*Computers, Materials & Continua*
DOI: 10.32604/cmc.2022.030878
***Article***

Tech Science Press

# Swarm Optimization and Machine Learning for Android Malware Detection

**K. Santosh Jhansi[1,2,*], P. Ravi Kiran Varma[2] and Sujata Chakravarty[3]**

[1]Centurion University of Technology and Management, Paralakhemundi, Odisha, India
[2]Maharaj Vijayaram Gajapathi Raj College of Engineering, Vizianagaram, India
[3]Centurion University of Technology and Management, Bhubaneswar, Odisha, India
*Corresponding Author: K. Santosh Jhansi. Email: santosh.jhansi@gmail.com
Received: 04 April 2022; Accepted: 08 June 2022

**Abstract:** Malware Security Intelligence constitutes the analysis of applications and their associated metadata for possible security threats. Application Programming Interfaces (API) calls contain valuable information that can help with malware identification. The malware analysis with reduced feature space helps for the efficient identification of malware. The goal of this research is to find the most informative features of API calls to improve the android malware detection accuracy. Three swarm optimization methods, viz., Ant Lion Optimization (ALO), Cuckoo Search Optimization (CSO), and Firefly Optimization (FO) are applied to API calls using auto-encoders for identification of most influential features. The nature-inspired wrapper-based algorithms are evaluated using well-known Machine Learning (ML) classifiers such as Linear Regression (LR), Decision Tree (DT), Random Forest (RF), K–Nearest Neighbor (KNN) & Support Vector Machine (SVM). A hybrid Artificial Neuronal Classifier (ANC) is proposed for improving the classification of android malware. The experimental results yielded an accuracy of 98.87% with just seven features out of hundred API call features, i.e., a massive 93% of data optimization.

**Keywords:** Android malware; API calls; auto-encoders; ant lion optimization; cuckoo search optimization; firefly optimization; artificial neural networks; artificial neuronal classifier

## 1 Introduction

Android, a Linux-based mobile Operating System (OS), is the most popular mobile OS worldwide. Unlike other operating systems that are subject to numerous laws and copyrights, Android is open-source, allowing developers from all around the world to contribute to it. Android is frequently targeted by malware because of its widespread usage. Malware poses a threat to the data privacy, integrity, and availability. Attackers take aid of malware infiltrated into a genuinely looking regular Application (App). App downloads, in particular from unauthorized or not from play store, are the most prevalent route for malware to infiltrate the android OS. When the victim installs an app from unprotected sources, there is a possibility of malware attack. To avoid these attacks, more sophisticated malware detection techniques [1–3] are required due to the huge quantity of harmful malware applications.

Predicting Android malware [4,5] is possible with a number of existing technologies. However, these methods rely on signatures, which are digital traces beneath the code. The signatures are taken from the app's Android Application Package (APK) and compared to malicious signatures in a database. This type of solution, on the other hand, is incapable of detecting malware that isn't in the database. As malware is becoming more prevalent, it is vital to present a solution that can accurately detect all varieties of malware [6], while using the least amount of time and resources possible.

Many attempts have been made to identify malware on android devices [7,8] traditional, signature-based malware detection methods compare the APK file signature to the signature of the malicious program in the malware database, which excludes malware that isn't in the database. In this scenario, there is a necessity to develop advanced detection techniques [9] for the effective identification of malware. This paper contributes to novel android malware detection based on API call features using a hybrid model of swarm optimization algorithms (ALO, CSO, and FO) along with auto-encoders evaluated on several ML classifiers. The main contributions of this paper are:

1. Detecting suspicious APIs for accurate classification of goodware and malware android apps.
2. Design and implementation of hybrid auto-encoders and swarm optimization wrapper feature minimization methods.
3. Evaluation of the proposed models using various ML classifiers including novel artificial neuronal classifier.
4. Determining the best algorithm for predicting android malware based on a variety of factors.

The remainder of the paper is ordered as follows: Section 2 presents the related work, methodology of the proposed work explained in Section 3, Section 4 details the experimentation setup, performance analysis and experimentation results are described in Section 5, and Section 6 presents the conclusion & future work.

## 2 Related Work

Although the android platform is the most comprehensive, the variety and number of malware have increased dramatically. As a result, researchers began looking for ways to detect and block these dangerous applications [10–12] Tehrany et al. [13] addressed the problem of imbalanced datasets in android malware detection using statistical analysis. Ranking methods, under-sampling, and Synthetic Minority Oversampling (SMOTE) are used for pre-processing and balancing the dataset. SVM, KNN, and Iterative Dichotomiser 3 (ID3) classifiers are used for the detection model. The SMOTE technique with KNN classifier outperformed other comparing methods with an accuracy of 98.69%. Dharmalingam et al. [14] experimented with Term Frequency-Inverse Document Frequency (TF-IDF) for the identification of malware in android. The permissions are ranked and graded using permission grader. The graded permissions are classified using Artificial Neural Networks (ANNs) and obtained an accuracy of 94.22% when compared to competing algorithms. Yildiz et al. [15] demonstrated a feature selection method based on linear regression for improving the performance of android malware classification. The experimented methodology resulted in improved accuracy of 96.1% with reduced training time.

Sarah et al. [16] introduced a model based on the recursive feature selection method combined with an ensemble classifier for android malware detection. The influential features are selected using the Recursive Feature Selection method and classified using the LightGBM classifier. The results indicate the experimented process proved its efficiency in classifying android malware with 99.5% accuracy. Elayan et al. [17] addressed the inefficiency of traditional malware detection techniques and upgraded to use deep learning techniques. Gated Recurrent Unit (GRU) is employed for the classification of good ware from malware. The obtained model outperformed the classical methods with 98.2% accuracy. Arif et al. [18] demonstrated a risk-based fuzzy approach using the Analytical Hierarchy Process (AHP) for mobile malware detection. Further to detect the malware, the risk is categorized into four different categories (very low, low, medium & high) and is also explored. The overall approach achieved an accuracy of 90.54%.

The suggested method in this paper focuses on incorporating artificial neural networks to improve the effectiveness of android malware application detection [19] and classification [20,21]. The most influential features for distinguishing good ware apps from malware apps are first recognized by employing auto-encoders in wrapper-based swarm optimization feature selection methods. Second, a new artificial neuronal classifier for productive android malware classification is tested by combining artificial neural networks and induction classifier.

## 3 Methodology

The architecture of the proposed wrapper-based feature selection technique using auto-encoders for android malware detection is highlighted in Fig. 1. The entire data is categorized into train and test sets in 7:3 ratio. The malware analysis process comprises two steps namely: feature selection & classification. In feature selection, swarm-intelligence-based ALO, CSO & FO algorithms are examined for iterative feature search selection. The selected features are passed onto auto-encoders to obtain a compressed form of input features. The auto-encoder's output is transferred to an induction algorithm to find the fitness of features in classifying malware from good ware. The induction algorithm induces a classifier by mapping feature space into a set of class values, useful in classifying future cases. Finally, in classification, the obtained reduced feature set is evaluated using popular induction algorithms and the proposed artificial neuronal classifier.

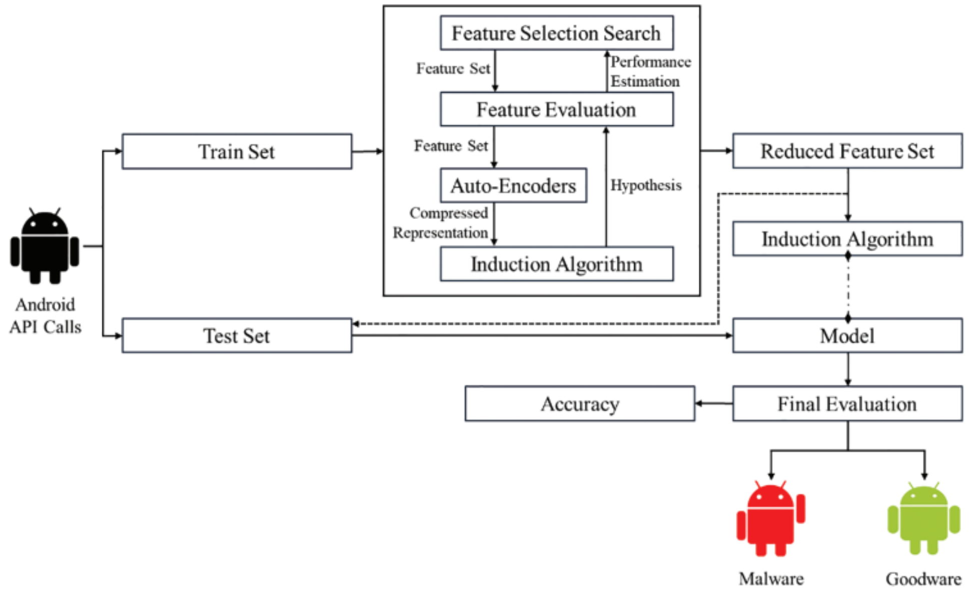

**Figure 1:** System architecture

### *3.1 Feature Selection*

The process of extracting the most consistent, relevant, and non-redundant features to employ in model creation is known as feature selection. The purpose of feature selection is to improve the model's performance while lowering the modelling cost. It reduces the number of input variables in a machine learning model by eliminating redundancies and unnecessary features and then restricting the set of features to those that are most appropriate for the model.

The primary advantages of completing feature selection in advance rather than relying on the ML model to determine which features are most important are: simple models, reduction in variance, reduced training time of the model & avoiding high dimensionality curse.

**Definition 1.** For a given inducer $I$ and data set $D$ having features $(x_1, x_2, x_3, \ldots, x_n)$ with distribution $D$ over a labelled instance space, the *optimal feature subset* $X_{opt}$ is the feature subset, which maximizes the accuracy of the induced classifier $C = I(D)$.

In unsupervised feature selection techniques, the wrapper-based approach identifies the best combination of features that maximizes the performance of the model. By analyzing different models with the addition and/or removal of features using the greedy approach, the most influencing features are chosen for model building. The wrapper-based feature selection process is represented in Fig. 2

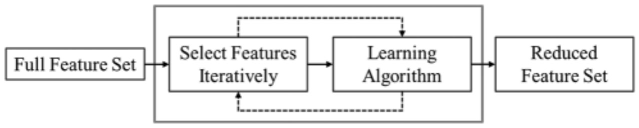

**Figure 2:** Wrapper-based feature selection

The swarm-intelligence-based ALO, CSO & FO feature selection search techniques are employed in order to replace the greedy approach. For the determination of near-optimal solution during fitness convergence, the choice of objective function plays a pivotal role. In the iterative approach of wrapper-based feature selection, the selected features count and the error obtained from the model after every iteration are considered to evaluate the fitness of the chosen features in the proposed method as shown in Eq. (1)

$$f(x) = error * \tau + \frac{u - s(l)}{u} * (1 - \tau) \tag{1}$$

where, $\tau$ represents the penalty given to learning algorithm for obtaining *error* during fitness calculation and $\tau \in [0, 1]$. The length of complete feature set is represented as $u$ & the length of the solution feature set is represented as $l$.

#### *3.1.1 Auto-Encoders*

Auto-encoders are a class of neural networks used to learn a compressed representation of input data. An auto-encoder consists of sub-models called an encoder and a decoder. The encoder model learns from input features and compresses them, while the decoder tries to regenerate the input from the compressed output of the encoder. Once the encoder model is trained, the decoder is ignored. The trained encoder is now used to extract features from raw data for the purpose of training machine learning models.

As shown in Fig. 3 the proposed auto-encoder comprises of an encoder with an input layer having $N$ nodes, two hidden layers having $[N * 2, N]$ nodes each, another hidden layer called latent space having $N/2$ nodes and a decoder with 2 hidden layers with $[N,\ N*2]$ nodes each and an output layer with $N$ nodes. Batch normalization is applied after each hidden layer and the LeakyReLU activation function is used in all the layers, whose mathematical representation is given in Eq. (2)

$$LeakyReLU: f\left(h_\theta\left(x\right)\right) = \begin{cases} 0.01 * h_\theta\left(x\right), & if\ h_\theta\left(x\right) < 0 \\ h_\theta\left(x\right), & if\ h_\theta\left(x\right) \geq 0 \end{cases} \tag{2}$$

where, $h_\theta\left(x\right)$ is obtained using Eq. (3)

$$h_\theta\left(x\right) = \sum_{i=1}^{n} w_i x_i + bias = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + bias \tag{3}$$

here, $x_i = (x_1,\ x_2,\ \ldots,\ x_n)$ indicates the inputs to the nodes, $w_i = (w_1,\ w_2,\ \ldots,\ w_n)$ represents the weights assigned to nodes (initially weights are assigned randomly between [0, 1] and they are adjusted during learning), a constant term *bias* is added to every layer, to assure the parameters are not passing through the origin. A node is fired if the output of Eq. (3) is above the *threshold* as shown in Eq. (4)

$$\text{Output} = f\left(x\right) = \begin{cases} \sum_{i=1}^{n} w_i x_i + bias \geq threshold \\ 0 < threshold \end{cases} \tag{4}$$

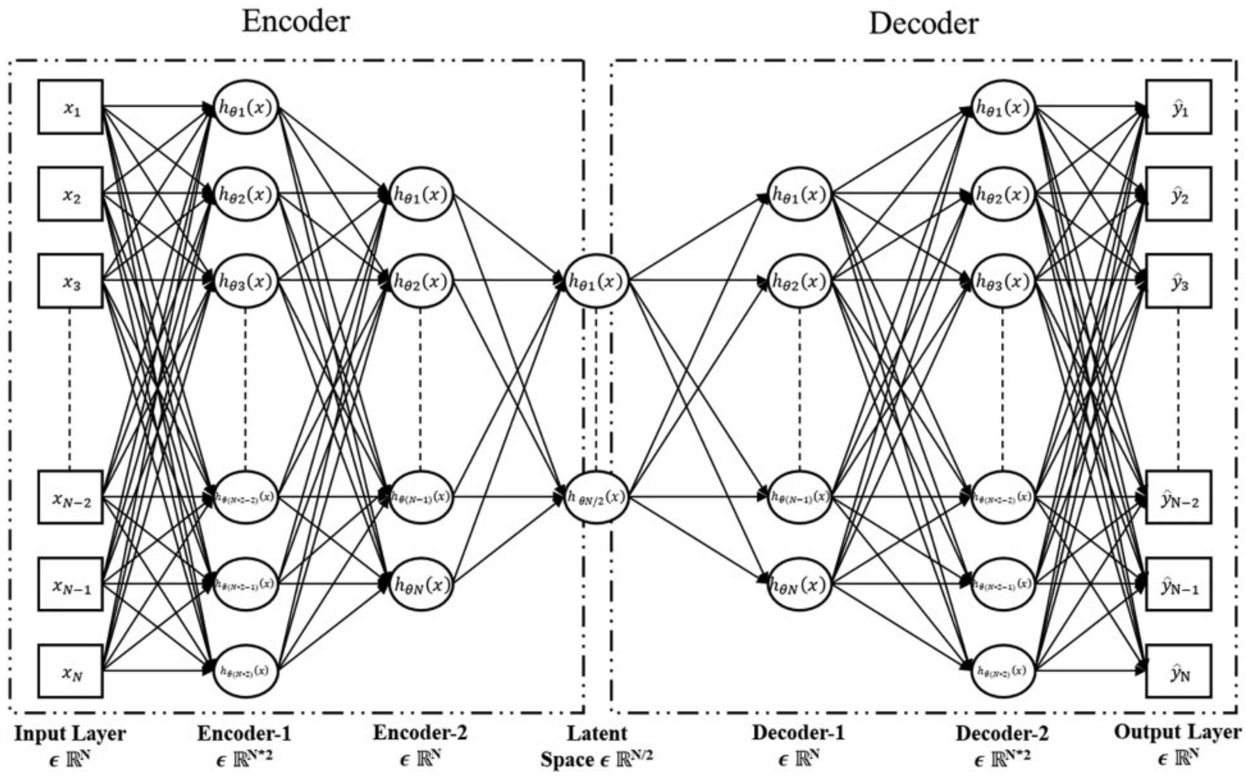

**Figure 3:** Architecture of proposed auto-encoder

### *3.1.2 Wrapper-Based Ant Lion Optimized Feature Selection*

The Ant Lion Optimizer developed by Seyedali [22] characterizes the hunting mechanics of antlions for ants in nature. ALO finds the ideal solution regardless of the initial values of the parameters. The convergence of ALO is fast and can handle integer & discrete constraints. The steps involved in prey hunting: ants random walk, building traps, entrapment of ants in traps, prey catching, and rebuilding the traps are implemented.

Initially, all the ants and antlions populations are initialized randomly. For each ant, antlions are selected based on the roulette wheel and a random walk is created and normalized using lines 9–10 and Eq. (5).

**Algorithm 1:** Wrapper-Based Ant Lion Optimized Feature Selection (WALOFS)

```
1.  Define objective function: $f(x) : x = (x_1, x_2, \ldots, x_d)$
2.  Random initialization of ants and antlions population
3.  Calculation of Ants and antlions fitness
4.  Identify best antlions and presume as elite
5.  While end criterion is not obtained or ($t <$ maxGeneration)
6.      For each ant
7.          Using Roulette Wheel: select antlion
8.          $d^t = \frac{d^t}{I}$ and $C^t = \frac{c^t}{I}$
9.          $X(t) = [0, cum\_sum(2r(t_1) - 1), cum\_sum(2r(t_2) - 1), \ldots, cum\_sum(2r(t_n) - 1)]$
10.         $X_i^t = c_i + \frac{(X_i^t - a_i) \times (d_i - c_i^t)}{(d_i^t - a_i)}$
11.         $Ant_i^t = \frac{R_A^t + R_E^t}{2}$
12.     End for
13. Calculation fitness for all ants
14. Replacement of antlion with equivalent ant, if it becomes fitter
15. If an antlion becomes fitter then
16.         $Antlion_j^t = Ant_i^t$ iff $(Ant_i^t > Antlion_j^t)$
17. End while
```

$$r(t) = \begin{cases} 1, & rand > 0 \cdot 5 \\ 0, & rand \le 0 \cdot 5 \end{cases} \tag{5}$$

In Algorithm 1, *cum_sum* refers cumulative sum, no. of iterations is represented as $n$, indicates current iteration is indicated using $t$, *rand* is a random number between [0, 1], $r(t)$ is a stochastic function, $a_i$ is the $i^{th}$ variable minimum random walk, $d_i$ is the $i^{th}$ variable maximum random walk, $c_i^t$ is $i^{th}$ variable minimum at $t^{th}$ iteration, $d_i^t$ is $i^{th}$ variable maximum at $t^{th}$ iteration, $c^t$ all variables minimum at $t^{th}$ iteration, $d^t$ represents a vector of all variables maximum at $t^{th}$ iteration. The position of the ant is updated using line 11. After every iteration, the fitness of all the ants is calculated and an antlion is replaced with its equivalent ant if it is a good fit using line 16. Here, $Ant_i^t$ indicates the position of $i^{th}$ ant at $t^{th}$ iteration, $I$ is a ratio, $Antlion_j^t$ indicates the position of $j^{th}$ antlion at $t^{th}$ iteration, $R_E^t$ represents the random walk elite at $t^{th}$ iteration chosen by roulette wheel, $R_A^t$ represents the random walk of antlion at $t^{th}$ iteration chosen by roulette wheel. The global minimum solution validated by the built-in wrapper classifier is returned after all iterations are complete.

### *3.1.3 Wrapper-Based Cuckoo Search Optimized Feature Selection*

The Cuckoo search algorithm developed by Yang et al. [23] depicts the egg-laying behavior of cuckoos in the nests of other host birds. Each cuckoo lays an egg and drops it in a randomly chosen nest. The best nests (solutions) with high-quality eggs are passed down to the following generation. The host nests that are accessible are finite, and a host's chances of discovering an alien egg are with probability (0, 1). All of the nests are randomly initialized during the initiation phase, however once the iterations begin, the Cuckoo traverses the solution space by modifying the nests through *Levy* flight as shown in line 4 here, the step size is tuned by $\propto$, sigmoid function is used to convert the continuous values generated by CSO in binary version of CSO, as shown in Eqs. (6) and (7).

**Algorithm 2:** Wrapper-Based Cuckoo Search Optimized Feature Selection (WCSOFS)

```
1.  Define objective function: $f(x) : x = (x_1, x_2, \ldots, x_d)$
2.  Initially generate population for $n$ host nests $x_i$ (i = 1, 2, 3, ..., $n$)
3.  while (end criterion not achieved) or ($t < maxGeneration$)
4.      Generate new solution by $E_i^{x+1} = E_i^x + \propto * Levy\ (\lambda)$, with randomly chosen cuckoo
5.      Evaluate the fitness of the solution $F_i$ [For maximizing, $F_i\ \alpha\ f(x_i)$]
6.      Randomly choose a nest $j$ among $n$
7.      if ($F_i > F_j$) then
8.          $j$ is replaced with new solution
9.      end if
10.     Abandon a part of worse nets by ($p_a$) fraction
11.     New nests are built in abandoned fraction ($p_a$) using $E_i^{x+1} = E_i^x + \delta * \left(E_1^x - E_2^x\right)$
12.     Lay aside the finest nests/solutions
13.     Obtain current best nest/solution by ranking them
14.     Current best solution is passed onto the next generation
15. End while
```

$$M\left(E_i^j\right) = \frac{1}{1 + e^{-E_i^j(x)}} \tag{6}$$

$$E_i^j(x+1) = \begin{cases} 1 & M\left(E_i^j\right) > 0.5 \\ 0 & otherwise \end{cases} \tag{7}$$

A few nests are abandoned and replaced with new nests at the end of an iteration using line 11 of Algorithm 2 where, $E_1^x$ and $E_2^x$ are randomly chosen nests, $\delta \in [0,\ 1]$. In wrapper-based CSO Feature Selection, the nests are initialized and fitness is calculated at the beginning. After every iteration, the iteration's best solution is represented as the global best solution, gleaning on fitness. As per the modalities of CSO, $p_a$ nests are abandoned and replaced using line 11. The global minimal solution, as verified by the embedded wrapper classifier, is returned once all iterations have been completed.

### *3.1.4 Wrapper-Based Firefly Optimized Feature Selection*

The Firefly Optimization algorithm developed by Lindfield et al. [24] mirrors firefly action to draw in different fireflies. The light intensity of the two fireflies is straightforwardly corresponding to their engaging quality, while the distance between them is contrarily proportionate. Assuming that there is definitely not a more brilliant firefly close by, the firefly will move aimlessly. The attraction between any two fireflies is influenced by the brightness of the firefly. The firefly with a reduced brightness shifts towards the firefly with a greater brightness. When there are no brighter fireflies, random movement is used. The attractive force between two given fireflies is calculated using line 15 of Algorithm 3 at a distance $r = 0$, $\beta_0$ represents attractiveness, the distance between fireflies $j$ and $k$ is indicated by $r_{jk}$ which is calculated using line 10. Here, $r_{ji}$ and $r_{ki}$ represent the spatial components of $i^{th}$ component at $j^{th}$ and $k^{th}$ fireflies, dimensions are shown by $n$. When one firefly is attracted to another, the movement of the firefly is determined using line 11 where, $r_j$ represents the current position of $j$, $random \in [0, 1]$. Fireflies move randomly based on $\alpha$ (mutation rate) when there are no brighter fireflies. The solution to the global minimum validated by the wrapper class built-in classifier is returned after all iterations are complete.

**Algorithm 3:** Wrapper-Based Firefly Optimized Feature Selection (WFOFS)

```
1.  Define Objective function: $f(x) : x = (x_1, x_2, \ldots, x_d)$
2.  Initial generation of fireflies' population $x_i$ (i = $1, 2, 3, \ldots, n$)
3.  Calculation of light intensities $I$ for fireflies
4.  Define light absorption coefficient $\gamma$
5.  while (end criterion is not obtained) or ($t < MaxGeneration$)
6.      for each firefly $i$ ($\forall\, i = 1, 2, 3, \ldots, n$)
7.          for every firefly $j$ ($\forall\, j = 1, 2, 3, \ldots, i$)
8.              Get light intensities of $I_i$ and $I_j$
9.                  if $I_i < I_j$ then
10.                     $r_{jk} = ||x_j - x_k|| = \sqrt{\sum_{i=1}^{n} \left(x_{j,i} - x_{k,i}\right)^2}$
11.                     $x_j = x_j + \beta_0 e^{-\gamma r_{jk}^2} * \left(x_j - x_k\right) + \alpha * \left(random - \frac{1}{2}\right)$
12.                 else
13.                     Randomly move firefly $i$
14.                 end if
15.                 Attractiveness is updated via $\beta_r = \beta_0 * e^{-\gamma r_{jk}^2}$
16.             New solution is evaluated & light intensity is updated
17.         end for
18. end for
19. Obtain current best by ranking fireflies w.r.t light intensities
```

### *3.2 Classifier*

The induction/classification algorithms [25,26] employed to evaluate the proposed android malware detection system are LR, DT, RF, KNN & SVM. Apart from these classifiers, a new hybrid classifier combining artificial neural networks with induction algorithm, called artificial neuronal classifier is proposed in this paper.

*Artificial Neuronal Classifier*

The architecture of the proposed artificial neuronal classifier is highlighted in Fig. 4. The artificial neuronal classifier is an association of artificial neural networks [27] with the induction classifier [28]. The features are given as input to the ANN and trained to understand the patterns among the features and the correlation between them. The acquired knowledge from ANN is transferred to the induction classifier. The induction classifier induces the knowledge space into the learning algorithm to maximize the accuracy in classifying malware from good ware.

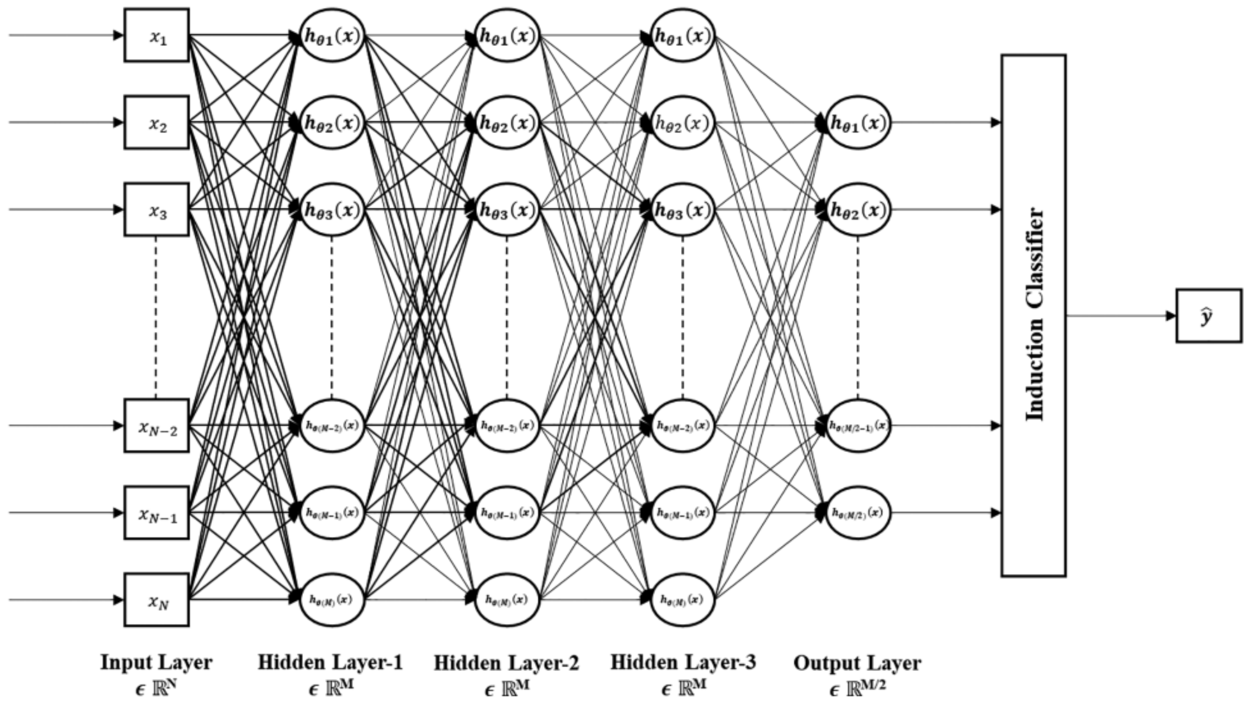

**Figure 4:** Architecture of proposed artificial neuronal classifier

After several experiments, the number of layers in ANN of ANC are configured as an input layer with $N$ nodes, three fully connected hidden layers having $M$ nodes each, another fully connected hidden layer with $M/2$ nodes, and an output layer with an induction classifier. The number of nodes in hidden layer are calculated using the Eq. (8)

$$M = \frac{2 * N}{\alpha} + 2 \tag{8}$$

where, $M$ represents no. of nodes in hidden layer, $N$ indicates no. of features, and $\alpha\ \epsilon\ [2,\ 10]$. The activation function decides whether a neuron is fire/activated or not. If the output is above a certain threshold, the neuron will be fired otherwise it is not fired. Except for induction classifier layer, the activation function used in all the layers is $ReLU$, which is given in Eq. (9)

$$ReLU: \ f\left(h_\theta\left(x\right)\right) = Max\left(0, h_\theta\left(x\right)\right) \tag{9}$$

here, $h_\theta\left(x\right)$ is calculated using Eq. (3). A neural network optimizer is a function or algorithm that alters the weights and learning rate of the network. The optimizer used in ANC is Adam, where the decay rate of mean of gradients $m_{ij}^{(t)}$ and the mean of the square of the gradients $v_{ij}^{(t)}$ for each weight $\omega_{ij}$ be $\beta_1$ and $\beta_2$ respectively. Also, let $N$ be the constraint learning-rate factor. Then, the update rules for Adam are as shown in Eqs. (10) and (11).

$$m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1 - \beta_1)\frac{\partial C^{(t)}}{\partial \omega_{ij}} \tag{10}$$

$$v_{ij}^{(t)} = \beta_2 v_{ij}^{(t-1)} + (1 - \beta_2)\left(\frac{\partial C^{(t)}}{\partial \omega_{ij}}\right)^2 \tag{11}$$

The normalized mean of the gradients $\hat{m}_{ij}^{(t)}$ and the mean of the square gradients $\hat{v}_{ij}^{(t)}$ are computed using Eqs. (12) and (13).

$$\hat{m}_{ij}^{(t)} = \frac{m_{ij}^{(t)}}{\left(1-\beta_i^t\right)} \tag{12}$$

$$\hat{v}_{ij}^{(t)} = \frac{v_{ij}^{(t)}}{\left(1-\beta_2^t\right)} \tag{13}$$

The final update rule for each weight is given in Eq. (14).

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\eta}{\sqrt{\hat{v}_{ij}^t} + \epsilon}\hat{m}_{ij}^{(t)} \tag{14}$$

After adjusting the weights of the neural network, the error between the predicted output and the actual output is calculated using a loss function. The Mean Absolute Error (MAE) is used in this model, which is given in Eq. (15).

$$MAE = \frac{1}{n}\sum\nolimits_{i=1}^{n}\left|y_i - \hat{y}_i\right| \tag{15}$$

here, $y_i$ is the actual, and $\hat{y}_i$ is the predicted output, and $n$ is the total number of outputs. Once the neural network is trained for certain epochs, the acquired knowledge from feature space is transmitted to the induction classifier to differentiate between malware and goodware.

## 4 Experimental Setup

All experiments are performed on a 64-bit Windows 10 operating system having 2.30 GHz Intel® Core™ i5 processor, 2TB Hard Drive and 8 GB RAM with Jupiter platform configured to support machine learning and deep learning packages. The programming language used is Python 3.7.

The API call sequence data used in the experiment is collected from IEEE data port. The data contains 43,876 API call sequences having 100 features, among them 42,797 are malware and 1,079 are good ware API call sequences. The size of the dataset is 17.1 MB. The experimented data is collected using the cuckoo sandbox environment and verified using virus total [29].

## 5 Performance Analysis and Experimental Results

The proposed system for android malware detection confirms the classification accuracy through various classification analysis metrics such as Mean Squared Error (MSE), Root Mean Square Error (RMSE), Precision, Recall, F1-Score, and Accuracy which are defined as follows:

$$Precision = \frac{True_{pos}}{False_{pos} + True_{pos}} \tag{16}$$

$$Recall = \frac{True_{pos}}{False_{neg} + True_{pos}} \tag{17}$$

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{18}$$

$$Accuracy = \frac{True_{pos} + True_{neg}}{True_{pos} + False_{pos} + True_{neg} + False_{neg}} \tag{19}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{20}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2} \tag{21}$$

where, $True_{pos}$ represents the samples identified correctly as good ware, $True_{neg}$ indicates the samples identified correctly as malware, $False_{pos}$ is the samples incorrectly identified as goodware, $False_{neg}$ is he samples incorrectly identified as malware, $\hat{Y}_i$ is the predicted output, $Y_i$ is the actual output, and $n$ represents the number of samples.

In experiment no. 1, the ALO, CSO & FO algorithms wrapped with LR, DT, RF, SVM, and KNN are evaluated for their performance on the API calls sequence dataset. All the experiments are run for 10 iterations with 10 agents. The results of the analysis are listed in Tab. 1 and its graphical illustration is given in Fig. 5. The FO optimizer, when wrapped with the KNN classifier obtained better results with an 88% reduced feature set and an accuracy of 98.29%.

**Table 1:** Accuracy comparison of ALO, CSO & FO wrapped with LR, DT, RF, SVM & KNN

| S. no | Classifier | Accuracy before feature selection | Feature selection method | Accuracy after feature selection | % Change in accuracy | Features selected | % Decrease in features |
|---|---|---|---|---|---|---|---|
| 1 | LR | 94.6311 | ALO | 98.1882 | 3.56% | 61 | 39% |
| | | | CSO | 98.2782 | 3.64% | 88 | 12% |
| | | | FO | 97.5387 | 2.91% | 14 | 86% |
| 2 | DT | 95.6485 | ALO | 98.0971 | 2.45% | 41 | 59% |
| | | | CSO | 98.4047 | 2.76% | 79 | 21% |
| | | | FO | 97.9831 | 2.33% | 18 | 82% |
| 3 | RF | 95.9985 | ALO | 97.6719 | 1.67% | 21 | 79% |
| | | | CSO | 97.7402 | 1.74% | 83 | 17% |
| | | | FO | 97.9666 | 1.97% | 13 | 87% |
| 4 | SVM | 94.9654 | ALO | 98.2224 | 3.26% | 91 | 9% |
| | | | CSO | 98.6098 | 3.64% | 71 | 29% |
| | | | FO | 98.3933 | 3.43% | 15 | 85% |
| 5 | **KNN** | 93.6521 | ALO | 98.4731 | 4.82% | 61 | 39% |
| | | | CSO | 98.5414 | 4.89% | 77 | 23% |
| | | | **FO** | **98.2981** | **4.65%** | **12** | **88%** |

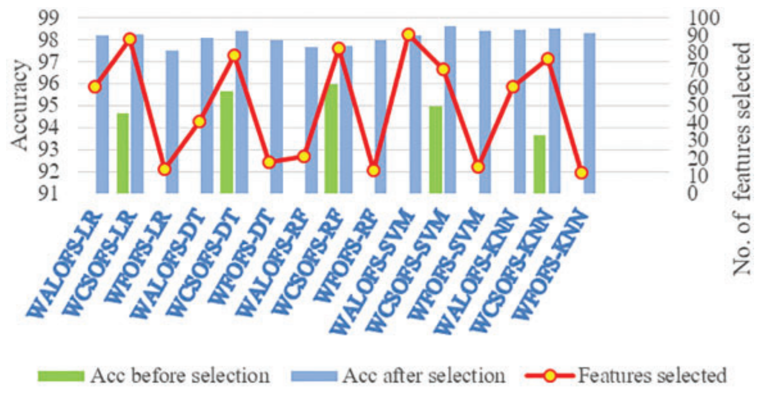

**Figure 5:** Performance comparison of ALO, CSO & FO wrapped with ML classifiers

In experiment no. 2, initially, all the features are passed on to the Auto Encoders to obtain latent space vector, which is passed on to the wrapper feature selection search methods. The FO wrapper search method produced the best accuracy of 98.53% with the KNN classifier. The FO optimizer showcased its supremacy with an 88% reduction in the dimensionality of the feature set. The experiment results are listed in Tab. 2 and its graphical illustration is shown in Fig. 6. Furthermore, in experiment no. 3, the ALO, CSO & FO are wrapped with the proposed artificial neuronal classifier to evaluate the performance of the classifier. Out of all the combinations experimented within in artificial neuronal classifier, the ANN combined with RF achieved a dominant accuracy of 98.87% with 93% reduced feature space. The experimental results are shown in Tab. 3 and its graphical illustration is shown in Fig. 7.

**Table 2:** Accuracy comparison of ALO, CSO & FO wrappers associated with auto-encoders

| S. no | Classifier | Accuracy before feature selection | Feature selection method | Accuracy after feature selection | % Change in accuracy | Features selected | % Decrease in features |
|---|---|---|---|---|---|---|---|
| 1 | LR | 94.6311 | ALO | 98.1426 | 3.51% | 64 | 36% |
| | | | CSO | 98.1198 | 3.49% | 76 | 24% |
| | | | FO | 97.5387 | 2.91% | 10 | 90% |
| 2 | DT | 95.6485 | ALO | 97.6754 | 2.03% | 13 | 87% |
| | | | CSO | 97.7803 | 2.13% | 85 | 15% |
| | | | FO | 97.7552 | 2.11% | 12 | 88% |
| 3 | RF | 95.9985 | ALO | 98.5198 | 2.52% | 73 | 27% |
| | | | CSO | 98.4984 | 2.50% | 76 | 24% |
| | | | FO | 98.5273 | 2.53% | 12 | 88% |
| 4 | SVM | 94.9654 | ALO | 98.3819 | 3.42% | 93 | 7% |
| | | | CSO | 98.4503 | 3.48% | 87 | 13% |
| | | | FO | 97.5387 | 2.57% | 13 | 87% |
| 5 | **KNN** | 93.6521 | ALO | 98.5186 | 4.87% | 58 | 42% |
| | | | CSO | 98.3933 | 4.74% | 73 | 27% |
| | | | **FO** | **98.5314** | **4.88%** | **8** | **92%** |

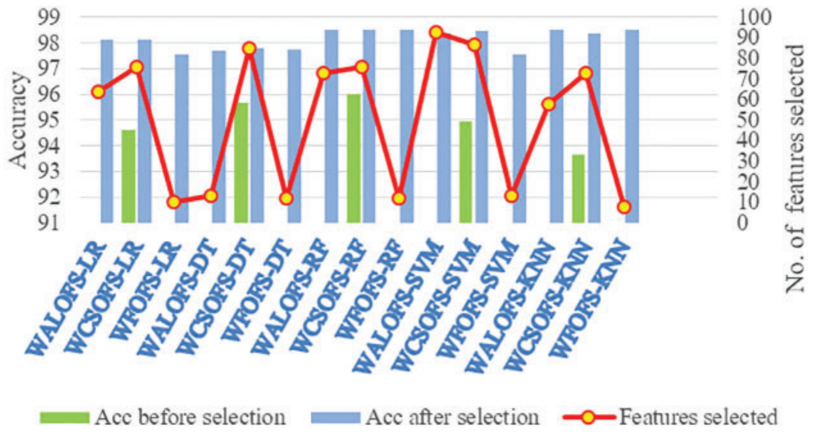

**Figure 6:** Performance comparison of ALO, CSO & FO associated with auto-encoders

**Table 3:** Accuracy comparison of ALO, CSO & FO wrapped with artificial neuronal classifier

| S. no | Classifier | Accuracy before feature selection | Feature selection method | Accuracy after feature selection | % Change in accuracy | Features selected | % Decrease in features |
|---|---|---|---|---|---|---|---|
| 1 | ANC • LR | 94.6311 | ALO | 98.2566 | 3.63% | 76 | 24% |
| | | | CSO | 98.1654 | 3.53% | 89 | 11% |
| | | | FO | 97.5387 | 2.91% | 11 | 89% |
| 2 | ANC • DT | 95.6485 | ALO | 97.6982 | 2.05% | 98 | 2% |
| | | | CSO | 97.7781 | 2.13% | 82 | 18% |
| | | | FO | 97.6412 | 1.99% | 31 | 69% |
| 3 | **ANC • RF** | 95.9985 | ALO | 98.5072 | 2.51% | 47 | 53% |
| | | | CSO | 98.4958 | 2.50% | 78 | 22% |
| | | | **FO** | **98.8728** | **2.87%** | **7** | **93%** |
| 4 | ANC • SVM | 94.9654 | ALO | 98.1654 | 3.20% | 31 | 69% |
| | | | CSO | 98.2654 | 3.30% | 79 | 21% |
| | | | FO | 98.0628 | 3.10% | 28 | 72% |
| 5 | ANC • KNN | 93.6521 | ALO | 98.3477 | 4.70% | 21 | 79% |
| | | | CSO | 98.3249 | 4.67% | 73 | 27% |
| | | | FO | 98.2566 | 4.60% | 10 | 90% |

The experimental results indicate that wrapper-based firefly optimized feature selection algorithm reduced the dimensionality of feature space by maintaining classification accuracy using ANC. The evaluation metrics of the artificial neuronal classifier with the WFOFS algorithm are presented in Fig. 8. The list of 7 features selected by wrapper-based firefly feature selection algorithm using artificial neuronal classifier are listed in Tab. 4.

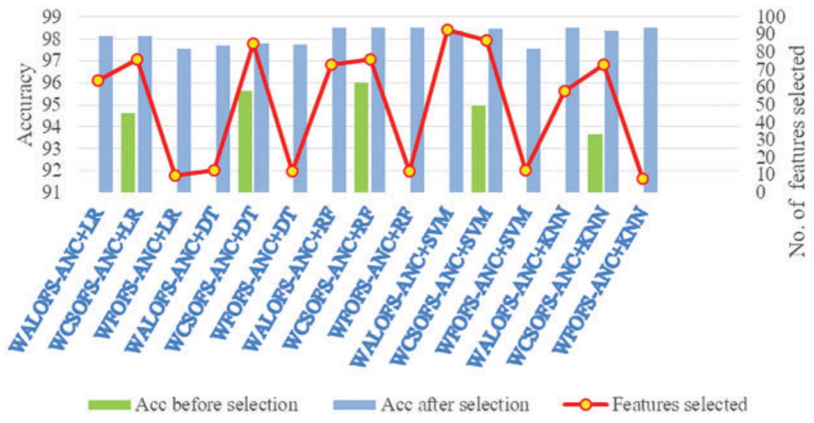

**Figure 7:** Performance comparison of ALO, CSO & FO wrapped with artificial neuronal classifier

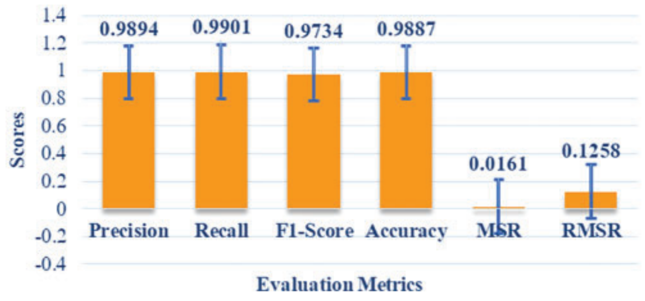

**Figure 8:** Evaluation metrics of WFOFS with artificial neuronal classifier

**Table 4:** APIs selected by WFOFS using ANC

| S. no. | API no. | API description |
|---|---|---|
| 1 | 15 | RegSetValueExW |
| 2 | 23 | HttpOpenRequestA |
| 3 | 40 | IsDebuggerPresent |
| 4 | 55 | CopyFileW |
| 5 | 70 | NtQueryMultipleValueKey |
| 6 | 83 | GetSystemTimeAsFileTime |
| 7 | 98 | CopyFileA |

The Area Under Curve_Receiver Operator Characteristic (AUC_ROC) curves are generated for FO algorithm wrapped with variants of ANC, which performed better than the comparison algorithms. When compared to the area under the complete feature set, the area under the AUC_ROC curve of the ANC classifier embedded with RF is smaller when employed with a reduced feature set. The AUC_ROC graphs for all the variants of the ANC classifier are represented from Fig. 9–13. According to the literature survey, there is no feature selection algorithm using wrapper-based firefly optimization on API call sequence data. Hence, AUC_ROC comparison based on related work on API call sequence data is presented in Tab. 5.

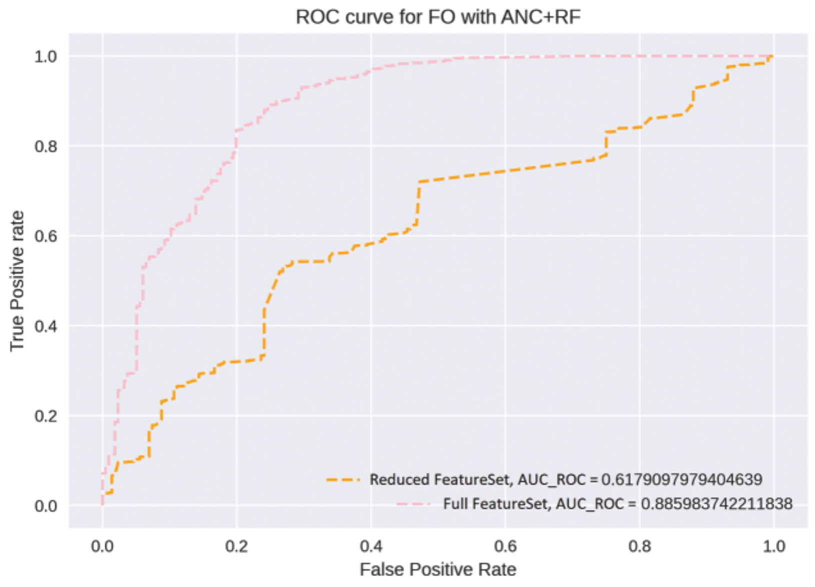

**Figure 9:** AUC_ROC curve of ANC + RF

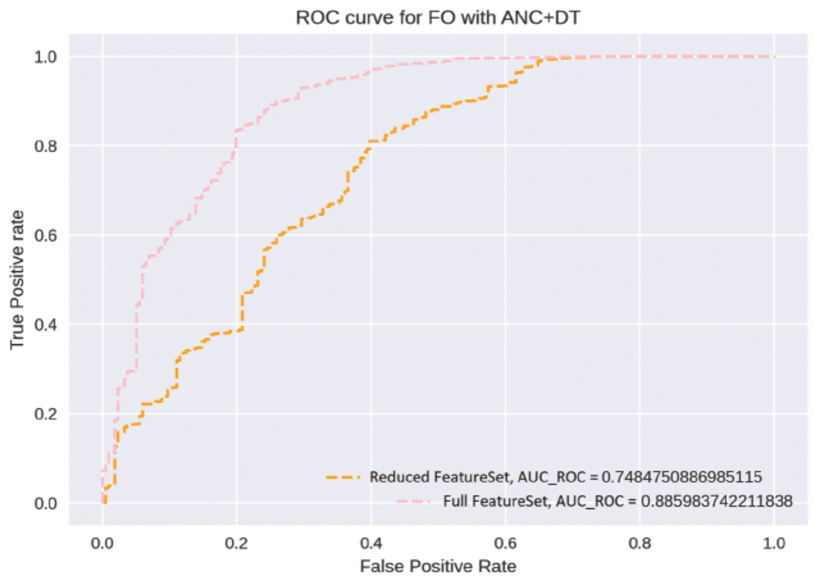

**Figure 10:** AUC_ROC curve of ANC + DT

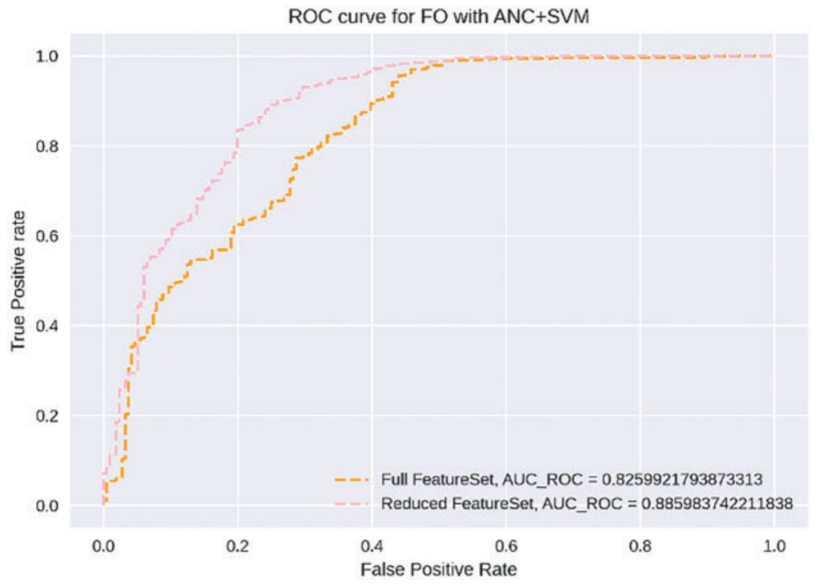

**Figure 11:** AUC_ROC curve of ANC + SVM

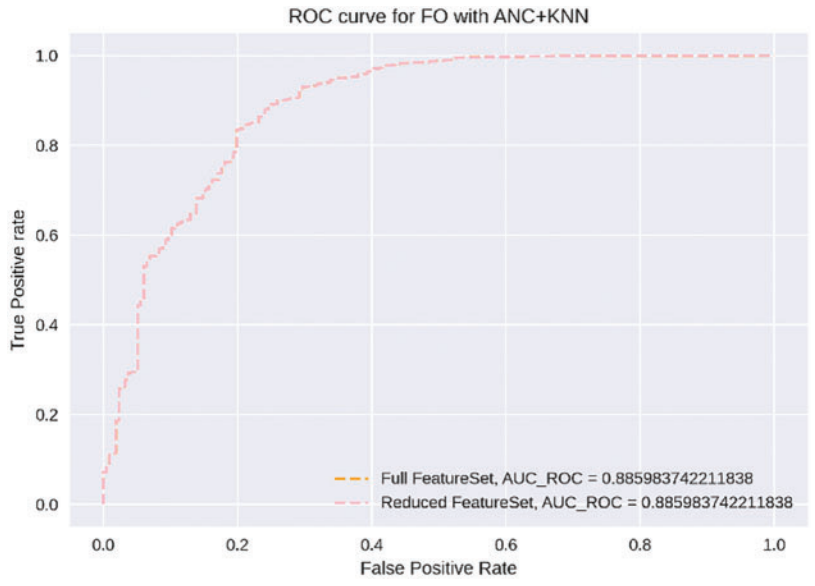

**Figure 12:** AUC_ROC curve of ANC + KNN

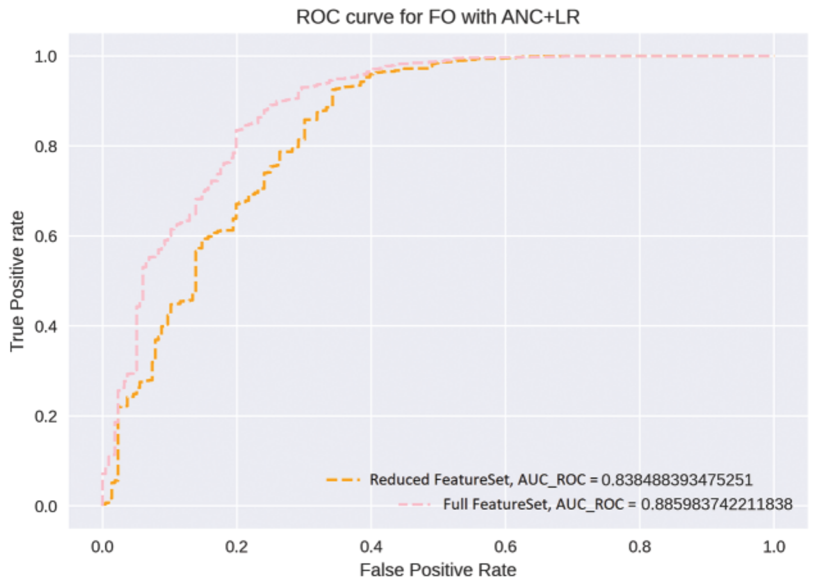

**Figure 13:** AUC_ROC curve of ANC + LR

**Table 5:** Accuracy comparison of related work

| Paper, Year | Classifier | Feature selection | Accuracy |
|---|---|---|---|
| [29], 2019 | Deep graph convolutional neural network (DGCNN) | - | 92.44% |
| [30], 2022 | Neural oblivious decision ensembles (NODE) | - | 90% |
| **This paper** | **Artificial neuronal classifier** | **WFOFS** | **98.87%** |

## 6 Conclusion and Future Work

This work investigates a hybrid dimensionality reduction strategy for feature space using an auto-encoder and swarm optimization. Initially, auto-encoders are given the entire feature set to investigate patterns among the features. To identify the most influential features, the gathered knowledge is fed into wrapper-based feature selection approaches. The machine learning model is subsequently trained to distinguish between good and bad Android applications using the restricted feature set. In addition to focusing on dimensionality reduction, this research proposes an artificial neuronal classifier, which combines artificial neural networks and machine learning approaches.

The wrapper-based feature selection techniques such as WALOFS, WCSOFS & WFOFS reduced the dimensionality of the feature space to a greater extent when incorporated with auto-encoders. Out of ALO, CSO & FO, the FO when wrapped with ML algorithms, the KNN classifier achieved better results in minimizing the size of the feature set to 88% having 98.29% accuracy without auto-encoders and to 92% reduced feature set size with 98.53% accuracy using auto-encoders. The ALO, CSO & FO when embedded with auto-encoders and wrapped with artificial neuronal classifier, the WALOFS outperformed other algorithms in reducing feature set dimensionality to 93% while maintaining an improved classification accuracy of 98.87%.

We will examine the performance of other deep learning models using a variety of feature extraction processes with dynamic datasets in the future. Furthermore, we would like to put classification systems to the test with adversarial threats based on malware imaging techniques. We would like to learn more about how neurons in each layer contribute to the feature extractor process for malware detection.

**Funding Statement:** The authors received no specific funding for this study.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the present study.

## References

[1] Q. Han, V. S. Subrahmanian and Y. Xiong, "Android malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3511–3525, 2020.
[2] A. D. Lorenzoa, F. Martinellib, E. Medveta, F. Mercaldobc and A. Santone, "Visualizing the outcome of dynamic analysis of android malware with VizMal," *Journal of Information Security and Applications*, vol. 50, pp. 1–8, 2020.
[3] J. Xu, Y. Li, R. Deng and K. Xu, "SDAC: A slow-aging solution for android malware detection using semantic distance based API clustering," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1–15, 2020.
[4] A. Mahindru and A. L. Sangal, "A feature selection technique to detect malware from android using machine learning techniques," *Multimedia Tools Applications*, vol. 80, pp. 13271–13323, 2021.
[5] H. Hasan, B. T. Ladani and B. Zamani, "MEGDroid: A model-driven event generation framework for dynamic android malware," *Information and Software Technology*, vol. 135, no. 106569, pp. 1–16, 2021.
[6] X. Liu, X. Du, Q. Lei and K. Liu, "Multifamily classification of android malware with a fuzzy strategy to resist polymorphic familial variants," *IEEE Access*, vol. 8, pp. 156900–156914, 2020.
[7] S. I. Rani and N. M. Sahib, "Detection of malware under android mobile application," in *2020 3rd Int. Conf. on Engineering Technology and its Applications*, Najaf, Iraq, pp. 179–184, 2020.
[8] J. Jiang, S. Li, M. Yu, G. Li, C. Liu *et al.,* "Android malware family classification based on sensitive opcode," in *IEEE Symp. on Computers and Communications (ISCC)*, Barcelona, Spain, pp. 1–7, 2019.
[9] W. Wang, Y. T. Li, T. Zou, X. Wang, J. Y. You *et al.,* "A novel image classification approach via dense-mobile net models," *Mobile Information Systems*, vol. 2020, pp. 1–8, 2020.
[10] N. Daoudi, K. Allix, T. F. Bissyande and J. J. Klein, "Lessons learnt on reproducibility in machine learning based android malware detection," *Empirical Software Engineering*, vol. 74, pp. 1–53, 2021.
[11] Z. H. Qaisar and R. R. Li, "Multimodal information fusion for android malware detection using lazy learning," *Multimedia Tools Applications*, vol. 81, pp. 12077–12091, 2021.
[12] H. Rathore, S. K. Sahay, P. Nikam and M. Sewak, "Robust android malware detection system against adversarial attacks using q-learning," *Information Systems Frontiers*, vol. 23, pp. 867–882, 2021.
[13] D. Tehrany and A. Rasoolzadegan, "A new machine learning-based method for android malware detection on imbalanced dataset," *Multimedia Tools Applications*, vol. 80, pp. 24533–24554, 2021.
[14] V. P. Dharmalingam and P. Visalakshi, "A novel permission ranking system for android malware detection-the permission grader," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 5071–5081, 2021.
[15] O. Yildiz and I. A. Dogru, "A novel permission-based android malware detection system using feature selection based on linear regression," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 2, pp. 245–262, 2019.
[16] N. A. Sarah, F. Y. Rifat, M. S. Hossain and H. S. Narman, "An efficient android malware prediction using ensemble machine learning algorithms," *Procedia Computer Science*, vol. 191, pp. 184–191, 2021.

[17] O. N. Elayan and A. M. Mustafa, "Android malware detection using deep learning," *Procedia Computer Science*, vol. 184, pp. 847–852, 2021.
[18] J. M. Arif, M. F. A. Razak, S. R. T. Mat, S. Awang and N. S. N. Ismail, "Android mobile malware detection using fuzzy AHP," *Journal of Information Security and Applications*, vol. 61, pp. 1–35, 2021.
[19] W. Wang, J. Wei, S. Zhang and X. Luo, "LSCDroid: Malware detection based on local sensitive API invocation sequences," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 174–187, 2019.
[20] H. Gao, S. Cheng and W. Zhang, "GDroid: Android malware detection and classification with graph convolutional network," *Computers & Security*, vol. 106, no. 102264, pp. 1–14, 2021.
[21] A. A. Taha and S. S. J. Malebary, "Hybrid classification of android malware based on fuzzy clustering and the gradient boosting machine," *Neural Computing and Applications*, vol. 33, pp. 6721–6732, 2021.
[22] M. Seyedali, "The ant lion optimizer," *Advances in Engineering Software*, vol. 83, pp. 80–98, 2015.
[23] X. S. Yang and S. Deb, "Cuckoo search via levy flights," in *World Congress on Nature & Biologically Inspired Computing*, Coimbatore, India, pp. 210–214, 2009.
[24] G. Lindfield and J. Penny, *Nature-inspired Optimization Algorithms*. Academic Press, Elsevier, pp. 85–100, 2017.
[25] A. Mahindru and A. L. Sangal, "MLDroid—framework for android malware detection using machine learning techniques," *Neural Computing & Applications*, pp. 5183–5240, 2020.
[26] P. Ravi Kiran Varma, M. S. K. Reddy, K. Santosh Jhansi, and D. Pushpa Latha, "Bat optimization algorithm for wrapper based feature selection and performance improvement of android malware detection," *IET Networks*, vol. 10, no. 3, pp. 131–140, 2021.
[27] M. Kinkead, S. Millar, N. M. Laughlin and P. O. Kane, "Towards explainable CNNs for android malware detection," *Procedia Computer Science*, vol. 184, pp. 959–965, 2021.
[28] A. Ananya, A. Aswathy, T. R. Amal, P. G. Swathy, P. Vinod *et al.,* "SysDroid: A dynamic ML-based android malware analyzer using system call traces," *Cluster Computing*, vol. 23, pp. 2789–2808, 2020.
[29] O. Angelo and S. R. Jose, "Behavioral malware detection using deep graph convolutional neural networks," *International Journal of Computer Applications*, vol. 174, no. 29, pp. 1–8, 2019.
[30] A. Cannarile, V. Dentamaro, S. S. Galantucci, A. A. Iannacone, D. Impedovo *et al.,* "Comparing deep learning and shallow learning techniques for API calls malware prediction: A study," *Applied Sciences*, vol. 12, no. 3, pp. 1–16, 2022.