

Malware detection and identification in Android using the meta-heuristic based neural network

Huda Faris Mohammed^{a,*}

^a Ministry of Education/General Directorate of Education Al Qadisiyah, Qadisiyah, Iraq.Email:huda.faris.muhammed@gmail.com

ARTICLE INFO

Article history:

Received: 12 /04/2026

Revised form: 11 /05/2026

Accepted : 13/05/2026

Available online: 30 /09/2026

Keywords:

Malware detection
Android
Fruit fly
PCA
MLP

ABSTRACT

The objective of this study is to improve the classification accuracy and lower the chances of false positives by fine-tuning the architecture of the Multilayer perceptron (MLP) network. Based on the Tunadromd dataset, Principal Component Analysis (PCA) will be used to reduce the dimensions of the data to find the most important attributes and cut down computing costs. To overcome the issues posed by the conventional "trial-and-error" method of choosing the right parameters, Fruit Fly Optimization Algorithm (FOA) will be employed to calculate the optimum number of hidden neurons in the Multilayer perceptron (MLP). The classification results reveal that the FOA-MLP model yielded an impressive accuracy rate of 98.9%. Additionally, its precision rate for malware and non-malware samples was recorded at 97.7% and 99.2%, respectively. In terms of sensitivity and the total rate of false-detection, this model produced 96.6% and 1.1%, respectively. It is important to note that this method outperformed popular models such as Extra Tree Classifiers and Stacked Ensembles by 1.67%.

MSC..

<https://doi.org/10.29304/jqcm.2026.18.32831>

1. Introduction

Current surveys show that the Android operating system is very popular and comprise more than 75% in the market share, so this operating system become a very important aim for breaching security and privacy [1]. With billions of users worldwide, the Android operating system has become one of the most widely exploited platforms globally. Inherently, this made it as a prime target for cybercriminals after Microsoft Windows. This weakness is clearly reflected in al existing types of threats; for example, Trojan attacks represented a staggering 90% of malware Android devices. This makes the analysis of Android malware critically important [2].

Because of evolving of digital landscape, the malware become more sophisticated, where it using advanced techniques to bypass traditional systems. Although the signature-based detection is common defense, but its have limitations which increasingly evident as attackers. Because of this limitation of conventional methods to keep pace, it is important to shift toward more adaptive solutions. Machine learning (ML) has therefore grown, which is offer the flexibility and analytical ability to detect and categorize modern malware in effective manner than traditional [3].

*Corresponding author: Huda Faris Mohammed

Email addresses: huda.faris.muhammed@gmail.com

Communicated by 'sub etitor'

Machine learning has transformed the defensive landscape by shifting from reactive, static analysis toward proactive, behavior-based detection. Where by using dynamic monitoring, ML allow for the granular classification of malicious payloads ,such ransomware, adware, and Trojans. These ability depends on the high specific mitigation protocols. These models are exceed in high velocity data environments, where they identify anomalies and zero-day vulnerabilities [4][5]. Through the ordering of high variance features, ML will improves the interpretability and scalability of detection machanisms. This already reducing the reliance on manual forensic intervention [6].

However, the applied of ML in the Android environment is also faced significant hurdles. Adversaries commonly employ obfuscation, polymorphism, and metamorphic techniques to dynamically alter a file's signature, complicating detection efforts. Thus, will complicating the feature extraction process [4][6]. From other side, the high dimensionality of mobile malware attributes often leads to a "curse of dimensionality". This results in increased computational demands, model overfitting, and reduced generalizability to previously unseen threats. Furthermore, the inherent hardware limitations of mobile devices create a tension between model depth and real-time execution, necessitating the development of lightweight, adaptive architectures capable of maintaining high detection accuracy under strict resource constraints [7].

The complexity of various malware programs has led to the proposal of numerous models for securing Android smartphones. However, these methods varied between models based on signature, heuristic and behavior analysis. Unfortunately, these models encounter problems with polymorphic and metamorphic malware that can change its code to appear as other applications. Other proposal methods have included using machine learning and deep learning to treat with such malware. The strength of these methods lies in their ability to analyze and compare massive amounts of data, enabling them to identify indicators of threat.

In this article, a new method based on neural networks is proposed to identify malware in Android devices. The proposed approach employs a neural network optimized using the Fruit Fly Algorithm to improve classification performance following data preprocessing and perform appropriate pre-processing on the inputs for data classification. By using the sense of sight and cooperating with others, the fly finds the exact position of the fruit. The innovative aspect is using the fruit fly algorithm to determine the neurons no existed in the hidden layer within neural network. In perceptron (MLP) neural network, the most important hidden layer of this neural network is because in this layer the main processing is done on the data, therefore, the optimal determination of the neurons no in this layer is also of great importance.

The rest of the paper is organized as follows. In Section 2, The discussion on certain related works were demonstrated. In Section 3, The dataset description was shown. Section 4 contain the proposed model, in Section 5, The Results were presented, analyzed and discussed. Section 6 presents comparative comparison with recent works, and finally, in Section 7, The conclusions and future work were presented.

2. Related works

The application of Artificial Neural Networks (ANN) in the Android malware detectionhas become as a cornerstone, that is because of the transitioning from straightforward classification to more composition behavioral modeling. The early researches in this field were utilized a Multi-Layer Perceptrons (MLPs) to classify static features, which were extracted from Android manifests and metadata. An example of such features is extracted from permissions and intent filters.

For instance, the work of [8] used an efficient DeepRefiner which based on combining MLPs to analyze the features extracted from XML. The work also used more complex structures for bytecode semantics. The system has proven effective on both Drebin and AMD datasets, but its reliance on static analysis weakens its efficiency and makes it vulnerable to modren obfuscation techniques. With the increasing of complexity of malware, researchers moved from traditional ANN architectures towrds Deep Neural Networks (DNNs). For example, the DNN with multi-input 'MINAD, was proposed in [9]. In this work, the fusion of heterogeneous feature into separate branches, in order to improve the classification stability and also reduce model overfitting. The weaknes of MINAD model will appear when dealing with the malware families that dynamicly evolution along time.

Recent improvements consist of integration of many specialized ANN. In the aim of capturing of structural and sequential nature of Android system. Where, in [10] convolutional neural networks (CNNs) was utilized to deal

with Dalvik opcodes as sequential text. The local correlations of each adjacent opcodes were captured by CNN layers. The model ignores the branching and non-linearity nature of code in real programs. This enables malware programmers to use code packing and instruction shuffling to get their aim.

Also, in [11], the CNN was used to learn from API call graphs to discover the features of local malicious. The model converts the API calls relationships into a format that a deep learning model can understand. The gap of model is focusing on pre-defined API patterns as native code execution, but malicious doesn't rely on standard API calls.

While in [12], the authors suggested to convert a raw Dalvik bytecode into a 2D representation, and then using CNN to analysis. The gap here is that, the CNN treats code as pixels, but two different-images might perform the same malicious action. So malware programmer can using "junk code" to change the image entirely, but it still have same malware's behavior.

A more recently, the researchers toward to model the complex structural-relationships within function call graphs as Hypergraph Neural Networks [13] and Graph Neural Networks (GNNs) [14]. Their technique offers better detection of obfuscated threats. However, the GNNs captures the pairwise interactions between functions, but fail with high-order ones. At opposite, the Hypergraph Neural Networks success in both sides. But the real problem is the high computational cost of hypergraph construction, especially with real-time malware detection.

Furthermore, proposed other authors adaptive and heterogeneous architectures to treat the variety of mobile threats. These approaches showed efficient way in combining the analysis of high-dimensional structural and adaptive learning to get robust performance in Android malware detection [15][16][17]. The work of [15] optimizes data utilization using Adaptive Deep Learning, while [16] emphasizes structural code analysis using Graph Convolutional Networks, and [17] leverages external relational context by using Heterogeneous Information Network. These methods actually need high computational resources for graph processing addition to need of consistent external metadata.

It is very noticeable in recent research trends that thereis a tendency to rely on the TUNADROMD (Technical University of Nigeria Android Malware Dataset) dataset due to its accuracy, modernity, and its overcoming of the shortcomings of previous data.

In [18] authors utilized an ensemble of Extra Tree Classifier and Logistic Regression to malware detection. The model used feature vectors of static API and frequencies of opcode captured from the TUNADROMD dataset. This approach depends on statistical distributions of code, thus causing problem with identifying zero-day threats. Author of [19] used TabLSTMNet architecture to enhance the classification of Android malware based on structured features. The authors suggested use Explainable AI (XAI) powered by attention mechanism determine which features related to the malicious label. However, the limitation of this work is the challenges for real-time application on mobile devices.

In [20] authors suggested Stacked Ensemble Framework that used Semantic Permission Aggregation as data. Although their model proved efficiency, but it dependent on the manifest file's declarations. Therefore, the malware that does not appear in the static permission list will bypass the system.

Furthermore, details of the neural network architectures and input representations which are used in above literatures to detect Android malware, were presented in Table 1.

Table 1 - Neural Network Architectures and Input Representations for Android Malware Detection.

Reference	Year	Input representation	Datset used	Model
Xu et al. [8]	2018	Using 2-layer : XML/Manifest layer and Malicious bytecode layer.	Drebin & AMD	Convolutional Neural Network CNN for feature extraction with lightweight classifier
Nguyen et al. [9]	2022	The heterogeneous features of: system, library, and user-level components, were fused into separate branches.	Android malware samples from multisource	Deep Neural Network (DNN)
Sun et al. [10]	2019	Treat Dalvik opcodes as sequences of text to capture the execution flow.	CIC-AndMal2017	Sequential CNN

Kim et al. [11]	2022	API call graphs, embedded static features.	Large-scale dataset from various repositories	Deep Learning (MAPAS)	
Zhang et al. [12]	2018	Dalvik opcodes converted into 2D images.	Drebin and several malware family repositories	Convolutional Network CNN	Neural
Zhang et al. [13]	2023	Hypergraphs modeling multi-function relations.	CIC-AndMal2017 and Drebin datasets	Hypergraph Network	Neural
Yumlembam et al. [14]	2022	API graphs with Adversarial Defense.	Medusa and various IoT-based malware samples of Android	Graph Neural Network (GNN)	Network
Wajahat et al. [15]	2023	Using labeled and unlabeled data for malware.	Drebin, CIC-AndMal2017, and AMD	Semi-supervised Learning	Deep
Zaidi et al. [16]	2024	Function Call Graphs and API graphs.	Drebin and AMD	Graph Convolutional Network (GCN)	
Ye et al. [17]	2018	Heterogeneous features for real-time detection.	Real-world traffic data	Heterogeneous Information Network (AiDroid)	Network
Negi et al. [18]	2025	Static feature vectors extracted from API+ Opcode frequencies extracted from TUNADROMD	TUNADROMD	Extra Tree + Logistic Regression	
Ambekar et al. [19]	2025	Sequence of feature embeddings.	TUNADROMD	TabLSTMNet (Attention + XAI)	
Pudke et al. [20]	2025	Semantic Aggregation and intent-based feature mappings.	TUNADROMD	Stacked Ensemble (XAI)	

3. Data Set

The ‘Tunadromd’ dataset was utilized in this work. The dataset contain 4,465 samples categories in two classes of malware and benign i.e non-malware. The samples of dataset collected between August 2010 and October 2012 made available using the MobileSandbox project. Each sample have 241 distinct features. This dataset is imbalanced, it contains 3565 of ‘Malware Samples’, which is about 79.8%, and contains 900 of ‘Benign Samples’, which is about 20.2%.

4. Proposed Method

This section details our proposed methodology, as illustrated in Figure 1. The diagram outlines the workflow designed to enhance Android malware detection accuracy by leveraging a neural network optimized via the fruit fly algorithm. Each stage of this flowchart will be discussed in detail in the following subsections.

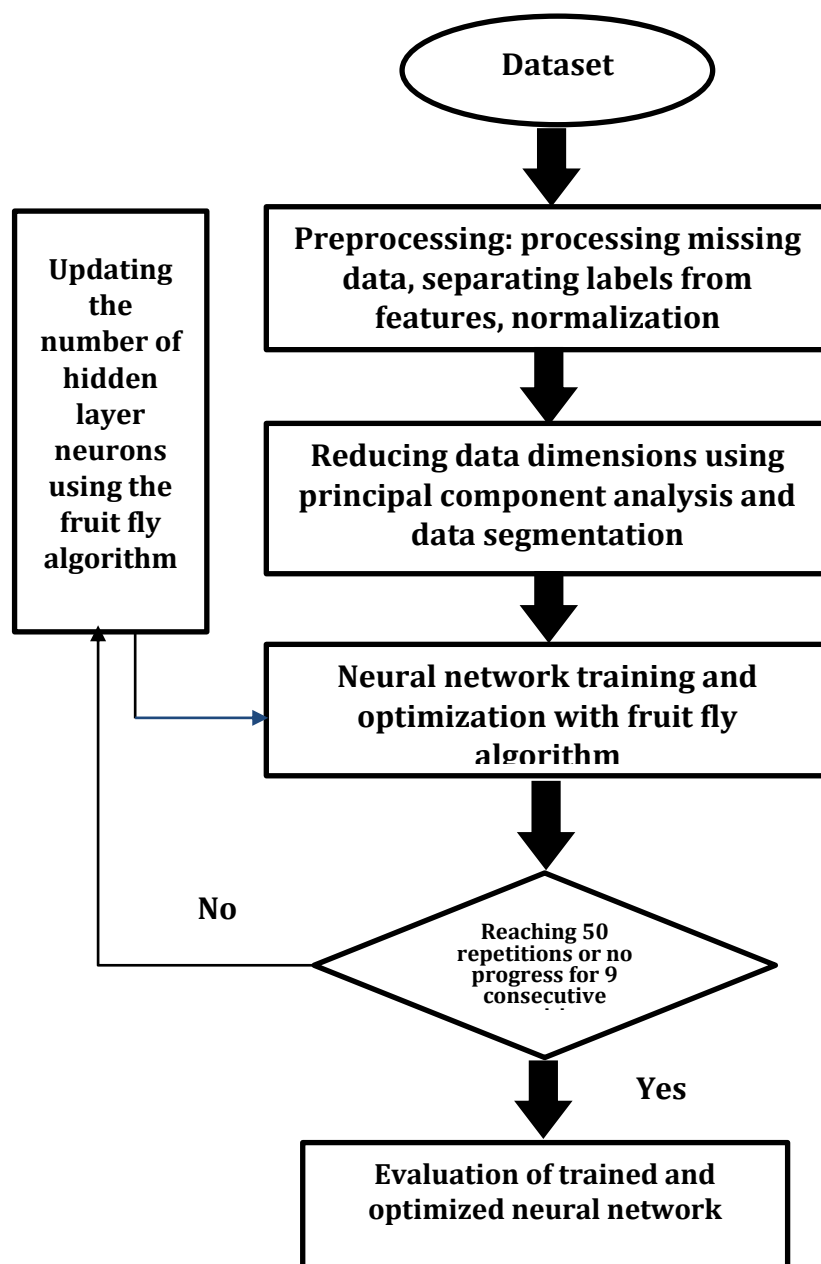


FIG. 1 Flowchart of the proposed method

4.1. Preprocessing of input data

Despite its apparent simplicity, the preprocessing stage is among the most critical steps in data classification, as it prepares all samples for subsequent neural network processing. If any of the steps of this stage, i.e. preprocessing, are not done correctly, the accuracy of the classifier will drop significantly. This step includes several sub-steps including: removing missing data as well as normalizing and separating labels from features. Each sub-step is explained in detail below.

4.1.1. Processing missing data

This stage is one of the most important pre-processing steps. In some cases, some data are recorded as non-existent or missing data in the data set due to reasons such as being noisy or stuck during the registration of the

data set. If these data are not processed in the pre-processing stage, when the neural network is faced with these data, it cannot correctly adjust its weight and bias value, so to speak, it is said that the network is overfitted and cannot be properly trained, and as a result, the accuracy of detection will drop sharply.

To solve the problem of missing data, there are several solutions, one of these methods is the averaging method on the nearest neighbors of the missing data in order to find the missing data value. One of the advantages of this method is that the replaced value instead of the missing data will be a value corresponding to the data in the data set and the neural network will be properly trained during training.

4.1.2. Separation of labels from features and normalization of feature matrix

After processing the missing data, in this step, firstly all the features considered for each sample will be stored in one matrix and the label specific to each data sample in another matrix, and then the values of the feature matrix will be normalized. Normalization ensures that all feature values fall within a consistent range, preventing any single feature from disproportionately influencing the network's weight updates during training. In other words, it can be said that some neurons reach the point of activity very quickly, but some of these neurons will not reach this point, and this causes the function approximation power and the accuracy of the neural network to decrease. Consequently, It is possible to state it that, to gain a logical and proper solution from the neural network, the information must be restricted to a specific scope by utilization a normalization technique.

In this work, the 'Min-Max Scaling' technique was used to normalize the features, the (equation 1) represents how this technique maps original value to a new value in a range defined by [L, H], as the standard equation described by [21].

$$X_n = \frac{X_i - X_{min}}{X_{max} - X_{min}} (H - L) + L, \quad i = 1, 2, \dots, N \quad (1)$$

X_i is the actual data value and X_n is the normalized value. L and H represent low and high limits of desired range.

4.2. Reducing the dimensions of unimportant data

Sometimes, all the features considered for an example in a data set are not of high importance, and in some cases, the presence of these low importance data, in addition to reducing the processing speed, cause the neural network to go astray during training. to be For this reason, removing or reducing this category of features increases the speed and accuracy of the final processing by the neural network. One of the existing methods in this field is the principal component analysis (PCA) algorithm. This algorithm reduces the volume of data by finding dimensions of samples that have the most variance or information.

In addition to reducing the dimensions of the data, this method reduces the correlation vector between the data by moving the data to a new space. This causes the network to encounter data that have lower correlations than each other during training, and each of the data transmits new information to the neural network. In this algorithm, the importance of the data is calculated according to the mutual covariance matrix of the data.

4.3. Training and optimization of perceptron classifier using fruit fly algorithm

This stage can be broken into three steps, which will be discussed in following sub sections:

4.3.1. Data Splitting

In this step, the data is divided into two categories, test and training. In general, the neural network uses the training data to train and adjust the appropriate value for the weights and parameters in this network. In this way, this group of data is given to the neural network along with the labels, and the network adjusts its weight and bias value by observing the input and output of each data so that the output is close to the real value. The process of adjusting the weight and bias of the neural network is called the process of training the network. After the neural network has been trained, in the next step to evaluate the trained neural network, the test data alone and without labels are provided to the neural network to evaluate the accuracy of the neural network. In this thesis, 70% of the data will be considered for training and 30% as test data.

4.3.2. Neural Network Training

The next step is to train and classify the data using MLP neural network. In fact, the perceptron neural network is a neural network without memory that does not have a round or loop in its structure and the data only moves forward. The multi-layer perceptron neural network has three layers of input, hidden and output. Neural network nodes are the computational parts of neural networks. In this neural network, the output of the input layer is used as the input of the hidden layer, and the output of the last hidden layer is used as the input of the output layer. It consists of a set of weights and biases that are adjusted for training and learning the neural network. In each layer, by multiplying the specific weight of each neuron by the value of the neuron and applying a constant term called bias, this value is transferred to the next layer, this process must continue until it reaches the output, and the output is the sum of all these multiplications to forms the output. As represented in the mathematical model of the artificial neuron by [22] and illustrated in (Equation 2).

$$O_j = f\left(\sum_{i=1}^n w_i X_i + b_j\right) \quad (2)$$

i, j : the index of the inputs/current neurons (ranging from 1 to n); X_i : the input value (feature); W_i : the weight assigned to input; b_j : the bias for the neuron.

After the output is obtained, it should be compared with the target function and the difference between the output and the true value forms the error. The error will be reduced. This training process is called 'error backpropagation training' in the neural network.

4.3.3. Neural Network Optimization

In the neural network, the number of neurons in the input layers is determined based on the number of features considered for each sample, and the number of neurons in the output layer is determined based on the number of available classes. The hidden layer of this neural network is the main layer in which data processing takes place. In many researches, the number of neurons in this layer is obtained by 'trial and error' method. One of the problems of the trial and error method is that it is not possible to achieve the optimal number of neurons and causes in prevent achieving the highest possible accuracy. Here the optimization process is to get the optimal number of neurons for the hidden layer of perceptron neural network. In this work, the fruit fly algorithm was used to discover the optimal number of neurons in the hidden layer of perceptron neural network.

The optimization algorithm of fruit flies is based on the search behavior of fruit flies for food, which behavior includes two stages. Fruit flies possess highly sensitive olfactory and visual capabilities, enabling them to locate food sources from distances of up to 40 kilometers. In the first stage, first, fruit flies randomly fly to some food sources around the group of fruit flies through the sense of smell. This phase is called the process of exploring by smell. In the second stage, which is called visual exploration, the fly flies towards the best food source and gets closer to the better food source. This step is repeated until the stop condition is reached. As a result, during these processes, the location of the group of fruit flies is updated. The fruit fly algorithm finds the exact state of the fruit by using the fly's sense of sight and smell and cooperating with others. This algorithm minimizes the possibility of getting stuck in local optima due to its search and exploration capabilities. The meaning of the search and discovery capability is that at the beginning of the work, the search coefficient is higher than the discovery, so the algorithm at the beginning of the optimization process focuses more on searching for the values that it can accept as the value of the hidden layer, and then the search rate is gradually reduced. And it discovers the values that had the most relevance to the optimization problem in the search process. To solve the optimization problem using this algorithm, it is enough to set only the parameters of the algorithm suitable for the optimization problem.

In this work, the optimization process implemented using three primary steps. The first step is to code the members of the population based on the parameter for which we want to obtain the optimal value. For this purpose, in this work, in the first step, the members of the algorithm were coded based on the number of hidden layer neurons. At the beginning of the algorithm, the population is placed randomly, because the fitness of all members is the same at the beginning of the work.

In the second step, after placing the members, the fit of each member is calculated by the fit function. In this thesis, the purpose of finding the number of hidden layer neurons is to reduce the classification error, so the fitness

function is defined based on the amount of classification error. The Mean Squared Error was used as proportionality function which its equation as given by [23]:

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - Y'_i)^2 \quad (3)$$

n: total number of samples; Y_i : actual value; Y'_i : predicted value.

In the next step, the population is updated based on the value of the fitness pan, that is, all the members will move to the value that has more fitness or less error. Next, the algorithm terminates or continues the optimization process by checking the stop condition. The stopping condition in this task will be defined as reaching the maximum amount of iteration. In this algorithm, if the stop condition is not established, the optimization process continues from the second step and this process will continue until the stop condition is reached. After the stopping condition is fulfilled, the algorithm will choose the amount that has the least error for the neural network as the optimal value of the number of neurons.

4.4. Evaluation Criteria

In this article, to quantitatively assess the performance of the proposed method, several evaluation criteria of accuracy were employed. These metrics are derived from the Confusion Matrix, addition to use of accuracy, precision (Correctness), sensitivity (recall), specificity (exclusiveness), and F-Measure (F1-Score).

These metrics can calculate using following equations as illustrated in [24]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

$$Sensitivity = \frac{TP}{TP + FN} \quad (6)$$

$$Specificity = \frac{TN}{TN + FP} \quad (7)$$

$$F1 - Score = 2 \times \frac{Precision \times Sensitivity}{Precision + Sensitivity} \quad (8)$$

Where, TP: True Positive, TN: True Negative, FP: False Positive, FN: False Negative.

5. Results, Analysis and Discussion

The result obtained of applying the pre-processing stage, i.e. normalization and removal of non-existent data, showed in Figure 2. As seen in this figure, all the data were placed in the specified range between -1 and 1, and there is no trace of non-existent data in this data set

	1	2	3	4	5	6	7
1	0	2.6198e-09	5.2396e-09	2.6198e-08	1.2863e-06	0	0
2	0	5.2396e-09	7.8594e-09	2.6198e-08	3.8249e-07	0	0
3	0	2.6198e-09	1.0479e-08	1.5719e-08	0	0	0
4	0	2.6198e-09	1.3099e-08	2.6198e-08	6.0779e-07	2.1359e-05	0
5	0	2.6198e-09	1.3099e-08	2.6198e-08	5.2134e-07	1.1003e-06	0
6	0	2.6198e-09	1.0479e-08	5.2396e-09	0	0	0
7	0	2.6198e-09	1.0479e-08	1.5719e-08	0	0	0
8	0	2.6198e-09	1.0479e-08	1.5719e-08	0	0	0
9	0	2.6198e-09	1.5719e-08	1.5719e-08	0	0	0
10	0	2.6198e-09	1.0479e-08	1.5719e-08	0	0	0
11	0	2.6198e-09	1.0479e-08	5.2396e-09	0	0	0
12	0	2.6198e-09	1.0479e-08	1.5719e-08	0	0	0
13	0	2.6198e-09	1.3099e-08	2.6198e-08	7.5188e-07	5.8972e-06	0
14	0	2.6198e-09	5.2396e-09	2.6198e-08	8.7501e-07	0	0
15	0	2.6198e-09	1.8339e-08	1.5719e-08	0	0	0

FIG. 2 Normalization and deletion of missing data of the dataset

The results of applying Principal Component Analysis (PCA) was filtering out the 10 least significant features. This process reduced the feature space from 251 to 241 dimensions, while maintaining the original 4,465 samples.

The parameters of the fruit fly algorithm which used to optimize the number of neurons in the hidden layer of the neural network are given in table 2

Table 2 - parameters set for the fruit fly algorithm.

Evaluation Function	Stop Condition	Primary Population
cost_{fun}=1-Accuracy	Reaching 50 repetitions or no progress for 9 consecutive repetitions	10

These parameters represent the search space with stop condition criteria for the FOA algorithm, also balancing computational efficiency. Figure 3 shows the optimization process of the fruit fly algorithm to find the optimal number of hidden layer neurons. The floor of this figure shows the horizontal axis of repetitions and the vertical axis of the error rate of each repetition. This diagram shows that in the initial stage of optimization, due to the random selection of the population and values, the error rate is high, and as the algorithm converges towards the optimal value, the error rate of the neural network is also decreasing.

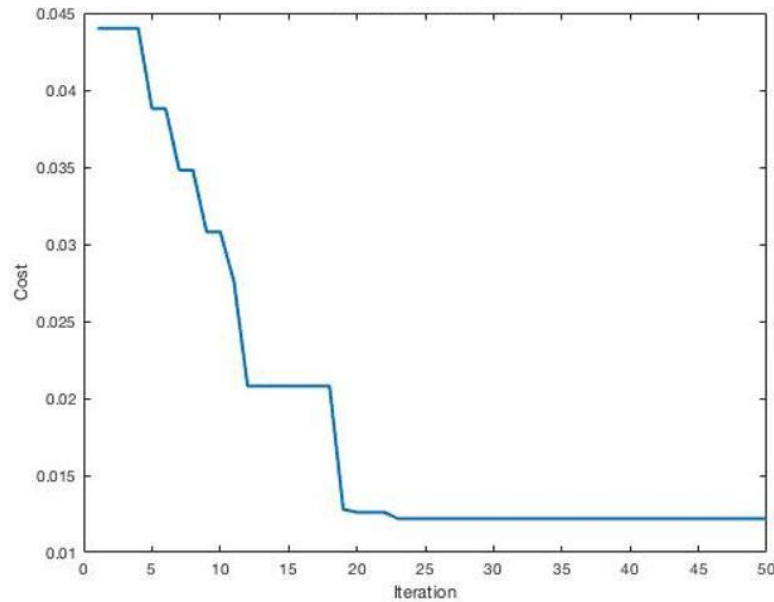


FIG. 3 Fruit fly algorithm optimization process to find the optimal number of hidden layer neurons

As shown in figure, high value of initial error rate represents the random initialization of population. Then, the algorithm's effectiveness in converging toward optimal neurons number in hidden layer showed by subsequent decline.

The structure of the perceptron neural network after the optimization process is represented in Figure 4. As seen in this neural network, the number of neurons in the first layer is equal to 241 neurons according to the number of features reduced after the next reduction step, and also the number of neurons in the first hidden layer is equal to 191 neurons and the second layer is equal to 100 neurons by the fruit fly algorithm. Also, the number of neurons in the output layer was determined to be equal to 2 neurons to the number of totals (malware or not).

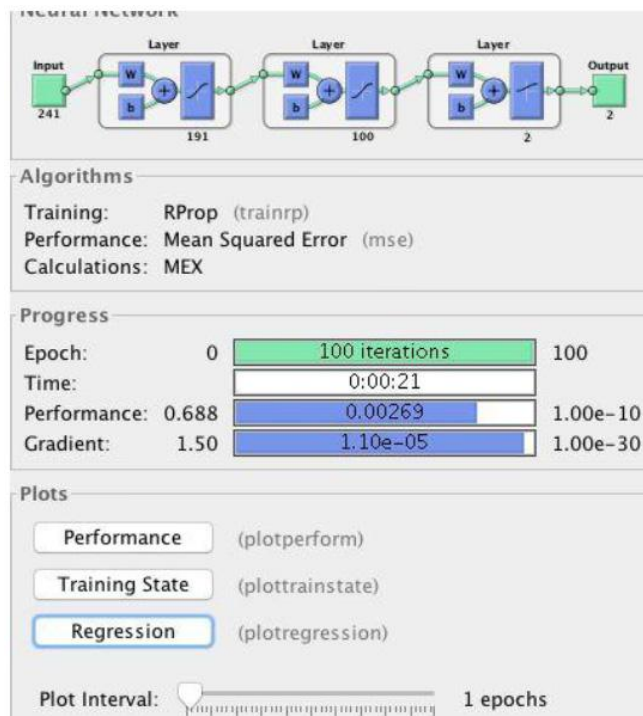


FIG. 4 Perceptron neural network structure after completion of the optimization process

The activation function of all layers is defined as hyperbolic tangent type. The results obtained from the training data represented by Clutter Matrix (Figure 5).

Output Class	0	721 20.2%	8 0.2%	98.9% 1.1%
	1	4 0.1%	2838 79.5%	99.9% 0.1%
	0	99.4% 0.6%	99.7% 0.3%	99.7% 0.3%
		Target Class		

FIG. 5 Confusion matrix of training data

As shown in confusion matrix, the correct detection of malware, of non-malware were 20.2% and 79.5% respectively. While, False detection was 0.2% for malware and 0.1% for non-malware. Finally, the model achieve accuracy of 99.7% and sensitivity or coverage was 99.4%.

In the following, we will use the graph of the data performance characteristic curve to further examine the output results of the test data. This curve shows the positive correct detection rate compared to the false positive detection rate. In this curve, the closer the graph is to the coordinates (1 and 0), it means that the rate of correct detection is higher than the rate of false detection, and as a result, the accuracy rate will be higher.

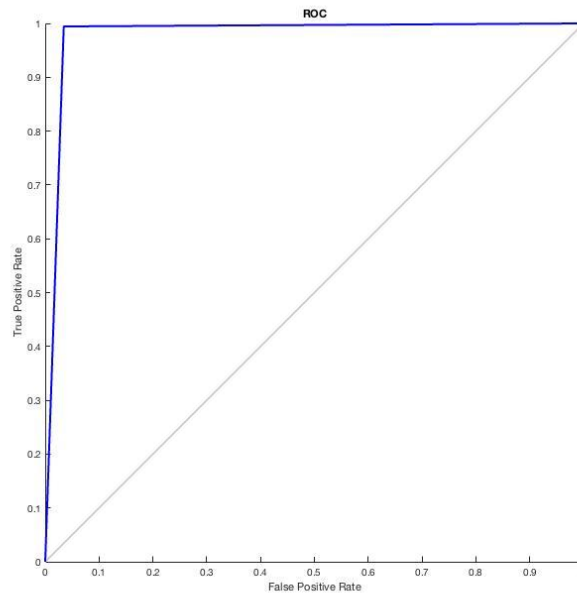


FIG. 6 Performance characteristic curve diagram of training data

The proposed model demonstrates exceptional performance across all primary evaluation metrics during the training phase (Figure 7). As shown in figure, high Precision and Recall reflect a low rate for both False Positives and False Negatives. This prove the reliability of the proposed system in distinguishing between malware and non-malware. Also, the value of F-Score which equal to 99.79% proves the balancing of model between precision and sensitivity and its stability even with unbalancing of data.

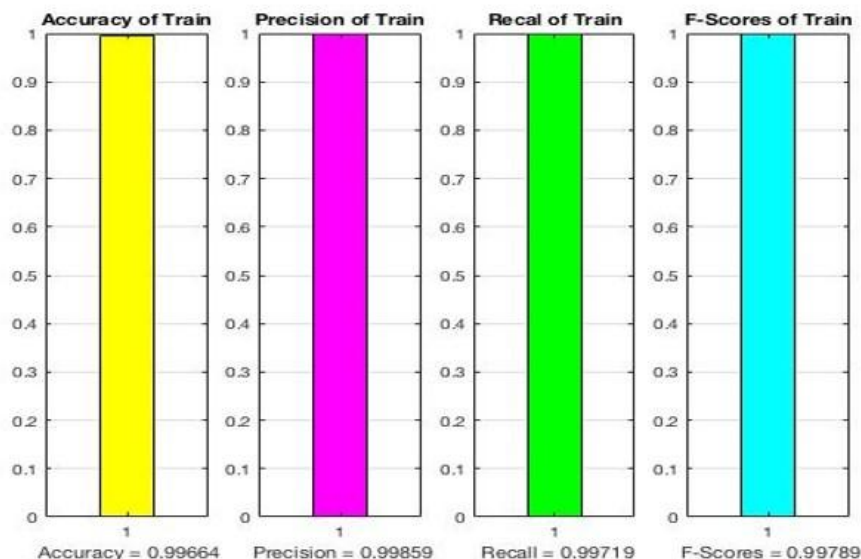


FIG. 7 The results related to the evaluation criteria obtained for the training data

From Figures 5,6,7 notice that the correct detection rate achieved from high accuracy with false detections of 0.2% for malware and 0.1% for non-malware. Also, the proximity of the curve to the top-left coordinates indicate same result. Similarly, the high value of F-Score, showed the stability of model even with unbalanced data.

After completing the training process, we provided the test data to the neural network, which will continue to analyze and check the results of this data.

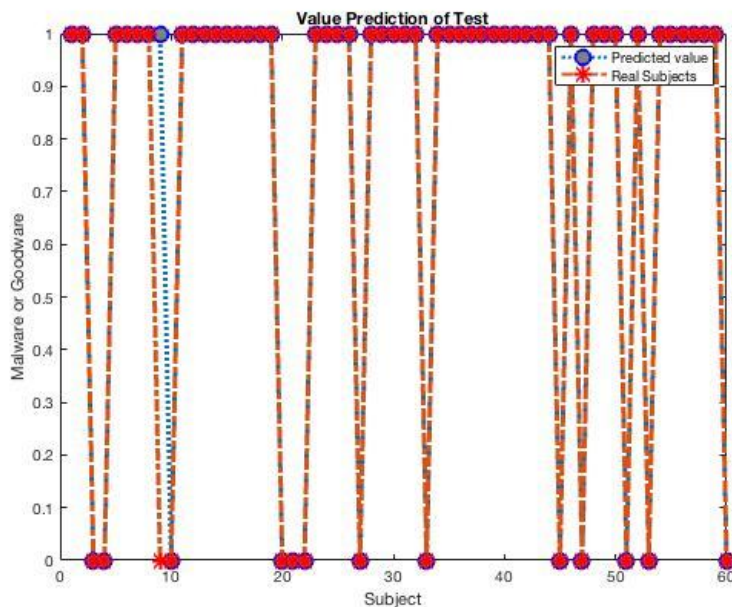


FIG. 8 Figure 8 Bar chart of test data output prediction

Figure 8 shows the level of classification accuracy for test data in the form of a bar graph. This diagram is drawn for 60 samples and as can be seen in this diagram, the red circles represent the real value of the data label and the blue circle represents the value predicted by the neural network. As it is known, the predicted value and the real value almost coincide except in some data. Next, to check the results of the whole test data, the confusion matrix of the test data is shown in Figure 9.

Output Class	0	168 18.8%	4 0.4%	97.7% 2.3%
	1	6 0.7%	715 80.1%	99.2% 0.8%
		96.6% 3.4%	99.4% 0.6%	98.9% 1.1%

FIG. 9 Confusion matrix of test data

As it is clear in this matrix, the accuracy obtained for the test data is equal to 98.9%. The Correct detection achieved 18.8% for malware and 80.1% for non-malware. While, False detection was 0.4 % for malware and 0.7% for non-malware. Finally, the model achieve accuracy of 98.9% and sensitivity or coverage was 96.6%.

For further investigation, the performance characteristic curve of the test data is given below. As can be seen from this curve, the blue graph is in the desired area and it shows the high accuracy for the test data.

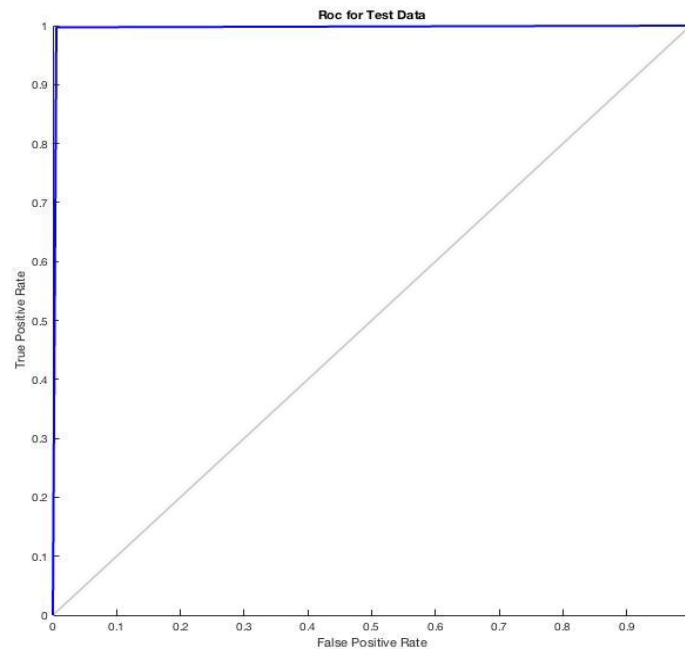


FIG. 10 Test data performance characteristic curve diagram

In Figure 11, the results obtained for the test data from different aspects.

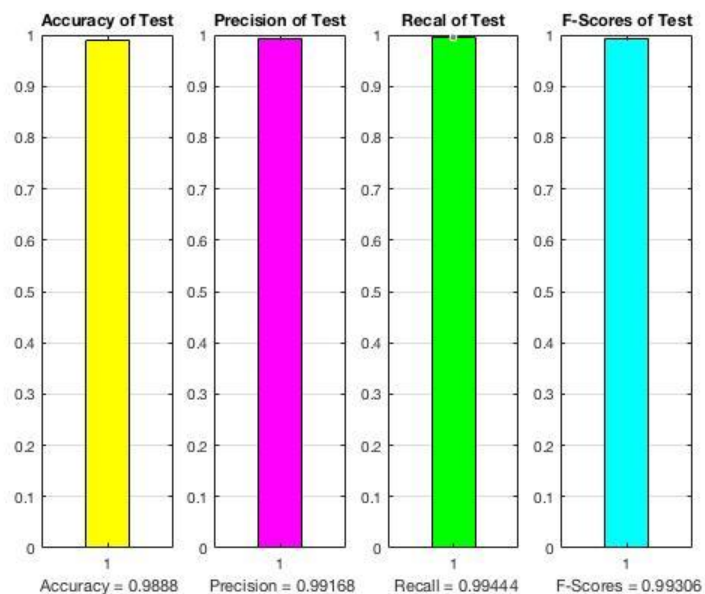


FIG. 11 The results related to the evaluation criteria obtained for the test data

The results presented in Figures 10 ,11 proved the high reliability and generalization capability of model. This, represented by trajectory which positioned close to the top-left coordinate (Figure 10). Also it supported by the comprehensive metrics in (Figure 11).

6. Comparative analysis

In this section, a comparative analysis with state-of-art works established. The recent works that, used Tunadromd dataset for malware detection on Android environment, were only adopted for comparison. The results of comparison were reported in Table 3.

Table 3 - Comparative Analysis.

Classification Technique	Accuracy	Reference
Extra Tree Classifier	97.23%	[18]
long short-term memory (LSTM)	98 %	[19]
Stacked Ensemble Framework	97.54%	[20]
MultilayerPerceptron (MLP)	98.9%	Proposed

7. Conclusion

This work successfully addressed the Android-based malware detection by proposing an optimized neural network technique. This study demonstrated that integrating heuristic optimization with classical deep learning architectures can significantly enhance the security of mobile systems.

The core contribution of this work is the optimization of a Perceptron Neural Network MLP using the Fruit Fly Optimization method. The test of proposed approach proved that the model outperformed the traditional method that used "trial-and-error" for hidden layer configuration. The application of Principal Component Analysis successfully reduced the dimensionality of the Tunadromd dataset, resulting in a more computationally efficient model.

The overall accuracy of proposed model was 98.9%. This accuracy indicate that, the performance gain of suggested methodology was 1.1% over baseline models with low false-positive rate of 1.1% at same time. These results refer to efficient of meta-heuristic algorithms using in aim of fine tuning of deep learning hyperparameter.

Future work include the exploration of applying Fruit Fly Optimization framework with larger and more heterogeneous datasets.

References

- [1] Alawneh, Heba, David Umphress, and Anthony Skjellum. "Android malware detection using neural networks&process control block information." 2019 14th International Conference on Malicious and Unwanted Software (MALWARE). 2019.
- [2] Musa, Hawraa O., and Muhanad T. Younis. "Effective Android Malware Detection Using CNN and LSTM Model with GWO-Based Feature Selection." *Babylonian Journal of Machine Learning* 2026 (2026): 1-22.
- [3] Goodfellow, Ian, et al. *Deep learning*. Vol. 1. No. 2. Cambridge: MIT press, 2016.
- [4] Musikawan, Pakarat, et al. "An enhanced deep learning neural network for the detection and identification of android malware." *IEEE Internet of Things Journal* 10.10 (2022): 8560-8577.
- [5] Baker del Aguila, Ryan, et al. "Static malware analysis using low-parameter machine learning models." *Computers* 13.3 (2024): 59.
- [6] Panja, Subir, et al. "An efficient malware detection approach based on machine learning feature influence techniques for resource-constrained devices." *IEEE Access* 13 (2025): 12647-12665.
- [7] Alhogail, Areej, and Rawan Abdulaziz Alharbi. "Effective ml-based android malware detection and categorization." *Electronics* 14.8 (2025): 1486.

-
- [8] Xu, Ke, et al. "Deeprefiner: Multi-layer android malware detection system applying deep neural networks." 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2018.
 - [9] Nguyen, Duc V., et al. "Minad: Multi-inputs neural network based on application structure for android malware detection." *Peer-to-peer networking and applications* 15.1 (2022): 163-177.
 - [10] Sun, XingPing, et al. "Android Malware Detection using Sequential Convolutional Neural Networks." *Journal of Physics: Conference Series*. Vol. 1168. No. 6. IOP Publishing, 2019.
 - [11] Kim, Jinsung, et al. "MAPAS: a practical deep learning-based android malware detection system." *International Journal of Information Security* 21.4 (2022): 725-738.
 - [12] Zhang, Yi, Yuexiang Yang, and Xiaolei Wang. "A novel android malware detection approach based on convolutional neural network." *Proceedings of the 2nd international conference on cryptography, security and privacy*. 2018.
 - [13] Zhang, Dehua, et al. "Android malware detection based on hypergraph neural networks." *Applied Sciences* 13.23 (2023): 12629.
 - [14] Yumlembam, Rahul, et al. "Iot-based android malware detection using graph neural network with adversarial defense." *IEEE Internet of Things Journal* 10.10 (2022): 8432-8444.
 - [15] Wajahat, Ahsan, et al. "An adaptive semi-supervised deep learning-based framework for the detection of Android malware." *Journal of Intelligent&Fuzzy Systems* 45.3 (2023): 5141-5157.
 - [16] Zaidi, Atif Raza, et al. "Deep learning-based detection of android malware using graph convolutional networks (gcn)." *STATISTICS, COMPUTING AND INTERDISCIPLINARY RESEARCH* 6.1 (2024): 57-73.
 - [17] Ye, Yanfang, et al. "AiDroid: When heterogeneous information network marries deep neural network for real-time Android malware detection." *arXiv preprint arXiv:1811.01027* (2018).
 - [18] Negi, Divya, and Sandeep Kumar. "A High-Performance Malware Detection Using Extra Tree Classifier and Logistic Regression on TUNADROMD Dataset." 2025 IEEE International Conference on Computer, Electronics, Electrical Engineering&their Applications (IC2E3). IEEE, 2025.
 - [19] Ambekar, Namrata Govind, et al. "TabLSTMNet: enhancing android malware classification through integrated attention and explainable AI." *Microsystem Technologies* 31.3 (2025): 695-713.
 - [20] Pudke, Priya, and Urvashi Bansal. "A Stacked Ensemble Framework for Android Malware Detection Using Semantic Permission Aggregation and Explainable AI." *Security and Privacy* 8.6 (2025): e70110.
 - [21] Han, Jiawei, Micheline Kamber, and Jian Pei. "Data mining: Concepts and." *Techniques*, Waltham: Morgan Kaufmann Publishers (2012): 2012-13.
 - [22] Haykin, Simon. *Neural networks and learning machines*, 3/E. Pearson education india, 2009.
 - [23] Hastie, T., et al. "An introduction to statistical learning Springer." New York (2013).
 - [24] Fawcett, Tom. "An introduction to ROC analysis." *Pattern recognition letters* 27.8 (2006): 861-874.