Article

Metaheuristics with Deep Learning Model for Cybersecurity and Android Malware Detection and Classification

Ashwag Albakri [1], Fatimah Alhayan [2], Nazik Alturki [2,*], Saahirabanu Ahamed [1] and Shermin Shamsudheen [1]

[1] Department of Computer Science, College of Computer Science & Information Technology, Jazan University, Jazan 45142, Saudi Arabia
[2] Department of Information Systems, College of Computer and Information Sciences, Princess Nourah Bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia
* Correspondence: namalturki@pnu.edu.sa

Abstract: Since the development of information systems during the last decade, cybersecurity has become a critical concern for many groups, organizations, and institutions. Malware applications are among the commonly used tools and tactics for perpetrating a cyberattack on Android devices, and it is becoming a challenging task to develop novel ways of identifying them. There are various malware detection models available to strengthen the Android operating system against such attacks. These malware detectors categorize the target applications based on the patterns that exist in the features present in the Android applications. As the analytics data continue to grow, they negatively affect the Android defense mechanisms. Since large numbers of unwanted features create a performance bottleneck for the detection mechanism, feature selection techniques are found to be beneficial. This work presents a Rock Hyrax Swarm Optimization with deep learning-based Android malware detection (RHSODL-AMD) model. The technique presented includes finding the Application Programming Interfaces (API) calls and the most significant permissions, which results in effective discrimination between the good ware and malware applications. Therefore, an RHSO based feature subset selection (RHSO-FS) technique is derived to improve the classification results. In addition, the Adamax optimizer with attention recurrent autoencoder (ARAE) model is employed for Android malware detection. The experimental validation of the RHSODL-AMD technique on the Andro-AutoPsy dataset exhibits its promising performance, with a maximum accuracy of 99.05%.

Keywords: cybersecurity; Android devices; malware detection; deep learning; feature selection; metaheuristics

check for updates

Citation: Albakri, A.; Alhayan, F.; Alturki, N.; Ahamed, S.; Shamsudheen, S. Metaheuristics with Deep Learning Model for Cybersecurity and Android Malware Detection and Classification. *Appl. Sci.* **2023**, *13*, 2172. https://doi.org/10.3390/app13042172

Academic Editor: Christina Thorpe

Received: 10 January 2023
Revised: 3 February 2023
Accepted: 3 February 2023
Published: 8 February 2023

1. Introduction

The explosive increase in the number of mobile devices operating on the Android platform has received significant intention among malware creators, since a massive quantity of private information (e.g., contacts, short messages, and e-mails) is typically stored on these devices. The availability of this information on many mass-market mobile devices renders them a desirable target for malware creators, making the security of mobile devices one of the significant, yet challenging, areas of research. Currently, Android remains one of the most prominent operating systems among mobile devices. An increasing number of devices with the Android operating system are prone to attacks [1]. This is particularly significant as smartphones authenticate critical activities (e.g., e-identity and e-banking) [2]. Several applications may have malware; thus, it is important to protect devices from that malware. The increasing number of mobile devices and their lack of safety have driven application developers to initiate higher levels of protection for the users' devices [3]. Android malware can simply refer to a malicious application that takes sensitive data without the knowledge of the users or executes any action which is not authorized by the user. Simultaneously, Android malware detection mechanisms are continuously developing.

Publications in the field of machine learning (ML) have presented a vast number of mobile malware detection solutions that are related to some kind of ML-driven technique [4]. Conventionally, ML is used in anomaly-based identification techniques. Anomaly-based identification involves two fundamental stages: the detection/testing stage and the training stage. It can be further classified based on the analysis type: hybrid, static, or dynamic. Static analysis can be executed in a non-runtime atmosphere [5] and analyzes the internal structure of the applications. The dynamic analysis adopted the opposite technique, carried out during the normal operation of the applications. By adding more features derived using dynamic analysis to malware detection methods, they can manage better the more challenging and newer types of malware [6]. However, hybrid analysis mechanisms are more complicated, because of the numerous extra elements mandated by the dynamic analysis, such as real or virtual platforms.

Conventional detection techniques have many disadvantages. With the wide applicability of deep learning (DL) in recent times, a method for identifying Android malware utilizing DL technologies has emerged, and the efficiency of malware detection has improved dramatically [7]. DL is a subdivision of machine learning (ML) and Artificial Neural Networks (ANNs). DL is at a high point of development and can be broadly utilized in natural language processing (NLP), computer vision (CV), and speech recognition [8]. The implementation of DL for Android malware detection has now become a trend. A common DL method is utilizing a deep neural network (DNN) that may use several hidden layers of multiple interconnected neurons for data processing. All layers in a DNN contain distinct neurons, each having varied weights and possibly distinct activation functions [9]. If the data were enforced to a neural network (NN), the loss function would compute the prediction error. The most prominent value of DL lies in the abstraction of features and their automatic extraction, abolishing the dreariness of manual feature extraction and automatically finding useful and sophisticated high-order features [10]. By summarizing the DL methods utilized in Android malware detection, it is concluded that the current detection methods are mainly recurrent neural networks (RNNs), deep belief networks (DBNs), deep autoencoders (DAEs), and convolutional neural networks (CNNs).

This work presents a Rock Hyrax Swarm Optimization with deep learning-based Android malware detection (RHSODL-AMD) model. The presented RHSODL-AMD model focuses on the differentiation of Android malware from benign applications and thereby accomplishes cybersecurity. The presented RHSODL-AMD model employs Application Programming Interfaces (API) calls and most significant permissions for effective discrimination between the good ware and malware applications. Moreover, an RHSO-based feature subset selection (RHSO-FS) technique was utilized for selecting a feature set. For Android malware classification, the Adamax optimizer with attention recurrent autoencoder (ARAE) model is employed. The simulation outcome of the RHSODL-AMD method is carried out utilizing a benchmark database.

The rest of the paper is organized as follows. Section 2 provides the related works, and Section 3 offers the proposed RHSODL-AMD model. Then, Section 4 gives the detailed result analysis, and Section 5 concludes the paper.

2. Literature Review

In [11], an Optimizing and effective Ensemble Learning-based Android Malware Detection technique named "OEL-AMD" was presented. This method utilizes Binary Grey Wolf Optimizer (BGWO)-based metaheuristic feature selection (FS) technique. Afterwards, distinct base learners can be trained to utilize hyperparameter tunes for boosting the inductive reasoning ability of the ensemble method for the classifier, and the aggregate efficiency was calculated. Sharma and Agrawal [12] propose a hybrid system for Android malware detection which decreases the dataset dimensionality to reduce resource-intensive computation while maintaining crucial data. The authors examined a new hybrid scheme for Android malware detection dependent upon metaheuristic (improved Intelligent Water Drop Algorithm (IWD)) and DL approaches. The authors [13] developed a new malware detection technique using the DL model [13]. Primarily, android malware database input was attained, and the normalized procedure was performed. The FS was executed together with the optimizer system Recurrent Tuna Swarm Optimizer (TSO).

Alzubi et al. [14] examine and test a new ML technique for Android malware detection. The presented method has collected Harris Hawks Optimization (HHO) and Support Vector Machine (SVM) techniques. Particularly, the HHO technique aims to optimize the SVM technique hyperparameter, but the SVM carries out the classification of the malware depending upon the best-chosen method and creates the optimum solutions for the weighted features. Bhagwat and Gupta [15] proposed a new malware structure in which dynamic features were utilized for android malware detection. When using the presented method, the author's purpose is to choose the correct subset of features to enhance their results. In the presented technique, the metaheuristic FS approach utilized the Gravitational Search Algorithm (GSA) and the Genetic Algorithm (GA), and a correlation was called the Correlated Genetic GSA (CGGSA) was also utilized. The optimizing features can be employed using the XGBoost and AdaBoost techniques for detecting malware. Elkabbash et al. [16] present a new detection method dependent upon the optimizer random vector functional link (RVFL), utilizing artificial Jellyfish Search (JS) optimization and then the dimensional reduction of the Android applications' features. JS was utilized for determining a better configuration of the RVFL for improving the classifier results.

In [17], an ML-based malware detection scheme was presented for distinguishing Android malware in benign applications. At the FS step of the presented malware detection scheme, it can be designed for removing unnecessary features while utilizing a linear regression-based FS system. Accordingly, the dimensionality of the feature vectors was decreased, the training time was decreased, and the classifier method was utilized in real-time malware detection methods. A dynamic detection system termed Artificial Malware-based Detection (AMD) was presented in [18]. The artificial malware pattern was created utilizing an evolutionary (genetic) algorithm. The latter develops a population of API call series to determine novel malware performances next to a group of specific evolution rules. In [19], an FS scheme dependent upon a rough set and improvised particle swarm optimizer (PSO) technique were presented for FS from the permission-based Android malware detection. The most important contribution of this effort is its mention of a novel arbitrary key encoder system that is utilized in the presented method (PSORS-FS) for converting typical PSO techniques in distinct domains. It also decreases the problems based on the maximal particle velocity and the sigmoid function that is connected to the binary PSO.

Dhabal and Gupta [20] presented a novel android malware recognition structure utilizing hybrid DL approaches. When using the presented structure, at primary t, preprocessed stages can be utilized for obtaining an optimized feature set. For malware detection, an optimizing feature-based database was utilized to train the presented hybrid of bi-directional long short-term memory (BiLSTM) and a merged sparse autoencoder (MSAE) with a softmax DL technique. Kim et al. [21] examine a MAPAS, a malware detection scheme that accomplishes a higher accuracy and an adjustable usage of computing resources. The MAPAS examines the performances of malicious applications using API call graphs with an exploiting CNN. Fallah and Bidgoly [22] introduced a model dependent upon LSTM for malware detection that is allowed to not only distinguish between malware and benign instances, but among the malware to identify and detect novel and unseen families. According to our understanding, this is the first time that traffic data were designed as an order of flows and that the sequential-based DL approach was exploited. In [23], the authors establish eight Android malware detection approaches dependent upon ML and DNN that are then examined for their robustness against an adversarial attack. With this intent, the authors generated novel variations of malware utilizing Reinforcement Learning (RL) that is misclassified as benign when using the current Android malware detection approaches.

Though several malware detection models are available in the literature, only a limited number of works have involved both FS and hyperparameter tuning processes. For the design of malware detectors, ML models can be employed to construct classifiers that are effective for discerning whether the application is benign or malware. Training this classifier requires large quantities of labeled datasets, which comprise categorized benignware and malware Android applications, characterized by a collection of features enable to define their behaviors. Even though various solutions have been put forward for the recognition of Android malware, feature selection approaches must be utilized in Android malware detection mechanisms. On the other hand, most of the existing Android malware detectors do not focus on the automated hyperparameter tuning process, which significantly affects their overall classification performance. More specifically, hyperparameters, which include learning rate selection, epoch count, and batch size, play a vital role in accomplishing effective outcomes. As the trial-and-error hyperparameter tuning process seems to be difficult, an automated hyperparameter optimizer needs to be employed. In summary, the design of automated Android malware detectors is yet to be well explored.

3. The Proposed Model

In this work, we have developed a new RHSODL-AMD technique for the effectual recognition of Android malware among benign applications, thereby accomplishing cyber-security. The presented RHSODL-AMD technique involves a series of operations, namely, feature extraction, RHSO-FS-based feature subset selection, ARAE-based classification, and Adamax hyperparameter optimization. Figure 1 shows the overall workflow of the RHSODL-AMD method.

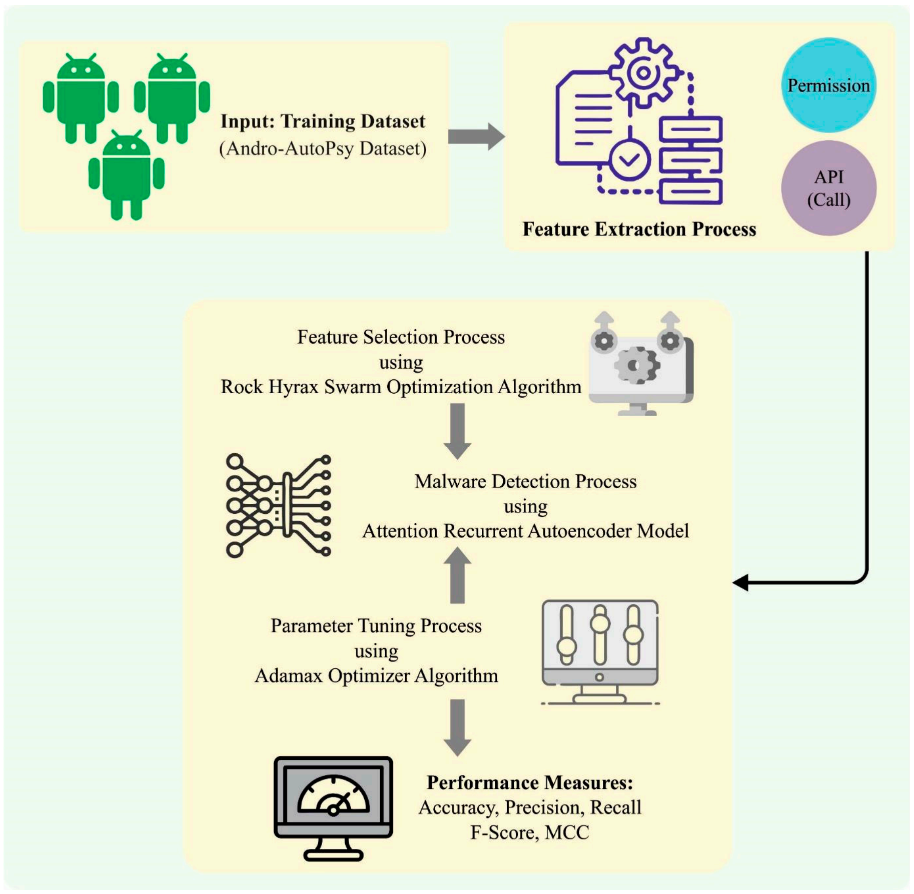
Input: Training Dataset
(Andro-AutoPsy Dataset)
Permission
API
(Call)
Feature Extraction Process
Feature Selection Process
using
Rock Hyrax Swarm Optimization Algorithm
Malware Detection Process
using
Attention Recurrent Autoencoder Model
Parameter Tuning Process
using
Adamax Optimizer Algorithm
Performance Measures:
Accuracy, Precision, Recall
F-Score, MCC

Figure 1. Workflow of the RHSODL-AMD system.

3.1. Feature Extraction

In this work, the presented RHSODL-AMD technique primarily exploited the API calls and most significant permissions, which results in effective discrimination between the good ware and malware applications. A large number of studies have integrated more characteristics to build the learning data and also use an integration of permissions and API calls. As a result, we have determined the features that depend on the abovementioned combinations and utilized them for ML. In particular, we acquired permission information determined in the AndroidManifest.xml files and the API class techniques and exploited it as a feature. The application permission and API class approaches are derived to attain the relevant attributes, involving data on 104 permissions in malicious apps discovered from the training data. Moreover, the study used binary encoding to implement the FS database and encrypted it as 1 once the feature dataset can be identified in the respective attributes, or else as 0. Malware data were encoded utilizing binary encoding, with 0 encoded for benign applications and 1 encoded for malware.

3.2. Algorithmic Procedure of RHSO-FS Technique

To select an optimal set of features, the RHSO-FS method is used. The RHSO technique is a metaheuristic based on the natural behaviors of a rock hyrax swarm [24]. The proposed technique simulates the group behavior of rock hyrax swarms in search of food and their unique method of searching for it. The rock hyrax lives in groups or colonies, with the dominant male closely following the colony to ensure its safety. The proposed model finds a better solution by integrating prior knowledge and local heuristics to construct a better subset of features for optimizing the performance of the classification.

The overall working process RHSOFS system splits the total data into testing and training sets. The training dataset was entered into the optimization method $(f(x))$ for finding the optimal feature. The classification technique is given the optimal set of features $(f(x))$, trained, and given the test dataset for evaluating the performance of the model. Equation (1) is utilized for representing the selection of optimum features. Equation (2) decreases the error in all the iterations using the specified features, which increases the performance of the classification.

The regulatory parameters for population-based algorithms are the initial weighting factors, population size, generation number, probabilities of crossover and mutation, and social and cognitive scaling factors. To accomplish good performance, these parameters needs to be finetuned, and the accuracy of the optimization method can be defined by the fine-tuning parameters; otherwise, the parameter values might result in an optimization technique leading to local optima, which would increase the computation cost of the optimization technique. The RHOSFS method is used for overcoming the aforementioned concern. This model is used for creating the optimum subset of input features for increasing the classification performance.

The presented RHOSFS approach is briefly discussed in the following:

First, select, examine, and generate a random sample of a $(0, 1)$ binary population for the overall amount of input attributes for FS. Generate a feature subset which is equivalent to 1 for all the representations of the input population. For computing the fitness values, the derived optimum input feature is given to the classification algorithm. The study aims at finding an optimal subset of input features which minimize the fitness model while enhancing the performance.

$$err\ (x_i) = actual\_output\ (x_i) - model\_estimated\_output(x_i) \tag{1}$$

$$fitness\ (x) = \frac{\sum_{x=0}^{n} err(x)}{n} \tag{2}$$

$$leader = r_1 \times x(leader_{pos}, j) \tag{3}$$

From the expression, r_1 is a randomly generated number ranging from zero to one, x specifies the preceding location of the leader, $leader_{pos}$ indicates the older location of the leader, and j represents the "each diminution". After updating the position of the leader, every member (or searching agent) updates its locations according to the following equation.

$$x(i,j) = (x(i,j) - (circ \times x(i,j) + leader)) \tag{4}$$

In Equation (4), *circ* indicates the circular movement, which attempts to replicate the circle system as follows:

$$circ = sqrt\left(n_1^2 + n_2^2\right) \tag{5}$$

$$n_1 = r_2 \times cos(ang) \tag{6}$$

$$n_2 = r_2 \times sin(ang) \tag{7}$$

where r_2 indicates the radius and denotes the random integer lies within [0, 1], and ang represents the angle of movement and is an arbitrary integer within [0, 360] in Equations (6) and (7). In each generation, the ang is also upgraded, and it depends on lb and ub, the lower and upper limits of the parameters, correspondingly.

$$dalta = random[lb,\ ub] \tag{8}$$

$$ang = ang + dalta \tag{9}$$

When the output values become greater than 360, or lesser than 0, then the angle (ang) is fixed as 360 or 0 to hold it within a desirable range.

Algorithm 1 describes the pseudocode of the RHSOFS. The RHSOFS initiates by randomly generating the binary population of *the P* agent and examining each feature. Next, it generates a feature subset equivalent to 1 for every instance of the population. This selected attribute is fed into classification models for calculating the fitness values. Equation (1) evaluates the $err(x)$ using the difference between a predicted and an actual value of the models, where $\chi = 1,\ 2, \ldots, n$ and n indicates the number of testing observations. The working process of the RHSO algorithm is shown in Figure 2.

The fitness model can be evaluated by dividing the sum of the errors by the number of observations, as demonstrated in Equation (2). Then, the technique attempted to upgrade the Leader location based on Equation (3), and the location of every search agent based on Equation (4). Next, using Equation (2), every search agent's new fitness can be defined. Based on Equations (6) and (8), these algorithms progress towards the angle updating. The bestX persons hold a minimal fitness value. Then, this model tries to upgrade every searching agent's location based on Equation (4). A new individual is selected as long as the new fitness value was larger than or equivalent to the previous fitness values, and the newest fitness values are reversed. For the next generation, only those with the lowermost fitness values are selected. Lastly, the technique chooses the best candidate. The fitness function (FF) utilized in the RHSO-FS scheme was aimed at taking a balance between the FS count from every result (lesser) and the classifier accuracy (higher) acquired by employing these FSs. Equation (10) exemplifies the FF for the measuring solution.

$$Fitness = \alpha\gamma_R(D) + \beta\frac{|R|}{|C|} \tag{10}$$

However, $\gamma_R(D)$ symbolizes the classifier rate of errors for the provided classifier (ARAE technique employed). $|R|$ denotes the cardinality of the chosen subset and $|C|$ stands for the whole feature count from the database, while α and β demonstrate the two parameters equal to the impact of the classifier quality and subset length. $\in$ [1, 0] and $\beta = 1 - \alpha$.

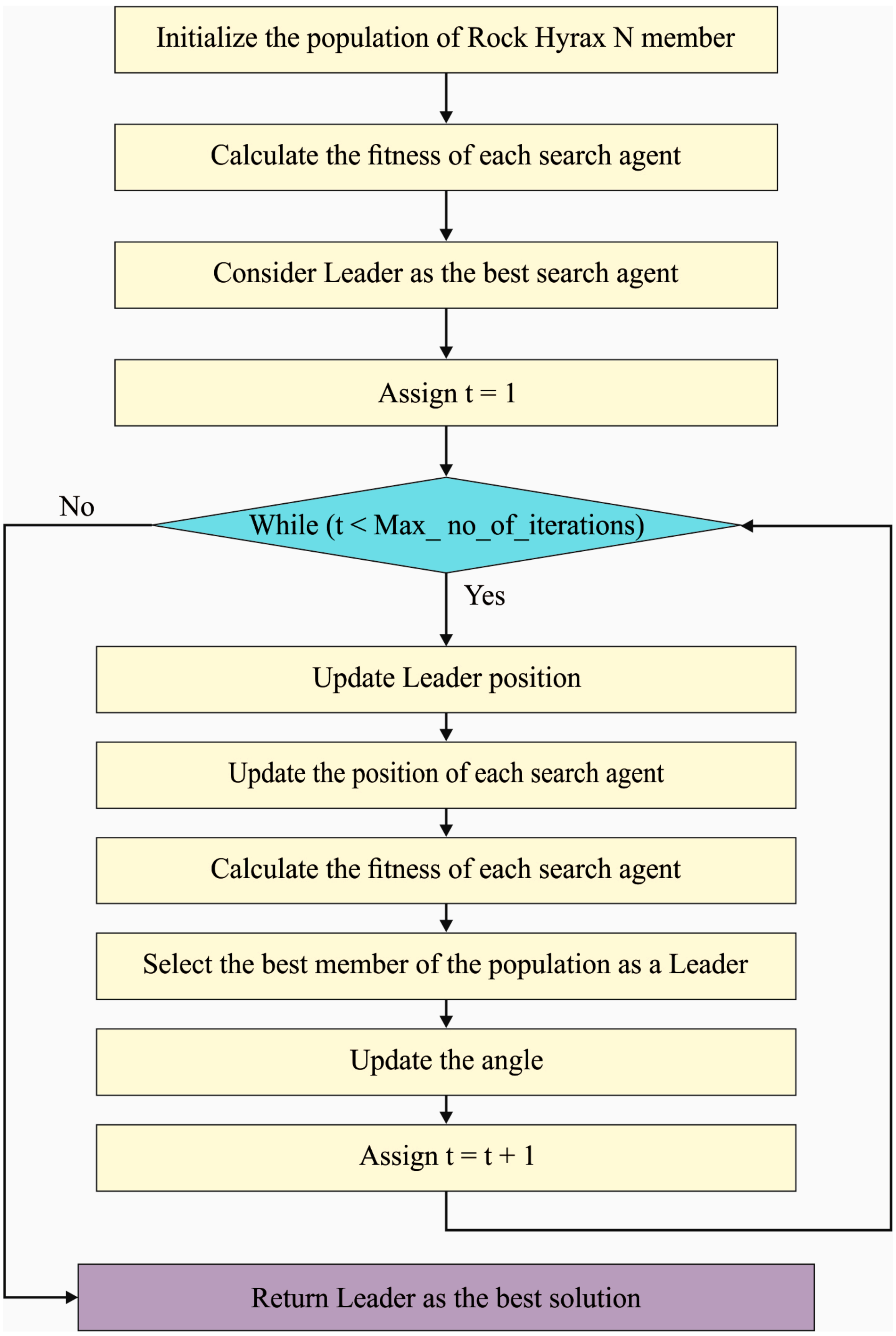
Initialize the population of Rock Hyrax N member
Calculate the fitness of each search agent
Consider Leader as the best search agent
Assign t = 1
While (t < Max_ no_of_iterations)
No
Yes
Update Leader position
Update the position of each search agent
Calculate the fitness of each search agent
Select the best member of the population as a Leader
Update the angle
Assign t = t + 1
Return Leader as the best solution

Figure 2. Flowchart of RHSO algorithm.

Algorithm 1: Proposed RHSO Algorithm

```
Generate a primary population of 0 and 1 of $P$ agents arbitrarily.
Fixed the dimensional of problems, $D = P$, whereas $P$ refers to the count of agents.
Fixed Low to 1 and High to $D$, whereas High and Low signify high and low dimensional, correspondingly.
Make the value of $rl$ and $r2$, in which $rl$ denotes the random number $(0,1)$ and $r2$ denotes the random radius $(0,360)$.
Make testing and training data.
Fixed $max\_$ iter = maximal count of iterations.
Compute all the agents' fitness.
Set Leader = $the$ optimum agents.
Set $t = 1$.
While $(t < max\_iter)$
    for $(i = 1$ to n$)$ do
    Upgrade Leader position.
    Upgrade the position of all the searching agents.
    Compute the Newfitness of all the searching agents.
    Choose the better member of the population $\rightarrow bestX = X(min(fitness))$
    Upgrade the angle.
        If $Newfitness\ (i) <= fitness\ (i)$; then
            Upgrade the position of all the searching agents.
            fitness $(i) = Newfitness\ (i)$.
        end if
    end for
    $t = t + 1$.
end while
Return the optimum agent
```

3.3. Malware Detection Using ARAE Model

To detect and classify Android malware, the ARAE model is exploited. Ma et al. [25] first created a long short-term memory (LSTM) cell by setting a long-time delay among the feedback, input, and gradient burst of the classical RNN module. An underlying concept of LSTM is to include the infrastructure of the cell state, input, forget, and output gates. The concrete calculation process and working principle of gate structure have been discussed in the following.

(1) Forget Gate: Firstly, the LSTM network links h_{t-1} hidden state and x_t input to $[h_{t-1},\ x_t]$. The vector f_t is computed to characterize how much data need to be "forgotten" from the C_{t-1} cell state at $the\ t-1$ time.

$$f_t = \sigma\Big(W_f \cdot [h_{t-1},\ x_t] + b_f\Big) \tag{11}$$

where σ indicates the sigmoid function, W_f denotes the weighted vector of "forget gate" and b_f shows the offset values of the forget gate.

(2) Input Gate: It computes cell state $\widetilde{C}_t$ to be inputted according to the input dataset $[h_{t-1},\ x_t]$ and similarly, calculates the vector to control how much data need to be inputted into the cell state $\widetilde{C}_t$.

$$\widetilde{C}_t = tanh\ (W_C \cdot [h_{t-1},\ x_t] + b_C) \tag{12}$$

$$i_t = \sigma(W_i \cdot [h_{t-1},\ x_t] + b_i) \tag{13}$$

where $\widetilde{C}_t$, W_C, and b_C characterize the value of cell state to be inputted from novel inputs, the upgraded weight of cell state, and cell state bias value, correspondingly. The resultant vector, weight matrices, and bias value of inputted gates are represented as i_t, W, and b_i, correspondingly. Figure 3 depicts the framework of ARAE.

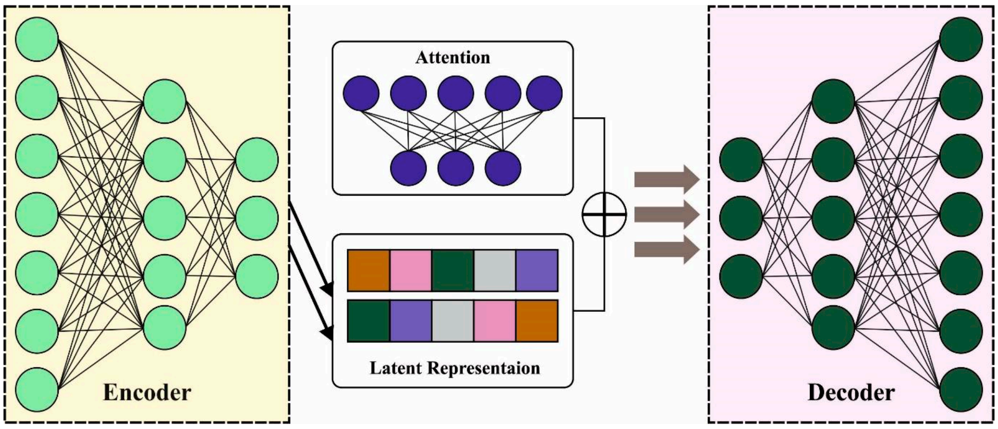

Figure 3. The Architecture of the ARAE model.

(3) Cell Status: The cell state updating can be defined at $t-1$. Afterwards, the computation outcomes of the forget gate f_t, the dataset C_{t-1} that is forgotten are defined [26]. Next, $\widetilde{the\ C_t}$ cell state is produced by the dataset at t time. With the computation result of the input gate, we adjust to define how much data will be inputted, and updating function of the C_t cell state at t time is

$$C_t = f_t \times C_{t-1} + i_t \times \widetilde{C}_t. \tag{14}$$

(4) Output Gate: The output 0_t of the LSTM was computed using the "output gate." Next, the cell state C_t at t time defines what data in the output are eventually transferred, and the results of the last output are h_t.

$$o_t = \sigma(W_o \cdot [h_{t-1},\ x_t] + b_o) \tag{15}$$

$$h_t = 0_t \times tanh\ (C_t) \tag{16}$$

where the output vector, weight vector, and offset values of output gate are represented as 0_t, W_o, and b_o, correspondingly. The resultant values of the LSTM at t time are represented as h_t.

An autoencoder (AE) is an unsupervised technique that encompasses the encode and decode process. Usually, it is exploited for feature extraction or dimension reduction. The LHS of the encode processes learns its inherent feature vector H from the raw input dataset X such that the encoder is denoted as $= f(X)$. The learning technique is understood to be that which reduces the cost function to guarantee minimal error among the raw inputs and the reconstruction dataset:

$$min\ \{L(X,\ g(f(X)))\} \tag{17}$$

In Equation (7), X indicates the input vector, f denotes the encoding procedure, g represents the decoding procedure, and L shows the cost function.

3.4. Hyperparameter Tuning

Finally, the hyperparameter tuning of the ARAE model takes place using the Adamax optimizer. Owing to the continual deepening of the model, the number of parameters of the DL models also increases quickly, which results in model overfitting. At the same time, different hyperparameters have a significant impact on the efficiency of the CNN model. Particularly, hyperparameters such as the epoch count, batch size, and learning rate selection are essential to attain effectual outcomes. Since the trial-and-error method for hyperparameter tuning is a tedious and erroneous process, the Adamax optimizer is used [27]. The Adamax optimizer is an extension of the Adam optimizer [27]. In this study, the infinity norm of the moment can be utilized instead of the second-order moment estimation to upgrade the weight parameter. As a result, the magnitude of the parameter updating has a simple bound in the Adam optimizer when compared to the momentum, and the weight upgrading rule is more stable. The weight update process is the same as the Adam optimizer, except for the subsequent difference: the term $\left(\eta_2/\left(1-\beta_1^k\right)\right)$ characterizes the learning rate with bias-correction for the first-order moment estimation, and uk indicates the infinity norm of the moment. Here, the p^{th} order moment estimation vk can be determined by:

$$vk = \beta_2^p v_{k-1} + \left(1-\beta_2^p\right)\left|g_k\right|_p \tag{18}$$

The iteration formula in (8) is formulated as follows:

$$v_k = \left(1-\beta_2^p\right)\sum_{i=1}^{k}\beta_2^{p(k-i)}\cdot|g_i|^p \tag{19}$$

Let $p\to\infty$, and adopted the description $uk=(v_k)^{1/p}$; then,

$$\begin{aligned} uk &= (\left(1-\beta_2^p\right)\sum_{i=1}^{k}\beta_2^{p(k-i)}\cdot|g_i|^p)^{1/p} \\ &= (1-\beta_2^p)^{1/p}(\sum_{i=1}^{k}\beta_2^{p(k-i)}\cdot|g_i|^p)^{1/p} \\ &= \sum_{i=1}^{k}(\beta_2^{(k-i)}\cdot|g_i|)^p)^{1/p} \\ &= max\left(\beta_2^{k-1}|g_1|,\ \beta_2^{k-2}|g_2|,\ \ldots,\ \beta_2|g_{k-1}|,\ |g_k|\right) \end{aligned} \tag{20}$$

Because $\beta_2\in[0,1]$, $(1-\beta_2^p)^{1/p}$ go to 1 if $p\to\infty$. Equation (20) is derived from the description of the infinity norm and later transformed into the subsequent recursive equation:

$$u = max\ (\beta\cdot u,\ |g_k|) \tag{21}$$

4. Performance Validation

The proposed model is simulated using the Python 3.6.5 tool on PC i5-8600k, GeForce 1050Ti 4GB, 16GB RAM, 250GB SSD, and 1TB HDD. The Android malware detection results of the RHSODL-AMD approach are tested using the Andro-AutoPsy dataset [28], which comprises 9000 benign samples and 13,000 malware samples as represented in Table 1. The Andro-AutoPsy is an anti-malware system which depends upon the similarity matching of malware-centric and malware creator-centric information. It is used for classifying malware samples into similar subgroups by exploiting the profiles extracted from integrated footprints, which are implicitly comparable to different behavioral characteristics. It is useful for benign and malicious applications and for classifying malicious applications into similar behavior groups. For experimental validation, the dataset is split into 70:30 and 80:20 training sets (TRS) and testing sets (TSS).

Table 1. Details of the dataset.

Class	No. of Samples
Benign	9000
Malware	13,000
Total Number of Samples	22,000

The confusion matrices offered by the RHSODL-AMD technique is shown in Figure 4. The results show that the RHSODL-AMD technique accurately identified the benign and malignant samples under varying training set (TRS) and testing set (TSS).

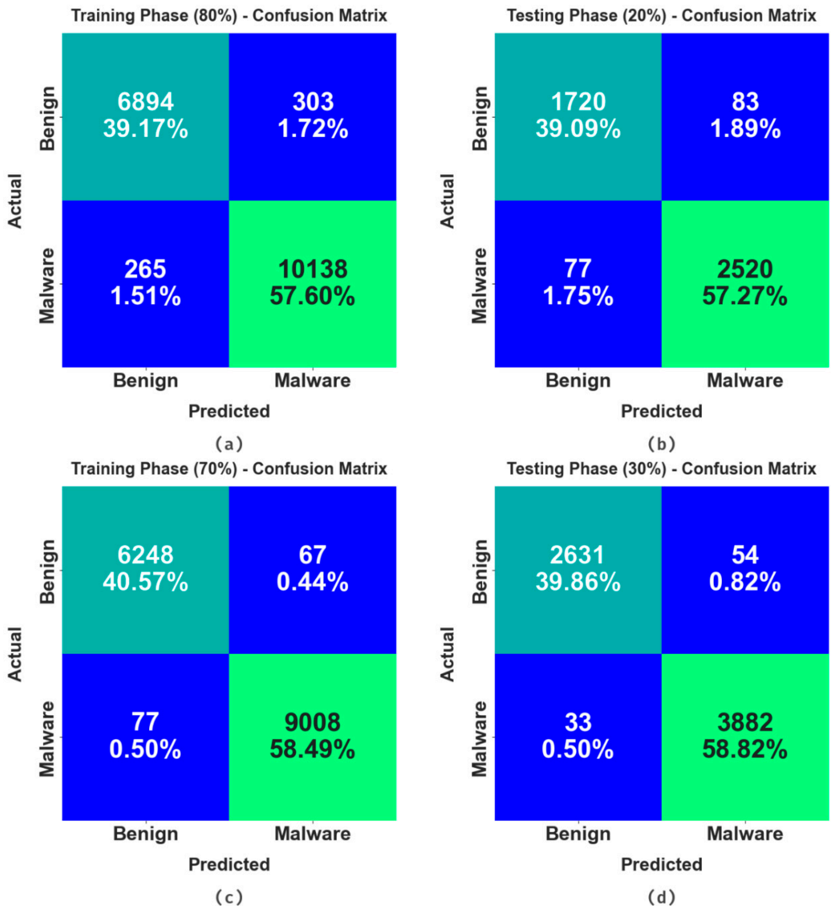

Figure 4. Confusion matrices of the RHSODL-AMD algorithm (**a**,**b**) TRS/TSS of 80:20 and (**c**,**d**) TRS/TSS of 70:30.

In Table 2, the overall Android malware detection results of the RHSODL-AMD technique are stated on 80:20 of TRS and TSS. On 80% of TRS, the results ensured that the RHSODL-AMD technique reaches the effectual classification of benign and malware samples. In addition, it is observed that the RHSODL-AMD technique offers an average $accu_{bal}$ of 96.62%, $prec_n$ of 96.70%, $reca_l$ of 96.62%, F_{score} of 96.66%, and MCC of 93.32%. Followed by, on 20% of TRS, the RHSODL-AMD system offers an average $accu_{bal}$ of 96.22%, $prec_n$ of 96.26%, $reca_l$ of 96.22%, F_{score} of 96.24%, and MCC of 92.48%.

Table 2. Android malware detection outcome of the RHSODL-AMD system on 80:20 of TRS/TSS.

Class	Accuracy$_{\text{bal}}$	Precision	Recall	F-Score	MCC
			Training Phase (80%)		
Benign	95.79	96.30	95.79	96.04	93.32
Malware	97.45	97.10	97.45	97.27	93.32
Average	96.62	96.70	96.62	96.66	93.32
			Testing Phase (20%)		
Benign	95.40	95.72	95.40	95.56	92.48
Malware	97.04	96.81	97.04	96.92	92.48
Average	96.22	96.26	96.22	96.24	92.48

The TACC and VACC of the RHSODL-AMD methodology on 80:20 of TRS/TSS are signified in Figure 5. The figure pointed out that the RHSODL-AMD system has revealed a better performance with higher values of TACC and VACC. It is noticeable that the RHSODL-AMD technique has reached superior TACC outcomes.

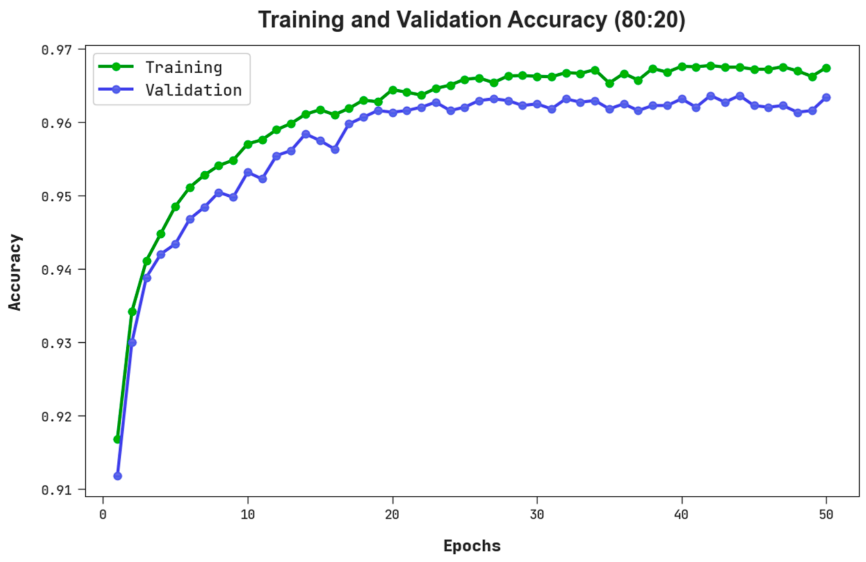

Figure 5. TACC and VACC outcome of RHSODL-AMD system on 80:20 of TRS/TSS.

The TLS and VLS of the RHSODL-AMD algorithm on 80:20 of TRS/TSS are given in Figure 6. The figure stated that the RHSODL-AMD system has exhibited a good performance with minimum values of TLS and VLS. It is observable that the RHSODL-AMD technique has resulted in decreased VLS outcomes.

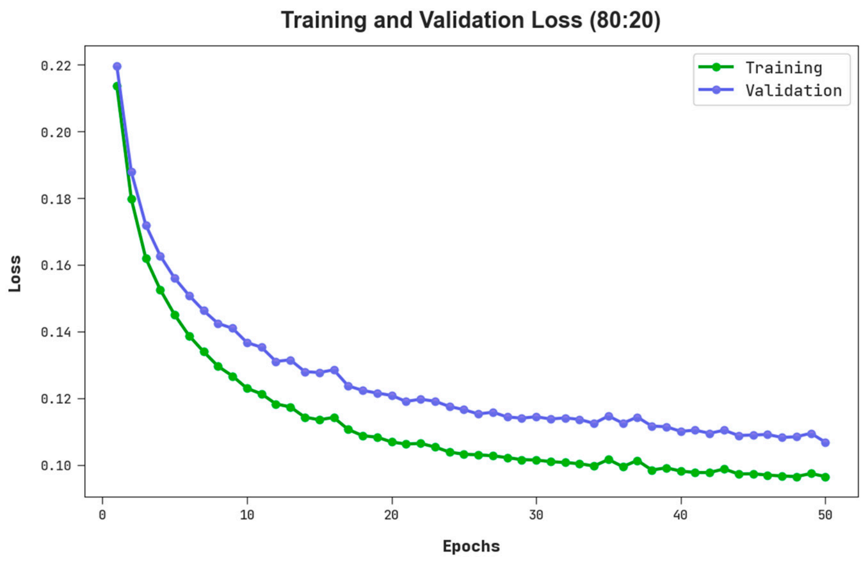

Figure 6. TLS and VLS outcome of RHSODL-AMD system on 80:20 of TRS/TSS.

In Table 3, an overall Android malware detection outcome forthe RHSODL-AMD system is stated at 70:30 of TRS and TSS. The outcome ensured that the RHSODL-AMD technique gains effective classification of benign and malware samples. On 70% of TRS, it is evident that the RHSODL-AMD system provides an average $accu_{bal}$ of 99.05%, $prec_n$ of 99.02%, $reca_l$ of 99.05%, F_{score} of 99.03%, and MCC of 98.07%. Moreover, on 30% of TSS, the RHSODL-AMD algorithm offers an average $accu_{bal}$ of 98.57%, $prec_n$ of 98.69%, $reca_l$ of 98.57%, F_{score} of 98.63%, and MCC of 97.27%.

Table 3. Android malware detection outcome of RHSODL-AMD system at 70:30 of TRS/TSS.

Class	Accuracy$_{bal}$	Precision	Recall	F-Score	MCC
Training Phase (70%)					
Benign	98.94	98.78	98.94	98.86	98.07
Malware	99.15	99.26	99.15	99.21	98.07
Average	99.05	99.02	99.05	99.03	98.07
Testing Phase (30%)					
Benign	97.99	98.76	97.99	98.37	97.27
Malware	99.16	98.63	99.16	98.89	97.27
Average	98.57	98.69	98.57	98.63	97.27

The TACC and VACC of the RHSODL-AMD approach on 70:30 of TRS/TSS have been represented in Figure 7. The figure inferred that the RHSODL-AMD system has exhibited a higher performance with maximum values of TACC and VACC. It is observable that the RHSODL-AMD method has gained higher TACC outcomes.

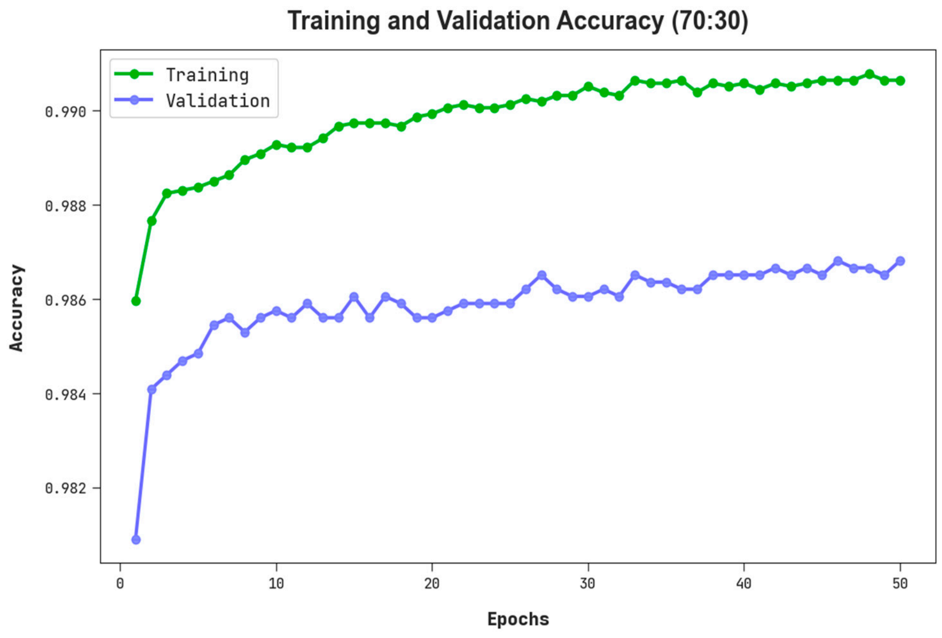

Figure 7. TACC and VACC outcome of RHSODL-AMD system on 70:30 of TRS/TSS.

The TLS and VLS of the RHSODL-AMD approach on 70:30 of TRS/TSS are exhibited in Figure 8. The figure implied that the RHSODL-AMD algorithm has revealed an improved performance with minimum values of TLS and VLS. It is noticeable that the RHSODL-AMD system has resulted in reduced VLS outcomes.

Figure 9 reveals the classifier outcomes of the RHSODL-AMD system under 80:20 and 70:30 of TRS/TSS. Figure 9a,b determines the PR investigation of the RHSODL-AMD method on 80:20 of TRS/TSS. The figures describes that the RHSODL-AMD methodology has gained maximal PR performance in several classes. Finally, Figure 9c,d exemplifies the ROC study of the RHSODL-AMD approach on 70:30 of TRS/TSS. The figure represents that the RHSODL-AMD technique has proficient results with higher ROC values under various classes.

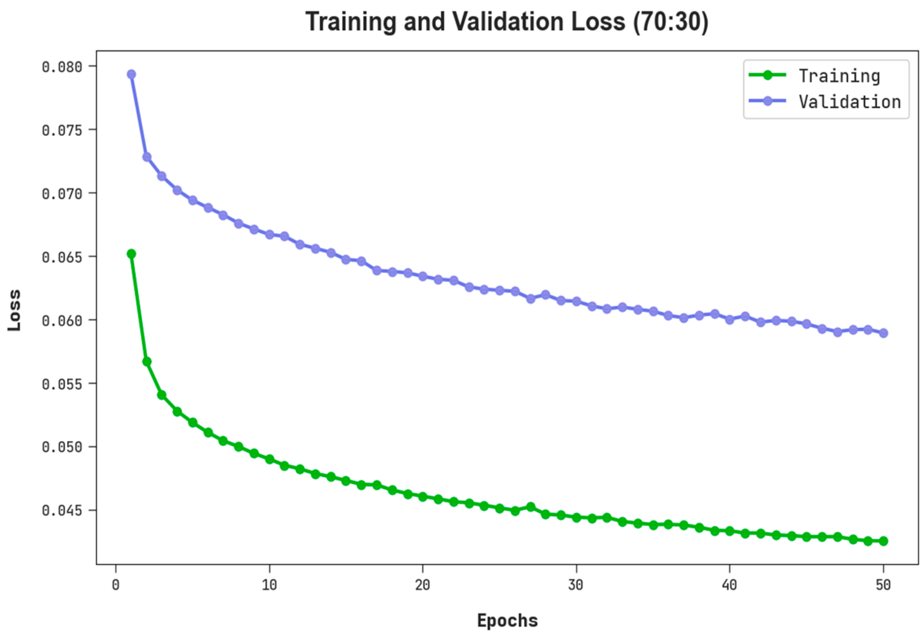

Figure 8. TLS and VLS outcome of RHSODL-AMD system on 70:30 of TRS/TSS.

To demonstrate the better performance of the RHSODL-AMD technique for Android malware classification, a detailed comparison study is represented in Table 4 [29]. The simulation values inferred that the J48, RF, MLP, and AdaBoost-M1 models reported poor classification performances. At the same time, the DBN, LSTM, Decision Table, NB, and SMO models attained moderately closer results. Although the logistic model has reached near optimal outcomes, the RHSODL-AMD technique outperformed the existing models with a maximum $accu_y$ of 99.05%.

Table 4. Comparative analysis of the RHSODL-AMD algorithm with other approaches [29].

Methods	$Accu_y$	$Prec_n$	$Reca_l$	F_{Score}
RHSODL-AMD	99.05	99.02	99.05	99.03
DBN Model	96.81	97.46	96.82	97.99
LSTM Model	96.37	95.45	95.91	97.20
J48 Model	94.85	94.26	94.92	94.09
RF Model	94.93	94.38	94.92	94.69
DecisionTable Model	96.40	96.17	95.95	97.45
NB Model	96.64	97.42	96.74	97.72
MLP Model	95.25	94.66	95.27	95.49
SMO Model	97.05	97.48	96.92	98.26
Logistic Model	98.17	98.27	97.47	98.46
AdaBoost-M1 model	95.88	95.18	95.42	96.82
Ibk Model	96.41	96.89	95.95	97.48

To ensure the computation time (CT) analysis of the RHSODL-AMD technique, a brief comparison study is made in Figure 10. The experimental results indicate that the MLP model has shown a poor performance with increased CT value. Next, the decision table and SMO models reported slightly reduced CT values. Although the remaining models exhibited closer CT values, the RHSODL-AMD technique outperformed the existing models with a minimal CT of 0.06s.

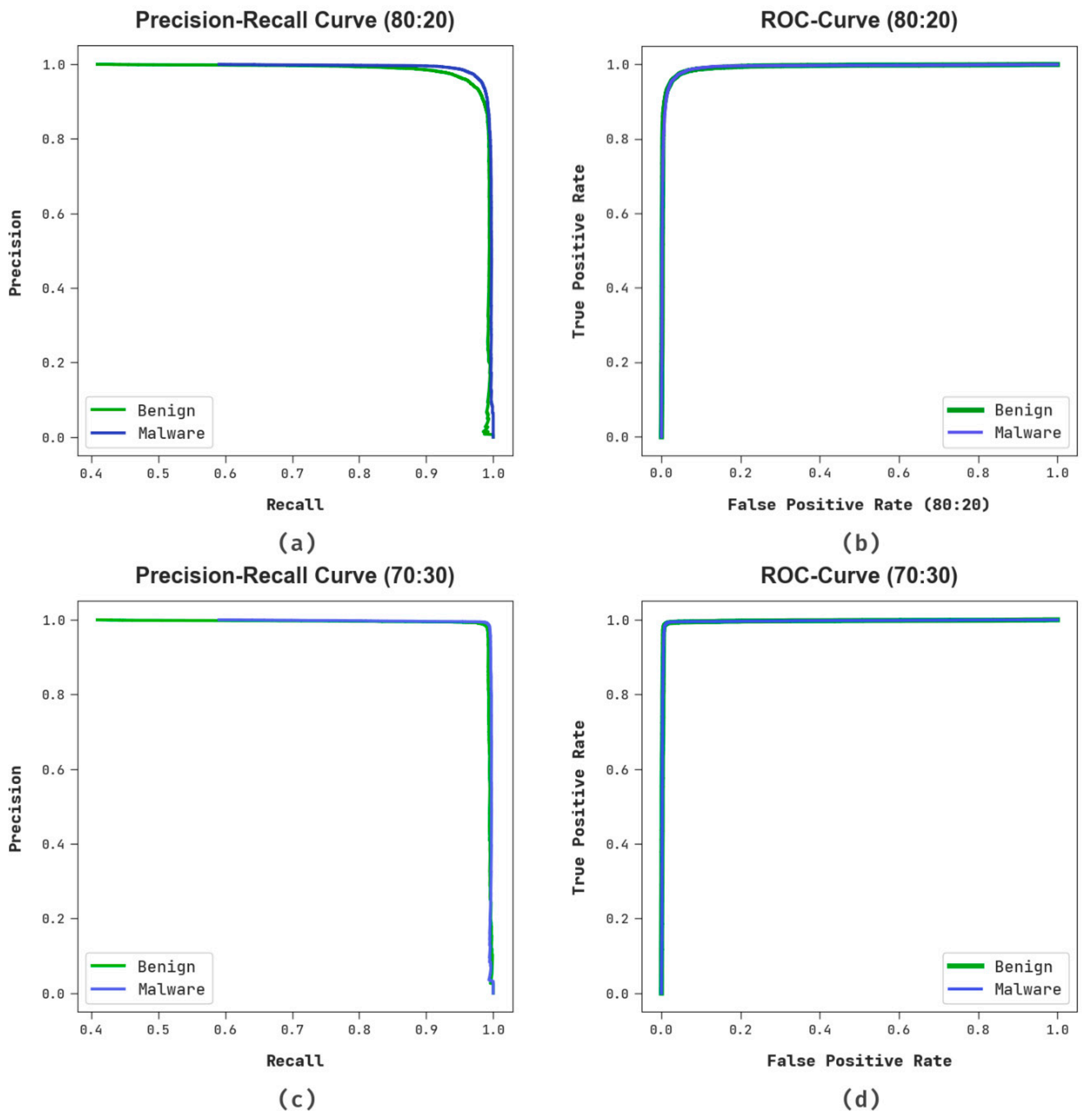

Figure 9. (**a**,**b**) PR and ROC curve at 80:20 of TRS/TSS and (**c**,**d**) PR and ROC curve at 70:30 of TRS/TSS.

The effectual performance of the RHSODL-AMD technique is due to the effectual selection of features and hyperparameters. The enhanced performance of the proposed model is because the lower number of features of the RHSO-FS technique and the Adamax-based hyperparameter optimizer. Therefore, the RHSODL-AMD technique can be employed for accurate Android malware detection, which can protect Android devices from mobile security threats and conceal private information (e.g., contacts, short messages, and e-mails).

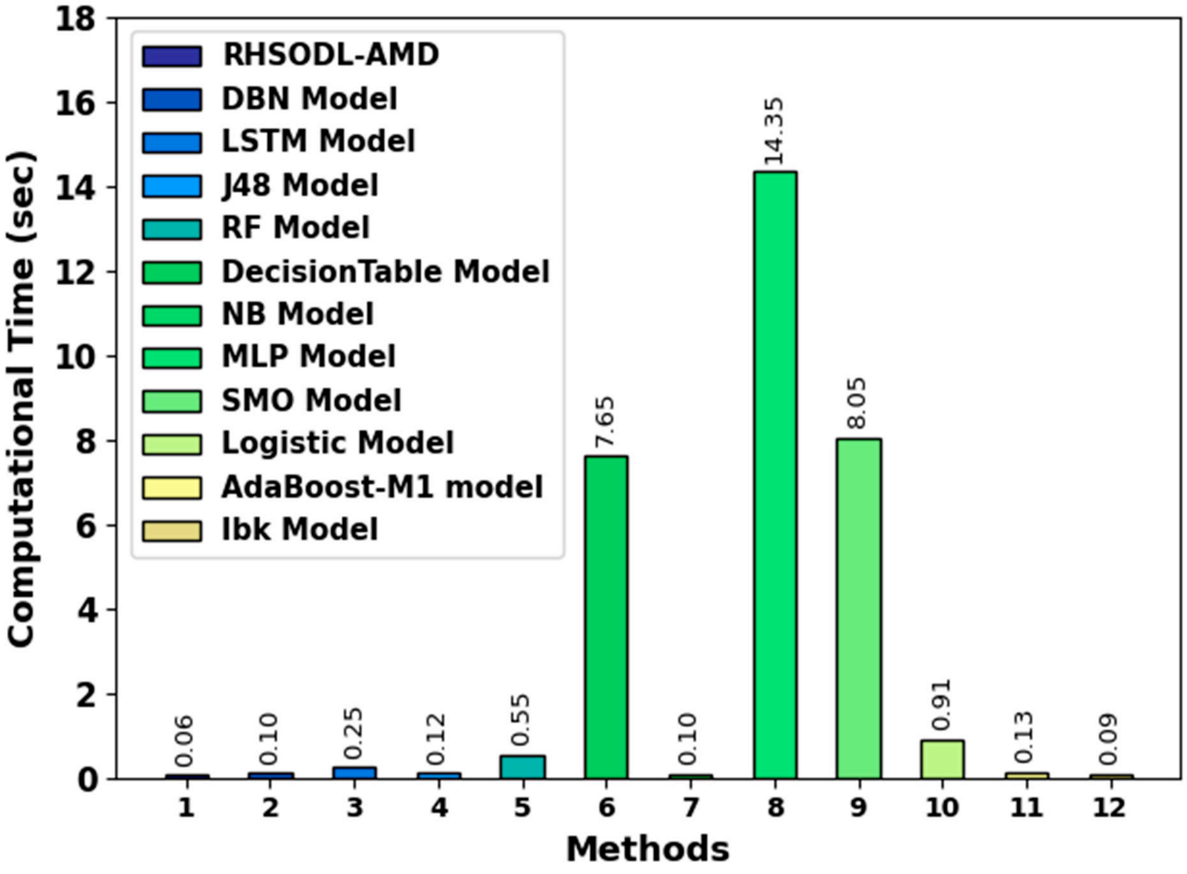

Figure 10. Comparative CT analysis of the RHSODL-AMD.

5. Conclusions

In this research work, we have developed the RHSODL-AMD technique for the effectual classification of Android malware from benign applications, thereby accomplishing cybersecurity. The presented technique primarily exploited the API calls and most significant permissions, which resulted in effective discrimination between the good ware and malware applications. Next, an optimal subset of features can be chosen by the RHSO-FS technique. Finally, the Adamax optimizer with the ARAE model is employed for Android malware detection, which helps to generate more precise and reliable outcomes in the classification of Android applications. The simulation outcome of the RHSODL-AMD approach was carried out utilizing the Andro-AutoPsy dataset. The experimental result demonstrates the improvement of the RHSODL-AMD algorithm over existing approaches, with a maximum accuracy of 99.05%. Thus, the proposed model can be employed for an accurate Android malware classification process. In the future, the performance of the RHSODL-AMD technique can be improved by the use of a metaheuristics-based hyperparameter tuning process. In addition, it is necessary to conduct a validation using a dataset consisting of recently released applications. In addition, the RHSODL-AMD algorithm can be extended to deal with crime data prediction and Ethereum fraud transactions.

Author Contributions: Methodology, A.A.; Formal analysis, S.A. and S.S.; Resources, F.A.; Funding acquisition, N.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R333), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Institutional Review Board Statement: Not applicable. This article does not contain any studies with human participants performed by any of the authors.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable to this article as no datasets were generated during the current study.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A Review of Android Malware Detection Approaches Based on Machine Learning. *IEEE Access* **2020**, *8*, 124579–124607. [CrossRef]
2. Zhao, S.; Li, S.; Qi, L.; Xu, L.D. Computational Intelligence Enabled Cybersecurity for the Internet of Things. *IEEE Trans. Emerg. Top. Comput. Intell.* **2020**, *4*, 666–674. [CrossRef]
3. Dovom, E.M.; Azmoodeh, A.; Dehghantanha, A.; Newton, D.E.; Parizi, R.M.; Karimipour, H. Fuzzy pattern tree for edge malware detection and categorization in IoT. *J. Syst. Archit.* **2019**, *97*, 1–7. [CrossRef]
4. Sicato, J.C.S.; Sharma, P.K.; Loia, V.; Park, J.H. VPNFilter Malware Analysis on Cyber Threat in Smart Home Network. *Applied Sciences* **2019**, *9*, 2763. [CrossRef]
5. Shah, Y.; Sengupta, S. A survey on Classification of Cyber-attacks on IoT and IIoT devices. In Proceedings of the 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 28–31 October 2020; pp. 406–413. [CrossRef]
6. Ali, S.; Bhargava, A.; Saxena, A.; Kumar, P. A Hybrid Marine Predator Sine Cosine Algorithm for Parameter Selec-tion of Hybrid Active Power Filter. *Mathematics* **2023**, *11*, 598. [CrossRef]
7. Aziz, R.M.; Mahto, R.; Goel, K.; Das, A.; Kumar, P.; Saxena, A. Modified Genetic Algorithm with Deep Learning for Fraud Transactions of Ethereum Smart Contract. *Appl. Sci.* **2023**, *13*, 697. [CrossRef]
8. Inayat, U.; Zia, M.F.; Mahmood, S.; Khalid, H.M.; Benbouzid, M. Learning-Based Methods for Cyber Attacks Detection in IoT Systems: A Survey on Methods, Analysis, and Future Prospects. *Electronics* **2022**, *11*, 1502. [CrossRef]
9. Aziz, R.M.; Hussain, A.; Sharma, P.; Kumar, P. Machine learning-based soft computing regression analysis ap-proach for crime data prediction. *Karbala Int. J. Mod. Sci.* **2022**, *8*, 1–19. [CrossRef]
10. Aziz, R.M.; Baluch, M.F.; Patel, S.; Kumar, P. A machine learning based approach to detect the Ethereum fraud transactions with limited attributes. *Karbala Int. J. Mod. Sci.* **2022**, *8*, 139–151. [CrossRef]
11. Smmarwar, S.K.; Gupta, G.P.; Kumar, S.; Kumar, P. An optimized and efficient android malware detection framework for future sustainable computing. *Sustain. Energy Technol. Assess.* **2022**, *54*, 102852. [CrossRef]
12. Sharma, R.M.; Agrawal, C.P. MH-DLdroid: A Meta-Heuristic and Deep Learning-Based Hybrid Approach for Android Malware Detection. *Int. J. Intell. Eng. Syst* **2022**, *15*, 425–435.
13. Jebin Bose, S.; Kalaiselvi, R. An optimal detection of android malware using dynamic attention-based LSTM classifier. *J. Intell. Fuzzy Syst.* **2018**, *34*, 1277–1288. [CrossRef]
14. Alzubi, O.A.; Alzubi, J.A.; Al-Zoubi, A.M.; Hassonah, M.A.; Kose, U. An efficient malware detection approach with feature weighting based on Harris Hawks optimization. *Clust. Comput.* **2022**, *25*, 2369–2387. [CrossRef]
15. Bhagwat, S.; Gupta, G.P. Android Malware Detection Using Hybrid Meta-heuristic Feature Selection and Ensemble Learning Techniques. In *International Conference on Advances in Computing and Data Sciences*; Springer: Cham, Germany, 2022; pp. 145–156.
16. Elkabbash, E.T.; Mostafa, R.R.; Barakat, S.I. Android malware classification based on random vector functional link and artificial Jellyfish Search optimizer. *PLoS ONE* **2021**, *16*, e0260232. [CrossRef]
17. Şahin, D.Ö.; Kural, O.E.; Akleylek, S.; Kılıç, E. A novel permission-based Android malware detection system using feature selection based on linear regression. *Neural Comput. Appl.* **2021**, *29*, 245–262. [CrossRef]
18. Jerbi, M.; Dagdia, Z.C.; Bechikh, S.; Said, L.B. On the use of artificial malicious patterns for android malware detection. *Comput. Secur.* **2020**, *92*, 101743. [CrossRef]
19. Bhattacharya, A.; Goswami, R.T.; Mukherjee, K. A feature selection technique based on rough set and improvised PSO algorithm (PSORS-FS) for permission based detection of Android malwares. *Int. J. Mach. Learn. Cybern.* **2019**, *10*, 1893–1907. [CrossRef]
20. Dhabal, G.; Gupta, G. Towards Design of a Novel Android Malware Detection Framework Using Hybrid Deep Learning Techniques. In *Soft Computing for Security Applications*; Springer: Singapore, 2023; pp. 181–193.
21. Kim, J.; Ban, Y.; Ko, E.; Cho, H.; Yi, J.H. MAPAS: A practical deep learning-based android malware detection system. *Int. J. Inf. Secur.* **2022**, *21*, 725–738. [CrossRef]
22. Fallah, S.; Bidgoly, A.J. Android malware detection using network traffic based on sequential deep learning models. *Softw. Pract. Exp.* **2022**, *52*, 1987–2004. [CrossRef]
23. Rathore, H.; Sahay, S.K.; Nikam, P.; Sewak, M. Robust android malware detection system against adversarial attacks using q-learning. *Inf. Syst. Front.* **2021**, *23*, 867–882. [CrossRef]
24. Padhi, B.K.; Chakravarty, S.; Naik, B.; Pattanayak, R.M.; Das, H. RHSOFS: Feature Selection Using the Rock Hyrax Swarm Optimization Algorithm for Credit Card Fraud Detection System. *Sensors* **2022**, *22*, 9321. [CrossRef] [PubMed]
25. Ma, X.; Tao, Z.; Wang, Y.; Yu, H.; Wang, Y. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transp. Res. C, Emerg. Technol.* **2015**, *54*, 187–197. [CrossRef]

26. Kong, X.; Li, X.; Zhou, Q.; Hu, Z.; Shi, C. Attention recurrent autoencoder hybrid model for early fault diagnosis of rotating machinery. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 1–10. [CrossRef]
27. Xiao, B.; Liu, Y.; Xiao, B. Accurate state-of-charge estimation approach for lithium-ion batteries by gated recurrent unit with ensemble optimizer. *IEEE Access* **2019**, *7*, 54192–54202. [CrossRef]
28. Jang, J.W.; Kang, H.; Woo, J.; Mohaisen, A.; Kim, H.K. Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information. *Digit. Investig.* **2015**, *14*, 17–35. [CrossRef]
29. Lee, J.; Jang, H.; Ha, S.; Yoon, Y. Android Malware Detection Using Machine Learning with Feature Selection Based on the Genetic Algorithm. *Mathematics* **2021**, *9*, 2813. [CrossRef]