application samples.	- Deep Learning, DNN, NB, KNN, DT, F1-score, SVM, Gradient Boosting, RF.	- They achieved F1-score best score of 0.88.
[27]	- This experiment has CICAndMal2017 and Drebin datasets. - Drebin Dataset comprises 15300 applications, two hundred and fifteen features, 9400 normal, and 5800 malware applications.	- Machine Learning, KNN, SVM, LDA, Deep-Learning, CNN-LSTM, RNN,	- The SVM Algorithm achieved an accuracy of 100%. - KNN achieved an accuracy of 82%.

the distance across intra-class variation persists because this group of works must consider the distance between the positive and negative pairings. These few-shot learning models frequently incorporate contrastive loss, which does not help reduce intraclass variation. It is true of much state-of-the-art that these models are now suggesting.

To detect Android malware, the study [24] created a brand-new fuzzy integral-based multi-classifier ensemble model. They used the XGBoost, RF, DT, AdaBoost, and Light-GBM methods to merge the classifiers' outputs across the Choquet fuzzy integral. The experimental outcomes revealed that their proposed approach based on the fuzzy integral technique achieved a higher performance, with an accuracy value of 96.56% compared to those of the classifiers used individually. The dataset contained 6100 malicious and 9500 benevolent applications, containing 9500 applications. This study [25] used the risk-based fuzzy analytical hierarchy process technique to create multi-criteria-based decision and

mobile virus detection systems. This strategy's objective was to increase user awareness of high-risk permissions by doing a static analysis of the permission-based features. The Drebin and AndroZoo dataset assessments have an accuracy rating of 92.95%.

SecureDroid is a solution that was [28] created to improve the security of machine learning-based Android malware detection. They offered a unique feature selection technique to make the classifier more challenging to manipulate, and they suggested an ensemble learning strategy by combining the separate classifiers. The research [29] developed a hybrid Android malware detection model combining network traffic characteristics and supervised learning (KNN and K-medoids) techniques. Using the KNN, [30] malware documents were categorized by proposing modeling malware as a language and evaluating the viability of extracting semantics from examples of that language.

In [31], the authors proposed an efficient malware detection approach that utilizes feature weighting based on Harris Hawks optimization. The study focuses on enhancing the accuracy of malware detection through optimization techniques. The authors of [32] presented MAPAS, a practical deep learning-based Android malware detection system. The research emphasizes the practicality and effectiveness of deep learning models in identifying malware instances. In another study [33], authors introduced a two-stage deep learning framework for image-based Android malware detection and variant classification. The study focuses on enhancing classification accuracy by considering variants of Android malware. The authors of [34] proposed a deep learning-based Android malware detection system using static analysis. The research investigates the application of deep learning models to identify malware based on static code analysis. The authors of [35] developed DroidRL, a feature selection approach for Android malware detection using reinforcement learning. The study explores the use of reinforcement learning to optimize feature selection in the detection process. The authors of [36] focused on Android malware detection using network traffic and sequential deep learning models. The research highlights the significance of analyzing network traffic patterns to identify malicious activities.

A research on the identification of Android malware in the context of the Internet of Things was presented by Babbar et al. [42]. They contributed to the continuing study of malware identification in IoT environments by using the K-Nearest Neighbor Algorithm for their investigation. A study on Android malware detection using machine learning classifiers was done by Raymond et al. [43]. Their study included an Enhanced PCA Algorithm, illuminating creative strategies to address the escalating difficulties in malware identification. The analysis and classification of malware for Android that is obfuscated was covered in depth by Aurangzeb and Aleem [44]. They used ensemble voting and deep learning approaches to provide insights into the changing malware obfuscation and detection landscape.

"ImageDroid," a ground-breaking deep learning program for effective Android malware detection, was introduced by Liu et al. [46]. Their efforts provide a substantial improvement in mobile security by automating the detection of harmful elements. For the objective of increasing cybersecurity and Android virus detection, Albakri et al. [47] investigated the combination of metaheuristics with deep learning models. In the context of malware categorization, their study provides insightful information about the possible synergy between optimization approaches and deep learning. In a study they conducted, Alkahtani and Aldhyani [27] investigated the use of artificial intelligence algorithms for malware detection in mobile devices running the Android operating system. They have made a sizable contribution to the ongoing work in mobile security with their work. Yumlembam et al.'s [48] research on IoT-based Android malware detection using adversarial defense mechanisms and graph neural networks. Their research is in line with the changing difficulties in mobile malware detection and Internet of Things security.

The list of previous literature in the field of android malware, including the dataset, technique, and findings, is shown in Table 1.

The literature evaluation identifies various research gaps in the area of Android malware detection. Traditional methods for static analysis, such as signature-based and permission-based approaches, are limited in their ability to show an application's behavior and have difficulty effectively capturing intra-class differences, which has an impact on the effectiveness of malware identification. Another key difficulty is managing limited data because typical machine learning models need a lot of labeled samples to train, which makes it challenging to collect enough data for uncommon or novel malware kinds. Addressing intra-class variation, necessary for accurate malware classification, remains a difficulty, even with the adoption of few-shot learning models like Siamese networks. Furthermore, even though some research has achieved excellent detection accuracy, it is still important to improve generalization across various datasets and unknown malware samples.

Innovative strategies that combine Siamese one-shot learning with visualization analysis have been put advanced in the literature to close these gaps. These methods seek to improve malware detection skills, particularly when data is scarce in the early phases of the generation of new malware. The suggested model outperforms conventional approaches in terms of accuracy and performance, suggesting that it can solve the problems facing current research. The model efficiently identifies hazardous and benign applications based on their properties by showing the correlations between various qualities. To fully evaluate the model's bridging of the stated research gaps, though, a thorough grasp of its design and experimental findings is required.

III. DATA COLLECTION

A list of Android apps classified as dangerous or benign is available to the public and is called the Drebin Dataset. The dataset was developed in 2014 due to a study on identifying malware on Android devices conducted by Northeastern University researchers [37]. Being one of the largest publicly available databases, the collection comprises more than 120,000 Android apps. Using static and dynamic analysis, the researchers classified every program in the dataset as either malicious or benign. 5,560 applications from 179 distinct malware families are included in the dataset. The collection comprises 13,106 samples from other sources, including Android websites, malware forums, and security blogs, in addition to 96,150 programs from the GooglePlay Store, 19,545 applications from various alternative Chinese Markets, 2,810 applications from alternative Russian Markets, and more. The more details about the Drebin Dataset is available in the paper [38].

Dynamic analysis includes executing an application in a controlled environment and watching its behavior, while static analysis is inspecting the code of an application without actually running it. The researchers used both methods to locate suspected malware, then carefully checked their results. In some studies, and research initiatives, the Drebin Dataset has become a well-liked source for those studying mobile security. Through the website of Northeastern University, the dataset is publicly accessible for download.

A. DATA DESCRIPTION

Table 2 in the dataset, part of the Drebin Dataset, lists the characteristics taken from each Android application. To create and test machine learning algorithms for malware detection on Android devices, these attributes were extracted to aid researchers.

The permissions, API calls, and string features in the Drebin Dataset may be generally divided into three areas.

Following's a brief overview of each of these group features:

1. **PERMISSIONS:** The Android operating system uses permissions as one of its primary security measures. It enables an application to use certain device resources or perform specific tasks. There is a list of all the permissions that each application in the dataset uses in the Drebin Dataset.
2. **API CALLS:** A collection of APIs (Application Programming Interfaces) allows Android applications to communicate with the operating system and other applications. Each application in the dataset's Drebin Dataset makes a list of all the API calls they performed.
3. **STRINGS:** In the application code, strings represent a thread of characters. A list of all the strings used in each application in the Drebin Dataset is included. A binary vector indicating whether an application utilizes specific permission, API call, or text describes each application's characteristics in the dataset. The total number of distinctive features in the dataset determines the overall vector size. Several key elements have been highlighted to aid in better understanding the dataset.
4. **TRANSACT:** The Android operating system provides a technique that enables an application to initiate a remote procedure call (RPC) to another process.
5. **ON SERVICE CONNECTED:** When a Service is linked to a component in an Android application, the callback function On Service Connected is activated.
6. **BIND SERVICE:** This method can be used to bind an Android application to a service.
7. **ATTACH INTERFACE:** Using this method, the Android allows you to attach an interface to a binder object
8. **SERVICE CONNECTION:** An application can connect to a service using the Service Connection interface in the Android operating system.
9. **ANDROID. OS.BINDER:** This class in the Android operating system offers a straightforward implementation of the IBinder interface.
10. **SEND_SMS:** This authorization enables an application to send SMS messages from the device.
11. **LJAVA. LANG.CLASS.GETCANONICALNAME:** This Java method retrieves the class's canonical name.
12. **LJAVA. LANG.CLASS.GETMETHODS:** This Java method returns an array of Method objects that correspond to the class's public methods.
13. **LJAVA. LANG.CLASS.CAST:** In the Java programming language, this method casts an object to the type of the provided class.
14. **READ_CONTACTS:** This permission enables an application to read the user's contacts.
15. **DEVICE_POWER:** This permission gives a program
16. control over the device's on/off status.
17. **HARDWARE_TEST:** An application may access low-level hardware diagnostic tests with the HARDWARE_TEST permission.
18. **ACCESS_WIFI_STATE:** This permission enables an application to get access to details about the Wi-Fi network.
19. **WRITE_EXTERNAL_STORAGE:** This permission enables a program to write to external storage.
20. **ACCESS_FINE_LOCATION:** This permission enables an application to obtain exact location data.
21. **SET_WALLPAPER_HINTS:** The permission SET_WALLPAPER_HINTS enables an application to set wallpaper hints.
22. **SET_PREFERRED_APPLICATIONS:** With this permission, an application can choose which apps the user prefers.
23. **WRITE_SECURE_SETTINGS:** This permission enables a program to alter secure system settings.
24. **CLASS:** This feature identifies the class of the Android application.

TABLE 2. Drebin's dataset.

HARD WARE_TES T	ACCESS _ WIFI_ STATE	WRITE_ EXTERNAL _ STORAGE	ACCESS FINE LOCATIO N	SET_ WALLPAPE R HINTS	SET PREFERRED APPLICATION S	WRITE_ SECURE_ SETTING S	CLAS S
0	0	1	0	0	0	0	S
0	0	1	0	0	0	0	S
0	0	0	0	0	0	0	S
0	1	1	1	0	0	0	S
0	1	0	1	0	0	0	S

Two hundred fifteen characteristics categorize malware assaults into 5,560 Drebin malware applications and 9,476 innocuous apps.

B. DATA VISUALIZATION

Numerous characteristics of the Drebin Dataset make it challenging to examine using conventional statistical techniques [38]. Visualization will help reveal these relationships when patterns and connections between characteristics are present but not always apparent through numerical analysis alone. The dataset's most significant characteristics may be found with visualization. Visually representing the links between various features and their influence on application categorization can provide insights into which elements are most crucial for differentiating between dangerous and benign apps. The graph showing the distribution of malware attacks in class is shown in Fig. 2.

With 5,560 Drebin malware applications and 9,476 benign apps, S and B are the two classified categories of the predicted class. It is the sole predicted class among the elements in the data frame.

There are 215 characteristics in the dataset, which is an essential distinction. Due to limitations, only the most significant characteristics will be displayed, as it is not feasible to show visualizations for all features. It should be noted that the attributed values are likewise categorical and fall within the range of 1 or 0.

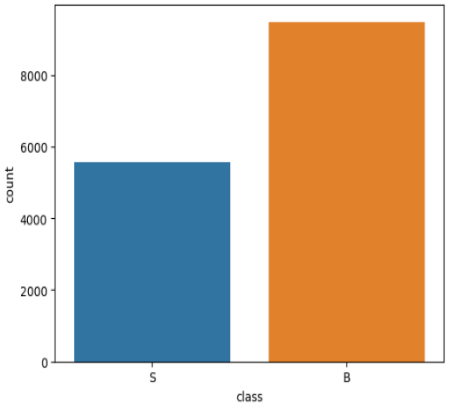


FIGURE 2. Distribution of malware attacks in class.

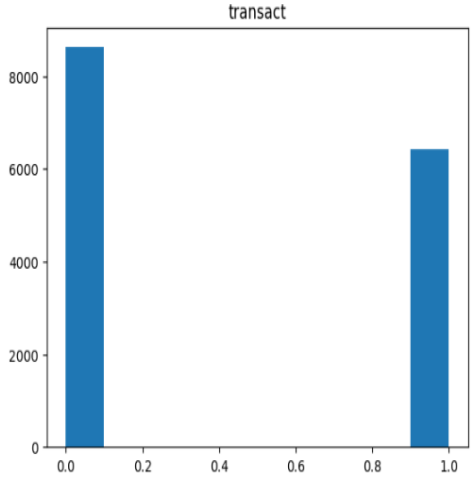


FIGURE 3. Transact feature values count.

In the Drebin Dataset, the "transact" feature can only take on binary values of 0 or 1. The usage of the "transact" technique of the Android Binder Inter-Process Communication (IPC) mechanism is indicated by this feature. When communicating between separate parts of an Android application, such as an activity and a service, one uses the "transact" method. Malicious programs can connect with other components using the "transact" technique that goes against Android security guidelines.

The Values Count of the Transact Feature graph is displayed above in Fig. 3. Use of the "Transact" approach is indicated by a value of "1" whereas "0" indicates not used. Researchers may learn more about how frequently malicious apps employ the "transact" approach compared to benign applications and how crucial this characteristic is for telling the two apart by examining this feature throughout the dataset.

The "onServiceConnected" feature in the Drebin Dataset Is also binary, which means it can have a value of either 0 or 1. The implementation of the "onServiceConnected" callback function, which is used to connect an Android service with an application component, such as an activity or a broadcast receiver, is represented by this feature.

A graph displaying the value of the service-connected feature is shown above in Fig. 4. The application implements

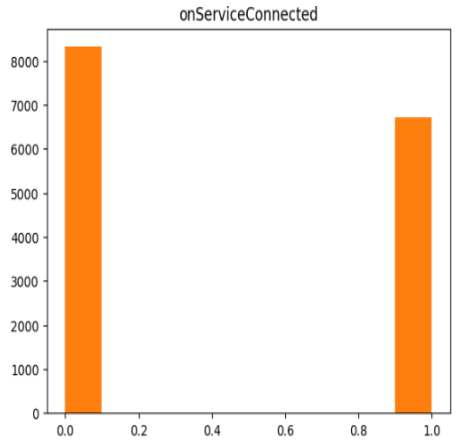


FIGURE 4. The value of the service-connected feature.

the ''onServiceConnected'' function if this feature has a value of 1, else it has a value of 0. Malicious programs may utilize the ''onServiceConnected'' function to connect to a vulnerable or malicious service, allowing them to conduct unauthorized operations on the device.

Another binary feature that can only accept values of 0 or 1 is the ''bindService'' feature in the Drebin Dataset, as seen in Fig. 5. The ''bindService'' function, which connects an Android service and an application component, is used to determine whether an application is using it.

When an application binds to a service using the ''bindService'' method, it can obtain a reference to the service's underlying ''Binder'' object, which can further be used to communicate with the service and carry out different operations, such as providing data and receiving callbacks. The ''bindService'' technique can be used by malicious apps to obtain confidential information or carry out illegal operations on the system. The value of 1 for the ''bindService'' feature denotes that the application uses the ''bindService'' method, whereas a value of 0 denotes the opposite.

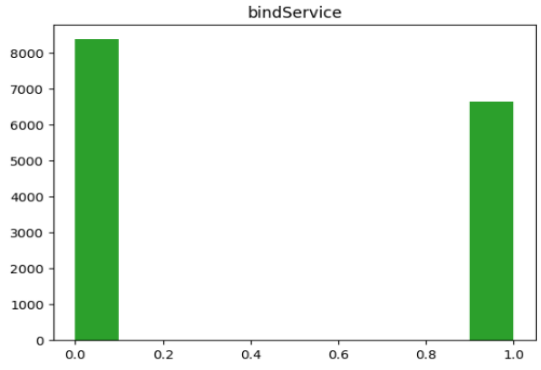


FIGURE 5. Value count for blind service feature.

The Android IPC system, which enables inter-application communication between various components of an application, is fundamentally dependent on the Binder object. A component can connect with other components Safely and in a more organized way by binding an interface to the Binder object using the ''attachInterface'' method. The ''attachInterface'' function may be used by malicious programs to access system resources without authorization or to connect with other components in a way that is against Android security guidelines.

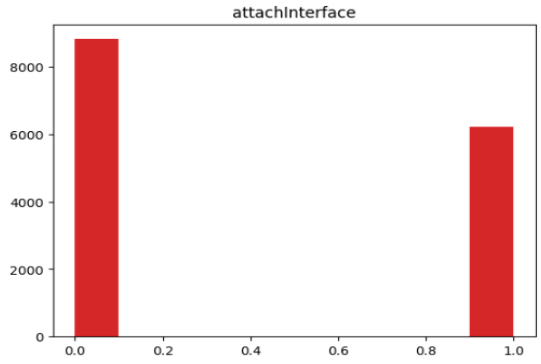


FIGURE 6. Value count for attack interface feature.

As shown in Fig. 6, an application employs the ''attachInterface'' technique if the ''attachInterface'' feature in the Drebin Dataset has a value of 1 instead of 0. The ''attachInterface'' function is used to attach an interface to an Android Binder object, and this feature indicates if the application uses that method.

When an application component connects to a service using the ''bindService'' function, it implements the ''ServiceConnection'' interface, which receives callbacks, relevant to the service connection lifecycle. Harmful apps may use the ''ServiceConnection'' interface to get unauthorized access to system resources or to carry out harmful operations on the device.

If the application in the Drebin Dataset has a value of 1 for the ''ServiceConnection'' feature, it implements the ''ServiceConnection'' interface otherwise, it does not. This attribute indicates that if the application implements the ''ServiceConnection'' interface, it will handle the callbacks associated with the connection between an Android service and an application component. The graph indicating the ServiceConnection Feature value count is displayed in Fig. 7.

SMaliciousapplications can exploit the ''SEND_SMS'' permission to send expensive messages, make money for the attacker, or even infect other devices with malware. That's why it is considered potentially harmful permission on Android. The Android operating system demands that users expressly grant this permission to apps before such applications may send SMS messages.

The value count graph for the Send SMS capability is shown in the above Fig. 8. When the ''SEND_SMS'' feature in the Drebin Dataset has a value of 1, then it means that the program makes use of the ''SEND_SMS'' permission, but when it has a value of 0, it does not. The ''SEND_SMS'' permission, which enables an application to send SMS messages

TABLE 3. Sample dataset.

	transact	On Service Connected	bind Service	attach Interface	Service Connection	android.os Binder	SEND SMS
0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
2	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
3	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
4	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
---			...	---	---		
15031	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
15032	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
15033	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
15034	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
15035	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

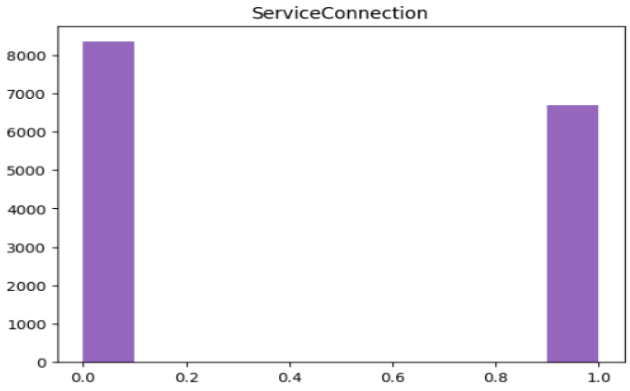


FIGURE 7. Value count for service connection feature.

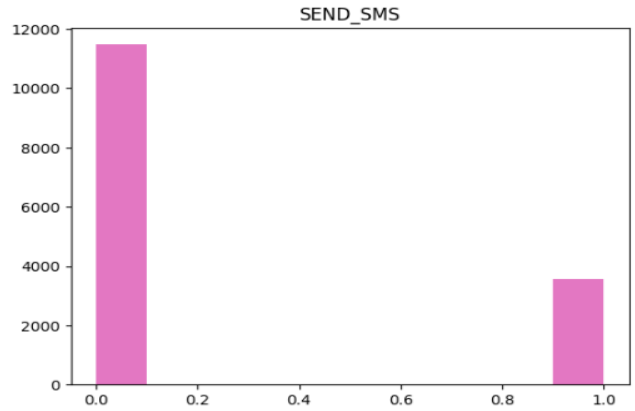


FIGURE 8. Value count for send SMS feature.

from the device without the user's permission, is represented by this feature.

While there are numerous features like Android. content.pm.packageinfo.kkeyspec, Dexclassloader, and secret key, the mentioned characteristics significantly influence the classification of malware infections as malicious or benign.

C. DATA PROCESSING

Any data analysis or machine learning effort, including the use of Drebin Dataset, must first analyze the data. Therefore, to process the dataset, you can follow these general procedures:

1. **DATA CLEANING:** The first stage in data processing is the cleaning of data. Duplicate entries may need to be removed, as well as any missing or null values and unnecessary or distracting characteristics [39].
2. **FEATURE SELECTION AND REDUCTION:** After creating new features, choose the ones that will help the machine learning model make the best predictions. Techniques like feature significance ranking, dimensionality reduction, and correlation analysis can be used. Critical data-processing stages like feature selection and feature reduction can enhance the effectiveness of machine learning models, lessen overfitting, and simplify the model [9].
3. **DATA SPLITTING:** The dataset has to be divided into training, validation, and testing sets before creating the machine learning model. The validation and testing sets are used to fine-tune the model's hyperparameters,

whereas the training set and testing set is used to train the model.

4. **NORMALIZATION AND SCALING:** It is important to normalize or scale the features so that they are at the same scale and have the same extent. Some machine learning models, such as the ones that use distance measurements, may perform better due to this.
5. **DATA CLEANING:** Finding invalid entries and other unrecognized symbols in the dataset is the initial stage in data cleansing. The dataset has no null entries, as seen in the table below.

The Null values information datastore is shown in Table 3 above. The next stage is to identify any symbols in the collection that are missing or unrecognized. To process the dataset, we will remove special characters like ''?'' and ''S'' or set them to NaN (Not a Number).

1) LABEL ENCODING

The categorical data must be transformed into numerical data before being utilized in a machine-learning model since many algorithms cannot handle them directly. For this purpose, a data preparation technique known as Label encoding is used to transform categorical data into numerical data. The two categorical items S and B in the class column of our dataset will be labeled encoded and turned into 0 and 1, as seen in the following example.

- Categorical Class
- B 9476
- S 5560
- Name: class
- Label Encoded
- 0 9476
- 1 5555

D. FEATURE SELECTION AND REDUCTION

By removing irrelevant features and concentrating on informative traits, reducing overfitting in high-dimensional datasets, and increasing computing efficiency, feature selection and dimensionality reduction are essential in machine learning for enhanced model performance. They also aid in model interpretability, deal with multicollinearity problems, handle scenarios involving little or no data, and enhance model generalization by simplifying the design. Additionally, by directing feature engineering efforts and helping to understand how features affect the target variable, these strategies help create models that are overall more effective and efficient. Due to their efficiency and applicability to the issue, SelectKBest and PCA were chosen for feature selection and dimensionality reduction in Android malware detection. To distinguish between legitimate and malicious apps, SelectKBest aids in the identification of the most significant features, enhancing predictive performance. However, PCA prevents overfitting and improves computational efficiency by deleting unimportant features and reducing data dimensionality by capturing key patterns. These techniques are well-known for enhancing model performance and interpretability and are often employed in machine learning, making them appropriate options for this study.

1) SELECTKBEST METHOD

SelectKBest is a feature selection technique that chooses the top k features based on their statistical importance with the target variable. This technique may be implemented using the SelectKBest class from the sci-kit-learn package. The fit transform function of this class takes the original data and the target variables as input and outputs the new dataset with the chosen top k features. Altogether, there are 215 features. During the first stage of feature reduction and selection, the choose K best approach was utilized to select the top 50 linked features. Despite the dataset's simplicity, the machine learning model may find it simple because each feature has two entries, 0 and 1. However, the dataset was standard-scaled to identify the most connected and standardized characteristics [40].

2) PCA (PRINCIPLE COMPONENT ANALYSIS)

The feature reduction technique PCA (Principal Component Analysis) lowers the dimensionality of the data by relocating it to a lower-dimensional space [41]. It functions by determining the main components of the data, which serves as directions in the feature space and represents the maximum variance in the data. This technique may be applied using the PCA class from the sci-kit-learn package. A new dataset with fewer features is returned by this class's fit-transform function, which takes given data as inputs. Using the PCA fit transform technique, the 50 features chosen from the chosen K best, were then put through PCA to minimize their number to normalize the remaining 40 features. Despite the initial dataset's size, we have selected only the top 40 features. When performing model training on the original data store without standardization, we expect to achieve the best results. However, the best practice for such a large dataset is to reduce the features to a reasonable size with the dataset standardized and normalized to achieve the top training and testing accuracy. The decreased feature datastore is shown in Table 4 below.

The outcomes of using the PCA approach to lessen the dataset's dimensionality are shown in Table 4. The table contains the PCA-reduced dataset with its original features. The columns PC32 through PC40 are the main components derived by PCA, and each row represents a particular occurrence or sample. The new orthogonal axes in the lower-dimensional space known as the principle components are what account for the majority of the variance in the data. The final column, ''labels,'' lists the appropriate class labels for each sample. The principal components in the table represent the new lower-dimensional representation of the data.

The performance of machine learning algorithms may be improved by reducing overfitting and the efficiency may be increased by lowering the dimensionality of the data, which

TABLE 4. PCA approach results.

PC32	PC33	PC34	PC35	PC36	PC37	PC38	PC39	PC40	labels
-0.769110	0.290244	0.572601	-0.259331	0.117823	0.382019	-0.011901	-0.215180	0.192850	1.0
0.087149	0.126745	-0.332061	-0.092596	-0.200895	-0.347682	0.028400	-0.202379	-0.629056	1.0
-0.153171	-0.399126	0.372040	0.208596	0.011325	-0.025678	0.341227	-0.379392	0.108309	1.0
-1.418605	-0.294062	-0.531261	0.508155	0.461734	-0.604897	-0.214293	-0.426843	0.508329	1.0
-0.457866	-0.130985	0.212512	0.041945	-0.363563	-0.122854	0.181935	0.582550	0.056341	1.0

SelectKBest and PCA are both excellent at doing. To ensure that the final dataset has enough information for the job at hand, it is crucial to consider that both procedures might lead to information loss. Therefore, it is important to consider how many features to select and reduce.

E. TRAIN-TEST SPLIT

In machine learning and data science, a technique known as the train-test split is employed to assess a model's performance. The train-test split is a technique that divides the available data into two sets: training the model and evaluating it.

An assigned percentage of the data is randomly split between a training set and a test set. A typical split distributes 70–80% of the data to the training set and the remaining 20–30% to the test set. However, the percentages might vary.

The dataset for our model is split into two parts: training and testing. 90% of the dataset is utilized for training and 10% for testing. These are the two new data stores with a total of 41 entries. The last item is of the response class, which contains two categorical answers, i.e., 1 or 0. The first 40 entries are the features, while the last entry is the response class.

- Train size = 13528
- Test size =1503.

F. MODEL DESIGN

One-shot learning aims to learn a classification problem using very little labeled data and in the case of one-shot learning tasks, a particular kind of neural network called a Siamese network is frequently employed. As a result, the application of the Siamese network is proved to be very effective since it is designed to learn the similarity between two input examples rather than their absolute categorization labels.

Although Siamese networks are frequently used for image data, they may also be utilized for other data types.

IV. MATERIALS & METHODS

The following is the process for designing a one-shot Siamese network with non-image data:

1. DATA PRE-PROCESSING: Before being sent to the Siamese network, non-image data must be pre-processed to guarantee it is in the proper format. Feature scalability, normalization, and categorical variable encoding are necessary for this situation.
2. DATA SPLITTING: A set of "support" and "query" instances The support cases are the labeled examples used to train the Siamese network. The network will be assessed using the query instances as the test cases.
3. MODEL ARCHITECTURE: The Siamese network comprises two identical sub-networks that use the same weights. A single input instance is taken by each sub-network, which then converts it to an embedding vector of a given length. Next, the two embedding vectors are contrasted using a distance function, such as cosine similarity or Euclidean distance. A binary classification judgment (i.e., whether the two occurrences are similar or different) is made using the obtained distance score.
4. Model training: Support instance pairs are used to train the Siamese network. The network calculates the distance score between the two embeddings for each pair to update the weights using a contrastive loss function. The network is encouraged to learn similar embeddings for the examples of the same class and dissimilar for cases of the other classes via the contrastive loss function.
5. MODEL EVALUATION: After training, the network may compare two query instances. The trained network is first used to embed the query instances, and then the comparison is performed using the same distance algorithm. Making a binary classification choice is possible using the resultant distance score.

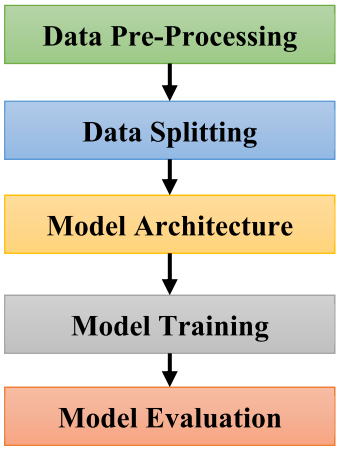


FIGURE 9. Steps for designing a one-shot siamese network.

Generally, a Siamese network for one-shot learning on non-image data entails pre-processing the data, dividing it into

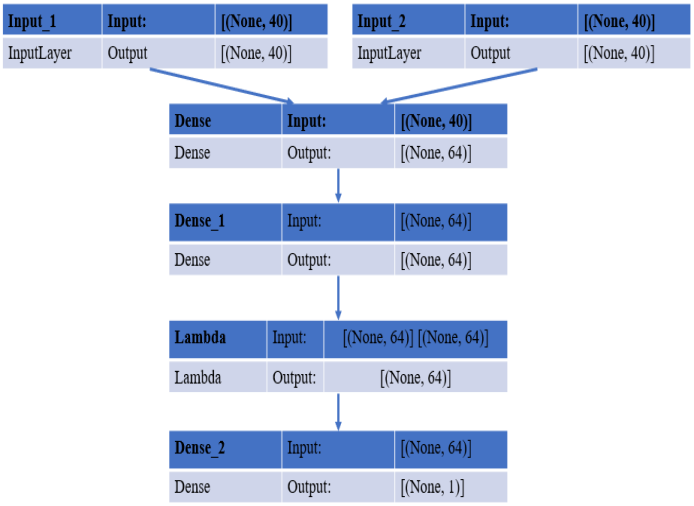


FIGURE 10. The siamese-shot model architecture of the proposed model.

support and query sets, constructing an appropriate architecture, training the network on pairs of support instances, and assessing the performance on pairs of query examples as shown in Fig. 9.

A. MODEL ARCHITECTURE

The one-shot learning that our Siamese network can do with non-image material is what it is intended for. Two identically shaped data points, each with a tuple of 40 values, serve as the model's input.

Two input layers were created, one for the left data points and another for the right data points, to process them. The variables' input-left, and input-right are given the definition of these input layers and the Input class from the Keras API.

Afterwards, we create a dense neural network using a ReLU activation function and 64 hidden units. The weights and biases used for both inputs are the same since the left and right input layers share this neural network. To do this, use the shared_dense_1 object to define the dense neural network as Dense (num_hidden_units, activation='relu') and then apply it to the left and right inputs. The suggested model's Siamese-Shot model architecture is shown below in Fig. 10.

Another dense neural network with 64 hidden units and a ReLU activation function is then used to repeat this procedure. This time, a separate shared_dense_2 object processes the left and right inputs, and the encoded left and corrected data points are saved in the encoded_left and encoded_right variables.

To determine their similarity, a lambda layer is created to compute the absolute difference between the encoded left and right data points. It is accomplished by utilizing the Lambda class from the Keras API, with the lambda function supplied as K.abs(tensors [0] - tensors [1]).

The output of the lambda layer is then sent through a fully connected layer with a sigmoid activation function to provide a prediction for binary classification. Using 1 output unit and a sigmoid activation function, the Dense class from the Keras API is used to do this.

The left and right input layers serve as the inputs and outputs of the Siamese network model, which is created using the Model class from the Keras API. Accuracy and a binary cross-entropy loss function are used as the evaluation metric during model construction.

The model is trained to utilize on-the-fly-generated training and validation data produced by a unique generating function. At the beginning of each training epoch, the generator function randomly chooses pairs of data points and the labels that go with them from the training or validation set, and then it feeds those pairs through the Siamese network. As a result, the model can be trained with an 8-batch size, which is effective even for big datasets.

In general, our Siamese network architecture is intended to learn a similarity metric between pairs of non-image input points, which may subsequently be used for one-shot learning tasks. ReLU and sigmoid activation functions guarantee that the model can learn intricate non-linear correlations between the input data points. At the same time, shared weights and bespoke data generators make it possible to train the model quickly on big datasets.

B. MODEL TRAINING

The Siamese network design, frequently employed for one-shot learning issues, was utilized for training our model. The model learned to predict whether two data points belong to the same class by using pairs of data points as its input. Two input layers were the first component of the model's design, followed by two shared dense layers with 64 hidden units each, a distance layer, and a sigmoid output layer. Utilizing stochastic gradient descent, the model was optimized after being trained using binary cross-entropy loss.

The training was conducted for 100 epochs using an 8-person batch size. Additionally, for each batch of data, this study employed a generator function to create pairs of data points, with 90% of the data being used for training and 10% for validation. To ensure the model was not overfitting the training data, the training and validation loss and accuracy were monitored during the training process.

Two efficient methods, stratified sampling, and class weights, were utilized to mitigate the influence of class imbalance during model training. Malware detection frequently faces the problem of class imbalance, where samples from one class outnumber those from other classes by a large margin. By using stratified sampling, each mini-batch used for training is guaranteed to contain a proportionate representation of the various classes, maintaining the original dataset's class distribution. This strategy ensures that the minority classes receive adequate attention during training and prevents the model from being skewed toward the dominant class. Class weights have also been included in the optimization process. The model concentrates more on correctly identifying data from the minority classes by giving these classes higher weights, which effectively addresses the class imbalance issue. These methods improve the model's capacity for generalization and correct classification of both majority and minority classes, making the Android malware detection system more effective and dependable.

A thorough evaluation of a machine learning model's performance must consider both the training and testing accuracies. It has several functions that contribute to the efficacy and dependability of the model. First off, it aids in the detection of overfitting, which occurs when a model becomes overly focused on memorizing training data and performs poorly when presented with new data. Overfitting can be detected by comparing the accuracies of training and testing data, and appropriate actions can be taken to correct it. The model's generalization ability, or how well it can categorize examples that have not yet been observed, is reflected in testing accuracy. A model can be trusted to function accurately in real-world circumstances if its testing accuracy is good. Thirdly, choosing the right model is aided by examining both accuracies. A model can be trusted to function accurately in real-world circumstances if its testing accuracy is good. Thirdly, choosing the right model is aided by examining both accuracies. A model may be excessively complex or overfitted if it has a high training accuracy but a low testing accuracy, in which case it should be simplified or regularized. On the other hand, low accuracy in training, as well as testing, may point to the need for more intricate models or better features.

Fig. 11 above shows the training and validation accuracy performance graph. The model's training accuracy reached above 98%, indicating that the model could learn the patterns in the training data very well. However, the model can overfit the training data, leading to poor performance on new data. To avoid overfitting, the validation loss and accuracy were monitored during training. The model's validation accuracy

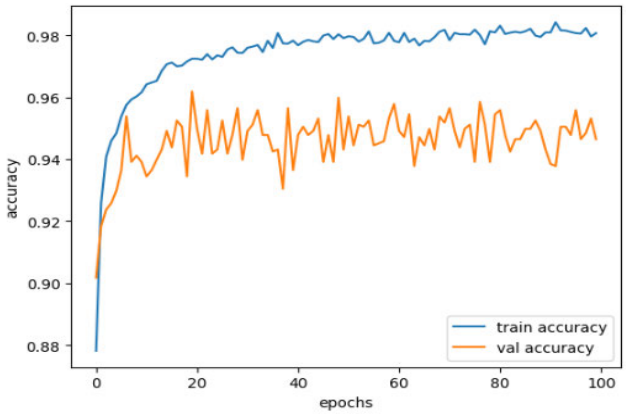


FIGURE 11. Training and validation accuracy performance.

reached around 95%, indicating its capability to generalize well to new data without overfitting the training data.

The performance of the Siamese-Shot model's training and validation losses is shown in Figure 12. Monitoring the loss function is essential since it gives information on the generalization and convergence abilities of the model. In Fig. 12, the training and validation loss both exhibit a declining trend, demonstrating that the model's weights are being efficiently adjusted and that the model is picking up new information from the data. The model successfully minimises the binary cross-entropy loss on the training data, as seen by the decreasing training loss, which also shows how well the model fits the training data. As the model generalizes to the validation data, the validation loss likewise declines, supporting this.

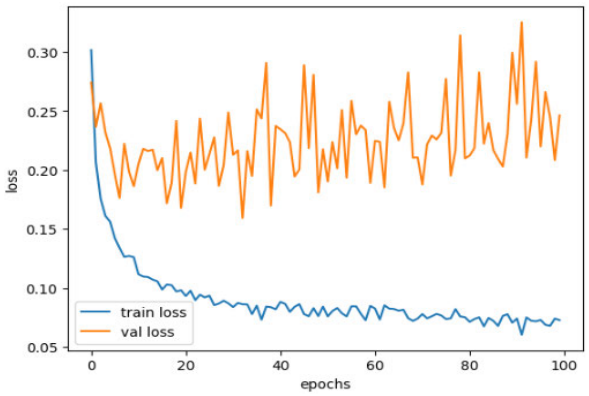


FIGURE 12. Training and validation loss performance.

C. MODEL ANALYSIS

Using Siamese one-shot learning, our work offers a unique method for identifying Android malware. On the Drebin dataset, the suggested model has an accuracy of 98.9%, outperforming modern methods. Our strategy is based on the idea of Siamese networks, where two identical neural networks are trained to learn a similarity metric between pairs of data. When there is a lack of accessible data, as is

frequently the situation when detecting malware, this method is quite helpful.

The Siamese one-shot learning method simultaneously trains two neural networks, each handling a separate input sample. The input samples are first encoded by the shared layers of the two networks into a fixed-size feature representation, which is then utilized to calculate a similarity score between the two samples. If the samples are members of the same class or not, the network's final layer generates a binary output to show this.

One advantage of our technique is that it can adequately categorize fresh samples even if it has only seen a single example of each class during training. Since the Siamese network is trained to acquire a general feature representation, any two inputs, regardless of their unique properties, may be compared using this.

The Siamese network is depicted as follows:

Let x1 and x2 be two input vectors of size n.

The feature representation of x1 is given by f(x1), and x2 is given by f(x2).

The distance between the two feature representations is given by:

$$d(f(x1), f(x2)) \tag{1}$$

The output of the final dense layer is given by:

$$y = \sigma(Wd * d + b) \tag{2}$$

where σ is the sigmoid activation function, Wd is the weight matrix of the dense layer, and b is the bias vector.

With positive results in identifying Android malware, our Siamese one-shot learning method can be used in other fields where one-shot learning is advantageous.

In contrast to current models, our method has some benefits. The first benefit is that it takes fewer data for training, which is crucial in malware detection because gathering vast amounts of labeled data can be difficult. As a result of the Siamese network's capacity to learn a strong similarity score, our technique can generalize well to fresh, previously undiscovered malware samples. Hence, the computational efficiency of our method makes it appropriate for usage on portable devices.

The Euclidean distance between the encoded feature representations of two input samples:

$$distance = sqrt(sum((encoded\_left - encoded\_right) \wedge 2)) \tag{3}$$

The contrastive loss function used during training encourages the network to learn a good similarity metric:

$$L = (1 - Y) * 0.5 * D \wedge 2 + Y * 0.5 * max(0, m - D) \wedge 2 \tag{4}$$

Y is the binary label (1 for a positive pair, 0 for a negative pair), D is the distance between the encoded feature representations of the two input samples, and m is the margin hyperparameter.

The sigmoid activation function is used in the final layer of the network to produce a binary output:

$$output = 1/(1 + exp(-z)) \tag{5}$$

where z is the output of the previous layer.

Siamese one-shot learning is used in our work to propose a unique method for identifying Android malware. The proposed model outperforms current models on the Drebin dataset and comes with several benefits, such as the capacity to acquire a reliable similarity metric with fewer data and generalization to new, undiscovered malware samples.

V. RESULTS

The n-way k-shot accuracy approach is used to assess the Siamese network. This method tests the model on n-way classification tasks using k-shot examples per class. Using a dataset with examples from n distinct classes, the model is trained before being tested to determine how well it can classify fresh instances from those n classes using k-shot examples for each class.

In a 5-way 1-shot task, the model is evaluated on its capacity to properly categorize a new sample into one of five classes, given just one example of each of those five classes as input.

A machine-learning model's performance on a test dataset is evaluated through a procedure known as model evaluation. Model assessment measures the model's ability to generalize results to new data and to assess whether it performs well enough for practical applications.

Metrics like recall, precision, F1 score, and confusion matrix may also be used to assess the model's performance. The fraction of true positive cases that are accurately marked as positive is the recall, which estimates how well the model can detect true positive cases.

Precision (i.e., the percentage of positive predicted positive cases) reflects the model's ability to identify genuine positives out of all anticipated positives correctly. An accurate evaluation of the model's performance is provided by the F1 score, which is a harmonic mean of recall and accuracy. Table 5 below displays the suggested model's assessment metrics.

TABLE 5. Evaluation metrics of the proposed model.

Evaluation metric	Performance value
Final n-way k-shot accuracy	0.994
Accuracy	0.985
Precision	0.989
Recall	0.988
F1 score	0.987

The number of n ways is [5, 10, 15]

The number of k shots is [1, 5]

The accuracy of n-ways and k-shots is as follows:

-way -shot accuracy: 0.995130086724483
-way -shot accuracy: 0.989601956860129
-way -shot accuracy: 0.9941205247943073
-way -shot accuracy: 0.9833311096286413
-way -shot accuracy: 0.9923015343562375
-way -shot accuracy: 0.9769179452968646

The true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) in the predictions are depicted in Fig. 13 mentioned above. The actual class labels are represented in the rows, and the predicted class labels are displayed in the columns. The ROC (Receiver Operating Characteristic) curve is another helpful indicator for assessing the model's effectiveness.

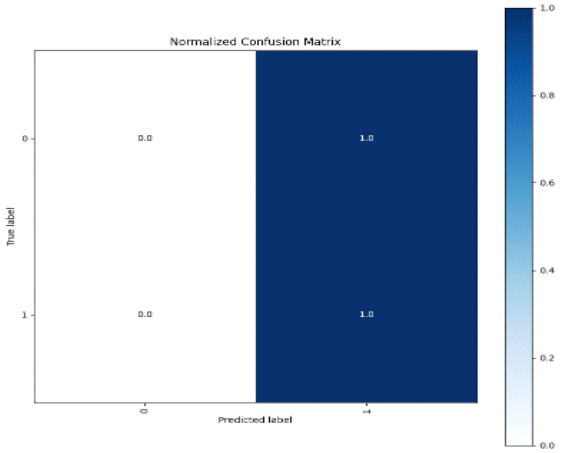


FIGURE 13. Confusion matrix based on test data.

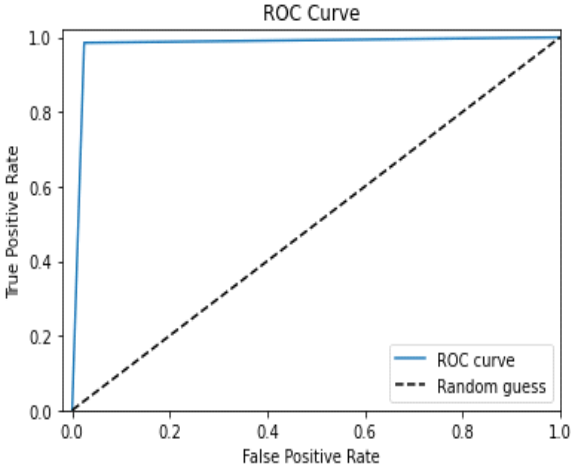


FIGURE 14. ROC curve based on test-data.

The true positive rate (TPR) vs. the false positive rate (FPR) at various threshold settings is displayed on the ROC curve plot in Fig. 14. An indicator of how well a model performs by correctly classifying both positive and negative cases, is the AUC (Area Under the Curve) on the ROC curve. A greater AUC indicates better performance. When describing the effectiveness of a classifier, the area under the ROC curve (AUC) is frequently employed as a summary statistic. An AUC of 0.5 means the classifier has no predictive power, whereas an AUC of 1.0 indicates a flawless classifier. The performance of a classifier is often more significant when the AUC is larger. Given that our model has an Area under the Curve of 1, it will likely produce accurate predictions.

A. MODEL COMPARISON ANALYSIS

Numerous strategies, including LSTM [9], random forest classifier [26], and SVM [27], have been investigated in malware identification. The accuracy of research on malware detection in 2023 that used deep learning and correlation-based feature selection was 94.59%. On the Drebin dataset, a better binary owl feature selection technique in 2022 that utilized a random forest classifier had an accuracy of 98.84%. On the same dataset, another research from the same year employed SVM and reached an accuracy of 80.71%. The accuracy of our method, which used a Siamese one-shot learning methodology, on the Drebin dataset was 98.9% (as shown in Table 6). Our technique outperformed earlier studies, and various issues may have played a role in this accomplishment.

TABLE 6. Proposed model comparison with related work.

Reference	Approach	Accuracy	Dataset
[9]	LSTM	94.59%	Drebin Dataset
[26]	Random Forest Classifier	98.84%	Drebin Dataset
[27]	SVM	80.71%	Drebin Dataset
[27]	LSTM	97.39%	Drebin Dataset
[48]	GNNs	98.68%	Drebin Dataset
Our approach	Siamese One-Shot Learning	98.90%	Drebin Dataset

First, the Siamese one-shot learning approach works well in various settings, including image recognition and natural language processing. It is an innovative strategy, as it's the first time this technology has been used to identify malware.

In addition, our method only needs a small amount of labeled data, which is frequently a bottleneck in malware identification. Also, it uses a one-shot learning approach, which trains the model to detect a new malware sample

using just one instance of each malware family. By using this method, data labeling costs and turnaround times may be drastically decreased.

Using a distance-based metric learning algorithm, our method assesses the similarity between two malware samples. In contrast to previous methods, this one can more accurately capture how similar malware samples are, while also assisting the model in distinguishing between related malware families.

Furthermore, overfitting or underfitting was successfully avoided with the employed method, which can be challenging with deep-learning models. Despite this, the model achieved excellent accuracy. Additionally, various measures were utilized to evaluate the model's performance, including recall, F1 score, accuracy, confusion matrix, and ROC curve.

Our method of applying the Siamese one-shot learning methodology is unique, effective, and efficient in detecting malware that beats earlier efforts in terms of accuracy and necessitates less labeled data.

VI. DISCUSSION

The adoption of smartphones with new capabilities and related Android applications has grown with the rapid advancement of technology. The increasing complexity and security flaws in Android applications can be exploited easily by hackers. This further leads to the creation of difficulties in terms of security measures, for the researchers and developers of these applications.

To prevent security gaps from arising inside this network, machine and deep learning methods are utilized to monitor the detection of harmful attempts made against Android applications. The current research contributes to cybersecurity by finding abnormalities in signature databases and enabling the system to identify unknown threats using the Siamese one-shot learning approach.

For this purpose, over 120,000 Android applications are included in one of the most extensive freely available databases. Researchers utilized this dataset with parameters extracted from each Android application to create and test machine-learning algorithms for malware detection on Android smartphones. With a 98.9% accuracy rate for recognizing Android malware on the Drebin dataset, Siamese one-shot learning proves to be a novel technique that outperforms all existing models. It involves training two neural networks concurrently, each analyzing a separate input sample. Also, the n-way k-shot accuracy approach may assess the Siamese network by training it with a dataset of n-shot samples for each class and then identifying fresh instances from those classes.

Furthermore, numerous methods, including LSTM [9], random forest classifier [26], and SVM [27], are being researched in the challenging field of malware identification. For example, Deep learning and correlation-based feature selection were used in a 2023 study on malware detection, and the accuracy was 94.59%. On the Drebin dataset in 2022, an improved binary owl feature selection strategy with a random forest classifier was successful at 98.84%, while another study used SVM at 80.71% using a Siamese one-shot learning method. Several reasons why all three strategies fared better than the earlier studies. Our Siamese one-shot learning technique fared better on the Drebin dataset than in previous work, with an accuracy of 98.9%. Additionally, this one-shot learning approach by the Siamese is a state-of-the-art malware detection technique that can detect new malware samples by training on only one instance of each malware family, without requiring large amounts of labeling data. A distance-based metric learning strategy that achieved high accuracy without overfitting or underfitting was used to determine how similar the two malware samples were. Our model's effectiveness was evaluated using various metrics, including recall, F1 score, accuracy, confusion matrix, and ROC curve. Therefore, the Siamese one-shot learning methodology, a novel and promising technique, may be used instead of traditional classification algorithms to detect Android malware. Other sectors where similarity-based learning is needed but labeled data is rare may use this method to improve their systems.

VII. CONCLUSION

Siamese networks, which have excelled at one-shot learning tasks even when working with non-image data, are leveraged by the model architecture used in this study. The model is specifically made to process pairs of data points, each of which contains a tuple with 40 values. The learning process is aided by the use of shared neural network layers, and the model can capture intricate non-linear relationships in the input data by utilizing ReLU and sigmoid activation functions. Importantly, the model architecture exhibits strong generalization abilities and avoids overfitting, as shown by the training and validation accuracy and loss performance metrics. By using pairs of data points as input, the Siamese network is trained to determine whether two data points belong to the same class. This training procedure uses a generator function to generate pairs of data points over the course of 100 epochs and an 8-person batch size. In the study, class weights and stratified sampling are used during model optimization to address issues with class imbalance that are frequently present in malware detection. The effectiveness of the model is thoroughly assessed, accounting for both testing and training accuracy.

Siamese one-shot learning emerges as a potent and successful method for Android malware detection, according to the model analysis. Notably, the model outperforms earlier techniques like LSTM, random forest classifier, and SVM, achieving an impressive accuracy rate of 98.9% on the Drebin dataset. It is highlighted how the Siamese network can build a strong similarity metric even with sparse labeled data and how well it generalizes to malware samples that haven't been seen before. The performance of the model is fully analyzed using a set of evaluation metrics, including recall, precision, F1 score, and confusion matrix. The model is also effective at differentiating between positive and negative cases, as shown

by the ROC curve and AUC, with an AUC of 1 indicating exceptional predictive abilities. Through comparison with earlier models, the model's superiority is confirmed, reiterating its status as a cutting-edge method for Android malware detection. In this study, we introduce a novel and effective Siamese one-shot learning-based methodology for Android malware detection. In addition to achieving remarkable accuracy, the model also addresses important issues like a lack of labeled data and class imbalance. It offers a promising solution for practical malware detection applications, offering improved security and reliability by outperforming current approaches.

Future research will concentrate on increasing accuracy with a higher N-way value and investigating Siamese network developments. To find the best implementation for early-stage malware prevention, a comparison evaluation of the various approaches used in the one-shot malware categorization procedure can also be done.

ACKNOWLEDGMENT

This project is funded by Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R319), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

REFERENCES

[1] M. Kim, D. Kim, C. Hwang, S. Cho, S. Han, and M. Park, "Machine-learning-based Android malware family classification using built-in and custom permissions," *Appl. Sci.*, vol. 11, no. 21, p. 10244, Nov. 2021.

[2] İ. Atacak, "An ensemble approach based on fuzzy logic using machine learning classifiers for Android malware detection," *Appl. Sci.*, vol. 13, no. 3, p. 1484, Jan. 2023.

[3] S. A. Nikale and S. Purohit, "Comparative analysis of Android application dissection and analysis tools for identifying malware attributes," in *Big Data Analytics and Intelligent Systems for Cyber Threat Intelligence*. Denmark: River Publishers, 2023, pp. 87–103.

[4] H. Gao, S. Cheng, and W. Zhang, "GDroid: Android malware detection and classification with graph convolutional network," *Comput. Secur.*, vol. 106, Jul. 2021, Art. no. 102264.

[5] J. Senanayake, H. Kalutarage, and M. O. Al-Kadri, "Android mobile malware detection using machine learning: A systematic review," *Electronics*, vol. 10, no. 13, p. 1606, Jul. 2021.

[6] K. Shaukat, S. Luo, and V. Varadharajan, "A novel deep learning-based approach for malware detection," *Eng. Appl. Artif. Intell.*, vol. 122, Jun. 2023, Art. no. 106030.

[7] A. Taha and O. Barukab, "Android malware classification using optimized ensemble learning based on genetic algorithms," *Sustainability*, vol. 14, no. 21, p. 14406, Nov. 2022.

[8] S. C. Sethuraman, "A comprehensive survey on deep learning based malware detection techniques," *Comput. Sci. Rev.*, vol. 47, Feb. 2023, Art. no. 100529.

[9] E. S. Alomari, R. R. Nuiaa, Z. A. A. Alyasseri, H. J. Mohammed, N. S. Sani, M. I. Esa, and B. A. Musawi, "Malware detection using deep learning and correlation-based feature selection," *Symmetry*, vol. 15, no. 1, p. 123, Jan. 2023.

[10] J. Zhu, J. Jang-Jaccard, A. Singh, P. A. Watters, and S. Camtepe, "Task-aware meta learning-based Siamese neural network for classifying obfuscated malware," 2021, *arXiv:2110.13409*.

[11] F. Alswaina and K. Elleithy, "Android malware family classification and analysis: Current status and future directions," *Electronics*, vol. 9, no. 6, p. 942, Jun. 2020.

[12] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing features for detecting Android malicious applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[13] V. J. Raymond and R. J. R. Raj, "Investigation of Android malware using deep learning approach," *Intell. Autom. Soft Comput.*, vol. 35, no. 2, pp. 2413–2429, 2023.

[14] N. Xie, Z. Qin, and X. Di, "GA-StackingMD: Android malware detection method based on genetic algorithm optimized stacking," *Appl. Sci.*, vol. 13, no. 4, p. 2629, Feb. 2023.

[15] D. Smith, S. Khorsandroo, and K. Roy, "Supervised and unsupervised learning techniques utilizing malware datasets," in *Proc. IEEE 2nd Int. Conf. AI Cybersecur. (ICAIC)*, Feb. 2023, pp. 1–7.

[16] B. A. Mantoo and S. S. Khurana, "Static, dynamic and intrinsic features based Android malware detection using machine learning," in *Proc. ICRIC, Recent Innov. Comput.*, 2020, pp. 31–45.

[17] S. Yen, M. Moh, and T.-S. Moh, "Detecting compromised social network accounts using deep learning for behavior and text analyses," *Int. J. Cloud Appl. Comput.*, vol. 11, no. 2, pp. 97–109, Apr. 2021.

[18] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of Android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124579–124607, 2020.

[19] P. Bhat and K. Dutta, "A survey on various threats and current state of security in Android platform," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 1–35, Jan. 2020.

[20] T. Sharma and D. Rattan, "Malicious application detection in Android—A systematic literature review," *Comput. Sci. Rev.*, vol. 40, May 2021, Art. no. 100373.

[21] M. A. Azad, F. Riaz, A. Aftab, S. K. J. Rizvi, J. Arshad, and H. F. Atlam, "DEEPSEL: A novel feature selection for early identification of malware in mobile applications," *Future Gener. Comput. Syst.*, vol. 129, pp. 54–63, Apr. 2022.

[22] V. Kouliaridis and G. Kambourakis, "A comprehensive survey on machine learning techniques for Android malware detection," *Information*, vol. 12, no. 5, p. 185, Apr. 2021.

[23] L. Cai, Y. Li, and Z. Xiong, "JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters," *Comput. Secur.*, vol. 100, Jan. 2021, Art. no. 102086.

[24] R. Thangaveloo, W. W. Jing, C. K. Leng, and J. Abdullah, "DATDroid: Dynamic analysis technique in Android malware detection," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 10, no. 2, p. 536, Mar. 2020.

[25] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in Android byte-code through an end-to-end deep system," *Future Gener. Comput. Syst.*, vol. 102, pp. 112–126, Jan. 2020.

[26] H. Alazzam, A. Al-Adwan, O. Abualghanam, E. Alhenawi, and A. Alsmady, "An improved binary owl feature selection in the context of Android malware detection," *Computers*, vol. 11, no. 12, p. 173, Nov. 2022.

[27] H. Alkahtani and T. H. H. Aldhyani, "Artificial intelligence algorithms for malware detection in Android-operated mobile devices," *Sensors*, vol. 22, no. 6, p. 2268, Mar. 2022.

[28] L. N. Vu and S. Jung, "AdMat: A CNN-on-matrix approach to Android malware detection and classification," *IEEE Access*, vol. 9, pp. 39680–39694, 2021.

[29] S. Millar, N. McLaughlin, J. M. del Rincon, and P. Miller, "Multi-view deep learning for zero-day Android malware detection," *J. Inf. Secur. Appl.*, vol. 58, May 2021, Art. no. 102718.

[30] A. Vishnoi, P. Mishra, C. Negi, and S. K. Peddoju, "Android malware detection techniques in traditional and cloud computing platforms: A state-of-the-art survey," *Int. J. Cloud Appl. Comput.*, vol. 11, no. 4, pp. 113–135, Oct. 2021.

[31] O. A. Alzubi, J. A. Alzubi, A. M. Al-Zoubi, M. A. Hassonah, and U. Kose, "An efficient malware detection approach with feature weighting based on Harris hawks optimization," *Cluster Comput.*, vol. 25, no. 4, pp. 2369–2387, Aug. 2022, doi: 10.1007/s10586-021-03459-1.

[32] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, "MAPAS: A practical deep learning-based Android malware detection system," *Int. J. Inf. Secur.*, vol. 21, no. 4, pp. 725–738, Aug. 2022, doi: 10.1007/s10207-022-00579-6.

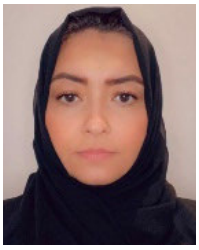
[33] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, "A two-stage deep learning framework for image-based Android malware detection and variant classification," *Comput. Intell.*, vol. 38, no. 5, pp. 1748–1771, Oct. 2022, doi: 10.1111/coin.12532.

[34] E. C. Bayazit, O. K. Sahingoz, and B. Dogan, "A deep learning based Android malware detection system with static analysis," in *Proc. 4th Int. Congr. Hum.-Comput. Interact. Optim. Robot. Appl.*, 2022, pp. 1–6, doi: 10.1109/HORA55278.2022.9800057.

[35] Y. Wu, M. Li, Q. Zeng, T. Yang, J. Wang, Z. Fang, and L. Cheng, "DroidRL: Feature selection for Android malware detection with reinforcement learning," *Comput. Secur.*, vol. 128, May 2023, Art. no. 103126, doi: 10.1016/j.cose.2023.103126.
[36] S. Fallah and A. J. Bidgoly, "Android malware detection using network traffic based on sequential deep learning models," *Softw., Pract. Exper.*, vol. 52, no. 9, pp. 1987–2004, Sep. 2022, doi: 10.1002/spe.3112.
[37] *The Drebin Dataset*. Accessed: Oct. 2012. [Online]. Available: https://www.sec.cs.tu-bs.de/~danarp/drebin/
[38] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.
[39] C. Liu, J. Lu, W. Feng, E. Du, L. Di, and Z. Song, "MobiPCR: Efficient, accurate, and strict ML-based mobile malware detection," *Future Gener. Comput. Syst.*, vol. 144, pp. 140–150, Jul. 2023.
[40] N. Peppes, T. Alexakis, E. Adamopoulou, and K. Demestichas, "The effectiveness of zero-day attacks data samples generated via GANs on deep learning classifiers," *Sensors*, vol. 23, no. 2, p. 900, Jan. 2023.
[41] Y. Sharma and A. Arora, "Ipanalyzer: A novel Android malware detection system using ranked intents and permissions," Delhi Technol. Univ., India, Tech. Rep., 2023.
[42] H. Babbar, S. Rani, D. K. Sah, S. A. AlQahtani, and A. K. Bashir, "Detection of Android malware in the Internet of Things through the K-nearest neighbor algorithm," *Sensors*, vol. 23, no. 16, p. 7256, Aug. 2023.
[43] V. J. Raymond and R. J. R. Raj, "Investigation of Android malware with machine learning classifiers using enhanced PCA algorithm," *Comput. Syst. Sci. Eng.*, vol. 44, no. 3, pp. 2147–2163, 2023.
[44] S. Aurangzeb and M. Aleem, "Evaluation and classification of obfuscated Android malware through deep learning using ensemble voting mechanism," *Sci. Rep.*, vol. 13, no. 1, p. 3093, Feb. 2023.
[45] H. H. R. Manzil and S. M. Naik, "Android malware category detection using a novel feature vector-based machine learning model," *Cybersecurity*, vol. 6, no. 1, p. 6, Mar. 2023.
[46] P. Liu, W. Wang, S. Zhang, and H. Song, "ImageDroid: Using deep learning to efficiently detect Android malware and automatically mark malicious features," *Secur. Commun. Netw.*, vol. 2023, pp. 1–11, Apr. 2023.
[47] A. Albakri, F. Alhayan, N. Alturki, S. Ahamed, and S. Shamsudheen, "Metaheuristics with deep learning model for cybersecurity and Android malware detection and classification," *Appl. Sci.*, vol. 13, no. 4, p. 2172, Feb. 2023.
[48] R. Yumlembam, B. Issac, S. M. Jacob, and L. Yang, "IoT-based Android malware detection using graph neural network with adversarial defense," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 8432–8444, May 2023.

FAHDAH A. ALMARSHAD is currently a Professor (an Assistant Professor) with Prince Sattam bin Abdulaziz University, Saudi Arabia. Her current research interests include computer science, machine learning, cyber security standards, information assurance, security risks, security policies, security culture, and the Internet of Things security.

MOHAMMED ZAKARIAH (Member, IEEE) received the bachelor's, master's, and Ph.D. degrees in computer science. He is currently a Senior Researcher with the College of Computer and Information Sciences, King Saud University. He has published 50 articles in reputed ISI-indexed journals. His current research interests include image processing, speech processing, signal processing, cybersecurity, and healthcare.



GHADA ABDALAZIZ GASHGARI received the master's degree in internet security and forensics from Curtin University, Australia, and the Ph.D. degree in computer science from the University of Southampton, U.K. She is currently an Assistant Professor with the Department of Cybersecurity, College of Computer Science and Engineering, University of Jeddah, Saudi Arabia. Her current research interests include cybersecurity, cybersecurity governance, information assurance, spam, malware, and phishing detection systems.

EMAN ABDULLAH ALDAKHEEL received the B.Sc. degree in computer science from Imam Abdulrahman Bin Faisal University, the M.Sc. degree from Bowling Green State University, and the Ph.D. degree in computer science from the University of Illinois at Chicago. She is currently an Assistant Professor and the Program Leader with the Department of Computer Sciences, Princess Nourah bint AbdulRahman University, Riyadh, Saudi Arabia. She has published her work in computer science in research publications, including the International Conference on Software Engineering (ICSE), the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), and the *Journal of Management Information and Decision Sciences*. Her current research interests include high-performance computing, automatic parallelization, cloud computing, concurrent and parallel software, and AI.

ABDULLAH I. A. ALZAHRANI received the B.S. degree in computer science from Taif University, in 2009, the M.S. degree in information security and assurance from Middle Tennessee State University, in 2016, and the Ph.D. degree in computer science from the University of Southampton, in 2020. He is an Assistant Professor with the Computer Science Department, College of Science and Humanities, Shaqra University, Alqwai'iyah, Saudi Arabia, and the Vice Dean for Educational Affairs with the College of Science and Humanities, Shaqra University. His research interests include information security, data science, and the Internet of Things (IoT).