mathematics

Article

A Novel Neural Network Architecture Using Automated Correlated Feature Layer to Detect Android Malware Applications

Amerah Alabrah

Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11451, Saudi Arabia; aalobrah@ksu.edu.sa

Abstract: Android OS devices are the most widely used mobile devices globally. The open-source nature and less restricted nature of the Android application store welcome malicious apps, which present risks for such devices. It is found in the security department report that static features such as Android permissions, manifest files, and API calls could significantly reduce malware app attacks on Android devices. Therefore, an automated method for malware detection should be installed on Android devices to detect malicious apps. These automated malware detection methods are developed using machine learning methods. Previously, many studies on Android OS malware detection using different feature selection approaches have been proposed, indicating that feature selection is a widely used concept in Android malware detection. The feature dependency and the correlation of the features enable the malicious behavior of an app to be detected. However, more robust feature selection using automated methods is still needed to improve Android malware detection methods. Therefore, this study proposed an automated ANN-method-based Android malware detection method. To validate the proposed method, two public datasets were used in this study, namely the CICInvestAndMal2019 and Drebin/AMD datasets. Both datasets were preprocessed via their static features to normalize the features as binary values. Binary values indicate that certain permissions in any app are enabled (1) or disabled (0). The transformed feature sets were given to the ANN classifier, and two main experiments were conducted. In Experiment 1, the ANN classifier used a simple input layer, whereas a five-fold cross-validation method was applied for validation. In Experiment 2, the proposed ANN classifier used a proposed feature selection layer. It includes selected features only based on correlation or dependency with respect to benign or malware apps. The proposed ANN-method-based results are significant, improved, and robust and were better than those presented in previous studies. The overall results of using the five-fold method on the CICInvestAndMal2019 dataset were a 95.30% accuracy, 96% precision, 98% precision, and 92% F1-score. Likewise, on the AMD/Drebin dataset, the overall scores were a 99.60% accuracy, 100% precision and recall, and 99% F1-score. Furthermore, the computational cost of both experiments was calculated to prove the performance improvement brought about by the proposed ANN classifier compared to the simple ANN method with the same time of training and prediction.

Keywords: Android malware detection; deep neural network; feature selection; malicious apps

MSC: 68T07

check for updates

Citation: Alabrah, A. A Novel Neural Network Architecture Using Automated Correlated Feature Layer to Detect Android Malware Applications. *Mathematics* **2023**, *11*, 4242. https://doi.org/10.3390/math11204242

Academic Editors: Oliviu Matei, Rudolf Erdei, Ioannis G. Tsoulos and Jie Wen

Received: 3 September 2023
Revised: 27 September 2023
Accepted: 7 October 2023
Published: 11 October 2023

1. Introduction

In a decade, tremendous technological improvements have transformed many manual interpretation tasks and procedures in different industries, enterprises, and government departments [1]. These have shifted the global population to using mobile phone technologies, and the U.N. estimated that, by the end of 2022, 82% of the total population will be using smartphones [2]. The use of mobile technology has not only improved the workloads in different fields by providing digital solutions, but it also increased the vulnerabilities of users' data. Android applications are uploaded to the Google Play Store, where very few or no restrictions are applied to app developers, making it easier for malicious apps to be uploaded and creating risks for user privacy and data [3]. Furthermore, malware attacks on devices cause economic losses [4]. Mobile technologies are welcomed despite their many risks. Among the different mobile technologies, the Android OS is the most prevalent and vulnerable [5]. In the mobile industry, the Android Operating System (OS) covered more than 70.9% of the global share until July 2023, as illustrated in Figure 1. The second competitor in terms of mobile OS is iOS, with almost 28.36% of the global share.

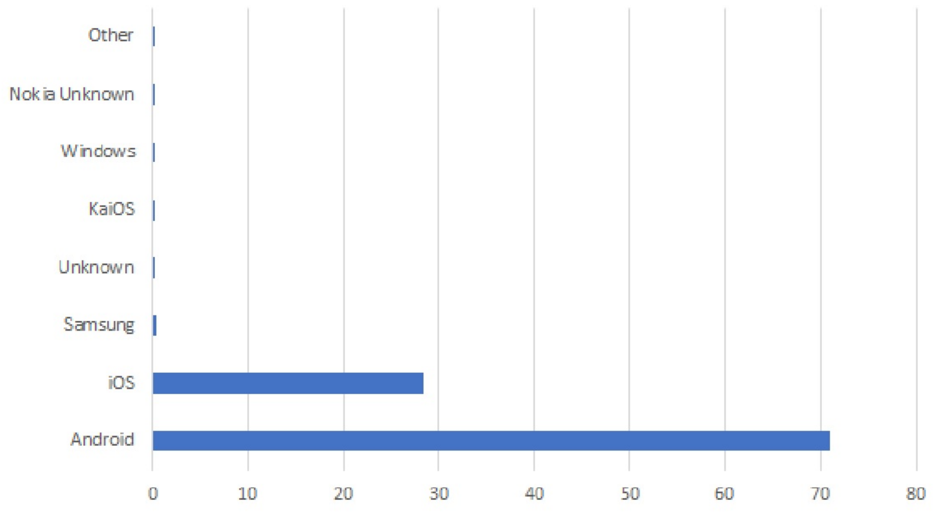

Figure 1. Statistics for the share of mobile operating systems from July 2022 to July 2023 [6].

Due to the abundant use of Android OS mobiles, they have become the prime target for malware experts to attack. One other cause of Android OS's vulnerability is its open-source nature [7]. It is estimated in the Kaspersky report of 2022 that 196,476 banking Trojans and 10,543 ransomware Trojans were found [8]. Trojan mobile applications invade the OS as legitimate programs but are fake. Most users now use mobile Internet banking, and through banking Trojans, the banking details of user accounts are shared via fake banking apps. Similarly, users' health information and sensitive personal information are also shared. To secure such information, many other techniques are currently in use, such as blockchain and edge computing methods [9]. Malware experts have hacked mobile devices and subsequently made them bots. These bots send spam emails and texts including malicious links, and also use them to execute distributed denial-of-service (DDoS) attacks. This illegitimate malware is developed using intelligent methods that make these attacks inevitable [10]. These botnets are a serious concern for Android OS security. However, according to the Security Institute [11], malware attacks are increased via Android packages. Hence, to enhance Android mobile security, signature-based approaches to the detection of malicious installation packages and malware using attribute information can be effectively adopted. Many malware experts are probably continuously developing ingenious methods to hide their attacks and to obtain user information. Therefore, it is becoming challenging to neutralize such new and complex methods [12]. Many techniques to detect the malware behavior of Android apps have been developed previously [13]. These schemes involve feature interaction to indicate feature importance and to identify malware attacks [14]. Likewise, the owl binary optimization algorithm has been used to select features from the Drebin dataset to detect malware attacks [15]. Android malware detection usage could significantly increase security against malicious attacks [16]. Furthermore, in recent years, many deep learning methods have been proposed to detect malicious apps. For example, S.C. Tan et al. proposed an optimal deep learning ensemble classifier using back-propagation (BP) and particle swarm optimization (PSO) methods. The optimization of the deep learning model was improved via a reduction in the computational cost. However, we can conclude that, to detect Android OS malware attacks, many methods have been proposed which ultimately involve selecting the most optimal features to increase the precision rates of malware detectors and, in parallel, try to reduce the computational cost. Therefore, to increase the precision of Android OS malware detection with optimal feature selection and reduce the computational cost, the conducted study proposed a framework based on deep learning. The use of deep learning has revolutionized many fields. The proposed study contributes in the following ways:

- Hidden aspects of Android permission-based features were explored via user permission indications.
- Instead of performing the manual calculation of features before applying a deep learning classifier, feature selection based upon the correlation score is embedded in the deep learning network.
- The precision of malware detection was improved while maintaining the same cost in terms of computational time using the proposed deep learning model for the Drebin and CICandMal2017 datasets.

The remainder of this manuscript consists of four sections. In Section 2, related work and a discussion are presented, highlighting the research gaps covered by the proposed study. In Section 3, the proposed methods are discussed. In Section 4, the results and discussion are presented, and comparisons with state-of-the-art studies are made. Section 5 presents the strengths and weaknesses of the proposed study.

2. Related Work

Numerous Android-OS-based intelligent malware detection methods have previously been proposed, and a few of the recent ones are discussed here. An Android malware detection system is proposed by training two parallel ensemble methods based upon three methods [17]. Two datasets have been used to test the applied approach, namely the CICandMal2017 and Drebin datasets. Static and dynamic analysis-based features were extracted, whereas in the ensemble approach, three classifiers, namely one-class-SVM, modified isolation forest, and local outlier factor methods, were used. A voting combination of benign and malware apps was used while using three methods as an ensemble approach. Ten different splits of training, validation, and testing were applied on both datasets and average scores of accuracy, precision, and recall were reported. The proposed fused model's reported scores were a 98.65% accuracy, 96.82% F1-score, and 0.9751 AUC on the CICandMal2017 dataset, whereas on the Drebin dataset, the scores were 98.70% accuracy, 95.67% F1-score, and 0.9325 AUC. The comparison with previous methods has shown that the proposed method achieved higher results. The dynamic and static features from the CIC-MalDroid2020 dataset are obtained by applying feature selection and a random forest classifier [18]. The applied ReliefF method results in higher accuracy compared to the original features, and 80 features were selected from the 470 original features. The selected features show a 97.4647% precision for malware detection. An image representation of Android malware using the Dex file is processed and used to detect malware [19]. Images were developed using bytecode extracted from Android Dex files, and deep features were extracted from them. Extensive experimentation is performed via 26 state-of-the-art pre-trained CNN models, of which EfficientNet-B4 provides the most useful features.The global average pooling is performed on the extracted deep features, while the softmax function is applied for classification. The binary classification results achieved a 95.7% score. A summary of the recent related work based upon Android malware detection, the datasets used, and the achieved results is shown in Table 1.

Table 1. Summary of recent studies on Android malware detection with their applied methods, datasets used, and achieved results.

Study	Year	Methods	Dataset	Results
[20]	2021	Hybrid deep learning method using CNN and Bi-LSTM	Combined dataset using Androzoo and AMD datasets	Multi-Class ACC = 99.05%, PRE = 99.39%, REC = 99.41%
[18]	2022	Feature selection using ReliefF method and random forest classifier	CIC-MalDroid2020	PRE = 97.46%, FAR = 0.1409
[19]	2022	Deep features extraction from Android Dex files via 26 pre-trained CNNs	Image development using Bytecode extracted via Android Dex files	Binary ACC = 95.7%
[17]	2023	Voting classifier based upon two parallel streams of benign and malware apps	CICandMal2017	ACC = 98.65%, F1 = 96.82%, AUC = 0.9751
			Drebin	ACC = 98.70%, F1 = 95.67%, AUC = 0.9325
[2]	2023	Fuzzy inference system-based ensemble ML method	Drebin	ACC = 99.33%, PRE = 98.68%, REC = 1.00, SPEC = 98.67%, F1 = 99.34%
[21]	2023	API calls, intent, and permission features extraction and ML or DL method-based classification	Combined dataset of CICMalDroid2020, CIC-InvestAndMal2019, and CICAndMal2017	Multiclass ANN (PreLU) ACC = 99.01%, REC = 99.4%, F1, PRE = 99.3%, AUC-ROC = 0.986.
[22]	2023	Efficient feature reduction and neural network-based classification	CICInvestAndMal2019	ACC = 96.40%

The timely detection of malware attacks using multi-vector representation is of extreme importance in Android devices [20]. For this purpose, a hybrid approach using the deep learning method and applying the CNN Bi-LSTM model is proposed. Additionally, the CNN-LSTM and CNN-GRU methods are also applied. Two datasets were used in this study, namely the Androzoo and AMD datasets. Data extracted from these datasets are divided into three classes, namely Trojan, Benign, and Backdoor. A new dataset was composed using these two datasets and contained 905 features in total. The validation scheme of 10-fold cross-validation is applied, and each fold's results are reported using accuracy, precision, recall, and F1-score. The highest 10-fold average results achieved using the CNN-Bi-LSTM method are 99.05% accuracy, 99.39% precision, and 99.41% recall, while the training time was reported to be 90 s with 1.5 s of testing time. An ensemble approach was applied using a fuzzy inference system, during which six ML methods were used in voting for the ensemble method, of which three methods were considered to be most appropriate [2]. The fuzzy inference system-based ensemble method has been proven to be better than the simple ML-based malware detection method. Regarding the dataset, the Debrin APK files dataset was downloaded and processed as a balanced dataset. Regarding the achieved results, using the FIS-based ensemble approach, the scores reached a 99.33%

accuracy, a recall of 1.00, a specificity of 98.67%, a precision of 98.68%, and an F1-measure of 99.34%.

Static analysis on the APK files of three datasets, namely CICMalDroid2020, CIC-InvestAndMal2019, and CICAndMal2017, was conducted [21]. From these files, API calls, intent, and permission-based features were extracted and 215 features were retained. According to the proposed method, five classes were used to classify. These five categories include APK files of Adware, Banking, SMS malware, Riskware, and Benign. The data splitting was performed as follows: 75% of the data were used for training and 25% for testing. Multiple classifiers were trained including decision tree, random forest, SVM, and neural network (NN) models based upon different activation functions, such as Sigmoid, PReLU, and TanH. However, the highest score was achieved by the NN models compared to the ML models. The highest results achieved by the PreLU method were 99.01% accuracy, 99.4% recall, 99.3% F1-score and precision, and 0.986 AUC-ROC. Another study used static analysis to detect Android malware and considered it as a first-line defense, as malware could be detected upon installation [22]. The CICInvesandMal2019 dataset was used to analyze malware detection, while efficient feature reduction was applied to reduce the feature set. The applied NN method shows that, with feature reduction, the results are improved. The reduction in features increased the accuracy from 95.2% to 96.40%.

From all the above-discussed studies, it is concluded that many of the methods have been applied to malware detection to classify the binary and multiclass categories. However, most of them have used deep learning methods and techniques, whereas, most of the time, the feature selection remains focused, and very few of the methods are considered timely and efficient methods. Furthermore, the ultimate purpose of all of the applied methods is to develop a robust, timely, efficient, and more accurate approach for malware detection. Therefore, the proposed method puts forward a timely and performance-efficient solution for malware detection. Manual feature selection methods have been used in previous studies, whereas this proposed study uses a feature selection layer in an ANN for feature selection using the correlation found in static features of Android APK files. The final results show that the feature correlation-based ANN method not only improves the results but also improves the time efficiency and reduces the number of features based on correlation while also reducing the space consumption.

3. Materials and Methods

The applied methodology in this study used two datasets of malware and benign apps to prove the effectiveness of the proposed method. The datasets were collected using the APK files of malware and benign apps. The static features of Android applications were used to make feature sets for both malware and benign apps. These static features include Android permission-based binary flags which are collected from the CICInvesAndMal2019 and AMD datasets. After obtaining feature sets for both datasets, two types of experiments regarding malware detection were conducted. In Experiment 1, the ANN classifier was applied using 5 cross-validations, while in Experiment 2, the custom feature selection layer-based ANN was applied for malware detection, which improves the ANN model performance and makes it timely and performance-efficient. The applied methods and techniques are shown in Figure 2.

In two experiments of malware detection and classification, the same ANN classifier was used in terms of layers, but in Experiment 2, a custom feature selection layer was introduced. Pearson correlation is used in feature target-based correlation calculation, and, subsequently, indices are selected. The selected feature is fed to the ANN classifier via the forward pass.

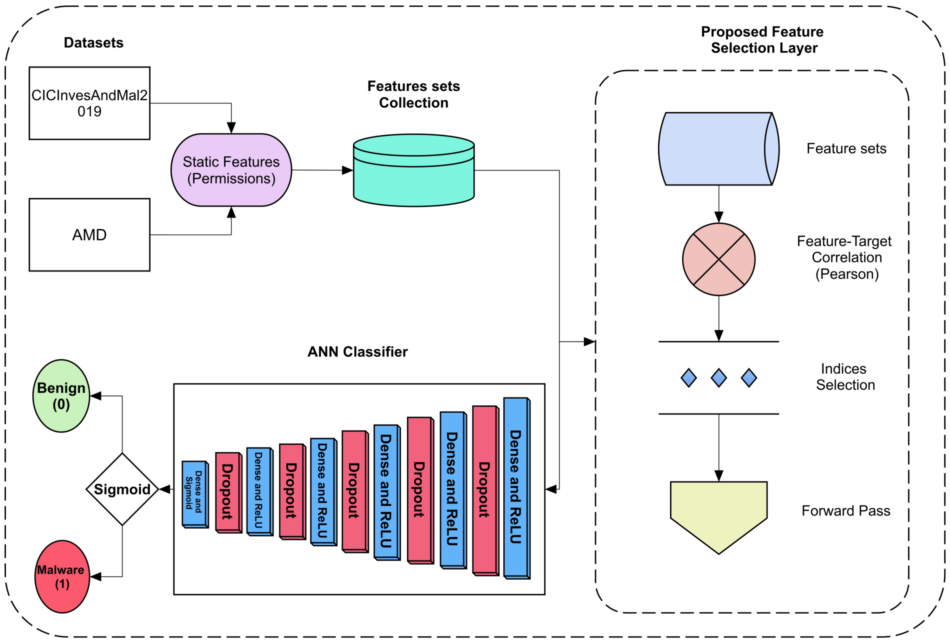

Figure 2. Flowchart of Android malware detection using the proposed feature-selected ANN method.

3.1. Data Preprocessing and Feature Set Extraction

Two publicly available datasets, namely CICInvesandMal2019 and AMD, were preprocessed and then used. The CICInvesandMal2019 dataset is found on the website in different formats, such as in CSV files containing descriptions, app names, etc. Similarly, APK files are also given in malware category-based directories. However, to obtain permission-based features, the APK files are iterated over Android. Permission-based properties and consequently a feature set of 0 and 1 as benign and malignant is built. Similarly, from the AMD dataset, the Drebin dataset-based malware apps and benign apps were collected. This dataset is available on the Kaggle website. This dataset contains permissions, API calls, Manifest, and other features. However, both datasets contain different kinds of statistical features. For the preprocessed apps based on their internal permissions, API calls, and manifest file data, the 0 and 1 indications are assigned. The 0 value indicates that permission is not granted and the 1 value indicates that permission is granted for certain API calls, permissions, and manifest files.

3.2. Proposed Custom Feature Selection Layer

The CICInvesAndMal2019 dataset contains 428 features and the AMD dataset contains 215 features. Substituting the most relevant ones could lead to more precise malware detection by improving time and computation consumption. Therefore, to include the most relevant features, a custom feature selection layer is defined, which includes feature selection based on Pearson correlation, which estimates the correlation between two variables. A threshold value is defined and represented as a k constant and different threshold values are tested, after which 0.01 was found to be the most optimal as it obtained the highest classification results. The feature set's correlation values corresponding to its target value are calculated and based upon the selected features, while the selected index-based forward pass is fed to the ANN classifier. The mathematical representation of the defined custom feature selection layer is shown below in Equation (1):

$$R_x = Corr(M_{i,j}, L) \tag{1}$$

The correlation value is estimated by using a feature value matrix ($M_{i,j}$), while i and j represent two rows and columns of a given feature matrix, and L is the label vector corresponding to the feature matrix of a given dataset. However, the correlation between the feature matrix and the labels is estimated using the Corr() function. The mathematical representation of the correlation-finding function is shown in Equation (2).

$$\mathrm{Corr}_i = \frac{\sum_{j=1}^{n}\left((M_{i,j} - \overline{M}_i)\cdot(L_j - \overline{L})\right)}{\sqrt{\sum_{j=1}^{n}\left((M_{i,j} - \overline{M}_{i,j})^2\right)}\cdot\sqrt{\sum_{j=1}^{n}\left((L_j - \overline{L})^2\right)}} \tag{2}$$

In Equation (2), the correlation value of each feature is calculated using a feature matrix. $M_{i,j}$ and its corresponding label vector (L). For instance, (j), $M_{i,j}$ represents the feature value of the ith index in a given feature set matrix.

In the nominator equation, a dot product is calculated using two terms, in which the first term is the subtracted values of the feature vector indices from the mean of the feature column ($\overline{M}_i$) and the second term is the same calculation applied for the labels ($\overline{L}$). To obtain an absolute positive correlation value, the dot product of the same terms is calculated in the denominator using their square roots. However, the indices are selected based on the defined threshold k. The mathematical representation of the selected indices is shown in Equation (3).

$$(\overline{M}_{i,j}) \Leftarrow |R_x| \geq k \tag{3}$$

In Equation (3), the selected indices are substituted into a new matrix ($\overline{M}_{i,j}$), and the subset of all feature sets for both datasets is then fed to the ANN classifier as a forward pass. The ANN classifier is the simpler model with dense and dropout layers and with ReLU and sigmoid activation functions.

3.3. *Malware Detection Using ANN Classifier*

In malware detection, the ANN classifier is trained and tested using five cross-validation methods for both feature sets of the CICInvesAndMal2019 and Drebin datasets. For malware detection, two experiments were conducted based on the same ANN architecture with input layer modifications only. In Experiment 1, both feature sets were fed to the ANN classifier, and the classification results were extracted using the 5 cross-validation methods. In Experiment 2, the proposed custom feature selection layer was first added as an input layer in which correlation-based features were selected and passed on to further layers of the ANN classifier. The proposed custom feature selection layer-based ANN classifier architecture is shown below in Table 2. In both datasets' training, the 100 epochs were initialized, and to compare the results, no stopping conditions were added, and similar hyperparameters of model training were used. Furthermore, the batch size was taken as 32 and verbose as 0.

Table 2. Proposed custom feature selection layer-based ANN architecture.

Number	Layer Name	Parameters/Value	Output
1	Feature Selection Layer	0	(None, X)
2	Dense 1	103,936	(None, 512)
3	Dropout 1	0.5	(None, 512)
4	Dense 2	131,328	(None, 256)
5	Dropout 2	0.5	(None, 256)
6	Dense 3	32,896	(None, 128)
7	Dropout 3	0.2	(None, 128)
8	Dense 4	8256	(None, 64)
9	Dropout 4	0.2	(None, 64)
10	Dense 5	2080	(None, 32)
11	Dropout 5	0.2	(None, 32)
12	Dense 6	33	(None, 1)

In Table 2, the ANN architecture used in Experiment 2 on both datasets is explained. The first layer, named the feature selection layer, does not initially receive any parameter or value; instead, it only obtains the input shape of feature sets such as 428 and 215 for the CICInvesAndMal2019 and AMD datasets, respectively. X, in the first layer output, shows how to shape the feature set, which is fed to the ANN classifier at that time. Based upon feature set dimensions or output shape, Dense 1 receives different parameters in layer 2, namely Dense 1 receives 103,939 parameters for the CICInvesAndMal2019 dataset and 107,520 for the AMD dataset. However, in the following layers, namely Dense 3, 4, 5, and 6, the number of parameters remains the same. The dropout value is changed during experimentation; in the first two dropout layers, namely dropout 1 and 2, the value given is 0.5, whereas in further dropout layers 3, 4, and 5, these values are changed to 0.2. In the Dense 1 to Dense 5 layers, ReLU activation is applied, whereas in the last and 12th layer Dense 6, the sigmoid activation function is applied due to which the output value is the only one used to predict malware or benign classes.

4. Results and Discussion

Two types of classification experiments were conducted on both datasets using the five-fold cross-validation method. However, in both experiments, the ANN classifier was used with the initial input layer modification. In Experiment 1, the ANN classifier with 12 layers was adopted for malware detection on both datasets, while the five-fold cross-validation method was adopted. In Experiment 2, the input layer of the ANN classifier was replaced with a custom feature selection layer based on Pearson correlation, as discussed in Section 3.2. The results of the Experiment-2-based proposed ANN classifier are improved in terms of accuracy, precision, recall, F1-score, and AUC values.

4.1. Datasets Description

Two datasets were used in this study to identify malware apps. The number of malware and benign apps varies in both datasets, and the frequency of both classes after processing the APK files is shown in Figure 3.

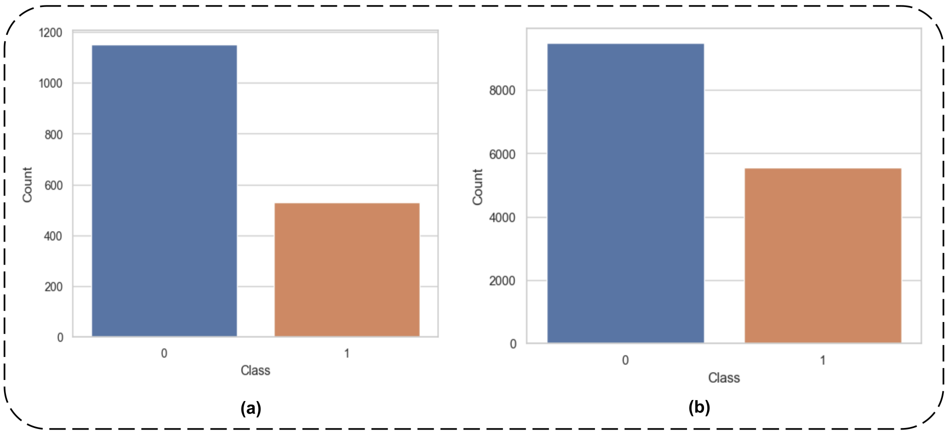

Figure 3. Frequencies of malware (1) and benign (0) app-based feature sets in (**a**) CICInvesAndMal2019 and (**b**) AMD datasets.

In (a), the CICInvesAndMal2019 APK file-based dataset contains 1150 benign and 529 malware app-based features sets, whereas 428 is its total number of features. The second dataset, AMD, contains the Drebin-dataset-based malware apps and benign apps, the frequency of which corresponds to 9476 and 5560, respectively. This dataset contains 215 features. Both datasets contain different numbers of instances and different numbers of features that are selected using the proposed custom feature correlation layer-based ANN classifier.

4.2. Experiment 1: Malware Detection Using ANN Classifier

In Experiment 1, the 12-layered ANN classifier is applied with five-fold cross-validation, and each fold's results are maintained. These K-fold-validation-method-based results put forward that dividing into different folds with shuffling or randomization is necessary to report confident results. Therefore, the applied approach was also tested on the same validation method. The results of a simple ANN classifier without adding an input feature selection layer are shown in Table 3.

Table 3. Malware detection classification results using ANN classifier via 5-fold validation method.

Dataset	Folds	ACC (%)	PRE (%)	REC (%)	F1 (%)	Training Time (s)	Testing Time (s)
CICInvesAndMal2019	Fold 1	89.58	92.00	93.00	83.00	18.40	0.32
	Fold 2	90.48	92.00	95.00	84.00	17.94	0.33
	Fold 3	91.37	92.00	96.00	86.00	18.07	0.32
	Fold 4	88.99	91.00	93.00	82.00	17.80	0.33
	Fold 5	92.24	91.00	98.00	86.00	17.01	0.32
	Average	90.53	91.60	95.00	84.20	19.34	0.17
AMD	Fold 1	99.00	99.00	99.00	99.00	162.29	0.69
	Fold 2	99.00	99.00	99.00	99.00	167.19	0.69
	Fold 3	99.27	99.00	100.00	99.00	166.92	0.77
	Fold 4	98.64	99.00	99.00	98.00	169.11	0.76
	Fold 5	98.70	99.00	99.00	98.00	163.19	0.71
	Average	98.92	99.00	99.20	98.60	174.99	0.75

It can be concluded from the results reported in the above table that they deviate throughout five folds, which shows that this method of validation is necessary to remove the bias from the given feature set. In fold 1 testing, we obtained 89.58% accuracy, 92% precision, 93% recall, and 83% F1-score. The accuracy reaches up to 90%, but the F1-score shows that, in negative class detection, the model does not perform well. The claiming negative class detection is derived via metrics used to obtain scores. The precision metric seemingly indicates the positive class predictions of the model, which is higher compared to the F1-score, whereas the F1-score computed with both positive and negative classes is lower. Therefore, a lower F1-score indicates that the score has been reduced while taking the positive and negative classes into consideration. Similarly, in the highest accuracy-achieving fold, namely fold 5 with 92.24% accuracy, 91% precision, 98% recall, and 86% F1-score, showed a lower F1-score. We can see that the F1-score does not improve while testing on single folds nor in the average scores of all folds. The overall average scores are all higher than 90% except for F1. Therefore, a few features must be excluded to increase the ANN classifier performance. To validate this argument, feature-selection-based Android malware detection could be performed. Therefore, Experiment 2 is conducted using the same architecture of the ANN classifier with the use of a custom feature selection layer. For Experiment 1, the overall average score shows a satisfactory performance of the ANN classifier compared to the individual fold results. An AUC and accuracy-curve-based visual illustration is shown in Figure 4. The visual illustration shows the deviation of fold-by-fold testing of the data used. The variation among each fold testing suggests that using this validation scheme is severely needed to make the model performance more confident. Likewise, for the AMD dataset, the average five-fold results show that the model achieved a 98.92% accuracy, 99% precision and recall, and 98% F1-score. It is noticed that the score needs to be improved for both datasets as it is lower than all other achieved scores reported by previous studies. The deviating behavior, although low in this dataset's results, does not become consistent throughout all the folds' testing. Upon closer examination of Table 3, the training and testing times of a simple ANN model are significantly different even when using the same model. The reason for this time difference is the greater number of instances or big data of the AMD dataset, as mentioned in Section 4.1. The AMD dataset took more time in each fold training and testing, whereas the average training time of all folds remained up to 174.99 s with less than a second for prediction, specifically, only 0.75 s. In the case of the CICInvesAndMal2019 dataset, the average time of training was 19.34 s and the average testing time was 0.17 s. The greater amount of time spent while training and testing a bigger dataset indicates that more data implies more training time; however, in Experiment 2, the features were selected based upon the proposed ANN classifier architecture and the same dataset was used to train and test. It is important to note that training and testing time is reduced regardless of feature reduction.

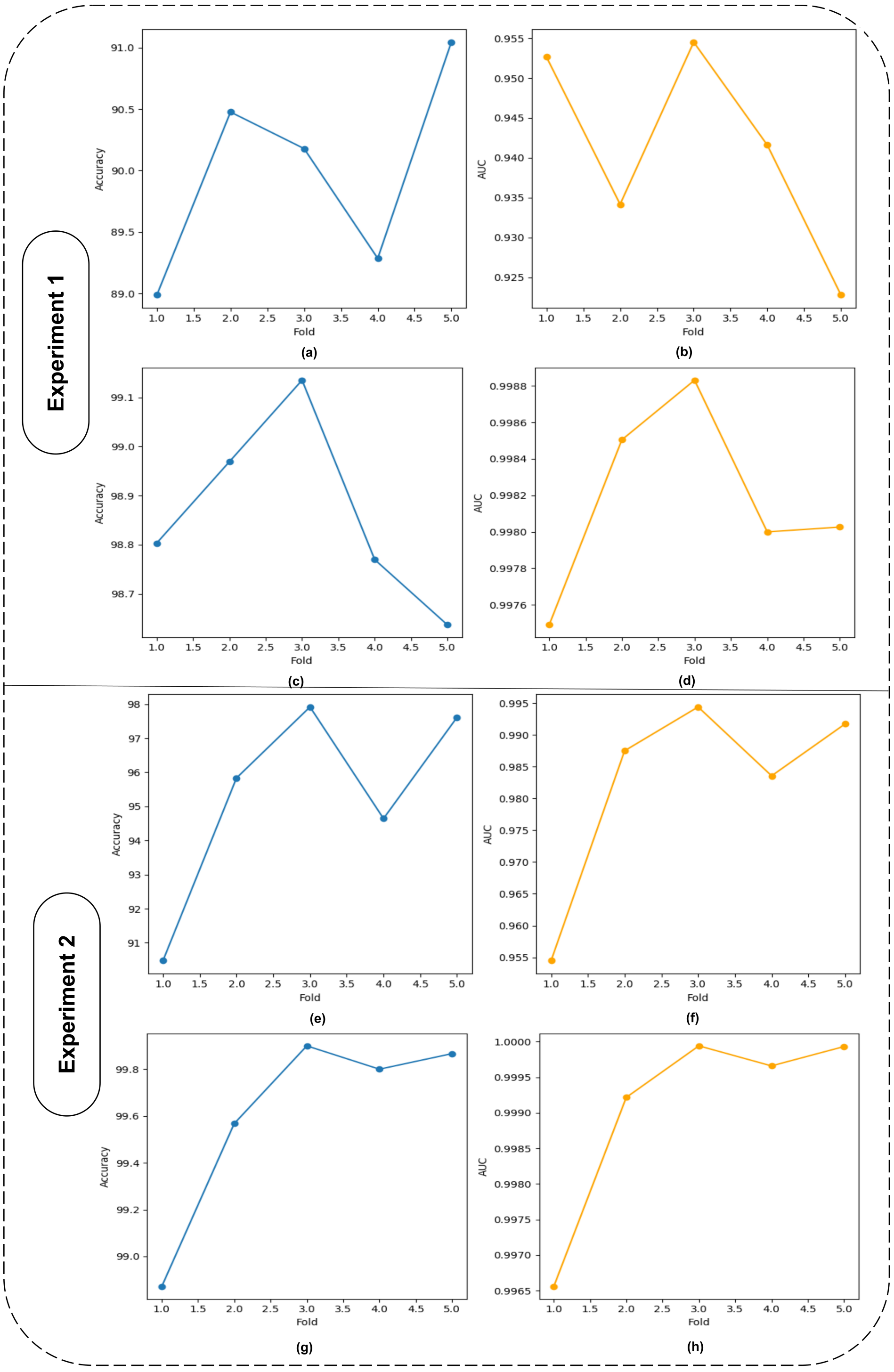

Figure 4. Experiment 1: Accuracy performance on 5 folds using ANN classifier for (**a**) CICInvesAndMal2019, (**c**) AMD datasets, and AUC scores of (**b**) CICInvesAndMal2019 and (**d**) AMD datasets. Experiment 2: Accuracy performance on 5 folds using proposed ANN classifier for (**e**) CICInvesAndMal2019, (**f**) AMD datasets, and AUC scores of (**g**) CICInvesAndMal2019 and (**h**) AMD datasets.

The AUC scores are given in (b) and (d), showing the deviation in even the fifth fold of both datasets. This means that features need to be optimized so that, when testing the model, consistency in the results is achieved. The deviation in each fold-based performance shows that there is some problem in the model architecture or feature set. The feature selection layer solved this issue and achieved better results on the proposed ANN classifier, which is discussed in the following section.

4.3. Experiment 2: Malware Detection Using Proposed ANN Classifier

In Experiment 2, the proposed custom feature selection layer is added and replaced with a simple input layer. The dimension of the feature set remains the same as in Experiment 1; however, it changes for both datasets due to the number of features. By applying the proposed custom feature selection layer, not only the performance of the model is improved, but also the consistency in the fold results is improved, which shows the robustness of the proposed layer-based ANN classifier. The achieved results using the proposed ANN classifier are shown in Table 4.

Table 4. Malware detection classification results using proposed ANN classifier via 5-fold validation method.

Dataset	Folds	ACC (%)	PRE (%)	REC (%)	F1 (%)	Training Time (s)	Testing Time (s)
CICInvesAndMal2019	Fold 1	90.48	93.00	93.00	85.00	19.79	0.35
	Fold 2	95.83	96.00	98.00	93.00	20.17	0.19
	Fold 3	97.92	98.00	99.00	97.00	19.10	0.22
	Fold 4	94.64	95.00	98.00	91.00	20.05	0.18
	Fold 5	97.61	97.00	100.00	96.00	19.59	0.18
	Average	95.30	95.80	97.60	92.40	19.74	0.23
AMD	Fold 1	98.87	99.00	99.00	98.00	184.64	0.79
	Fold 2	99.57	100	100	99.00	174.37	0.67
	Fold 3	99.90	100	100	100	178.34	0.68
	Fold 4	99.80	100	100	100	168.54	0.71
	Fold 5	99.87	100	100	100	176.85	0.82
	Average	99.60	99.80	99.80	99.40	176.55	0.74

In Table 4, we can see that, while testing on the same fold in both datasets, the results are improving. For dataset CICInesAndMal2019, the fold 1 accuracy was 89.58%, which improved in folds 2 and 3, but remained slightly lower in fold 4, while it rose again in fold 5. Similarly, the other scores' precision, recall, and even F1-score are improved. The average score on the CICInvesAndMal2019 dataset reached up to 95.30% accuracy, 95.80% precision, 97.60% recall, and 92.40% F1-score. The results continue to increase gradually in all folds except fold 4, where they decrease slightly. However, the gradual increase in the results leads us to conclude that feature selection in the model is appropriate. The results increase when selecting the appropriate features in the testing of the folds. The training and testing time of all folds on both datasets is shown in Table 4 when using a custom ANN classifier. The feature selection on each fold is performed while performing training and testing on both datasets. If we look at the average results of the CICInvesAndMal2019 dataset, it can be assumed that no significant change was observed compared to Experiment 1 based on the training and testing times. However, if we look at the results of the AMD dataset, only a 0.2 s time difference is observed in terms of average training, while testing time is reduced by 0.1 s. Although the time difference in both datasets' training and testing times is not significant, the results have nonetheless been improved. This leads us to conclude that the proposed architecture does not increase any computational cost on the overall training and testing on both datasets, with the same time cost, and that the results of both datasets have been improved. The overall average results are more satisfactory. To look at another measure, namely AUC, the positive class prediction during all five folds can be seen in Figure 4 from Experiment 2.

In Figure 4, Experiment-2-based section (f) shows that the results of AUC remain up to 95 while starting from fold 1. Likewise, they continuously increase during all folds except fold 4. The overall results of AUC also reach 0.98, which is significant to classify the positive class malware. The second dataset, AMD, includes malware app data from the Drebin dataset and Google-Play-Store-based benign app data. It shows remarkable performance on all the folds' testing and never even decreases. If we look at Table 4, fold 1 accuracy values start from 98.87%, with a 99% precision, recall, and 98% F1-score. It keeps increasing throughout the folds with a value of 99.87 until fold 5. The overall performance reflecting the performance of the proposed ANN classifier shows 99.60% accuracy, 100% recall and precision, and 99% F1-score, and if we look at the AUC scores in the (h) section, we see that it starts from 99.65% in fold 1 and reaches up to 100 in fold 5, while the average or overall score remains 100% for AUC too.

The proposed method shows a significant number of benefits over previous methods to detect Android malware. It includes useless feature reduction and dependent feature selection in a dynamic way that enhances the performance of the model. Furthermore, the robust and invariant method (K-fold validation) of classification is applied, which makes the results more satisfactory in terms of validation. A comparison of previous studies' results was conducted, which shows the better performance of the proposed method. The comparison is shown in Table 5.

Table 5. Comparison of proposed model-based achieved results with state-of-the-art methods applied to different datasets.

Reference	Year	Method	Datasets	Results
[20]	2021	Hybrid deep learning method using CNN and Bi-LSTM	Combined dataset using Androzoo and AMD datasets	Multi-Class ACC = 99.05%, PRE = 99.39%, REC = 99.41%
[14]	2021	Code DE obfuscation-based useful information-retrieval-based classification	Drebin	ACC = 99.51%, F1 = 94.61%
[17]	2023	Voting classifier based on two parallel streams of benign and malware apps	CICandMal2017	ACC = 98.65%, F1 = 96.82%, AUC = 0.9751
[2]	2023	Fuzzy inference system-based ensemble ML method	Drebin	ACC = 98.70%, F1 = 95.67%, AUC = 0.9325
			Drebin	ACC = 99.33%, PRE = 98.68%, REC = 1.00, SPEC = 98.67%, F1 = 99.34%
[22]	2023	Efficient feature reduction and neural network-based classification	CICInvestAndMal2019	ACC = 96.40%, PRE = 96.40%, REC = 96%, F1 = 96.59%
		Proposed feature-selection-layer-based ANN model	CICInvestAndMal2019	ACC = 95.30%, PRE = 96, REC = 98, F1 = 92
			Drebin/AMD	ACC = 99.60%, PRE = 100, REC = 100, F1 = 99

The first comparison contains the AMD dataset extended form including the AndroZoo dataset. Multiclass classification was conducted in this study, wherein the ACC was 99.05%, with PRE 99.39%, and REC 99.41%. The dataset performs multiclass classification, but the results are lower than in the conducted study. The second comparative study used the Drebin dataset. Two datasets were used, but the authors explicitly chose the Drebin dataset for its performance achievement. The achieved scores of the Drebin dataset are of 99.51% with a 94.61% F1-score. This study also applied five-fold cross-validation to validate its achieved results and the proposed method achieved higher scores in terms of accuracy and F1-score. Similarly, in the third and fourth comparisons, the Drebin dataset obtained lower scores than the proposed method-based AMD/Drebin dataset scores. In the fifth comparison, the CICInvestAndMal2019-dataset-based results show more accuracy at 96.40%, but analyzing only accuracy for comparison is not a fully appropriate validation method; it may be required to report other metrics too, such as precision, recall, F1-score, and AUC, with a bias-free validation method. Therefore, the proposed method achieved higher results compared to previous studies in terms of robustness, timely efficiency, and confidence.

5. Conclusions

Android malware detection has become an alarming and challenging issue due to the increasing use of mobile devices. The open-source nature of Android devices makes them more vulnerable compared to other mobile OSs. However, the malware detection method using the static method could be applied due to the on-spot detection of apps while installing them on any device. Many of the previous studies have proposed using static and dynamic features for Android malware detection. However, timely, efficient, robust, and more accurate malware detection remains an open challenge. Therefore, the proposed study uses an automated method of feature selection by proposing a custom input layer of the ANN classifier. Two datasets were used in this study, namely the CICInvestAndMal2019 and Drebin/AMD datasets. For the sake of model validation, two types of experiments were conducted to detect malware. In Experiment 1, the ANN classifier was used using a simple input layer, whereas the five-fold cross-validation method was applied to calculate the overall performance of the model. In Experiment 2, a custom feature selection layer was placed in the ANN classifier as an input layer, and a malware detection method was built for both datasets. The performance of the proposed ANN classifier was better than those from previous studies. Furthermore, the time computation on both experiments was calculated, which led us to conclude that the proposed custom ANN classifier increases the results on both datasets within the same training and testing time as per a simple ANN. However, more robust techniques in terms of feature selection and data transformation could be applied to increase the model's efficiency and timeliness. Similarly, a combination of static and dynamic features could be used to increase confidence in the applied approach.

Funding: There is no external funding for this research.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Dataset is publicly available.

Acknowledgments: This researchwas supported by the Researchers Supporting Project (RSP2023R476), King Saud University, Riyadh, Saudi Arabia.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Bai, H.; Xie, N.; Di, X.; Ye, Q. Famd: A fast multifeature Android malware detection framework, design, and implementation. *IEEE Access* **2020**, *8*, 194729–194740. [CrossRef]
2. Atacak, İ. An Ensemble Approach Based on Fuzzy Logic Using Machine Learning Classifiers for Android Malware Detection. *Appl. Sci.* **2023**, *13*, 1484. [CrossRef]
3. Chopra, R.; Acharya, S.; Rawat, U.; Bhatnagar, R. An Energy Efficient, Robust, Sustainable, and Low Computational Cost Method for Mobile Malware Detection. *Appl. Comput. Intell. Soft Comput.* **2023**, *2023*. [CrossRef]
4. Niu, W.; Wang, Y.; Liu, X.; Yan, R.; Li, X.; Zhang, X. GCDroid: Android Malware Detection Based on Graph Compression with Reachability Relationship Extraction for IoT Devices. *IEEE Internet Things J.* **2023**. [CrossRef]

5. Kouliaridis, V.; Kambourakis, G.; Peng, T. Feature importance in android malware detection. In Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 29 December 2020–1 January 2021; pp. 1449–1454.
6. Mobile Operating System Market Share Worldwide | Statcounter Global Stats—gs.statcounter.com. Available online: https://gs.statcounter.com/os-market-share/mobile/worldwide (accessed on 22 August 2023).
7. Oh, T.; Stackpole, B.; Cummins, E.; Gonzalez, C.; Ramachandran, R.; Lim, S. Best security practices for android, blackberry, and iOS. In Proceedings of the 2012 the First IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT), Seoul, Republic of Korea, 18 June 2012; pp. 42–47.
8. Mobile Cyberthreat Report for 2022—securelist.com. Available online: https://securelist.com/mobile-threat-report-2022/108844/ (accessed on 22 August 2023).
9. Ren, Y.; Leng, Y.; Cheng, Y.; Wang, J. Secure data storage based on blockchain and coding in edge computing. *Math. Biosci. Eng* **2019**, *16*, 1874–1892. [CrossRef] [PubMed]
10. Detecting and Eliminating Chamois, a Fraud Botnet on Android—Android-developers.googleblog.com. Available online: https://android-developers.googleblog.com/2017/03/detecting-and-eliminating-chamois-fraud.html (accessed on 22 August 2023).
11. Malware Statistics & Trends Report | AV-TEST—av-test.org. Available online: https://www.av-test.org/en/statistics/malware/ (accessed on 22 August 2023).
12. Kouliaridis, V.; Kambourakis, G. A comprehensive survey on machine learning techniques for android malware detection. *Information* **2021**, *12*, 185. [CrossRef]
13. Sawadogo, Z.; Dembele, J.M.; Mendy, G.; Ouya, S. Android malware detection: An in-depth investigation of the impact of the use of imbalance datasets on the efficiency of machine learning models. In Proceedings of the 2023 25th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, Republic of Korea, 19–22 February 2023; pp. 1460–1467.
14. Chen, Y.C.; Chen, H.Y.; Takahashi, T.; Sun, B.; Lin, T.N. Impact of code deobfuscation and feature interaction in android malware detection. *IEEE Access* **2021**, *9*, 123208–123219. [CrossRef]
15. Alazzam, H.; Al-Adwan, A.; Abualghanam, O.; Alhenawi, E.; Alsmady, A. An Improved Binary Owl Feature Selection in the Context of Android Malware Detection. *Computers* **2022**, *11*, 173. [CrossRef]
16. Guerra-Manzanares, A.; Bahsi, H.; Luckner, M. Leveraging the first line of defense: A study on the evolution and usage of android security permissions for enhanced android malware detection. *J. Comput. Virol. Hacking Tech.* **2023**, *19*, 65–96. [CrossRef]
17. AbuAlghanam, O.; Alazzam, H.; Qatawneh, M.; Aladwan, O.; Alsharaiah, M.A.; Almaiah, M.A. Android Malware Detection System Based on Ensemble Learning. 2023. Available online: https://www.researchsquare.com/article/rs-2521341/v1 (accessed on 22 August 2023).
18. Kshirsagar, D.; Agrawal, P. A study of feature selection methods for android malware detection. *J. Inf. Optim. Sci.* **2022**, *43*, 2111–2120. [CrossRef]
19. Yadav, P.; Menon, N.; Ravi, V.; Vishvanathan, S.; Pham, T.D. EfficientNet convolutional neural networks-based Android malware detection. *Comput. Secur.* **2022**, *115*, 102622. [CrossRef]
20. Haq, I.U.; Khan, T.A.; Akhunzada, A. A dynamic robust DL-based model for android malware detection. *IEEE Access* **2021**, *9*, 74510–74521. [CrossRef]
21. Gómez, A.; Muñoz, A. Deep Learning-Based Attack Detection and Classification in Android Devices. *Electronics* **2023**, *12*, 3253. [CrossRef]
22. Chaudhary, M.; Masood, A. RealMalSol: Real-time optimized model for Android malware detection using efficient neural networks and model quantization. *Neural Comput. Appl.* **2023**, *35*, 11373–11388. [CrossRef]