SLUJST
Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

An Android Malware Detection System Based on Hybrid Artificial Neural Network and Decision Tree Algorithm

Hambali, Moshood Abiola[1*] and Oforjetu Chukwudi Peter[2]
Department of Computer Science, Federal University Wukari, Nigeria.
[1] hambali@fuwukari.edu.ng; [2] chukwudioyoko@gmail.com
*Corresponding Author: hambali@fuwukari.edu.ng

Abstract

***Background**: The widespread adoption of cloud technology has brought significant benefits in terms of information exchange. However, this progress has also introduced opportunities for malicious actors, to exploit the cloud for illicit purposes, including the theft of personal data. In response to the malware threat. **Aim:** This study explores the application of machine learning, specifically Artificial Neural Networks (ANN) and Decision Tree (DT) algorithms to detect and classified Android malware. **Method**: A stacking technique is employed to combine ANN and DT algorithms and hence, create a hybrid model. The ClaMP and Android Network Traffic datasets from Kaggle are utilized for the analysis. Model performance is measured in several ways, but accuracy is the major indication, and features are selected using information gain. **Results**: The results indicate that the hybrid model achieved the highest accuracy scores of 96% and 99% on the ClaMP malware and Android network traffic datasets, respectively. Additionally, comparisons are made between the best model's performance and that of cutting-edge algorithms, demonstrating that the hybrid model outperforms them with an impressive accuracy score of 99%. This study highlights the effectiveness of combining machine learning techniques to address malware-related challenges and enhance data security in cloud-based systems.*

***Keywords:** Android malware, Malware Detection, ANN, Decision tree, Hybrid*

1. Introduction

The growth of Android smart devices has increasingly empowered internet usage for the past decades as the amount of private, sensitive, and otherwise crucial information travelling across the worldwide network has increased spectacularly (Kim et al., 2022). The proliferation of smartphone app shops like Google Play and Apple Store has stoked interest in mobile internet technologies. Although this evolution facilitates and supports human day-to-day lifestyles and activities, it also has its drawbacks and thus suffers security and privacy issues via malicious attacks (Wazid et al., 2019).

The Android operating system is a mobile platform that has been made available as open-source software, using the structural framework of the Linux kernel (Arslan et al., 2019). Currently, the android operating system has the highest share and sales volumes. With the increasing usage of android mobile devices, users store their personal information and critical information on their mobile device, and these devices have been a target for malicious application developers via app store download which requires permissions or internet usage (Arslan et al., 2019). Thus, some of these applications are becoming more suspicious as they keep demanding some specially designated unnecessary permissions. For this reason, the security mechanisms provided for the use of permission in application pools can easily be overcome making the system vulnerable to malicious attacks.

Malware referred to as malicious software is recognized as one of the greatest threats against modern computer systems that currently underpin much of societal activities (Afianian et al., 2019). Recent years have seen many high-profile cyber-attacks on global corporations, which consequentially have resulted in financial, operational, and reputational damage (Chen & Bridges, 2017). Because of developments in anonymous payment mechanisms like

© The Author(s), under exclusive license to SLUJST 2024

Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Bitcoin, ransomware most notably has become a prevalent malware strain, this implies that individual files can be encrypted and hence held for a ransom fee until payment is made to acquire the decryption key. Such malicious software further refrains authorized users from information access by masquerading as a legitimate application. These attacks hamper users' confidentiality, data integrity, authenticity, and data availability, and hence other resources of the system. Consequentially, sensitive information meant to be private may be leaked, changed, or unavailable even to approved individuals.

Additionally, most malicious attacks attempt to intrude on a network, device, or system and thus have access to important information that can greatly affect the operation of the systems and disrupt data confidentiality due to the security gaps in the systems (Daniel et al., 2020). Thus, these attacks are becoming more complex and advanced. Therefore, the ever-increasing diversity and attempts of malicious attacks have led to the essential need of providing efficient and effective techniques to harness these attacks.

There have been many attempts by researchers to come up with malware detection methods capable of detecting attacks efficiently and effectively, few of which are based on rule-based and pattern-matching techniques while recently the majority of these malware detection methods presented in several kinds of literature have focused on malware detection using artificial intelligence (deep learning) (Daniel et al., 2020; Li et al., 2022; Nisa et al., 2020; Yeo et al., 2018) methods and various machine learning methods (supervised, unsupervised and semi-supervised learning methods) (Lu et al., 2013; Mahindru & Singh, 2017; Mathur et al., 2021) to automatically recognize normal and abnormal events happening in the systems and networks.

Nevertheless, despite the diligent endeavours of various antivirus vendors and researchers to combat the proliferation of malware and its subsequent expansion, the prevalence of android malware attacks persists as a pervasive issue, resulting in substantial economic and intellectual property detriment. Research conducted revealed that the prevailing advanced malware opposition software utilized by various organizations via machine learning and deep learning methods for malware analysis and detection is facing new challenges due to the evolution of malware software operations (Xie et al., 2023). Several researchers have been proposing a new solution to effectively and efficiently defend against such revolving and blackmailing malware techniques (Dixit & Silakari, 2021). Although several researchers have developed systems to protect against malware, the degree of such software reliabilities to malware defence still requires optimal and guaranteed solutions against malware attacks (Xie et al., 2023). As a result, this study proposed a more potent, effective, and efficient approach to the detection of the revolving malware techniques using ensemble of Decision Tree (DT) and the Artificial Neural Network (ANN) with feature selection using information gain.

2. Review of Related Works

Lu et al. (2013) utilized and combined the two types of permissions controls on android devices. These permissions are the required permission and requested permission. Using this permission to identify malicious software, they built a two-tiered system and used machine learning techniques. The authors analysed a total of 28,548 permission combinations, including 1,536 malicious apps and permission pairs, from the combined permission controls. The authors were able to strike a compromise between the detection efficiency and the detection speed of their proposed classifier thanks to the two-layered technique. The researchers conducted two phases of their experiment, with the first using the requested permissions and the J48 Decision Tree method for detection, and the second using the requested permit pairs and the J48 Decision Tree technique for detection. In case of discrepancy between the two sets of findings, the dataset was reclassified using the previously employed permission pairs and J48. The authors

© The Author(s), under exclusive license to SLUJST 2024

Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

asserted success with the proposed method and suggested granting access to individual components rather than whole applications in order to spot fraudulent behaviour more effectively.

Using the Application Programming Interface (API) class and machine learning techniques including the random forest method and the support vector machine, Rosmansyah & Dabarsyah (2015) suggested a malware detection system for mobile devices. Based on the permissions provided in the used-permission label of the manifest.xml file, the authors analysed API calls from 205 safe and 207 dangerous application files. By deducing from the author's tests, the authors learn that telephone Manager and connection Manager are the most sought-after capabilities by 97% of malware requests. When using cross-validation as the feature selection technique and SVM obtaining 91.4% with percentage split as the feature selection algorithm, the authors of this study found that using the random forest classification algorithm on the extracted dataset yielded a 92.4% accuracy rate.

Malware analysis for Android devices was examined by Sheen et al. (2015). A flexible recognition module based on multi-feature collaborative decision fusion (MCDF) was developed by the team. To improve the model's performance, the authors took into account the unique characteristics of a malicious record, such as consent-based features and API call-based features, with the intention of providing a better discovery by combining their choices using a collective approach based on the likelihood hypothesis and hybridising classifiers (J48, Support Vector Machine, Naive Bayes). The performance of the model was measured against a dataset (2000 in total) consisting of Android-based malware, from Google Play and android Malware services. The outcome showed that the proposed method offers a higher execution accuracy of 98.91 percent.

To bridge the gap between static and dynamic analysis, Yuan et al. (2016) introduced a deep learning approach to Android app analysis. In addition, they developed a Droid-Detector, an Android malware detection engine built on the deep learning technique (Deep Belief Network). As a result, their malware detection technology can tell whether a file is harmful or not. Droid-Detector was evaluated using data from the "Google Play store" and the android Malware genome project (a combined total of 1860 apps) to ensure its efficacy with a large sample of Android apps. The researchers then dove headfirst into the challenges the Deep Belief Network has when trying to accurately depict malware. With the availability of extra preparation data, the system was able to reach a detection accuracy of 96.76%, proving that deep learning approaches are suitable for detection of Android malware.

Nikolopoulos & Polenakis (2017) introduced a graph-based approach that identifies malicious software as one of a set of predefined malware families based on the relationships between system calls obtained from an unknown software sample. Clients employed the System-call Dependency Graphs (ScD-graphs) that were gathered from dynamic taint investigation traces with great accuracy. ScD-graph guarantees the capacity to assemble disparate subsets of its vertices, therefore the authors used recognition and arrangement systems on a weighted coordinated network (in this case, a Group Relation Graph; Gr-graph) to make their model robust against drastic alterations. In order to improve their model's discoverability, the authors offered the Delta-comparability metric, and they provided the SaMe-similitude and NP-similarity metrics that make up the SaMe-NP closeness for the classification method. Finally, they measured the detection rates and classification accuracy of their model and found that it performed well against malicious software, with an accuracy of 95.9% and a high dependency analysis for calls, demonstrating its potential.

Using an unidentified packed executable, Bat-Erdene et al. (2017) developed a technique characterised by several packing algorithms. The executable's entropy estimates were tallied, and the entropy estimates of a region of memory were converted into more familiar forms. Using controlled learning order methodologies, such as credulous

© The Author(s), under exclusive license to SLUJST 2024

Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Bayes and support vector machines for recognising urgent computations, they ordered the delivery of pictures using symbolic aggregate approximation (SAX), which is known to be effective for big information changes. They used 324 rushed charitable initiatives and 326 crammed malicious software with 19 hurried computations. The authors found that their method has a high precision of 95.35%, a recall of 95.83%, and an accuracy of 94.13% when identifying pressing computations of a particular executable.

Tong & Yan (2017) developed a hybrid strategy for detecting malware on Android mobile devices. The authors used a dynamic method to gathering data on runtime system calls across a sample of well-known malicious and benign applications. Specifically, they modified the Android OS source code to collect system-calling data in real time. In addition, they sorted through the data to find similarities between malicious and benign system calls or sequences of system calls, and then used those patterns to build malicious and benign patterns. That is, malicious and normal patterns are generated by calculating the ratio of the average frequency of a sequential system call in the set of malware to the average frequency of the same sequential system calls in the set of benign apps. The authors claim that their method of detection has a 90% success rate.

Mahindru & Singh (2017) used machine learning to investigate Android Malware detection methods that relied on dynamic permissions. The authors found that owing to Android apps' widespread use and open architecture, they were easy targets for malware. The writers used a two-pronged strategy to actualizing their vision. They created a new dataset for detecting malicious Android applications by extracting a set of 123 dynamic permissions granted by 11000 Android applications and testing a number of machine learning classification techniques on it, including Naive Bayes (NB), Random Forest (RF), J48, Simple Logistic (SL), Decision Trees (DT), and K-star. According to the results of their experiments, SL outperforms RF and J48 by a little margin when it comes to malware categorization.

D'Angelo et al. (2020) created an Autoencoders and API-Images-based malware detection solution for mobile platforms. The dataset for the study is the API-pictures, which are sparse matrices in the shape of pictures that record the sequences of API calls called by applications throughout their execution and may be thought of as fingerprints of the apps' activity over time. The scientists used a neural network trained with artificial intelligence to determine whether malware was benign or malicious based on properties extracted from these matrices using the sparse autoencoder technique. In addition to outperforming more complex and sophisticated machine learning (J48, Nave Bayes) approaches in malware classification with an outstanding accuracy rate of 0.94%, the authors found that performing features extraction using autoencoder greatly enhanced the model's effectiveness in detecting malware.

Ren et al. (2020) developed two end-to-end Android malware detection methods based on deep learning. In their method, the authors resample the raw bytecodes of the classes.dex files of Android applications and then preprocess them as inputs to the two deep learning models namely the Dalvik Executable-Convolution Neural Network (Dex-CNN) and the Dalvik Executable -Convolution Recurrent Neural Network (Dex-CRNN). The models were trained and evaluated with a dataset containing 8000 benign applications and 8000 malicious applications collected from the Google play store. The result of the author's experiments revealed that the models achieved 93.4% and 95.8% detection accuracy for Dex-CNN and Dex-CRNN respectively. They further compared their models with some existing methods and identified that their models are not limited by input file size, no manual feature engineering, and low resource consumption and thus concluded that their methods are more suitable for application on Android IoT devices with an added advantage of the end-to-end learning process.

© The Author(s), under exclusive license to SLUJST 2024

Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

In an effort to identify malware using a neural network, Hossain et al. (2021) compared the efficacy of three distinct neural networks. Convolutional Neural Networks (CNNs), Long Short Term Memory Networks (LSTMs), and Gated Recurrent Units (GRUs) were some of the strategies used by the writers to identify malware. Malware samples from virusshare.com, portableapps.com, and the Windows 7 x 86 folders were utilised to create this work. When comparing the models used by the authors, CNN's 83% accuracy in malware recognition was the highest, followed by LSTM's 66% and GRU's 76%.

Malware detection using Autoencoder and deep learning was suggested by Xing et al. (2022). To create their dataset, the authors took malicious apps from VirusShare and benign ones from the Google Play store. An autoencoder network and a grey-scale visual representation of malware were combined to create this model. Using the autoencoder's dimensionality reduction characteristics, the authors were able to classify malicious from safe software. Their analysis of the grey-scale picture technique was based on the autoencoder's reconstruction error. The authors claimed that their suggested detection model has a 96% accuracy rate and a 96% stable F-score. The Summary of literature review were presented in Table 1.

Damodaran et al., (2017) highlighted the limitations of signature-based antivirus programs in detecting mobile viruses and the need for advanced detection methods. Ren et al., (2020) demonstrated the effectiveness of deep learning methods in Android malware detection. Rosmansyah and Dabarsyah (2015) applied machine learning algorithms for smartphone malware detection thus emphasizing the importance of feature selection. Xing et al., (2022) utilized autoencoders and deep learning for malware detection while indicating the potential of deep learning in the domain of malware detection. Hossain et al., (2021) compared different neural networks for malware detection, highlighting the relevance of deep learning approaches. However, the deep learning method takes a lot of time to train and retrain before performing effectively (Yuan et al., 2016). To prevent such defects with the deep learning method, this study proposed using an information gain method to autonomously extract relevant features to reduce the overhead of training and retraining a model before passing the relevant extracted features to an artificial neural network with backpropagation techniques. This is because the effectiveness of a classification model strongly depends on the choice of selecting the right features D'Angelo et al., 2020). Ensembles generally achieve higher predictive performance than individual models by leveraging the strengths of multiple learning algorithms. Also, ANNs can smooth out noise due to their ability to learn complex functions, while Decision Trees can handle outliers through clear decision rules.

3. Method and Materials

For feasible and effective implementation of a solution to any machine learning task, it is of great significance to adopt a methodology for the problem in concern. The idea of adopting a methodology is to provide a framework for the effective and efficient implementation of the proposed problem and in conclusion, results in the best feasible result. Hence, the methodology for this study encompasses three basic approaches namely the data preprocessing, model application, and performance evaluation stage.

The first stage, which is data preprocessing, entails the removal of unwanted characters and symbols inherently present in the acquired dataset, these data include missing values, noisy data, etc. Furthermore, a filtering technique namely the information gain is proposed in the data preprocessing phase to evaluate and select all possible combination of features that greatly influences the prediction of malware as benign or malignant before performing data encoding and scaling. When it comes to machine learning problems, feature selection is essential for speeding up the training of a machine learning algorithm, simplifying a model for better interpretation and prediction, and minimising overfitting.

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

In the second phase (model application), the preprocessed and selected input dataset that has been encoded and split into training and test data is applied to the two adopted models and their ensemble namely the ANN and the DT and the outcome from each of the model is a trained model that can be tested and evaluated.

Table 1. Summary of Literature Review

S/N	Author(s)	Method	Results/Findings
1	Lu et al. (2013)	J48 Decision Tree method for detection.	Suggested granting access to individual components rather than whole applications in order to spot fraudulent behaviour more effectively.
2	(Rosmansyah & Dabarsyah, 2015)	SVM and RF	SVM yielded 91.4% while RF obtained 92.4% accuracy rate
3	Sheen et al. (2015)	MCDF	The proposed method offers a higher execution accuracy of 98.91 percent
4	Yuan et al. (2016)	Deep learning technique (Deep Belief Network).	The detection accuracy of 96.76% was obtained.
5	Nikolopoulos & Polenakis (2017)	System-call Dependency Graphs (ScD-graphs)	An accuracy of 95.9% was obtained.
6	Bat-Erdene et al. (2017)	symbolic aggregate approximation (SAX)	An accuracy of 94.13% was achieved
7	Mahindru & Singh (2017)	Naive Bayes (NB), Random Forest (RF), J48, Simple Logistic (SL), Decision Trees (DT), and K-star were used	Their results showed that SL outperforms RF and J48 by a little margin.
8	D'Angelo et al. (2020)	Neural network with sparse autoencoder technique for feature extraction	An outstanding accuracy rate of 0.94% was obtained.
9	Ren et al. (2020)	Dex-CNN and Dex-CRNN	The models achieved 93.4% and 95.8% detection accuracy for Dex-CNN and Dex-CRNN respectively.
10	Hossain et al. (2021)	Compare three distinct neural networks: CNNs, LSTMs, and GRUs	CNN yielded 83% as highest accuracy followed by LSTM's 66% and GRU's 76%.
11	Xing et al. (2022)	An autoencoder network and a grey-scale visual representation of malware were combined to create this model.	The model achieved 96% accuracy rate and a 96% stable F-score.

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

The final step is the performance evaluation phase where the trained model is tested to classify malware as malignant or benign from the test data and the trained model is further evaluated based on some machine learning classification metrics such as the accuracy score, precision metric, etc. The three discussed processes for the methodology can be seen in Figure 1.

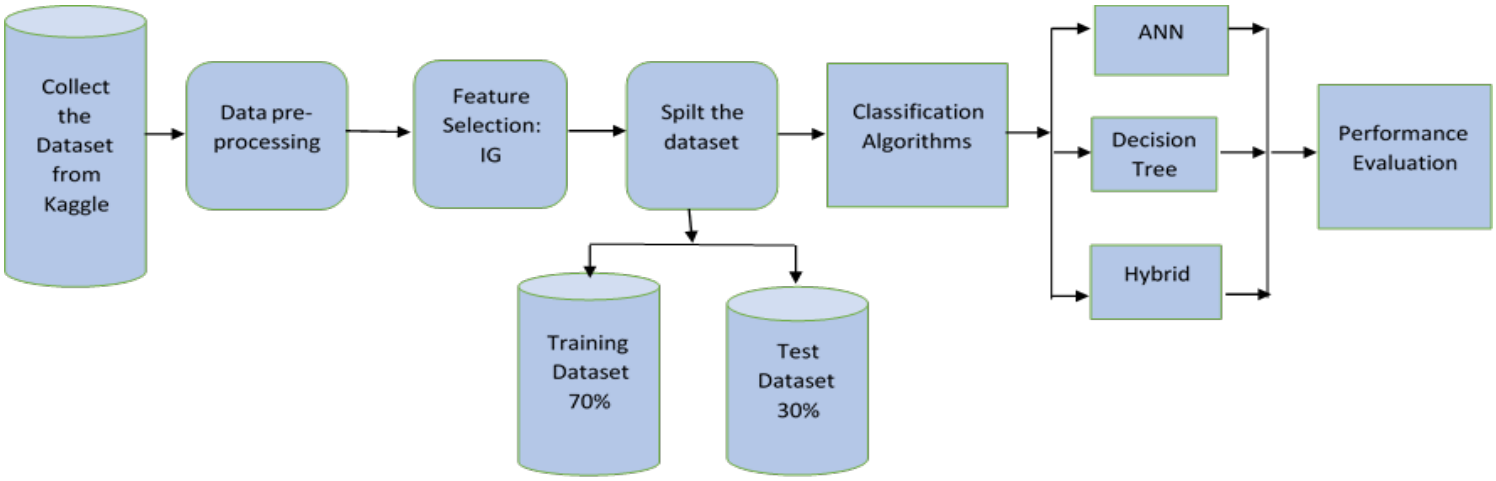


Figure 1. Research Methodology

3.1 Data Source

Malware classification models are trained using the ClaMP "(Classification of Malware with PE headers)" unified dataset available at the Kaggle ML repository[1]. The dataset included a record of 69 characteristics, 54 of which were raw features and 15 were derived features, and a sample of 5000+ malware files with the class label Malware or Benign.
The second dataset is the Android network traffic dataset[2] also sourced from the Kaggle machine learning repository (López et al., 2018). DroidCollector, from which all network traffic is obtained in pcap (packet capture) files, serves as the foundation for this collection. Furthermore, the Android network traffic dataset has a total of 4,704 benign records and a total of 3,141 malignant malware records.

Note: [1] https://www.kaggle.com/datasets/saurabhshahane/classification-of-malwares
[2] https://www.kaggle.com/datasets/xwolf12/network-traffic-android-malware

3.2 Data Pre-processing

As real-world data in most cases are incomplete, noisy, and inconsistent. It becomes necessary to filter and eradicate all inconsistencies in the dataset to achieve an accurate and concise model. Hence, the data pre-processing steps adopted by this study include:

i. The identification and handling of missing values, and unwanted characters.
ii. Encoding the categorical data into numerical values rather than characters.
iii. The identification and selection of relevant features relevance using information gain as the feature selection technique.
iv. Scaling the selected dataset to a range between 0 and 1.
v. Separating the training data from the test data in the dataset based on standard machine learning training and test proportions split.

© The Author(s), under exclusive license to SLUJST 2024

SLU JST
Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

3.3 Information Gain

According to the findings of study that was carried out by Hambali et al. (2022), one of the ways of attribute assessment that is utilised the most often is knowledge acquisition. Therefore, information gain is a method of univariate filtering that creates a ranked list of all characteristics and chooses attributes based on a value that is threshold. In particular, the information gain method creates a metric to measure how much knowledge can be gleaned from a dataset via the classification of malware presence or absence. As a result, the whole spectrum of malicious software characteristics is divided into two categories according to the threshold that was chosen. The first section defines the features above the set threshold whereas the second section defines the features below the set threshold. Hence, the feature above the threshold is then utilized as the selected data attribute to be passed to the model. The mathematical expression for information gain is:

$$IG_z = \sum_{k=0}^{1}\sum_{m=1}^{2}\frac{i_{km}}{N}\log_2\frac{i_{km}}{n_{vm}} - \sum_{k=0}^{1}\frac{i_{k1}+i_{k2}}{N}\log_2\frac{i_{k1}+i_{k2}}{N} \quad (1)$$

where IG defines the information gain, n_{vm} is the number of instances of class k in region m, and i_{km} is the number of expression levels of malware z in region m.

Algorithm 1: *Information Gain (IG)*

Input: *Dataset D with features F (x_1, x_2, ……x_n)*
Θ Threshold value
Parameters: *N – Quantitative amount of bootstrap samples*
M – Total number of features
m – number of selected features
Step 1: Perform attribute filtration on Dataset D
Step 2: Compute Information Gain using equation 1
Step 3: ***if*** *(IG > Θ)*
Step 4: ***Select*** *subset x with the highest IG*
Step 5: Create N bootstrap samples from the selected subset x.
Step 6: ***increment*** *m where m < M*
Step 7: ***end***

Step 1: Perform attribute filtration on Dataset D
This step implies that some initial preprocessing or filtration of the features in the dataset D is performed. This could involve removing irrelevant or noisy features.
Step 2: Compute IG using equation 1
IG is a measure used in feature selection. It provides a numerical measure of the decrease in entropy or uncertainty that is brought about by segmenting the data according to a given characteristic. Equation 1 likely represents the formula used to calculate IG for each feature.
Step 3: if (IG > Θ)
Check whether the computed IG for a feature is greater than the specified threshold θ. This step determines whether a feature is informative enough to be considered for selection.
Step 4: Select subset x with the highest IG
If the IG of a feature is greater than θ, select that feature as part of the subset x with the highest IG. This subset x will initially contain the most informative features.
Step 5: Create N bootstrap samples from the selected subset x.

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Bootstrap sampling involves creating N random samples (with replacement) from the selected subset x. Each bootstrap sample is likely used for further analysis.

Step 6: Increment m where m < M.

Increment the number of selected features (m) as long as it is less than the total number of features (M). This step suggests that the algorithm may progressively add more features to the selected subset based on their IG values.

Step 7: End

The algorithm concludes when the selected subset x has reached a predetermined number of features (m = M) or when there are no more features with IG > θ to consider.

3.4 Classification Algorithm

The problem of malware detection proposed to implement a viable solution to the problem was thus identified to be a classification problem. Classification in machine learning defines a technique for the identification of a model that describes and distinguishes a class label. In essence, every machine learning model varies depending on the type of problem applicable. For instance, the logistic regression is much more suitable for regression problems whereas the decision tree is viable for both classification and regression problems. Hence, this study proposed the adaptation of two classification models to evaluate their performance in malware detection. The machine learning algorithm proposed includes the ANN, the DT and their ensemble models. Both model performances are accessed by evaluating distinct evaluation metrics after conducting data pre-processing which is to filter irrelevant detail and scale the dataset before feeding them to the models.

3.4.1 Artificial Neural Network

ANN was found to be a computer model that simulates the biological brain system in order to carry out extensive data processing. This was discovered via research that was carried out. Dendrites, which are found in biological neural networks, are used to represent inputs in ANN; cell nuclei are used to represent nodes; synapses are used to represent weights; and axons are used to represent outputs. As shown in Figure 2, the ANN has three layers namely, the “input”, “hidden”, and “output” layers with multiple connections between each adjacent layer and a weight assigned to each connection. The numerous layers of the ANN algorithm most often make it referred to as the MLP (Multi-Layer Perceptron). The hidden layers are often called the distillation layer because they choose the most important patterns from the inputs and pass them on to the subsequent layer for further processing. By identifying the most important data from the input and filtering out the rest, this speeds up the network and reduces its overall energy consumption.

The suggested ANN method relies heavily on determining what values of W-weights will result in the least amount of prediction error. To do this, the "backpropagation algorithm" adapts ANN into a learning algorithm that incorporates previous errors into its current predictions. The "gradient descent" method is offered as a means to optimise the system. Gradient descent provides a numerical measure of the errors in predictions. The optimal value of W may be found by experimenting with different values of W and seeing how this affects the prediction errors. Since adjusting W beyond that point has little effect on error rate, we settle on those numbers as optimal. Algorithm 2 depicts the suggested algorithm for the ANN network.

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

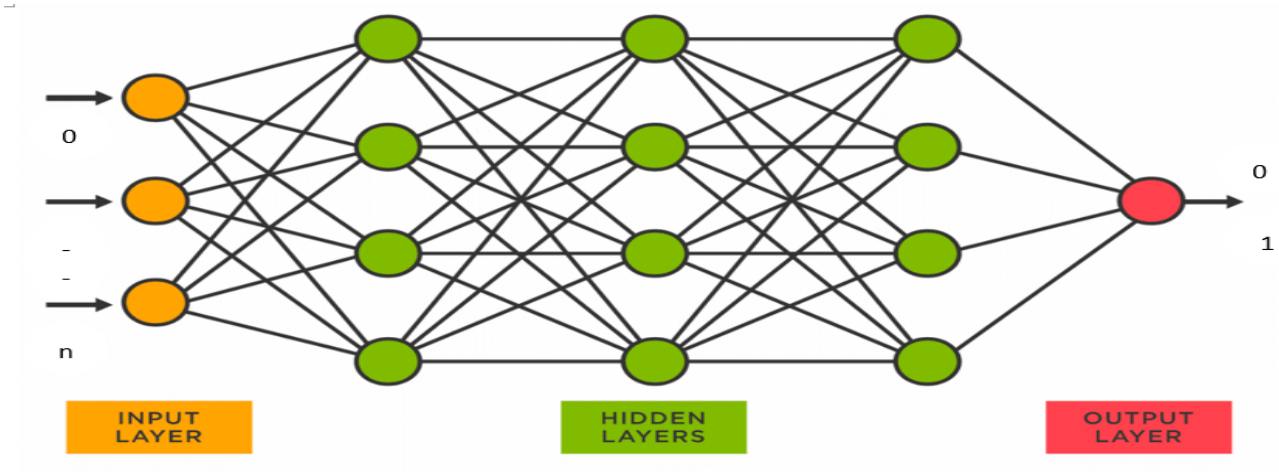


Figure 2. Artificial Neural Network

3.4.1.1 ANN Architecture

It has been established that an ANN's node layers consist of an input layer, a hidden layer(s), and an output layer(s). Each artificial neuron, or node, is linked to the others and has its own weight and threshold. Nodes in a network are activated and send data to higher levels when their output reaches a certain threshold. If this is not the case, no data will be sent to the next layer of the network.

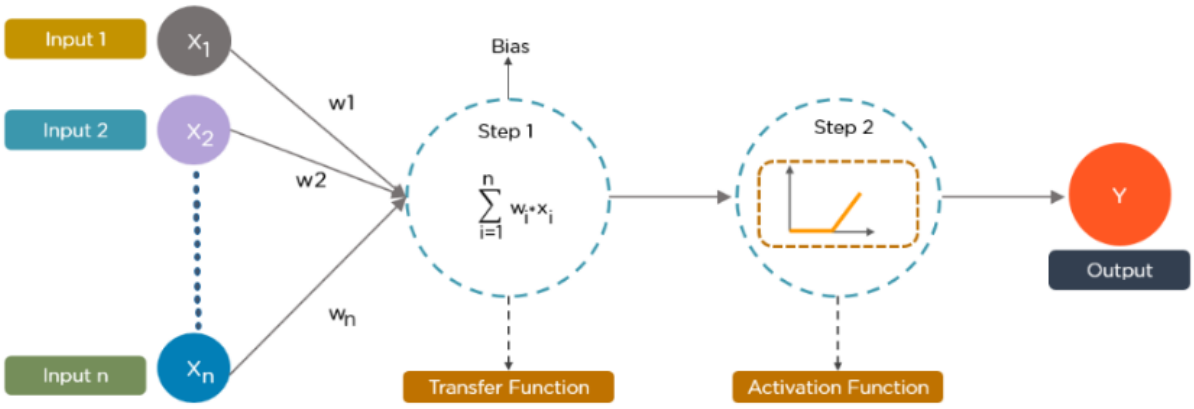


Figure 3. ANN Architecture (Yasin et al., 2024)

The output of ANNs is mostly governed by certain factors such as the “weights”, “biases”, “learning rate”, “batch size”, and other parameters, and the performance of a neural network is impacted by a number of parameters and hyperparameters. The ANN's weights are assigned to each individual node. Therefore, a transfer function is used to determine the output as a function of the inputs and the bias. The activated function receives the weighted sum as an input and produces the output. Node firing behaviour is determined by activation functions. The activated weights are candidates for the final layer. Several activation functions can be used depending on the problem domain. Numerous activation functions, including RELU, Softmax, Sigmoid, tanh, etc., are used in ANNs.

© The Author(s), under exclusive license to SLUJST 2024

SLU JST
Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Algorithm 2: ANN Algorithm

Step 1: *pass the input with some weight to the input layer* $(x^1x^2, \dots \dots x^6)$
Step 2: *Connect all the inputs to each neuron*
Step 3: *perform computation at the hidden layers*
    Step 3.1: *Get the summation of all input with their weight (check figure 3)*
    Step 3.2: *Get bias (check figure 3).*
    Step 3.3: *Get the threshold unit (check figure 3).*
Step 4: *Repeat step 3 for each of the hidden layers*
Step 5: *Pass the result to an output layer*
Step 6: *Get predictions from the output layers and hence calculate the performance metrics.*
Step 7: *Calculate the error, i.e., the difference between the actual and predicted output.*
Step 8: *End*

Here's an explanation of each line from Algorithm 2:
Step 1: pass the input with some weight to the input layer $(x^1x^2, \dots \dots x^6)$
This step represents the initial input to the neural network. The input features $(x^1x^2, \dots \dots x^6)$ are passed to the input layer with certain weights associated with them.
Step 2: Connect all the inputs to each neuron
In a neural network, each input is connected to every neuron in the subsequent layer. This line indicates that connections are established between all input features and neurons in the hidden layers.
Step 3: perform computation at the hidden layers
This step involves the calculations that happen in the layers underneath the network's surface that do the real job of analysing data.
Step 3.1: Get the summation of all input with their weight (check Figure 3)
Within each neuron of the hidden layers, the weighted sum of the input features is computed. This line refers to the summation of all inputs multiplied by their respective weights, as depicted in Figure 3 of the algorithm.
Step 3.2: Get bias (check Figure 3).
In neural networks, a bias term is usually added to the weighted sum. This line indicates that the bias term is obtained, as shown in Figure 3.
Step 3.3: Get the threshold unit (check figure 3).
The threshold unit, often referred to as the activation function, is applied to the weighted sum plus bias to determine the output of the neuron. This line suggests that this activation function is applied, as shown in Figure 3.
Step 4: Repeat step 3 for each of the hidden layers
Neural networks can have multiple hidden layers. This step implies that the calculations performed in Step 3 are repeated for each hidden layer in the network.
Step 5: Pass the result to an output layer
The results computed in the hidden layers are passed to the output layer for making predictions.
Step 6: Get predictions from the output layers and hence calculate the performance metrics.
In the output layer, predictions are made based on the results obtained from the previous layers. After making predictions, various performance metrics can be calculated to evaluate the model's accuracy and effectiveness.
Step 7: Calculate the error, i.e., the difference between the actual and predicted output.
Finally, the algorithm calculates the error by comparing the model's predicted output to the actual expected output. This error is used in the training process to adjust the weights and biases in the network.

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

3.4.2 Decision Tree

To solve both classification and regression issues, DTs are among the most used machine learning methods. DTs, as their name suggests, make judgments based on information and the way it is used. Thus, the performance is not reliant on the linearity of the data, since the decision is made using a tree-like structure based on a conditional statement on the presence or non-occurrence of an event, as illustrated in Figure 4. In order to separate the two-class label (Malware or Benign) contained in the dataset to the fullest purity by lowering the entropy threshold, a broad decision tree can be defined as an illustration of all possible solutions to a decision based on certain conditions. At each step, also known as a node of a DT.

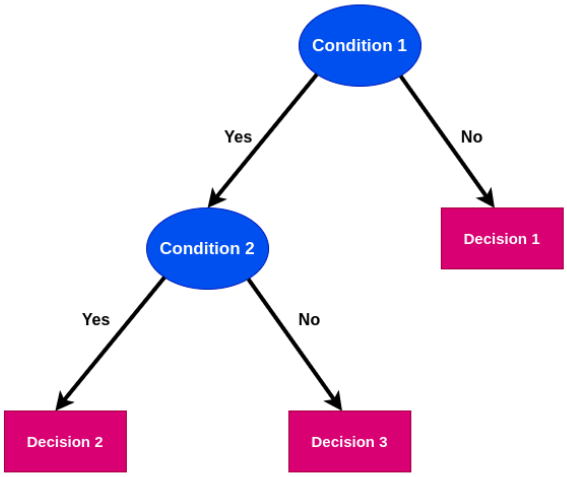


Figure 4. Decision Tree

Algorithm 3: Decision Tree

Step 1: *Start the tree with the root node, assuming X, which contains the complete dataset.*
Step 2: *Find the best attribute in the dataset using Attribute Selection Measure (ASM).*
Step 3: *Divide X into subsets that contain possible values for the best attributes.*
Step 4: *Generate the decision tree node, which contains the best attribute.*
Step 5: *Recursively generate new decision trees using the subsets of the dataset created.*
Step 6: *Continue this process until a stage is reached where the model cannot further classify the nodes which are mostly referred to as the final node as a leaf node.*
Step 7: *End.*

Here's a detailed explanation of each step of the Algorithm 3:
Step 1: Start the tree with the root node, assuming X, which contains the complete dataset.
This step initializes the DT with the entire dataset as the root node.
Step 2: Use Attribute Selection Measure to zero down on the most important feature of the dataset.
In this step, the algorithm evaluates each attribute (feature) in the dataset to determine which one is the best for splitting the data. This selection is based on a measure called the Attribute Selection Measure, which assesses how well each attribute separates the data into distinct classes. The attribute with the highest measure is chosen as the best attribute.
Step 3: Divide X into subsets that contain possible values for the best attributes.

© The Author(s), under exclusive license to SLUJST 2024

SLU JST
Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Once the best attribute is selected, the dataset is divided into subsets or branches based on the possible values of that attribute. Each branch corresponds to a specific value of the chosen attribute.
Step 4: Create the best attribute node in the determination tree.
At this point, a decision tree node is created, representing the best attribute selected in Step 2. This node will serve as an internal node in the tree structure.
Step 5: Create more decision trees in a recursive fashion using the newly derived subsets of the dataset.
For each branch created in Step 3, the algorithm recursively applies the decision tree construction process. It repeats Steps 2 to 5 on each subset of data. This recursive process continues until certain stopping criteria are met, such as reaching a predefined depth of the tree or if further splitting does not improve classification significantly.
Step 6: Keep going until you reach a point when the model is unable to categorise any more nodes (this is the "leaf node" stage).
The recursion continues until a stopping condition is met. When the algorithm reaches a leaf node, it means that further splitting doesn't provide meaningful information, and that node is considered a final prediction or a "leaf" in the decision tree. Leaf nodes typically represent the class label that most instances in that branch belong to.

3.4.3 Hybrid Algorithm

The hybrid was created using ensemble (stacking method). The stack ensemble model combines ANN and DT for the classification of Android malware detection involves several steps which were depicted in Algorithm 4.

Algorithm 4: Hybrid ANN-DT

```
# Define a function to create the Keras model
def create_ann_model():
    model = Sequential()
    model.add(Dense(100, input_dim=20, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
# Create the KerasClassifier wrapper
ann_model = KerasClassifier(build_fn=create_ann_model, epochs=100, batch_size=10, verbose=0)

# Define the Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)
# Define the meta-model
meta_model = DecisionTreeClassifier(random_state=42)
# Create the stacking classifier
stacking_model = StackingClassifier(
    estimators=[('ann', ann_model), ('dt', decision_tree_model)],
    final_estimator=meta_model
)
# Train the stacking classifier
stacking_model.fit(X_train, y_train)
# Make predictions
y_pred = stacking_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}'
End
```

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

The major steps involve define Base Models, that is, create and train ANN and DT models. Next is to create Meta-Model that is, ensemble. The meta-model will take the base models as inputs and produce the final output. After that, the base models were combine using a stacking technique.

3.5 Performance metrics

In order to compare the effectiveness of the ANN, DT and their ensemble, confusion matrix was used as foundation to compute, validate and assess the precision, F-measure, recall, and accuracy produced by the models.

Confusion Matrix: the confusion matrix is used to determine each model's correctness and accuracy and is hence mostly applicable to classification problems (for either a binary or multiclass classification) as in the case of this study. The major indicators used to calculate the Confusion Matrix are shown in Table 2.

Table 2. Confusion Matrix

		Actual	
		Positive	**Negative**
Predicted	Positive	TP	FP
	Negative	FN	TN

From Table 1 above, the description of the TP, TN, FP, and FN is:

True Positives (TP): True Positives are instances where the model correctly predicted the positive class. In other words, it correctly identified a condition or an event when it was present.

True Negatives (TN): True Negatives are instances where the model correctly predicted the negative class. It means the model correctly identified the absence of a condition or an event when it was indeed absent.

False Positives (FP): False Positives occur when the model incorrectly predicts the positive class. It means the model falsely indicates the presence of a condition or an event when it is not actually present.

False Negatives (FN): False Negatives occur when the model incorrectly predicts the negative class. It means the model fails to identify a condition or an event that is actually present.

Precision: Precision is a metric that focuses on the positive class and assesses how many of the predicted positives were actually true. It answers the question: "Of all the instances predicted as positive, how many were actually positive?" Equation 2 depicts the mathematical expression for the precision metrics.

$$precision = \frac{TP}{TP+FP} \quad (2)$$

Recall: Recall, also known as sensitivity or true positive rate, evaluates how many of the actual positives were correctly predicted. It answers the question: "Of all the actual positives, how many were correctly predicted as positive?" The formula is shown in equation 3:

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

F-measure (or F-score): The F1 score is the harmonic mean of precision and recall. It provides a balanced evaluation of a classification model, taking into account both false positives and false negatives. It is particularly useful when dealing with imbalanced datasets. Equation 4 shows the mathematical formular for the F-score.

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

$$F - Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (4)$$

Accuracy: Accuracy is a commonly used metric to evaluate the overall performance of a classification model. It measures the proportion of correctly classified instances (both true positives and true negatives) out of the total number of instances as shown in equation 5:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (5)$$

4. Results and Discussion

To identify and classify malware attacks, from some Android malware experimental (ClaMP malware and Android network traffic) datasets sourced from the Kaggle machine learning repository. This section presents the results of the experiment conducted using an ANN and DT algorithms, with their ensemble (ANN-DT) of the two models. The method utilized for the ensemble is the stacking method. The performance of the models, including the ensemble model, is assessed via the use of assessment criteria such as accuracy, recall, and F1-score.

4.1 Environmental Setup

The implementation of the Android malware detection models used the Python programming language inside the Anaconda programming environment. This implementation was conducted on a Windows operating system, using a dual-core Intel Core i5 CPU with 4GB RAM. The integrated model comprises the ANN, the DT algorithm, and an ensemble approach that combining both methods.

4.2 Parameter Setting

The parameters utilized for the ANN, DT, and their ensemble model are presented in Table 3. From the parameters, the batch size defines the number of the dataset input instances passed to the model per unit layer. The batch size was set to 32 units. The max epochs define the maximum number of epochs conducted and were set to 10 epochs with a learning approach of 0.0001, with Adam set as the optimizer and drop out of 0.2 for each of the layers of the model (Essentially, it is important to note that the values for the defined parameters were tweaked for several ranging values, the presented values are the ones that gave the best performance based on the several experiments conducted). When training a neural network, each epoch represents a full pass over all of the training data. When training a model, one of the hyperparameters that controls the amount of the updates to its parameters is the learning rate. It determines how quickly the model adapts to new information. To enhance the model's precision and efficiency, the optimizer tweaks its weights and biases.

Table 3. Parameter Setting

Parameter	Value
Batch Size	32
Max Epochs	10
Initial learning rate	0.001
Optimizer	Adam
Drop	0.2
Loss	Sparse Category Cross Entropy

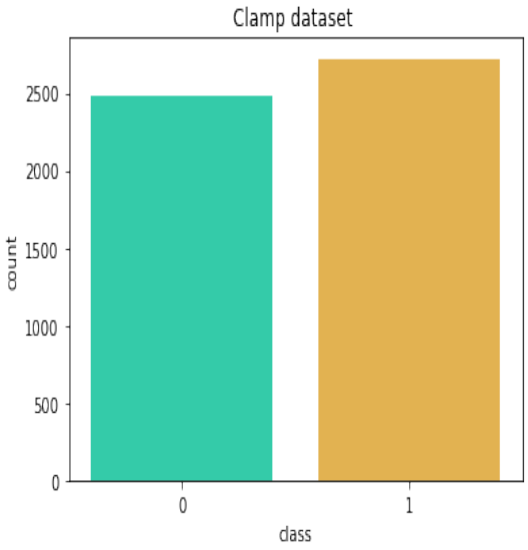
© The Author(s), under exclusive license to SLUJST 2024

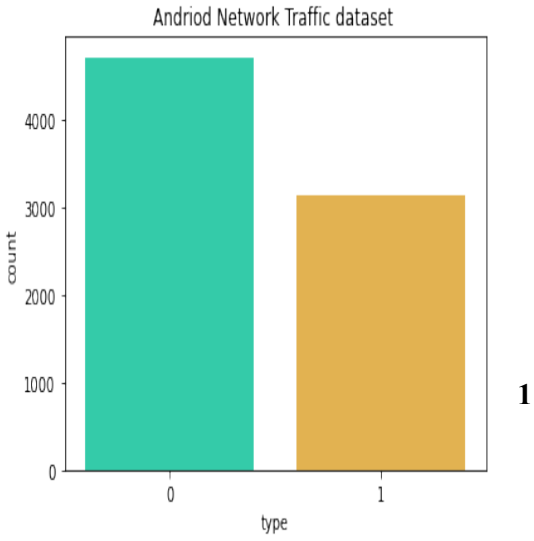
Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

4.3 Data Exploration

This module as an approach to data exploration visualizes some of the dataset features. The visualization enables the experiment to identify data patterns and relationships between the different dataset variables.

Considering an exploratory data analysis, the ClaMP (first) malware dataset contains almost an even distribution of benign and malignant malware labels. The bar chart in Figure 5(a) shows that the benign label has a total of 2,722 records whereas, the malignant has 2,488 records. The second bar chart portrays the Android network traffic dataset, with a total of 4,704 benign records and a total of 3,141 malignant malware records (Figure 5b).





1

Figure 5(a). Distribution of the ClaMP Dataset. Figure 5(b). Distribution of the Andriod Traffic Dataset

4.4 Data Scaling

Data scaling a preprocessing step in machine learning that involves transforming the values of variables (features) into a specific range or distribution. The goal of data scaling is to standardize the features so that they can be more effectively used by certain algorithms or models.

The essence of the data scaling in this study is to avoid feature dominance as larger scales can dominate the learning process and can thus lead to a biased result. As a result, equal weight is given to all characteristics in the learning process when the dataset is scaled while ensuring faster convergence and stable training for the utilized ANN and DT algorithms. Furthermore, the fact that some of the dataset features have ranging units or data types, the need for scaling becomes important to ensure a consistent scale. Hence, one of the methods for normalizing data to a zero-mean, 1-unit scale is to use the standard scaler library included in the Sklearn package.

4.5 Feature Selection

As aforementioned the study incorporates a feature selection technique using the information gain algorithm. The essence of the feature selection is to reduce the dataset dimensionality and further eradicate less correlated features in the detection and classification of Android malware. The algorithm IG allows the specification of the number of

© The Author(s), under exclusive license to SLUJST 2024

SLU JST
Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

features to be selected using a value called the k-threshold. The value of k as a threshold was set to 10 during the experiment. Table 4 shows the selected features of both datasets.

4.6 Percentage Split Technique

In this research, the datasets were split into training and test sets; the training set was utilized to train the ANN, DT, and ANN-DT algorithms, while the test set was utilized to assess the model's performance. Seventy percent (70%) of both the Android network traffic and the ClaMP datasets were utilized to train the models, while the remaining thirty percent (30%) was used to assess the models' performance.

Table 4. Selected Features

ClaMP Dataset	Android Network Traffic
FH_char12	Name
Minor Linker Version	TCP-Packets
Address Of Entry Point	Dist-Port-TCP
Check Sum	External IPs
OH_DLLchar2	DNS-Query-time
Size Of Stack Reserve	UDP-packets
E-text	Source App Packets
File size	Remote App Packets
E-file	Source App bytes
E-data	Remote App Bytes

4.7 Result Presentation

The performance of the ANN, DT, and the ensemble of the two models on the adapted datasets will be evaluated based on certain performance indicators, such as the accuracy score, precision, recall, and the F1-Score. Table 5 and 6 presents the result of the ANN, DT, and hybrid based (Ensemble) on whether an instance is considered malignant or benign (0/1) as labelled in the headers of the tables. The metrics result was evaluated using percentages as the degree of measurement.

Hence, considering the ClaMP malware dataset (Table 5), ANN has an accuracy of 82%, with a precision score of 85 and 81% for the malignant and benign class labels, ANN also has a recall score of 77 and 88% for the malignant and benign class labels, and lastly, and F1-Score of 81 and 84%. Furthermore, DT has a 93% accuracy with a precision of 97 and 89% for the malignant and benign labels, a recall of 87 and 98%, and also an F1-Score of 92 and 93% for both the malignant and benign class labels. The hybrid model has an accuracy of 96%, with precisions, recalls, and f1 scores of 96% respectively for both the benign and malignant class labels.

Table 5. ClaMP Dataset Result

Algorithm	Accuracy (%)	Precision (0/1) (%)	Recall (0/1) (%)	F1-Score (0/1) (%)
ANN	82	85/81	77/88	81/84
DT	93	97/89	87/98	92/93
Hybrid	96	96/96	96/96	96/96

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Taken into cognizance Android network traffic datasets (Table 6), ANN has an accuracy of 71%, with a precision score of 68 and 83% for the malignant and benign class labels, a recall score of 95 and 34% for the malignant and benign class labels, and lastly, an F1-Score of 79 and 48%. Furthermore, DT has a 94% accuracy with a precision of 97 and 89% for the malignant and benign labels, a recall of 92% and 96%, and also an F1-Score of 95 and 92% for both the malignant and benign class labels. The hybrid model has an accuracy of 99%, with precisions and an f1 score of 99%, and also a recall of 100% and 98% for the benign and malignant class labels.

Table 6. Android Network Traffic Dataset Result

Algorithm	Accuracy (%)	Precision (0/1) (%)	Recall (0/1) (%)	F1-Score (0/1) (%)
ANN	71	68/83	95/34	79/48
DT	94	97/89	92/96	95/92
Hybrid	99	99/99	100/98	99/99

4.9 Graphical Result

The graphical result shows a graph plot of the comparison models' performance for each of the datasets based on accuracy metric. From the bar chart, the x-axis defines the models in comparison whereas, the y-axis depicts the accuracies of the model. The graphical plot for the models can be visualized in Figures 6 and 7.

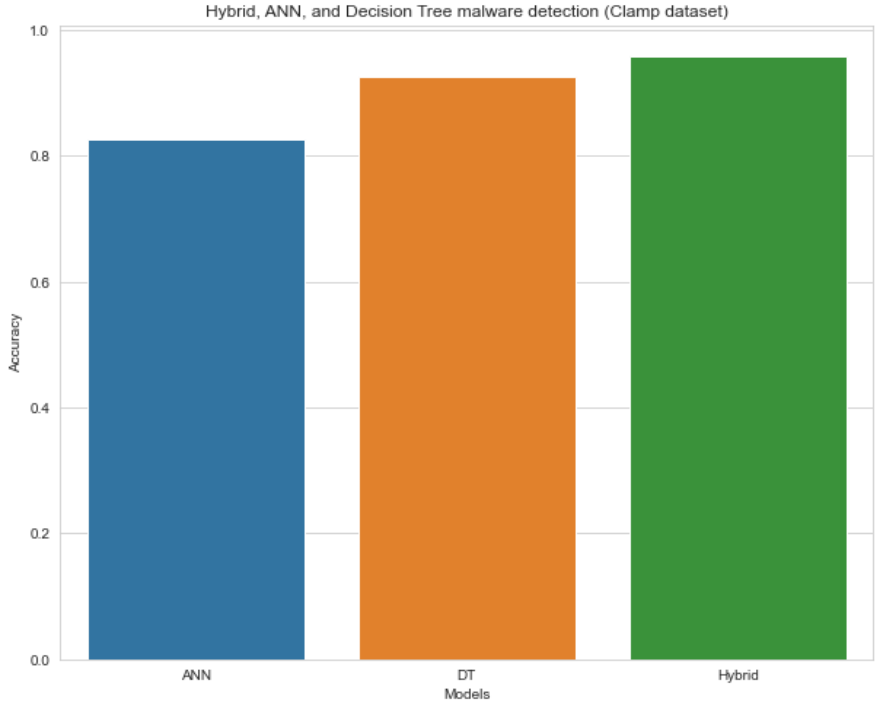


Figure 6. Bar-Chart Comparison of ClaMP Dataset for ANN, DT and Hybrid Models

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Figure 6 shows that hybrid model has the highest accuracy of 96%, followed by DT while the lowest accuracy of 82% was achieved in ANN model.

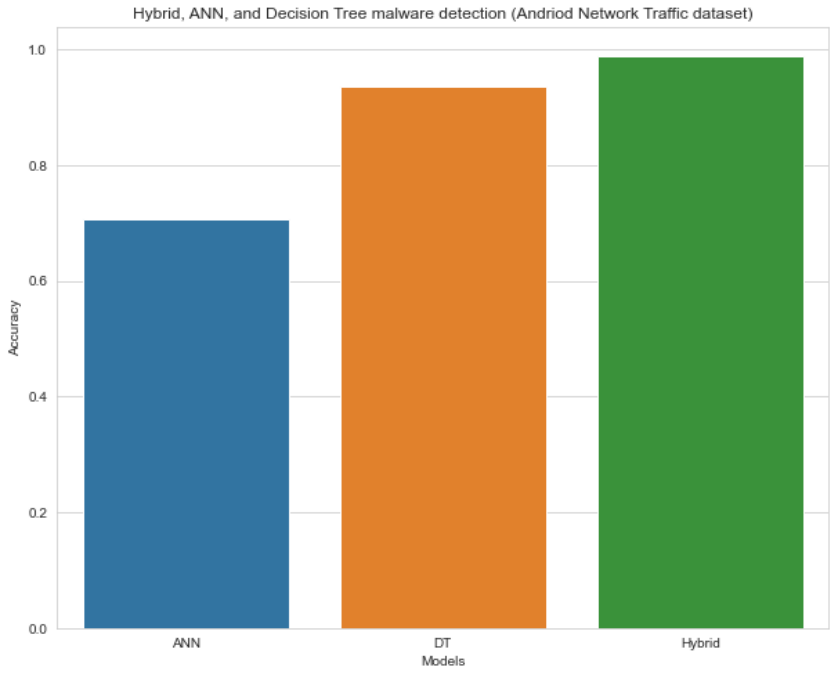


Figure 7. Bar-Chart Comparison of Android Network Traffic Dataset for ANN, DT and Hybrid Models

Figure 7 shows that hybrid model has the highest accuracy of 99%, followed by DT while the lowest accuracy of 71% was achieved in ANN model.

4.9 Receiver Operating Characteristics

To visualize the performance of the developed model on the applied Android network and ClaMP malware datasets, the receiver operating characteristic was applied. The receiver operating characteristic (ROC) curve is a graphical representation that illustrates the performance of a classification model across various categorization criteria. The curve often represents two parameters, namely the rates of true and false positive. From the diagram in Figure 8 – 10, the ‘a’ part of the diagrams represents the performance of each model on the Android network traffic dataset and the ‘b’ part depicts the performance of the models on the ClaMP datasets. It is important to note that the AUC of each model for the respective dataset is appended to the diagrams in Figures 8 - 10.

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

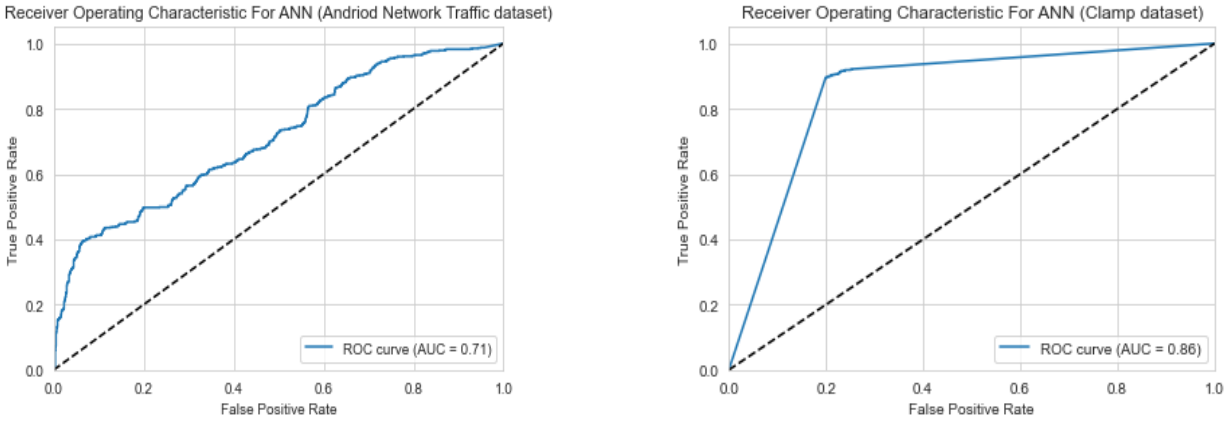


Figure 8. ANN Malware Model (ROC and AUC)

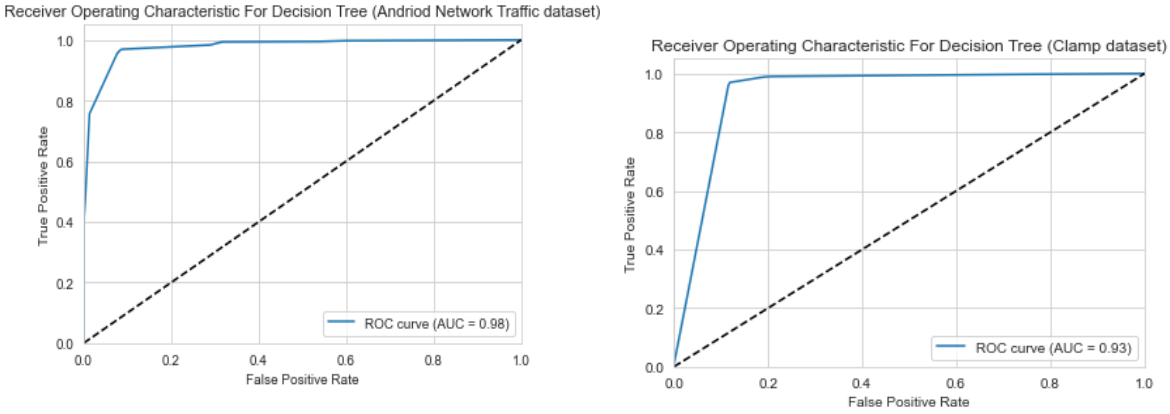


Figure 9. DT Malware Model (ROC and AUC)

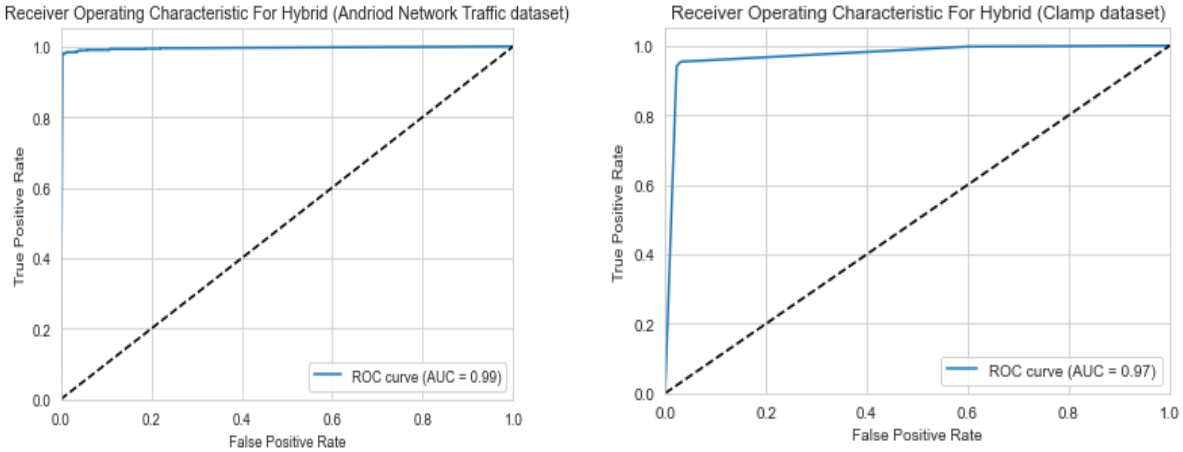


Figure 10. Hybrid Malware Model (ROC and AUC)

© The Author(s), under exclusive license to SLUJST 2024

Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Figure 8 shows ROC curve for ANN in both datasets. First diagram indicated that ROC for Android network traffic dataset while the second one is for Clamp dataset. For Android network traffic, ANN perform very poor because the curve is very close to the diagonal from left-button to right-top which indicating a low sensitivity and specificity. While that of Clamp dataset performer better above the average.

Figure 9 shows DT ROC curve for both datasets. The first diagram is for Android network traffic dataset while the second one is for Clamp dataset. Both curves indicated a good performance with the ROC curve far away from diagonal from left-button to right-top which indicating a high sensitivity and specificity. A model that performs excellently has an AUC of closer to 1.

Figure 10 shows hybrids ROC curve for both dataset. The first diagram is for Android network traffic dataset while the second one is for Clamp dataset. Both curves indicated a good performance with the ROC curve far away from diagonal from left-button to right-top which indicating a high sensitivity and specificity. A model that performs excellently has an AUC of closer to 1. The best model based on ROC is the hybrid model because the AUC is almost 1 in the both datasets.

4.10 State-of-the-Art Comparison

The comparison of state-of-the-art algorithms for the three utilized model, as well as four other algorithms applied by different researchers, is presented in Table 7. The table includes five features as columns for the state-of-the-art comparison, namely authors, the dataset used, the algorithm employed, the best algorithm chosen by each author, and the corresponding scores in the final column. The performance evaluation of the applied ANN, DT, and ANN-DT (Ensemble) algorithms is conducted on the selected features from two datasets, namely the Android Network Traffic and ClaMP dataset retrieve from the Kaggle ML database. When compared to the algorithms applied by the four other authors in the field of machine and deep learning, the hybrid ANN-DT model trained in this study exhibited the best performance, achieving an accuracy of 99%. Figure 11 shows a graphical chart of the state-of-the-art algorithms compared based on AUC Score. The hybrid of ANN-DT combines the complex pattern recognition of ANNs with the interpretability and rule-based decision-making of DT. This results in improved performance, reduced overfitting, and better handling of various data characteristics, making it a superior choice for classification tasks like Android malware detection. These benefits are well-documented in the literature, providing a strong justification for the ensemble approach.

Table 7. State-of-the-Art Comparison

Authors	Dataset Used	Algorithm Used	Best Algorithm	Accuracy (%)
Salehi & Amini, (2017)	-	RF	RF	96
Ren et al., (2020)	Classes.dex Android files Google play store	Dex-CNN and Dex-CRNN	Dex-CRNN	95.8
Hossain et al., (2021)	Virusshare and portableapps malware dataset	LSTM, GRU, and CNN	CNN	83
Xing et al., (2022)	VirusShare and Google Play Store	AE	AE	96

© The Author(s), under exclusive license to SLUJST 2024

SLU JST
Sule Lamido University Journal of Science & Technology Vol. 8 No. 2 [Dec, 2024], pp. 45-68
https://doi.org/10.56471/slujst.v7i.482
Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

Current Study	Android Network Traffic	ANN, DT, ANN-DT (Hybrid)	ANN-DT (Hybrid)	**99**
	ClaMP Dataset		ANN-DT (Hybrid)	96

KEYs: Artificial Neural Network (ANN), Decision Tree (DT), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), Auto Encoder (AE), Dalvik Executable – Convolution Neural Network (Dex-CNN), Dalvik Executable – Convolution Recurrent Neural Network (Dex-CRNN)

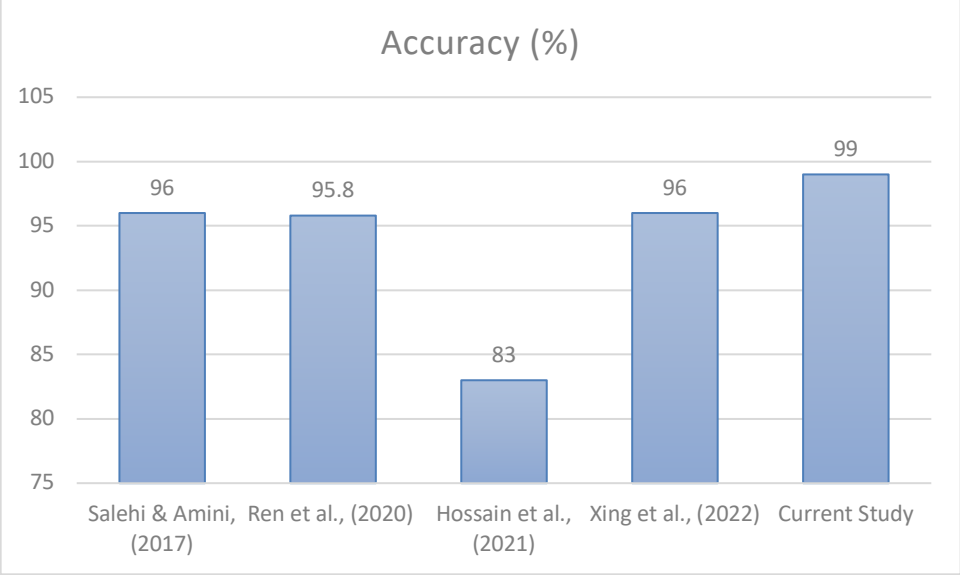


Figure 11. Comparison of State-of-the-Art Algorithms

5. Conclusion

This study after an extensive survey of the state-of-the-art model on malware detection techniques applied machine and deep learning algorithms namely the ANN and DT algorithm while hybridization of the two algorithms via a stacking technique. Furthermore, the performance of the developed models after applying information gain as a feature selection technique was evaluated based on some metrics with the accuracy score metrics as the major indicators. The comparison of the implemented models revealed an accuracy of 82%, 93%, and 96%, for ANN, DT, and the hybrid respectively on the ClaMP malware dataset. Considering the Android network traffic dataset, ANN, DT, and the hybrid has an accuracy of 97%, 94%, and 99%. To further scrutinize the performance of the best performing, the best model performance was compared with some state-of-the-algorithms. The result of the comparison revealed the hybrid model was developed to outperform the state of the algorithms with an accuracy score of 99%. Therefore, the hybrid approach is recommended for Android malware detection system. In the future we would like consider malware attacks mutate over time, the study suggests the application of recent machine learning techniques including the ResNet50, Alex-net, Google Net, and others to evaluate the performance of the adapted datasets and suggest the use of other feature selection techniques such as the use of meta-heuristic methods like the artificial particle swarm optimization, bee colony, and bat algorithm.

© The Author(s), under exclusive license to SLUJST 2024

Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

References

Afianian, A., Niksefat, S., Sadeghiyan, B., & Baptiste, D. (2019). Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, *52*(6), 1–28.

Arslan, R. S., Doğru, İ. A., & Barişçi, N. (2019). Permission-based malware detection system for android using machine learning techniques. *International Journal of Software Engineering and Knowledge Engineering*, *29*(01), 43–61.

Bat-Erdene, M., Park, H., Li, H., Lee, H., & Choi, M.-S. (2017). Entropy analysis to classify unknown packing algorithms for malware detection. *International Journal of Information Security*, *16*, 227–248.

Chen, Q., & Bridges, R. A. (2017). Automated behavioral analysis of malware: A case study of wannacry ransomware. *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 454–460.

D'Angelo, G., Ficco, M., & Palmieri, F. (2020). Malware detection in mobile environments based on Autoencoders and API-images. *Journal of Parallel and Distributed Computing*, *137*, 26–33.

Damodaran, A., Troia, F. Di, Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, *13*, 1–12.

Daniel, G. L., Carles, M. P., & Jordi, P. C. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenge. *Journal of Network and Computer Applications, 2020, Vol. 153, 102526*.

Dixit, P., & Silakari, S. (2021). Deep learning algorithms for cybersecurity applications: A technological and status review. *Computer Science Review*, *39*, 100317.

Hambali, M. A., Oladele, T. O., Adewole, K. S., Sangaiah, A. K., & Gao, W. (2022). Feature selection and computational optimization in high-dimensional microarray cancer datasets via InfoGain-modified bat algorithm. *Multimedia Tools and Applications*, *81*(25), 36505–36549.

Hossain, H., Kayum, S. I., Paul, A., Rohan, A. A., Tasnim, N., & Hossain, M. I. (2021). Malware Detection Using Neural Networks. *2021 5th International Conference on Electrical Information and Communication Technology (EICT)*, 1–6.

Kim, Y., Lee, J. J., Go, M.-H., Kang, H. Y., & Lee, K. (2022). A systematic overview of the machine learning methods for mobile malware detection. *Security and Communication Networks*, *2022*.

Li, C., Lv, Q., Li, N., Wang, Y., Sun, D., & Qiao, Y. (2022). A novel deep framework for dynamic malware detection based on API sequence intrinsic features. *Computers & Security*, *116*, 102686.

López, C. C. U., Villarreal, J. S. D., Belalcazar, A. F. P., Cadavid, A. N., & Cely, J. G. D. (2018). Features to detect android malware. *2018 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 1–6.

Lu, Y., Zulie, P., Jingju, L., & Yi, S. (2013). Android malware detection technology based on improved Bayesian classification. *2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*, 1338–1341.

Mahindru, A., & Singh, P. (2017). Dynamic permissions based android malware detection using machine learning techniques. *Proceedings of the 10th Innovations in Software Engineering Conference*, 202–210.

Mathur, A., Podila, L. M., Kulkarni, K., Niyaz, Q., & Javaid, A. Y. (2021). NATICUSdroid: A malware detection framework for Android using native and custom permissions. *Journal of Information Security and Applications*, *58*, 102696.

Nikolopoulos, S. D., & Polenakis, I. (2017). A graph-based model for malware detection and classification

© The Author(s), under exclusive license to SLUJST 2024

Online ISSN: 2736-0903
Print ISSN: 2736-089X
www.slujst.edu.ng

using system-call groups. *Journal of Computer Virology and Hacking Techniques*, *13*(1), 29–46.

Nisa, M., Shah, J. H., Kanwal, S., Raza, M., Khan, M. A., Damaševičius, R., & Blažauskas, T. (2020). Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features. *Applied Sciences*, *10*(14), 4966.

Ren, Z., Wu, H., Ning, Q., Hussain, I., & Chen, B. (2020). End-to-end malware detection for android IoT devices using deep learning. *Ad Hoc Networks*, *101*, 102098.

Rosmansyah, Y., & Dabarsyah, B. (2015). Malware detection on android smartphones using API class and machine learning. *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, 294–297.

Salehi, M., & Amini, M. (2017). Android malware detection using Markov Chain model of application behaviors in requesting system services. *ArXiv Preprint ArXiv:1711.05731*.

Sheen, S., Anitha, R., & Natarajan, V. (2015). Android based malware detection using a multifeature collaborative decision fusion approach. *Neurocomputing*, *151*, 905–912.

Tong, F., & Yan, Z. (2017). A hybrid approach of mobile malware detection in Android. *Journal of Parallel and Distributed Computing*, *103*, 22–31.

Wazid, M., Das, A. K., Rodrigues, J. J. P. C., Shetty, S., & Park, Y. (2019). IoMT malware detection approaches: analysis and research challenges. *IEEE Access*, *7*, 182459–182476.

Xie, N., Qin, Z., & Di, X. (2023). GA-StackingMD: Android Malware Detection Method Based on Genetic Algorithm Optimized Stacking. *Applied Sciences*, *13*(4), 2629.

Xing, X., Jin, X., Elahi, H., Jiang, H., & Wang, G. (2022). A malware detection approach using autoencoder in deep learning. *IEEE Access*, *10*, 25696–25706.

Yasin, N., Naqvi, S. M. D., & Akhter, S. M. (2024). Simultaneous spectrophotometric determination of Co (II) and Co (III) in acidic medium with partial least squares regression and artificial neural networks. *Heliyon*, *10*(4), e26373. https://doi.org/10.1016/j.heliyon.2024.e26373

Yeo, M., Koo, Y., Yoon, Y., Hwang, T., Ryu, J., Song, J., & Park, C. (2018). Flow-based malware detection using convolutional neural network. *2018 International Conference on Information Networking (ICOIN)*, 910–913.

Yuan, Z., Lu, Y., & Xue, Y. (2016). Droiddetector: android malware characterization and detection using deep learning. *Tsinghua Science and Technology*, *21*(1), 114–123.

© The Author(s), under exclusive license to SLUJST 2024