EXAM: Explainable Models for Analyzing Malicious Android Applications

K. A. Asmitha, P. Vinod(✉), K. A. Rafidha Rehiman, Raman Prakash Verma, Rajkishor Kumar, Surbhi Kumari, and Nishchaya Kumar

Department Computer Applications, Cochin University of Science and Technology, Kochi, India
asmitha@pg.cusat.ac.in, {vinod.p,rafidharehimanka}@cusat.ac.in

Abstract. The open source nature and high performance have made Android smartphones popular world wide. On the other hand, the ease of usage and popularity has prompted malware creation. The proposed method presents a lightweight solution capable of detecting unknown malware on Android smartphones based on static analysis of android.apk files. Here we extract three different kinds of features i.e. permissions, activities and receivers, in order to evaluate if individual features are effective in detecting malware. Experiments suggest that our proposed deep learnig detection method is able to identify Android malware with an overall classification accuracy of 97.35% using boolean representation of the feature vector table. Comparative analysis of individual features recommends that the deep learning model resulted in better detection rate with permission feature. We also performed obfuscation of selected malware.apk files and found that the detection rate of our trained model is about 100%. Moreover, we also show how explainability helps the analyst to assess different models.

Keywords: Android malware · Static analysis · Obfuscation · Explainable models · Deep learning

1 Introduction

Android is the most popular mobile operating system in the world with an 84 percent market share for smartphones. The open-source nature of Android applications and easiness of usage have led to an increase in the prevalence of security attacks. According to statistics, more than 50 million instances of malware and potentially unwanted apps for Android have been found [18]. Various commercial antivirus solutions, including McAfee, Avast, Kaspersky, BullGuard, Avira and Bitdefender were developed as a solution for the consequences of the threat. However, because they rely on the signatures of known malicious apps, they have a serious flaw that prevents them from detecting new malware. In order to safeguard users from evolving malware, the academic community has been emphasizing on designing effective methods for malware detection that employ machine learning or deep learning algorithm [1–3]. Many malware detection methods

S. Rajagopal et al. (Eds.): ASCIS 2022, CCIS 1760, pp. 44–58, 2022.
https://doi.org/10.1007/978-3-031-23095-0_3

leveraging deep learning algorithms have recently been proposed. The researchers have addressed the malware detection that are based on a certain collection of attributes extracted from mobile applications in different approaches. The characteristics chosen for the proposed model may vary from static ones like commonly used APIs, permissions, and libraries to more action-related aspects like system call graphs. Nevertheless, these methods cannot provide the explanations of any kind of decision considered by the proposed model.

Machine learning algorithms create a predictive model to map features to classes through a training phase. However, biases might lead to inaccurate and unfair choices. The biggest drawback of black box method is that machine learning developers can't explain how the model came to a particular conclusion, especially when the model made a wrong decision. There comes the importance of Explainable Artificial Intelligence(XAI) and it is becoming a research topic in recent times [8]. XAI consists of techniques that helps the professionals to understand the results of their solutions by explaining the models.

This article suggests an obfuscation-resilient explainable method aimed at detecting Android malware, allowing security analysts to interpret and evaluate the predictions immediately. The main contributions are as follows:

- Propose a Deep learning based Android malware detection method using Permission, Activities and Receivers as features.
- Present an extensive research on several conventional machine learning models for Android malware detection.
- Analyze the effect of obfuscation on the effectiveness of the proposed approach.
- Demonstrate the attributes that our framework has learnt using explainable SHAP in order to assess how well it can distinguish between malware and benign.

The remaining sections of the article are arranged as follows: Sect. 2 presents the Related Works. The proposed method is introduced in Sect. 3. Section 4 details the experiments and test results. The conclusion and future work are covered in Sect. 5.

2 Related Works

Koli et al. [13] present a static analysis-based method for identifying malware on the Android operating system. It uses risky permission combinations and dubious API requests to train the SVM algorithm.

Abdulrahman et al. [9] proposed a new method based on deep learning for identifying malware applications that employ pseudo-dynamic analysis. The researchers instead developed an API call graph to represent the execution routes that malicious apps may follow during its entire duration. By comparing multiple methods and fine-tuning various network setup settings, they also focused on increasing network efficiency.

To identify Android malware, Suleiman et al. [10] devised a categorization technique based on parallel machine learning. Total 179 features were collected and separated into API calls, instructions and permissions. A parallel collection of heterogeneous classifiers, including Simple Naive Bayes, Logistic, RIDOR, PART, and Decision Tree, were used

to create a composite classification model. PART surpassed all the other classifiers, achieving accuracy rates of 96%.

A lightweight machine learning-based system was proposed by Long et al. in [11] to distinguish between benign and malicious applications using both static and dynamic techniques. Additionally, he suggested a novel strategy for reducing the dimensionality of the features called PCA-RELIEF. By utilizing both their SVM model and the newly presented model to lower the dimensions, their study demonstrated a high degree of effectiveness in identifying malware and improving the detection rate.

Alhebsi, Mohamed Salem [12], proposed a technique to scan the application to identify malware using two types of features permissions and signature. They found that K-Nearest Neighbour(KNN) and Random Forest (RF) classifiers are effective in terms of detection rate.

The detailed review of the above papers shows enough space to enhance and construct new solutions for detecting Android malware employing explainable machine learning/deep learning models. Explainable models in Android malware detection are crucial as deep learning models are challenging to evaluate, since they cannot be broken down into simple, intuitive parts. Moreover, obfuscation-resiliency is needed as developers always use sophisticated obfuscations to conceal their dangerous activities.

The challenges mentioned above are addressed in our study by evaluating the model's resistance against obfuscation and generating model explanations. Additionally, our article compares and thoroughly examines the effectiveness of various conventional classification and deep learning techniques in identifying Android malware applications.

3 Proposed Method

In this section, we demonstrate our Android malware detection framework based on machine learning and deep learning models. We have extracted three prime features specifically-permissions, activities and receivers by disassembling.apk files using Androguard [4] tool. Experiments are conducted using boolean feature vector tables created individually for different feature category and prepared models using machine learning as well as deep learning. Figure 1 depicts the complete flow of our proposed system and we describe the different components in the following subsections.

3.1 Data Set Preparation

The research is based on 5000 malware.apk files gathered from Drebin dataset [7] and 7000 benign samples obtained from a variety of sources. Each benign.apk file examined using professional antivirus software to ensure that they were all trustworthy.

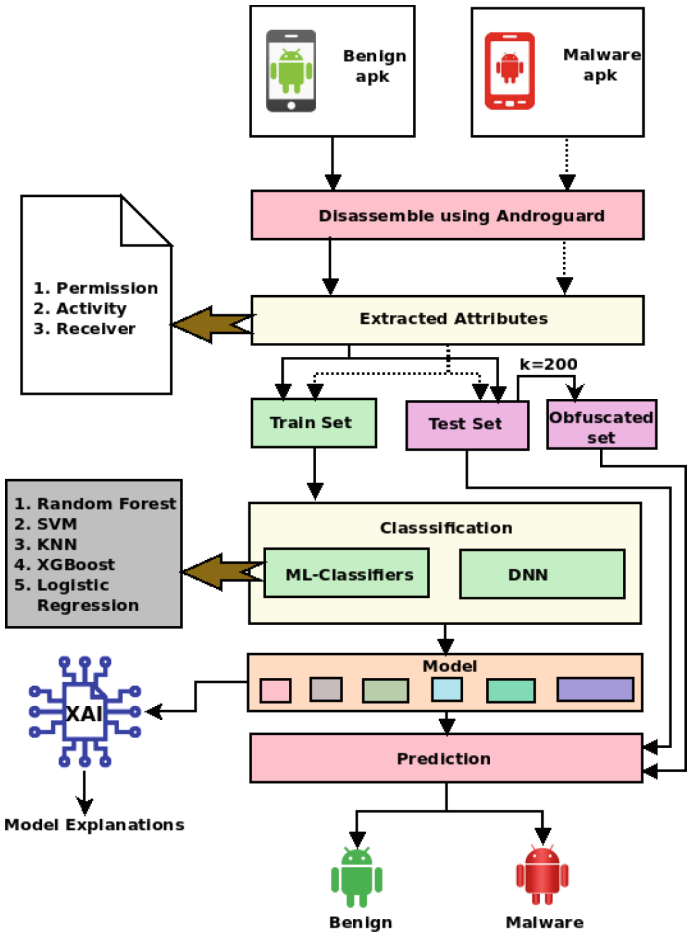

Fig. 1. Proposed method architecture

3.2 Feature Extraction

In the initial stage, the features are extracted from benign as well as malicious Android applications without execution. The disassembler tool called andro-guard gets the.apk files as input and an adroguard function called Analyze apk() which returns an object that has the capability to extract necessary features. The generation of model on each individual feature needs three different feature sets (Fig. 2):

1. **Permissions**: When a program wants to access sensitive user data or system functionalities, the authorisation is obtained through permissions. These are described statically in the AndroidManifest.XML file.
2. **Activities**: Activities are indeed the starting point for user interaction and launched to identify what should happen next when a user or another app launches an application. The activity names in the application are the features.

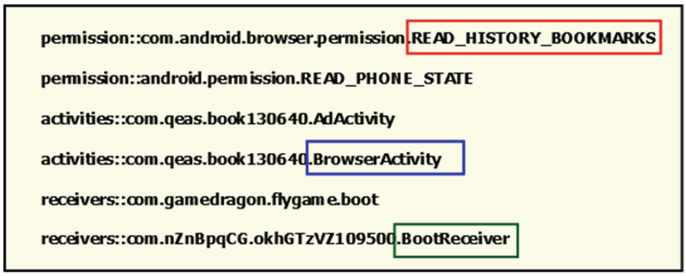

Fig. 2. Extraction of features

3. **Recievers**: These are areas where messages are received from a variety of sources and enable to register for system and application events. Application receives a message when the phone is in airplane mode is an example.

3.3 Pre-processing the Feature Set and FVT Creation

In this step, first we eliminate the irrelevant features and subsequently consider union of features from both the classes (*i.e. malware and benign files*). After extracting the features, we convert the feature set into a vector of 0's and 1's where 1 indicates that the feature is present in the app and 0 indicates that it is not.

3.4 Model Preparation and Classification

In this step, we aim to classify the unknown applications and it involves both training as well as testing. The model generation process using the feature vector tables generated in the previous step can be divided into: (a) Training ML classifiers (b) Training using deep neural networks. Former uses five different machine learning classifiers namely-Random forest, Support Vector Machine, K-nearest neighbour, XGBoost and Logistic Regression. Latter uses our own classifier for the classification The learned models will be used to predict the unseen samples.

3.5 Obfuscation of Samples

In our approach, we further tried to inspect the impact of obfuscation by generating obfuscated samples using the advanced obfuscator for Android app called *Obfuscapk* [14]. It will decompile the original input apk file and give an obfuscated app having same functionality as output. For obfuscating the samples, we have used four types of process called (1) Call indirection: It alters the control-flow graph (CFG) without compromising the semantics of the function. (2) Re-build: Rearrange the bytecode without altering its meaning but maintaining the app's original behavior (3) New alignment: The output is a restructured application with an improved file structure for Android device compatibility (4) New signature: Re-sign phase is the final step after applying obfuscation since Android mandates that all APKs be digitally signed with certificates or updated.

3.6 Interpretation of Results Using SHAP

Explainable AI (XAI) is a developing area of study in machine learning with the goal of enabling people to comprehend, believe in, and efficiently manage the next-AI solutions [5, 6]. The majority of XAI techniques created in recent times aim to describe supervised machine learning models. The SHAP [6] technique is a unified strategy that attempts to describe the output data using shapely values to determine the respective contributions of various coalition members. In order to develop effective solutions, the explanation for the result is particularly crucial in malware detection. In this investigation, we employ the SHAP technique that is especially well-suited for explaining machine learning models.

4 Experiments and Results

The investigations are performed on a computer with Ubuntu 21.10 as OS, Intel core i9 CPU and 32 GB RAM. The extensive experimentation involves two parts (1) Prepared models using the real samples considering individual features and (2) Models are tested against obfuscated samples.

4.1 Evaluation Measures

The confusion matrix which summarises the classifier's prediction outcomes was used for empirical evaluation of its performance and effectiveness. In order to forecast unknown samples, many assessment measures such as accuracy, precision, recall, and F-measure are calculated.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F - measeure = 2 * \frac{P * R}{P + R} \tag{4}$$

Here, the malicious app that has been identified as malicious is referred to as TP, and the malicious app that has not been detected as malicious is referred to as TN. The Benign app that has been designated as benign iteself is FN and FP denotes the number of benign app's that have been misclassified as malicous.

4.2 Research Questions

The following five major research questions are addressed in this paper:

RQ1: How effective our android malware detection framework in detecting malicious apps?

RQ2 Whether the model prepared is resilient against obfuscation?
RQ3: Can proposed method interpret the classification results?
RQ4: Which type of permissions, activities, recievers are used more?

***RQ1*: Our proposed Android malware detection framework is effective enough in detection of malicious apps?** First, we carry out tests to see whether our proposed method is capable of categorising basic Android malware. Both deep learning and conventional machine learning classifiers have been used in our experiments. We first use the dataset to train the classifiers, test and finally,assess the results. The classifications in both trials are based on features from permissions, activities, and receivers categories. A feature's presence or absence is recorded (i.e. Boolean features) to create the classification models.

Conventional Approach: From Table 1 we can observe that, RF classifier is getting highest F-measure of 0.97 for permissions. Receivers category of features also have highest F-measure of 0.90 with RF classifier. From Table 4, it is evident that the model is classifying samples with an F-measure of 0.95 when RF is used. The above observations shows that the model designed using permissions features is performing well with highest F-measure when Random Forest classifier is used. An app's functionality is dependent on the rights (permissions) it requests, and all malicious applications require some permissions that are different from those required by benign.apk files.

Table 1. Evaluation measures for permissions

Classifiers	Accuracy	Recall	F-measure	Precision
RF	97.04	97	**97**	97
SVM	88.42	78.53	90.5	94.12
XGBoost	87.86	97.59	86.0	84.03
KNN	82.55	90.04	80.5	77.26
LR	88.47	98.42	89.2	84.30

Deep Learning: From the Table 3, it is clear that using activities and Recievers we were able to classify the samples with 0.85%, 0.88% F-measure respectively with a 100% Detection Rate. Moreover, our deep learning technique has highest F-measure of 0.973% and successfully categorized Android malware with 100% detection rate using permissions as features. The results demonstrate that deep learning techniques have improved model performance and outperform those of conventional machine learning classifiers.

Summary of RQ1: *Our proposed deep learning approach classifies more accurately than conventional approach. However, both studies produced outstanding results, making the models based on permissions look promising for detecting Android malware.*

Table 2. Evaluation measures for recievers

Classifiers	Accuracy	Recall	F-measure	Precision
RF	90	90	**90**	92
SVM	85.95	87.53	87.82	88.13
XGBoost	82.36	84.40	84.72	85.05
KNN	83.94	84.38	85.89	87.47
LR	85.60	86.93	87.49	88.06

Table 3. Performance measures for deep learning model

Classifiers	Activity		Permissions		Receivers	
	F-Measure	DR	F-Measure	DR	F-Measure	DR
DNN	85.86	100	97.35	100	88.21	100

Table 4. Evaluation measures for activities

Classifiers	Accuracy	Recall	F-measure	Precision
RF	95.34	95	**95**	95
SVM	88.42	97.59	89	84.03
XGBoost	82.55	99.04	80.50	77.26
KNN	87.86	97.59	89.00	84.03
LR	88.47	98.42	89.2	84.30

RQ2: Whether the model prepared is resilient against obfuscation? Next, we evaluate the effectiveness of our proposed strategy in classifying Android malware that has been obfuscated using Obfuscapk, which obfuscates Android apps automatically. Specifically, we use 80% samples from the dataset to train the model. Then, select 200 samples correctly predicted as malware from the test set and use them for obfuscation. Table 5 demonstrates the evaluation results after obfuscation. RF classifier can correctly classify the obfuscated samples with a detection rate of 100% using permissions features. Using Activities, Receivers can also classify obfuscated samples with 88.5%, 86.6% detection rates, respectively. On comparing Table 1, we noticed that the proposed model outperforms with obfuscated samples when permissions are used as features. When Activities are used, all the obfuscated samples can correctly classify with an average true positive rate of 88.5% using RF. Therefore the effectiveness is marginally reduced.

Summary of RQ2: *When permissions are used as features, even if the samples are obfuscated, the samples are correctly classified with high detection accuracy. Additionally, obfuscation has seriously affected the detection rate of obfuscated samples when other types of features are used. Utilizing deep learning on permissions may increase the detection rate and robustness of the model.*

Table 5. Detetction rate of obfuscated samples

Classifiers	Detection rate		
	Permissions	Activities	Recievers
Random Forest	**100**	88.5	86.6
SVM	96.01	80.51	79.20
KNN	96.75	80.51	79.87
XGboost	96.00	75.97	68.83
LR	96.00	79.87	77.92

RQ3: Can proposed method interpret the classification results? Since Random Forest is performing well in terms of true positive rate in all the feature categories, we decided to generate visual explanations to interpret our classification results. Security analysts can use these interpretations to better comprehend the reasons behind a malware samples classification. Specifically, we use SHAP summary plots to display the importance and effects of the features. Each point on the summary plot represents a feature's Shapley value for the prediction. Red indicates a feature's value is higher whereas blue denotes features with a lesser value. Based on the distribution of the red and blue dots, we may generalize the directionality influence of the features. a request to the external storage-Fig. 3 reveals that the permission called read_phone_state have a greater and a positive impact in prediction of a malware. The features such as read_sms and send_sms are positively correlated with prediction but the permission called access_network_state is negatively correlated with prediction.

The Fig. 4 shows that the messageReceiver is positively correlated with prediction but HireBaseInstanceIdReceiver has a negative impact on prediction. From the Fig. 5, it is clear that the Adactivity and MainActivity is negatively contributing to the prediction. Since deep learning on permissions outperforms all other methods, we construct the SHAP summary plot (Refer Fig. 6). The Send_Sms permission have a high and positive impact on predicting malware and low impact when it is having a low value. Similarly, Read_Phone_State also have a negative correlation with the class malware.

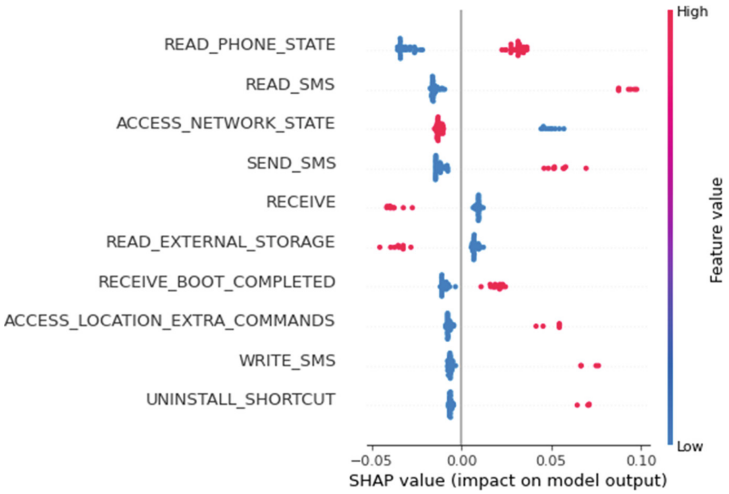

Fig. 3. Summary plot of Random Forest classifier using SHAP

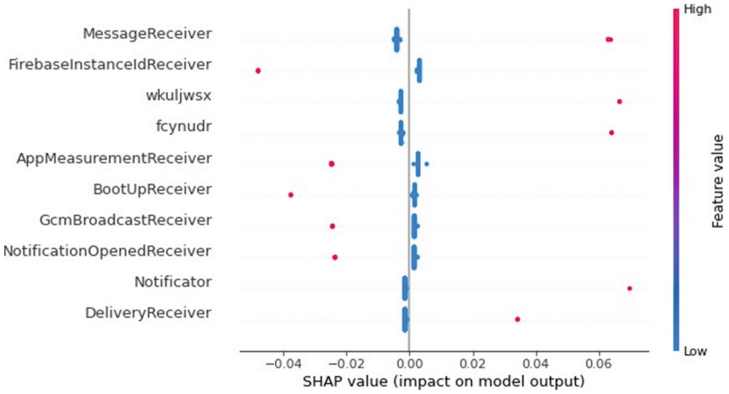

Fig. 4. SHAP summary plot of Random Forest classifier for Receivers

Summary of RQ3: *The proposed method uses a graphical tool called SHAP to interpret the findings of the malware categorization. Since it provides information about the contribution of each feature in predicting malware, we can even distinguish the malware from benign files directly through the summary plots* (Table 6).

RQ4: Which type of permissions, activities, recievers are used more? The features with high absolute Shapley values are the most prominent ones and are arranged in the order of importance. The SHAP feature importance for the deep learning model is shown in the Fig. 7. Send_sms, Read_phone_state, Recieve, Read_External_Storage, Access_network_State, Read_Sms are the most prominent permissions. We can observe from the results that the malware always captures personal data and writes it into newly

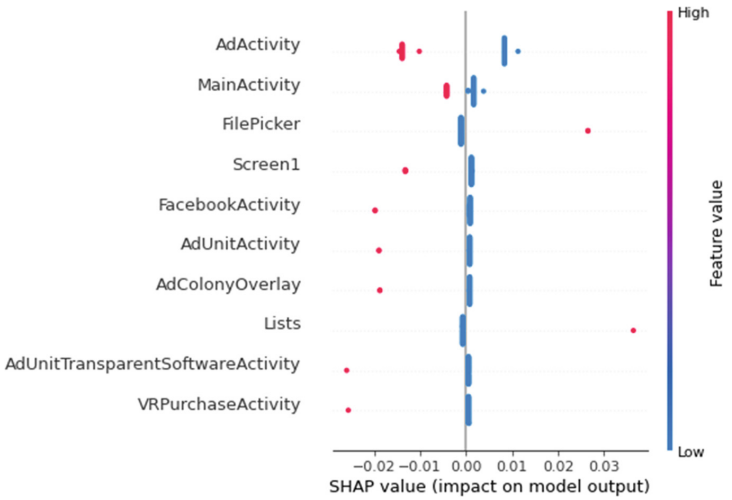

Fig. 5. SHAP summary plot of activities

Table 6. The prominent features and its descriptions

Feature name	Descriptions
SEND_SMS	Enables an app to send SMS
READ_PHONE_STATE	Accesses phone state read-only
RECEIVE	Obtain information from the Internet
READ_EXTERNAL_STORAGE	Authorization for a request to the external storage for writing
ACCESS_NETWORK_STATE	Permission to access network data
READ_SMS	App permission to read SMS

generated files. Then, these files are uploaded to the network or kept on other external storage systems and can be misused to discover the victim's location or to distinguish between the actual system and sandboxes. This can be considered as a suspicious behavior and qualifies it as malware.

Summary of RQ4: The findings show that the explanation technique (SHAP) is capable of evaluating the model as well as assisting us in learning about the most prevalent and harmful features for malware classification.

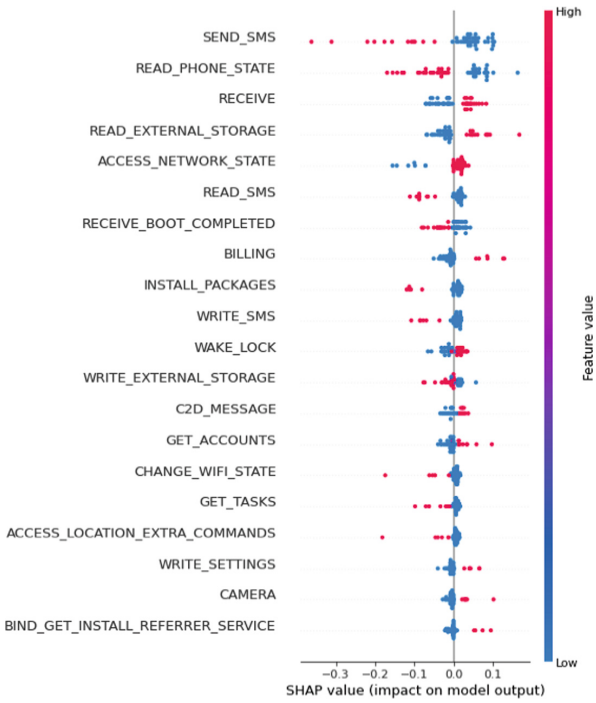

Fig. 6. SHAP summary plot for deep learning using permissions

4.3 Comparison with State-of-the-Art Approaches

Table 7 presents the comparison of the proposed scheme to other state-of-the-art techniques in terms of F1 sore reported in [15–17] which is tested on DREBIN dataset. Our detection scheme accurately classifies the samples with an F1 of 97.35% using explainable deep neural network. Also, the obfuscated samples are detected using Random Forest classifier with 100% detection rate.

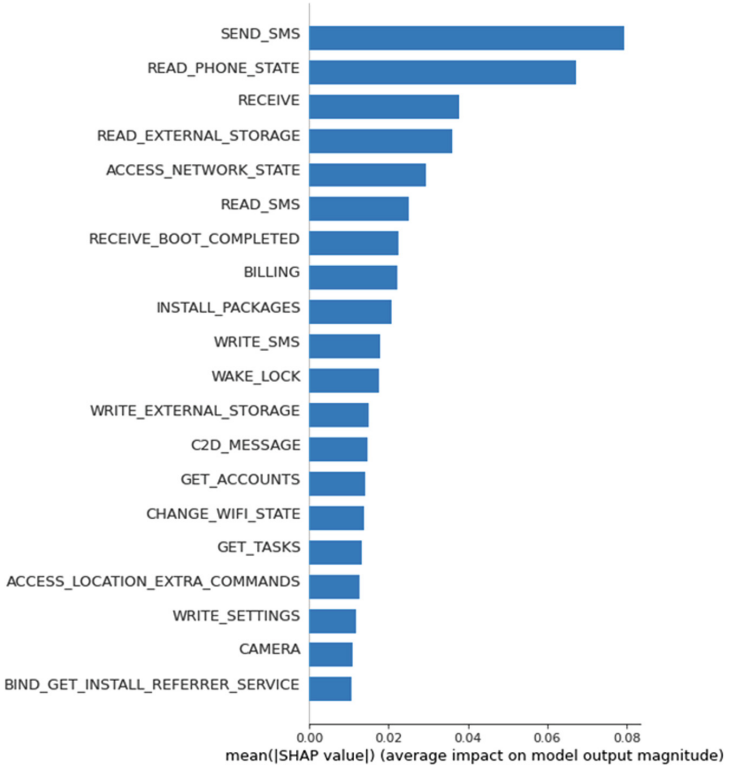

Fig. 7. The prominent features (permissions) selected through deep learning

Table 7. Comparative analysis

No.	Author	Remarks	Explainable model	Obfuscation
1	Masum et.al. [15]	Droid-NNet, a neural network-based framework with a L^2 regularization approach, early stopping criterion, and the mini-batch gradient descent method, is used to train the Droid-NNet. Obtained F1 score of 0.98	✗	✗
2	Sharma et.al. [16]	Hybrid technique based on Deep learning and Binary Particle Swarm Optimization (BPSO) Obtained an F1 score of 92.39% for DREBIN dataset	✗	✗

(*continued*)

Table 7. (*continued*)

No.	Author	Remarks	Explainable model	Obfuscation
	Shiqi et.al. [17]	The deep residual LSTM-based sequence model known as MalResLSTM is then used to identify and categorize Android malware. Using static features obtained an F1 score of 0.92%	✗	✗
	Proposed method	Obtained an F-measure of 0.973 using Deep learning Achieved 100% detection rate for obfuscated samples	✓	✓

5 Conclusion

This research presents an explainable deep learning model based on static features using SHAP to detect Android malware. The proposed system is helpful for the primary classification of malicious samples. Additionally, classic machine learning and deep learning techniques are used to select the most effective model for identifying Android malware. Evaluation results reveal that the deep learning model outperforms all conventional machine learning algorithms with an F-measure of 0.973 and a detection rate of 100%. Furthermore, we have examined the effect of obfuscation on the model's efficacy, and the proposed method can categorize all obfuscated malware samples with a 100% detection rate. Such a finding implies that our model is resistant to obfuscation.

Future of this work extends to combine static and dynamic features for ensuring greater accuracy. Additionally, it is necessary to test different obfuscation techniques in combination. Future assessments of the effects of adversarial attacks can also be estimated using explainable techniques.

References

1. Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D.: Android malware detection using deep learning on API method sequences. Comput. Sci. arXiv:1712.08996v1 (2017)
2. Kim, T., Kang, B., Rho, M., Sezer, S., Im, E.G.: A multimodal deep learning method for android malware detection using various features. IEEE Trans. Inform. Forensics Secur. **14**(3), 733–788 (2018)
3. Hou, S., Saas, A., Chen, L., Ye, Y.: Deep4MalDroid: a deep learning framework for android malware detection based on linux kernel system call graphs. In: IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), pp. 104–111 (2016)
4. Androguard: http://code.google.com/p/androguard/ (2019). v3.3.5
5. Gunning, D.: In: Defense Advanced Research Projects Agency (DARPA), nd Web 2. Explainable artificial intelligence (xai) (2017)

6. Scott, M.L., Su-In, L.: A unified approach to interpreting model predictions. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17), pp. 4768–4777. Curran Associates Inc., Red Hook, NY, USA (2017)
7. Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K.D.: Effective and explainable detection of android malware in your pocket. In: Proceedings of the 21st annual network distributed system security symposium (NDSS). The Internet Society (2014)
8. Arrieta, A.B., et al.: Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Inform. Fusion **58**, 82–115 (2020)
9. Pektaş, A., Acarman, T.: Deep learning for effective android malware detection using API call graph embeddings. Soft Comput. **24**(2), 1027–1043 (2019). https://doi.org/10.1007/s00500-019-03940-5
10. Yerima, S.Y., Sezer, S., Muttik, I.: Android malware detection using parallel machine learning classifiers. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 37–42. IEEE (2014)
11. Wen, L., Yu, H.: An android malware detection system based on machine learning. In: AIP Conference Proceedings, vol. 1864. AIP Publishing LLC (2017)
12. Alhebsi, M.S.: Android Malware Detection using Machine Learning Techniques. Thesis. Rochester Institute of Technology (2022)
13. Koli. J.D.: RanDroid: android malware detection using random machine learning classifiers. In: International Conference on Technologies for Smart City Energy Security and Power (ICSESP) IEEE (2018)
14. Aonzo, S., Georgiu, G.C., Verderame, L., Merlo, A.: Obfuscapk: an open-source black-box obfuscation tool for android apps. SoftwareX **11**, 100403 (2020)
15. Masum, M., Shahriar, H.: Droid-NNet: Deep learning neural network for android malware detection. In: 2019 IEEE International Conference on Big Data (Big Data). IEEE (2019)
16. Sharma, R.M., Agrawal, C.P.: A BPSO and deep learning based hybrid approach for android feature selection and malware detection. In: 2022 IEEE 11th International Conference on Communication Systems and Network Technologies (CSNT). IEEE (2022)
17. Shiqi, L., et al.: Android malicious code classification using deep belief network. KSII Trans. Internet Inform. Syst. **12**(1), 454–475 (2018)
18. https://www.mcafee.com/content/dam/global/infographics/McAfeeMobileThreatReport2021.pdf