[1] M.Karthika*

[2]Dr.M.Anusha

Ohmn: Android Malware Detection Method using Opcode Highway Memory Network

Abstract: - In recent times, there has been a notable trend among Android malware to prioritise fast replication, hence expediting the acquisition of critical data. Adversaries have the potential to introduce hazardous modifications to code via the act of replicating it. In this research, we propose a deep learning-based network"Opcode highway memory network" (OHMN) that may detect variants of malware that are connected to one another without needing the analyst to have a background in mathematics or methodology.Input data may be standardised using sequence patch normalisation, and features can be discovered with truth interference fuzzy clustering by employing software birthmarking of malcode sequences. Once malware samples have been grouped together, the data is passed to the OHMN process, which searches for various types of malware. Malware detection on Android has been proved to work successfully in both experimental and observational settings. The Mal Radar dataset, an android malware dataset retrieved from zenodo in a python environment, was used for these tests. The simulation results demonstrated that the suggested method outperformed the state-of-the-art technique.

Keywords: Android malware, sequence patch normalization, Truth interference fuzzy clustering, Opcode highway memory network,

I. INTRODUCTION

The global population of Android users exceeds 1.4 billion individuals, and during the first quarter of 2016, the Android platform facilitated over 290,000 smartphone transactions. Regrettably, a significant majority of mobile malware, over 97%, exhibits a special focus on Android handsets with the intention of illicitly acquiring monetary resources or personal data. The act of transmitting unwanted SMS messages to premium-rate lines has the potential to jeopardise an individual's personal data, including their calendar, inbox, contact list, social media profiles, files, and banking credentials. Malicious programmers might potentially elude discovery by using obfuscation methods like as polymorphism and metamorphism, whereby they make inconspicuous alterations to attributes like file hash, function names, and variable names. In 2016, a study conducted by researchers revealed a 40% increase in the number of daily instances of Android malware. The process of creating detection signatures for antimalware systems may encounter delays as a result of manual analysis, which is more challenging owing to the escalating volume of novel malware samples. The use of novel algorithms that possess the capability to autonomously amalgamate comparable samples into distinct groups is the only viable approach for effectively managing the extensive quantities of malware. There are several notable benefits associated with this approach. Firstly, it allows for the consistent application of removal techniques to samples that have been identified as belonging to established families. Secondly, it enables security analysts to concentrate their manual investigations on the limited number of new samples that do not fall into any known family category. Lastly, it contributes to the enhancement of anti-malware tool signatures by facilitating a more comprehensive compreheThe primary focus of the bulk of academic articles pertaining to mobile malware analysis is around the detection procedure. Numerous well published research efforts have been undertaken to ascertain the identities of Android malware lineages. The task of characterising malware families might provide challenges due to the utilisation of old information or the adoption of low-level, intricate traits such as API call patterns, which may be difficult to decipher. The current body of research offers valuable insights; however, it is limited by its focus on large families (e.g., size > 10) where training data is readily accessible. Consequently, it neglects smaller families, which

[1] [1]Research Scholar, PG & Research Department of Computer Science, National College (Autonomous),

Affiliated to Bharathidasan University, Tamilnadu, India, (E-mail-karthikaparthi2009@gmail.com)

[2]Assistant Professor, PG & Research Department of Computer Science, National College (Autonomous),

Affiliated to Bharathidasan University, Tamilnadu, India, (E-mail-anusha260505@gmail.com)

***Corresponding Author:** M.Karthika

*(E-mail – karthikaparthi2009@gmail.com)

encompass distinct malware variants that warrant attention from security analysts. In order to effectively classify families of malware samples, we suggest the use of a memory network referred to as Opcode highway. The study starts by doing a thorough performance analysis of the Mal Radar dataset, which is regarded as the largest publicly available Android malware dataset including labelled families. This dataset is used to classify various Android malware families. This study seeks to evaluate and examine many advanced classification and clustering techniques in order to categorise families, using a wide range of static and dynamic characteristics. By leveraging the advantages of both classification and clustering, our technique demonstrates superior performance compared to other state-of-the-art baselines. This research has made many significant contributions.

(i), In this study, we compare and contrast a broad variety of cutting-edge supervised and unsupervised methods for categorising Android malware families.

(ii) We combine clustering and classification results to get around issues in existing malware classifiers.

By showing how to extract traits that are unique to each family, we show how to do identification across different types of malware.

The rest of the paper's structure may be found in the outline below. In Section 2, we discuss the studies conducted to far on the subject of malware identification and classification. In Section 3, the problem is outlined. In Section 4, we detail the methodology behind the proposal. We conducted a thorough evaluation of the suggested approach in Section 5. Sections 6 and 7 provide a summary and suggestions for further study.

II. RELATED WORKS

In recent times, there has been a significant emphasis placed on the matter of malware detection on Android mobile devices.Based on the available data, the study conducted by [1] successfully determined the optimal permissions and aims for detecting Android malware. The use of Random Forest resulted in an increase in accuracy to 94.73%. Nevertheless, their approach would not adequately consider the presence of malicious or geographically-restricted infections. The researchers in reference [2] achieved a high level of accuracy in identifying Android malware, with a rate of 98.3%. Additionally, they were able to remember viruses with a rate of 98.1% by combining the results obtained from many deep neural networks. In the web-based approach proposed by [3], a total of twenty-one algorithms, six feature ranking methodologies, and four feature subset selection processes were used to determine the ultimate set of top features. The success rate of the efforts is 99.2 percent. The number of permissions and API calls required for the suggested strategy to effectively identify malware remains uncertain. In a study conducted by [4], a three-tiered trimming approach was used to identify the essential permissions necessary for the detection of android malware. During the testing phase, the system demonstrated an efficacy rate of 93.62 percent in mitigating malware, including both novel and pre-existing threats. One of the significant limitations of the SIGPID tool is its failure to identify malware that does not have a signature stored in its database. In their study, the researchers [5] used a methodology that included analysing both the string features and structural elements in order to provide a comprehensive definition of the enduring behaviour shown by Android applications. The use of string features alone results in an accuracy of 95.8%, which improves to 98.4% when combined with structural characteristics. In contrast, the use of structural characteristics alone yields an accuracy of 90.6%. One shortcoming of this technique is in its incapacity to do real-time assessments. In this study, our objective was to identify the optimal algorithm for malware prediction and determine the key attributes that contribute significantly to its effectiveness.In the seventh section of our study, we introduced DroidMalwareDetector, a system designed for detecting Android malware. This system utilises convolutional neural networks as its underlying methodology. In their study, the authors provide a technique referred to as MINAD (Multi-Inputs Neural network based on application structure for Android malware Detection) that is designed to identify malicious software inside Android apps [8]. The first step involves comprehensive categorization of all conceivable aspects pertaining to Android apps. Subsequently, these aspects are classified into three distinct classes, namely "System-based," "Library-based," and "User-based," depending on the degree of alignment between their respective definitions and the inherent characteristics of the Android platform. Furthermore, we try to replicate the well-established attributes that contribute to the distinctiveness of each community. The creation of the deep multi-input network involves two distinct stages that are dedicated to the abstract exploration of feature sets. In order to conduct our evaluation, we use a comprehensive collection of more than 155,000 data points sourced directly from reputable datasets such as the Google Play Store, Drebin, and

AMD. According to the reference [9], machine learning is used for the purpose of identifying and detecting malicious software, often known as malware, on Android-based mobile devices. The achievement of our 98.98% detection rate was accomplished by the analysis of a dataset consisting of 10,010 clean applications and 10,683 malicious applications, using the C4.5, DNN, and LSTM algorithms. The study conducted by TLAMD [10] used a thorough approach to generate adversarial samples and examined the extent of disruption caused on android samples. The construction of the black-box model included the use of eight distinct attributes derived from both the manifest file and the decompiled code. Nevertheless, the completion of lengthy requests may provide a challenge for the model. The study conducted by researchers aimed to investigate the need of acquiring extra permits for the unauthorised transmission of certain programming in [11]. The combined use of static and coding analysis yielded an overall accuracy rate of 91.95 percent. TFDroid [12] use support vector machines (SVMs) for the purpose of analysingprogramme descriptions and identifying malware. The acquired datasets were classified into pertinent categories, and anomalies were extracted from them. By only using secure apps as the training data and employing a limited dataset for cross-validation, we achieved a significant improvement in the accuracy of detection, reaching 93.65 percent. The authors of [13] proposed an ensemble learning approach to detect unfamiliar APKs. This approach included the integration of sensitive permissions and API data with two classifiers: a decision tree classifier and a KNN classifier. In a study conducted by [14], several machine learning techniques, including K-Nearest Neighbours (KNN), Random Forest (RF), and Support Vector Machines (SVM), were used to evaluate the robustness of the rules of association across API abstractions. When juxtaposed with another model, MaMaDroid [15], which was constructed using identical characteristics and machine learning methodologies, this model exhibited superior outcomes. Datasets including both benign and malicious software were collected and annotated with timestamps to provide a comparative analysis of the efficacy of several machine learning classifiers, namely Naive Bayes (NB), J48, Support Vector Machines (SVM), Random Forest (RF), and Simple Logistic (SL) classifiers [16]. GefDroid [17] use unsupervised learning techniques to generate graph embeddings via the analysis of malware families. The process of abstracting the semantics of the programme into a set of sub-graphs enables the creation of a detailed behavioural model. The AndroDialysis system (18) employs rights and intentions as mechanisms for the identification of malware. The stark disparity between pragmatic solutions and theoretical ideals became readily apparent. Subsequently, the Bayesian Network methodology is developed with the intention of expediting accurate diagnoses for medical practitioners. The permission-based approach employs a detection model that has undergone training using a dataset including both benign and malicious apps. Based on the findings of ICCDetector [19], it has been identified that there are five discrete categories of Android malware. In this particular instance, a malware detection model using a support vector machine (SVM) is used. The model has been trained on a dataset consisting of 5,264 instances of hazardous software and 12,026 instances of benign software. The authors of [20] presented a machine learning methodology for the identification of Android malware families. The use of attributes that are influenced by the specific context in which an Android API is employed encompasses several advantages, such as the incorporation of capabilities derived from reflection and the integration of native app binaries. In a study conducted by researchers [21], a bi-objective Generative Adversarial Network (GAN) was used to provide a unique methodology for launching attacks on data via the use of adversarial instances. The paper by [22] introduces the use of HyGNN-Mal for the purpose of automatic Android malware detection. This study does an in-depth analysis of the source code of an Android application, examining its structure and execution simultaneously. In the meanwhile, we have successfully included conventional static components, safety measures, and application programming interfaces (APIs). The HyGNN-Mal framework employs a Deep-TNN model to analyse the fundamental structure of the code.

III.PROBLEM STATEMENT

In an effort to find the links between Android features, previous research has sought to employ machine and deep learning models. This means they may be able to mimic the complex and nuanced differences between malicious and benign software. Features like as application programming interfaces, system calls, and permission lists were essential to the operation of machine learning programmes. Due to the usage of these techniques by experts for feature engineering, they may undergo surprising changes. Even if future changes to feature patterns are detected, the algorithms may incorrectly categorise them. This leads to more uncertainty in ML detection techniques. The use of insufficiently large or imbalanced datasets also posed a challenge, since it muddled conclusions regarding

the efficacy of specific procedures in real-world settings. No approach has been put through its paces with the toughest potential scenario. There are also nuanced distinctions between benign and malicious Android software, particularly Adware. Thus, it is essential to create detection algorithms that are specifically designed to identify various kinds of malware. The scholarly literature on malware analysis indicates that familiarity with API calls or system calls is essential for understanding malicious process activity. In spite of the fact that the API/system calling sequences may provide an apparently descriptive picture of an application's activity, there are no benchmark datasets comprising tens of thousands of goodware and dangerous applications. The sheer quantity of operations that must be controlled might be an issue. When malware detects the presence of virtual environment instrumentation, the harmful heart is generally concealed by antiemulation or anti-sandbox measures. Deep learning models have been shown to be very good at learning features from time-series data. Deep learning, on the other hand, has been demonstrated to be useless when long sequences, such as those involving application programming interface (API) or system calls, are taught. One notable contributor is deep learning's extensive calculation of error rates. Current approaches for detecting Android malware include the following shortcomings:

• Using both classic machine learning and deep learning approaches was typical procedure.To find the optimal collection of static characteristics to obtain, most research has focused on filtering approaches such as information gain and chi-squared tests.

• It is unrealistic to anticipate a certain amount of features to be used in any feature selection technique.

As a result, in our study, we created a malware prediction model that, by grouping negative features together, can predict whether or not a certain software would participate in hostile conduct.

IV. PROPOSED WORK

“Figure 1 below depicts the Overall prediction procedures. Here initially the data was preprocessed using sequence patch normalization then the features can be clustered using the truth interference fuzzy clustering finally the malware data can be classified using the OHMN. Modelling malware, evaluating models, and extracting features from raw Android APK files are all included. We demonstrate the steps required to put into action a memory network for opcodes.

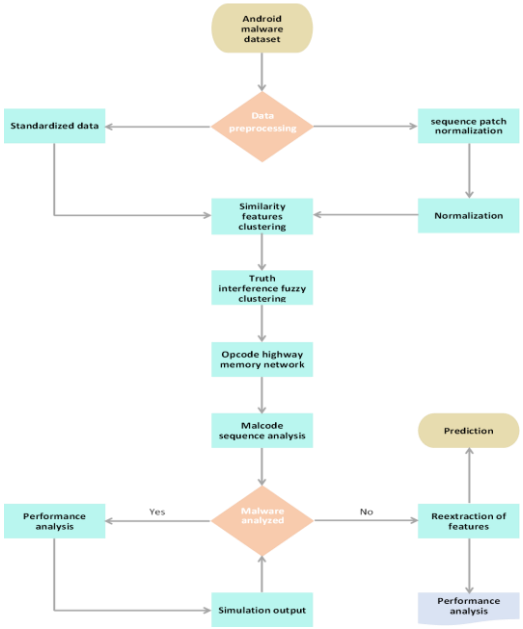

Figure 1 schematic representation of the suggested methodology

a. Android malware dataset

Our community has done a lot of research on mobile virus detection. A trustworthy and up-to-date malware dataset is required for assessing malware detection methods. Cybersecurity specialists should authenticate the infection

and classify its hazardous capabilities. Malware benchmarks are extensively utilised in our community (MalGenome, Drebin, Piggybacking, and AMD, to name a few), but each of these resources has its own set of flaws, such as being out of date, too large, lacking in coverage, or untrustworthy.

MalRadar, a continually updated Android malware dataset, is maintained as accurate as possible by collecting malware based on the analytical reports of security professionals. They automated the process of discovering articles that describe new Android malware and include Indicators of Compromise (IoC) by monitoring the websites of the ten most prominent security businesses that publish such discoveries. MalRadar, a collection of 4,534 distinct Android malware samples (including apks and metadata) produced between 2014 and April 2021 and validated by security experts through in-depth behavioural research, had been successfully collected by the time this report was released. More information may be found at https://malradar.github.io/.

The dataset includes the following files:

(1) sample-info.csv

In this file, they list all the detailed information about each sample, including apk file hash, app name, package name, report family, etc.

(2) malradar.zip

They have packaged the malware samples in chunks of 1000 applications: malradar-0, malradar-1, malradar-2, malradar-3. All the apk files name after the file SHA256.

b. Data preprocessing

For normalizing the data sequence patch normalization was used. Normalising the data to fall within the range 0-1 using sequence patch normalisation is advised for a more easy investigation of malware patterns. The residual studentized method's normalisation is based on the computation of the standard deviation. Several data distributions and regression analysis are used to normalise the prognosis of heart illnesses. Eq. (1) provides the normalisation regression equation:

$$Q = \alpha_0 + \alpha_1 Q + \epsilon_j, \; for \; j = 1,2, \ldots n \qquad (1)$$

The proposed model may be used to explain any potential permutation of random data points. This equation (2) describes the model used to predict data:

$$Q_j = \alpha_{0_0} + \alpha_{0_j} Q + \epsilon^*_j \qquad (2)$$

Data samples with the same variance and error may be represented by Equation (2), where $\alpha_{0_0}$and $\in_i$ are the least-squares values. Solving equations (1) and (2) allows us to determine the residual value, given by (3). Mean and median values are calculated by subtracting the standard deviation from the data sample. To get a rough average, we may use the formula.

$$\mu = \frac{\sum_{j=1}^{n} Q_j}{N} \qquad (3)$$

where $Q_i$is the raw data and N is the data repetition frequency. Normalisation of data may be calculated using the following formula:

$$M_r = \frac{\epsilon^*_j}{\sigma^{\cdot}_j} \; M_r = \frac{Q_j - \mu^*_j}{\sigma^{\cdot}_j} \qquad (4)$$

where:

ϵ^*_j -error residual value

$\sigma^{\cdot}_i$–error variance

Finally the data can be standardized

b. Feature extraction and clustering

This truth inference fuzzy clustering approach uses a dynamic computational process known as the truth interference to identify and classify properties of malware.Here in which the features are identifies and it can be clustered depend on its characteristics . Fuzzy clustering is a powerful method that combines the strengths of both fuzzy logic and logic interference. The available data reveals that with an appropriate set of criteria, the fuzzy can make accurate predictions in the vast majority of situations. The fuzzy approach has seen widespread use, although its processing requirements and complexity make it less than ideal. The number of changeable factors and rules may grow exponentially due to the large number of inputs. Furthermore, the gradient-based learning that tends to become trapped in local minima is an integral part of the regular learning process in fuzzy. In addition to the above fundamental fuzzy principles, equations (11) and (12) also contain

Rule 1: If T_1 is B_j and T_2 is Z_j, then

$$Rules_j = sT_j + t_jT_{j+1} + u_j$$

Rule 2: If T_1 is B_{j+1} and T_2 is Z_{j+1}, then

$$Rules_{j+1} = s_{j+1}T_j + t_{j+1}T_{j+1} + u_{j+1}$$

where B_i, Z_i, B_{i+1}, and $Z_{i+1}$are fuzzy sets. T_i and T_{i+1}stand for the numerous feature values that were gathered in the previous step. Parameters $s_i, t_i, u_i, s_{i+1}, t_{i+1}$, and $u_{i+1}$are likely to alter during training. For better outcomes, the interference approach is utilised to fine-tune the required parameters. The next section provides a brief overview of the interference strategy for optimising parameters in fuzzy sets. The fuzzy cluster's several layers are detailed below.

Layer 1:Fuzzification refers to the most superficial level. The input values and membership function (MF) are calculated by the fuzzification layer using equation (6):

$$L_{1,j} = \mu_j(T_j) \quad (6)$$

where:

$$F_j - input\ j\ \ \mu_{f_j} - membership\ function$$

The MF input represents the membership level, which is output by each layer. This framework's membership function is the Gaussian kernel, which is described by the following equation (7).

$$\mu_j = exp\left(-\frac{\|s_j - t_j\|^2}{2u_j^2}\right) \quad (7)$$

where s_j, t_j, and $u_j$are the MF parameters in charge of determining the membership function's. The (s_j, t_j, and u_j)Premise parameters are the parameters of a membership function.

Layer 2: The rules' strengths are generated by this rule layer. The output for this layer ($L_{2,j}$) is represented by the product of all the input datas, as shown in equation (8).

$$L_{2,j} = H_j = \mu_{f_j}(T_j) \times \mu_{f_j}(T_{j+1}) \quad (8)$$

Layer 3: The data features has been normalised by this layer. The normalisation, H _i, is calculated at Node i as per equation (9).

$$L_{3,j} = I^-_j = \frac{I_j}{I_1+I_2+I_3+I_4+I_5+I_6}, \quad (9)$$

$$i = 1,2 \ldots 6$$

Layer 4:The fourth layer (a collection of parameter values for the results) receives the adjusted values. The adaptive node's layer-4 node function is represented by Equation (10).

$$L_{4,i} = I^-_j \cdot Rules_j \quad (10)$$

where

I^{-}_{j} −normalized data features

Rules $_{j}$ - system rules

Layer 5: To create the final output, layer five receives the defuzzified data from layer four. The sum of all the input data is used to calculate the overall result. The circular node in this layer is labelled as follows, as indicated in equation (11):

$$L_{5,i} = \sum_j \quad I_i \, Rules_j = \frac{\sum_j \quad I_j \, Rules_j}{\sum_{hj} \quad I_j} \qquad (11)$$

Equation (12) calculates the Grouped Variable as follows.:

$$\omega = \frac{\delta_{\text{final}}}{\delta_0} \qquad (12)$$

The difference is 1, as the process iterates, therefore assuming $\delta_0 = 0$ results in equation (13).

$$V_j^{\,j} = \frac{1}{2}\left(V_j^{\,j} + \eta V_j^{\,j-1}\right), j \geq 2 \qquad (13)$$

Equation (14) demonstrates how to calculate the truth interference fuzzy clustering's total running time.

$$O(t(d * M + COF * M) + O(1) + O(M) + O(M^2) \approx O(M^2) \qquad (14)$$

Finally the malware related features are isolated and grouped together.

c. Malware Classification

The OHMN was used to identify the most prevalent malware combinations. Our primary focus is to discern potentially detrimental characteristics rather than to ascertain their precise geographical positions. A convolution network-based architecture was used in order to enhance the ability to capture global location independence.Here we are integrating the structure of the memory neural network with the CNN for prediction of theandroid malware was consider as the novelty of the OHMN.

The OHMN was provided with sequences of layers that can be depicted below. The input was first processed by an embedding layer, which turned each byte into a feature vector. The aforementioned vectors were used throughout the convolutional process. The convolutional section consists of the first convolutional layer and the final max-pooling layer. Our technique uses the Rectified Linear Activation Function (ReLU). The feature vectors were used for the purpose of extracting specialised features in the convolutional layer. The visual representation of the features obtained from the first convolutional layer may be interpreted as n-grams derived from the training data. In this scenario, the appropriate number for n is determined by the width of the convolution kernel. The dimensionality of the malware features is reduced by the use of max-pooling after each convolutional layer. The detection approach we propose is capable of accommodating a wide range of convolutional layers, including those with arbitrarily high sizes. Following the conclusion of the last convolutional layer, a concealed fully connected layer is used to discern and establish significant connections of higher complexity among the extracted data. The probability of the labels are then computed by using a softmax layer. During the training process, backpropagation is used to automatically adjust the bulk of the parameters inside the neural network.

a) Embedding Layer: The primary goal of the embedding layer is to separate instructions that have significantly different goals, while also ensuring that instructions with comparable objectives and semantic information are placed in close proximity to one another.

In the first layer of the representation, a singular hot vector xn is used to symbolise each command group. Subsequently, we proceed to write one-hot vectors for each of the n following instances$\{y1, y2, \ldots, yn\}$. Due to the fact that the index of instruction groups might range from 1 to 206, it is necessary for each hot vector to have 206 items. The magnitude of a singular vector is denoted as K and measures 206 units in length. To further reduce the length of the new feature vector, the initial hot-coded vector was subjected to multiplication by a weight matrix, denoted as v_E, of dimensions D*K.

The entire technique is shown by the use of Formula 15.

$$Y = y * v_E \quad (15)$$

Upon completion, the whole of the process may be symbolically represented by the matrix Y. The dimension of Y is n times greater than that of the embedding space E. Following the embedding layer, instructions with unique functions will be projected to distant locations in the embedding space, while instructions with similar functionalities will be projected to adjacent spots. It is crucial to acknowledge that the variable E has significant importance in the interpretation of the classification results. The selection of dimensions is contingent upon the specific data sets that will be used. In many cases, it is essential to include a larger dimensionality in order to accommodate the increased volume of data used for training purposes.

b) A stratum of convolutions: Convolutional layers are used for the purpose of extracting several features from the input data. A network including several convolutional layers has the capability to iteratively extract more intricate characteristics from smaller input data. The proposed architectural design facilitates the integration of many layers of convolutional processing. The convolutional layers are sequentially labelled from 1 to L. The embedding matrix Y, which has dimensions n*E, is inputted into the first convolutional layer. The number of convolution filters used in the layer is represented by the symbol m1 for the lth convolutional layer. The size of the filter in the first convolutional layer is denoted as s1*E, where s1 is the maximum length of executable instructions that can be accurately identified. The deeper convolutional layers receive input from the preceding layer and use their own output as input. The dimensions of the filter used in the subsequent convolutional layer are determined by the product of the spatial dimensions of the input feature map in the previous layer, denoted as sl, and the number of channels in that feature map, denoted as m(l-1). Once convolution filters are applied to the variable Y, the activation function is then applied to the data in order to preserve relevant characteristics while reducing unnecessary information. In this particular context, the ReLU (Rectified Linear Unit) or rectified linear activation function is used.

$$Re\,LU = max(0, x) \quad (16)$$

Each convolutional filter in convolution layer l (of size m) generates an activation map $a_{l,m}$ of size n*1. $W_{l,m}$ and $b_{l,m}$ is the mth convolutional filter in layer l, and its weight and bias parameters are and. In the first Convolutional layer, for instance, Conv indicates the convolutional operation of the filter on the input matrix Y.

$$B_{l,m} = \mathrm{Re\,LU}\left(\mathrm{Conv}(Y)_{W_{l,m},b_{l,m}}\right) \quad (17)$$

The resulting matrix, denoted as A_1, is formed by aggregating the activation maps produced by $n * m_l$ convolution filters. The matrix A_L represents the output of the last convolutional layer. After the final convolutional layer, a maxpooling layer is implemented. The max-pooling technique produces a vector that comprises the maximum value extracted from each activation map inside the last layer, denoted as A_L. The creation of the max-pooling layer was motivated by two principal objectives. One aspect that is considered is the maintenance of sample invariance. An alternative approach involves limiting the input to the subsequent layer while ensuring a uniform output length. After the inclusion of the max-pooling layer, we successfully generated a β of magnitude m_L..

$$\beta = \left(\max(B_{L,1})|\max(B_{L,2})| \ldots | \max(B_{L,m})\right) \quad (18)$$

c) Hidden Layer and Output Layer:

Using the max-pooling approach, a vector β of length m_L the data was produced/generated. After being retrieved from the convolutional layer, the vector is sent to a fully-connected hidden layer, where it might potentially establish connections with the extracted features. The activation function of the buried layer remains as Rectified Linear Unit (ReLU).

V_h and b_h the weight and bias of the fully-connected hidden layer are denoted as the parameters that reflect its characteristics. The expression denoted as Formula 19 encapsulates this concept.

$$H_L = \mathrm{Re\,LU}(V_h\beta + b_h) \quad (19)$$

H_L is the resulting vector is the output vector. Upon transitioning from the hidden layer to the softmax layer, the vector undergoes a process of normalisation. The size of the output vectors of the softmax layer is dependent on the number of classes. In this study, our primary objective is to ascertain if the programme has negative or neutral

effects. As a result, the softmax layer generates a vector with two dimensions. The designation of the variable is denoted as "h."

$$h_\Theta(z) = \frac{1}{e^{\Theta_1^T z+b_1}+e^{\Theta_2^T z+b_2}}\left[e^{\Theta_1^T z+b_1}\ e^{\Theta_2^T z+b_2}\right] \quad (20)$$

The probability of the current sample belonging to each class is represented by each element of h. The probability of the malware opcode data is calculated using the following method:,

$$Q(o=1) = \frac{e^{\Theta_1^T z+b_1}}{e^{\Theta_1^T z+b_1}+e^{\Theta_2^T z+b_2}} = \frac{1}{1+e^{(\theta_2-\theta_1)^T z+(b_2-b_1)}} \quad (21)$$

Where θ_i and $b_i$are the parameters of the classifier specific to the $O_{1,2}$ class. In order to construct the OpCodes graph, it is necessary to compute the edge values, which indicate the probability that the original code is malicious. When $V_j$and V_i adhere to the OpCode sequence of the given sample, $E_{i,j}$is commonly computed by adding 1 to the preceding value of $E_{i,j}$.

If the value of the edge between i and j is 0.2, the total $E_{i,j}$ is 0.2, according to Formulation(22).

$\mathrm{OpCode_i} = \text{call}$ and $\mathrm{OpCode_j} = \text{sub}$ is 0.2.

$$\mathrm{E_{i,j}} = \sum_{O\in So} \frac{2}{1+\alpha*e^{\min(|s-t-1|)}} \quad (22)$$

$$O = \{\text{"index of all OpCode appearances"}\ V_{V_i}\ \text{"Sequence of sample's OpCode "}\}\ t\epsilon\{\text{"index of all OpCode appearances"}\ e_{V_j}\ \text{" sampl} \quad (23)$$

The malware codes are identified precisely using the layers of the OHMN.

d) Cost Function: Formula 24 represents the softmax regression's cost function,

$$\mathrm{C} = -\frac{1}{\mathrm{b}}\sum_{\mathrm{i=1}}^{\mathrm{n}} \sum_{\mathrm{o=1}}^{\mathrm{O}} 1\{\text{label}_{\mathrm{i}} = \mathrm{o}\}\log p(\text{label}_{\mathrm{i}} = \mathrm{o} \mid \mathrm{H_i})) \quad (24)$$

Because 1 is the indicator function, $1\{\cdot\}$ is comparable to 1 and 0 is equivalent to 1.Label$\mathrm{H_i}$ is the right label that the hidden layer produced for the ith sample. The symbol represents the neural network's weights and bias. During the training phase of a normal gradient descent implementation, we would use formula 25 to update after each iteration.

$$\delta := \delta - \alpha\frac{\partial \mathrm{C}}{\partial \delta} \quad (25)$$

The αformula (25) is used to determine the pace of learning. Training data is presented to the network in random order many times until the parameters converge.

ALGORITHM: OPCODE HIGHWAY MEMORY NETWORK

"Input: Extracted features used for training.

Output: Malware family classes

Start:

data process

Do

if

Train data

For (Class)

INITIALIZE (data array) * Size [..]

Data= $= \{y1, y2, \ldots, yn\}$

Class= mask[1..ml

return (data)

End For

Classify data

$$\delta := \delta - \alpha \frac{\partial C}{\partial \delta}$$

Append (data Patches) / Count Labels with patches))

Return labels

End For

End For

update labels {OHMN}

returns: Sample(labels)

End

End”

V. Performance Analysis

We objectively and systematically compare the opcode highway memory network to baseline models by analysing a variety of detection metrics on a testing dataset in a Python environment.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	CF
1	Flow ID	Source IP	Source Pc	Destinatio	Destinatio	Protocol	Timestam	Flow Dur	Total Fwc	Total Bacl	Total Leng	Total Leng	Fwd Pack	Fwd Pack	Fwd Pack	Fwd Pack	Label
2	10.42.0.15	10.42.0.15	58063	104.46.62.	443	6	13/06/201	209484	1	2	913	309	913	913	913	0	BENIGN
3	10.42.0.15	10.42.0.15	58063	104.46.62.	443	6	13/06/201	31308	2	0	85	0	85	0	42.5	60.10408	BENIGN
4	137.116.15	10.42.0.15	36864	137.116.19	443	6	13/06/201	13333	2	0	85	0	85	0	42.5	60.10408	BENIGN
5	137.116.15	137.116.19	443	10.42.0.15	36864	6	13/06/201	45	2	0	0	0	0	0	0	0	BENIGN
6	172.217.2.	10.42.0.15	34482	172.217.2.	443	6	13/06/201	22164	1	1	0	0	0	0	0	0	BENIGN
7	172.217.2.	10.42.0.15	56284	172.217.2.	443	6	13/06/201	22194	1	1	0	0	0	0	0	0	BENIGN
8	172.217.2.	10.42.0.15	34483	172.217.2.	443	6	13/06/201	22171	1	1	0	0	0	0	0	0	BENIGN
9	172.217.6.	10.42.0.15	43824	172.217.6.	443	6	13/06/201	36804	1	1	0	0	0	0	0	0	BENIGN
10	172.217.15	10.42.0.15	39937	172.217.19	443	6	13/06/201	118922	1	1	0	0	0	0	0	0	BENIGN
11	10.42.0.15	10.42.0.15	35047	104.41.208	443	6	13/06/201	64883	2	0	85	0	85	0	42.5	60.10408	BENIGN
12	10.42.0.15	104.41.208	443	10.42.0.15	35047	6	13/06/201	6928	2	0	0	0	0	0	0	0	BENIGN
13	172.217.0.	10.42.0.15	45711	172.217.0.	443	6	13/06/201	22358	1	1	0	0	0	0	0	0	BENIGN
14	172.217.0.	10.42.0.15	35363	172.217.0.	443	6	13/06/201	22358	1	1	0	0	0	0	0	0	BENIGN
15	172.217.3.	10.42.0.15	55798	172.217.3.	443	6	13/06/201	36704	1	1	0	0	0	0	0	0	BENIGN
16	172.217.0.	10.42.0.15	37583	172.217.0.	443	6	13/06/201	22465	1	1	0	0	0	0	0	0	BENIGN
17	172.217.0.	10.42.0.15	52935	172.217.0.	443	6	13/06/201	22461	1	1	0	0	0	0	0	0	BENIGN
18	172.217.2.	10.42.0.15	45286	172.217.2.	443	6	13/06/201	22476	1	1	0	0	0	0	0	0	BENIGN
19	172.217.3.	10.42.0.15	59415	172.217.3.	443	6	13/06/201	36254	1	1	0	0	0	0	0	0	BENIGN
20	172.217.3.	10.42.0.15	56713	172.217.3.	443	6	13/06/201	37038	1	1	0	0	0	0	0	0	BENIGN
21	172.217.0.	10.42.0.15	38200	172.217.0.	443	6	13/06/201	22541	1	2	23	0	23	23	23	0	BENIGN
22	172.217.0.	10.42.0.15	47230	172.217.0.	443	6	13/06/201	22217	1	2	23	0	23	23	23	0	BENIGN
23	172.217.0.	10.42.0.15	35101	172.217.0.	443	6	13/06/201	271	2	0	31	0	31	0	15.5	21.92031	BENIGN

test

Figure 2 Sample input

Figure 2 shows the instances of input that was provided.

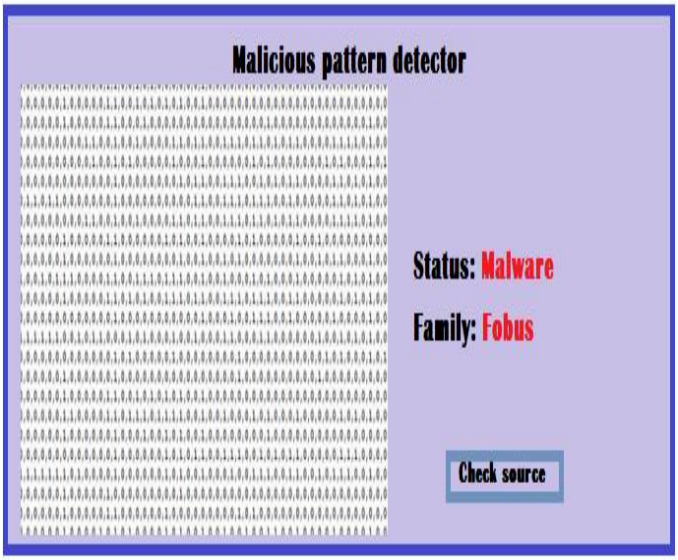

Figure 3 Simulated output

In figure 3, we can see the whole results of the simulation. Figure 3 demonstrates that the suggested method successfully classifies and clusters malware from a variety of testing sets.

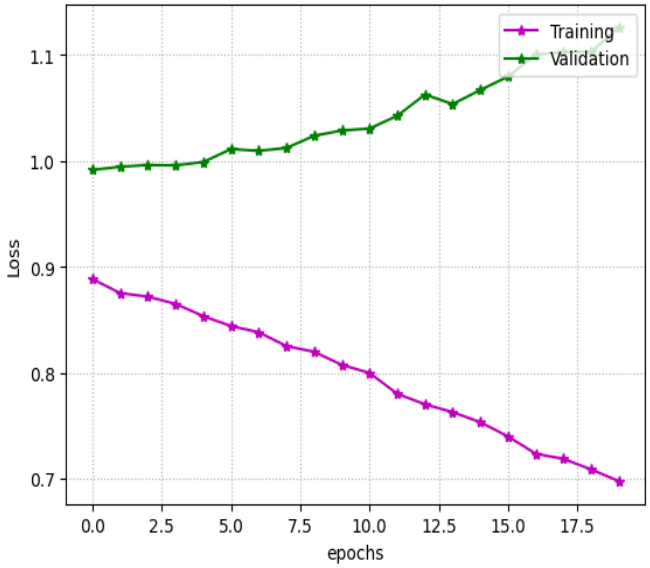

Figure 4 Epochs Vs. Loss

When applied to all of the data in an epoch, the loss function provides a quantitative estimate of the loss experienced throughout that time period. The iterative curve creation process inherently results in some degree of loss. The resulting curve shows that compared to other methods, the time and effort spent training and testing the classifier was minimal. Our model may be underfitting if there is a large discrepancy between the training loss and the validation loss. The training loss may be lowered if more data is used in the process. (either in terms of total layers or individual neuron counts inside each layer). The information in Figure 4 was used to calculate the validation loss. On the other side, the validation loss measure assesses how well a model 7006eneralizes to new data in the validation set. A subset of the data has been retrieved and labelled as the validation set in order to assess the efficiency of the model. The testing loss is computed by adding together the number of false positives seen in the training set and the validation set. The suggested method results in a dramatic decrease in losses under the prevailing conditions..

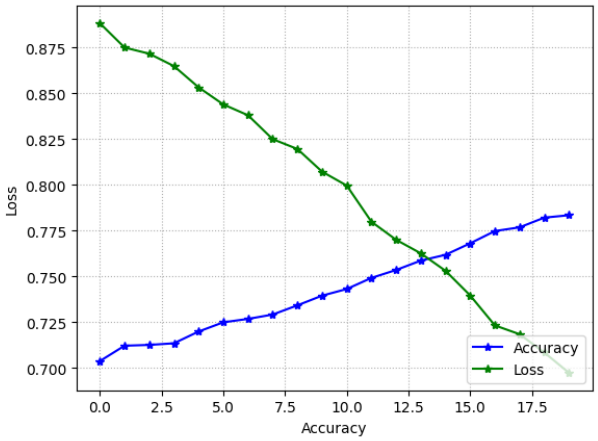

Figure 5 Accuracy Vs Loss

Figure 5 shows that validation accuracy is typically lower than training accuracy because the model is already familiar with the training data but must learn to interpret the validation data, which contains new data points.Applying the suggested classifier, as shown in Figure 5, may help with the reliable detection of perspectives.

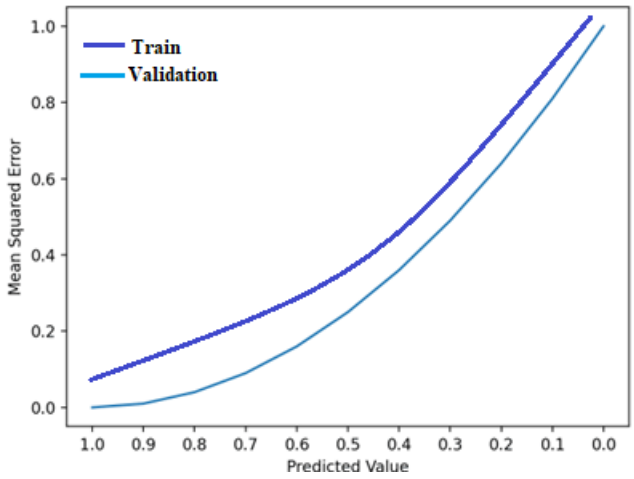

Figure 6 Error analysis

As of from the figure 6 the mean absolute error of the suggested methodology over train and test set was low prove the methodology effectiveness.

We conducted a comparison study to determine the viability of the proposed classifiers. Let TP stand for the malware sample that was properly recognised, TN for the benign sample that was correctly identified, FP for the benign sample that was erroneously identified, and FN for the malware sample that was correctly detected. Malware samples have the positive label, whereas benign samples bear the negative label. Typically, total accuracy is used to gauge a method's dependability. Four more measures are also used, including accuracy, precision,Recall, and F1.

The following definitions are provided for performance metric formulae:

- To calculate total accuracy, divide the number of samples by the fraction of samples where predictions were right.

$$\text{Accuracy} = \frac{TP+TN}{TP+FN+TN+FP} \quad (26)$$

- Precision is defined as the percentage of properly categorised positive samples divided by the total number of positive samples (sensitivity).

$$\text{Precision} = \frac{TP}{TP+FP} \quad (27)$$

- The recall statistic measures how many actual positive samples were compared to how many were predicted..

$$\text{Recall} = \frac{TP}{TP+FN} \quad (28)$$

- F1 is the harmonic mean (weighted) of precision and recall. F1 may take on the value of 1 at its most extreme and 0 at its most minimal. F1 is proportional to both accuracy and recall in a binary classification problem like the one at hand.

$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (29)$$

The sensitivity of a screening test refers to its capacity to accurately identify true positive (TP) results. This statistic quantifies the level of uncertainty present in the model's output.

Sensitivity= TP/(TP+FN)*100 (30)

- Specificity refers to the capacity of a screening test to accurately identify true negatives (TN).

Sensitivity= TN/(FP+TN)*100 (31)

Table 1 Performance analysis of the suggested methodology

Method	Year	Performance metrics			
		Acc	P	R	F1
Bi-LSTM+GCNs [25]	2017	99.23	99.54	99.11	99.32
AndrEnsemble [27]	2019	96.40	96.35	97.12	96.73
IndRNN+GCNs [28]	2020	99.15	98.82	99.21	99.01
$Bi-LSTM+CNN[26]$	2021	98.68	99.43	99.25	99.33
Bi-GRU+Deep_TNN+Self-attention[22]	2022	99.62	99.67	99.50	99.58
Proposed	-	99.8	99.88	99.9	99.8

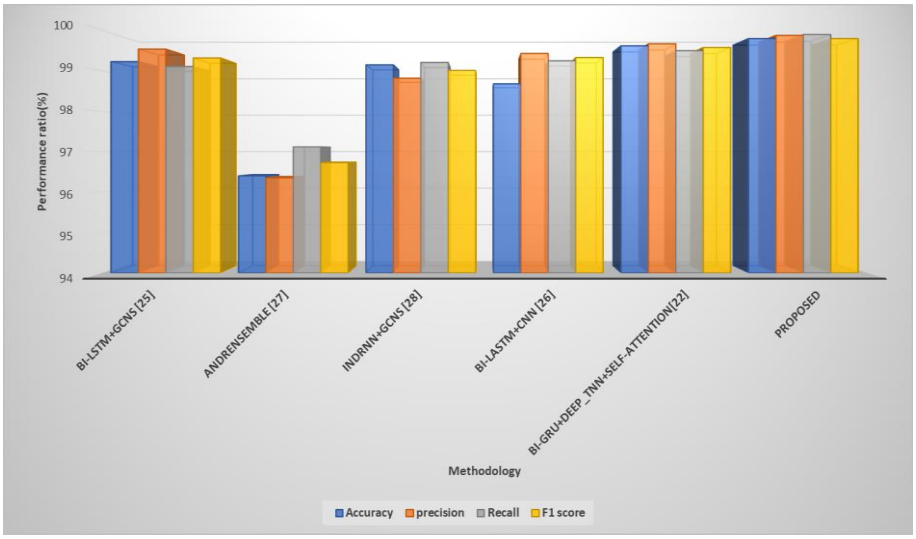

Figure 7 Comparative performance analysis

Table 1 and figure 7 show the methodology performance. Comparing the proposed technique to known mechanisms illustrates that the suggested methodology outperforms by obtaining high range of precison , accuracy, recall and F1 score.

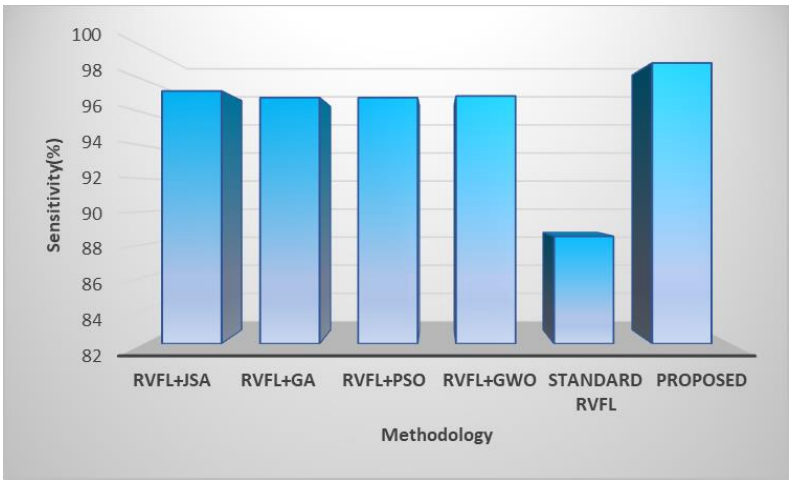

Figure 8 Sensitivity analysis

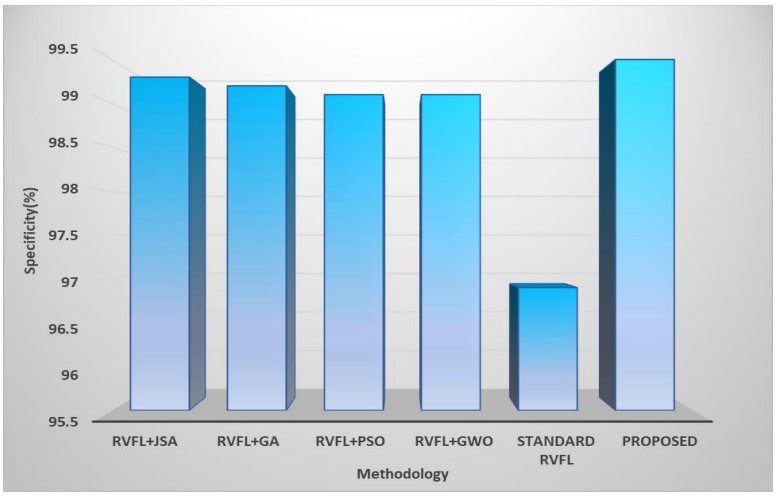

Figure 9 Specificity analysis

Based on the findings shown in Figure 8 and Figure 9, the proposed approach demonstrates a notable degree of sensitivity and specificity in comparison to other mechanisms currently used.

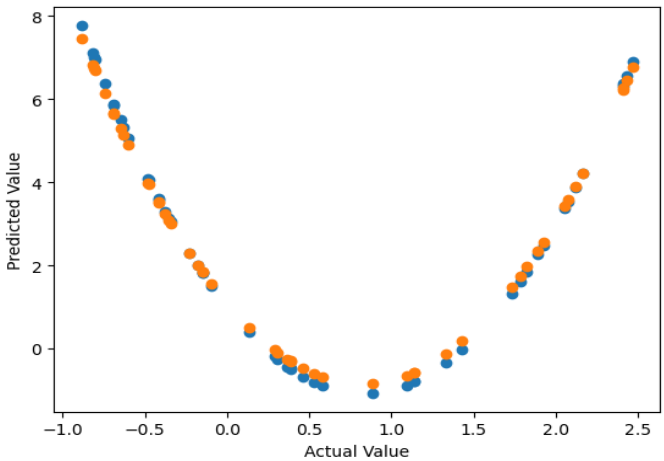

Figure 10. Simulated vulnerability data prediction output

The simulated output of the vulnerable values in the dataset by the suggested algorithm was demonstrated using a sample that was illustrated in figure 10.

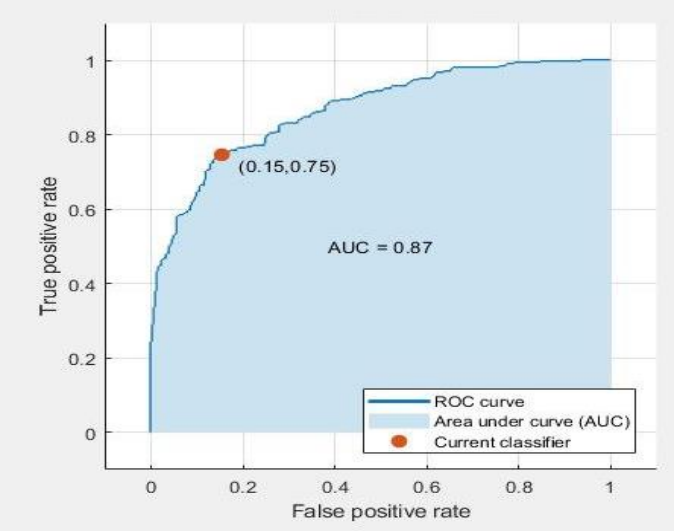

Figure 11 AUC analysis

The ROC curve is shown in a two-dimensional style, as seen in Figure 11. The threshold exhibits a continuum between 0 and 1, with higher values positioned towards the right end and lower values positioned towards the left end. The x-axis is indicative of positive rates, while the y-axis corresponds to true positive rates. The outcomes of each threshold categorization are visually shown on a graph. An AUC score of 87% indicates that the classifier demonstrates a considerable degree of accuracy, expected to be about 99%.

Table 2(a)Comparison with existingworks (Refer table 1 in [23] and [24])

"Year	Analysis	Dataset	No. of Class	Method	Accuracy
2020	Dynamic	CICMalDroid2020	5	Semi Supervised PLDNN	97.840
2021	Static	Malgenome	2	KNN, SVM	99.0
		- 14		NB, DT	
	Static	CICMalDroid2020	2	KNN, SVM	86.00
				NB, DT	
2021	Static	Drebin	2	LSTM	92.940
	Dynamic	CICMaldroid 2020	2	LSTM, SMOTE	94.220
2022	Static	CICMalDroid2020	2	DNN	98.180
	Dynamic	CICMalDroid2020	2	DNN	93.50
	Static	CCCS-CIC-AndMal-2020	2	DNN	97.720
	Dynamic	CCCS-CIC-AndMal-2020	14	DNN	78.820
2022	Static	CICMalDroid2020	2	Hist GB, SMOTE	94.090
	Static	CICMalDroid2020	2	Hist GB	95.250
2022	Static	CICMaldroid 2020	2	XGBoost	95.30
2022	Dynamic	CICMalDroid2020	4	Ensemble of NB, SVM	99.110
	o			DT, LR, RF	
2021	Static	CICMalDroid2020	2	CNN	99.0
2022	Dynamic	CICMalDroid2020	5	K-means, PCA	88.0
2022	Static	CICMalDroid2020	2	Ontological Semantic	91.0
				Environment	
2020	Dynamic	CICMalDroid2020	2	GB	99.350
	Static	Drebin	2	GB	96.350
2020	Static	CICMalDroid2020	2	CNN	95.90
2021	Static	CCCS-CIC-AndMal-2020	14	Random Forest	89.0
2020	Dynamic	CCCS-CIC-AndMal-2020	12	Semi Supervised Deep	93.360
				Image Learning	
	Dynamic	CCCS-CIC-AndMal-2020	12	Ensemble ML	95.0
2023	-	android Package Kit bytecode	-	LSTM GAN	99.0
-	-	Malradar	12	Opcode highway memory network	99.850"

Table 2(b)Both Manifest and Code based Static Analysis with ML.(Refer table 4 in [30]

Year	Detection Approach	Feature Extraction Method	Used Datasets	ML Algorithms/ Models	Selected ML Algorithms/ Models	Model Accuracy	Strengths	Limitations/Drawbacks
2017	Using customized method named Waffle Director	Manifest Analysis for Sensitive permissions and API calls	Tencent, YingYong Bao, Contagio	DT, Neural Network, SVM, NB, ELM	ELM	97.06 %	Fast Learning speed and Minimal human intervention	Combination of permissions and API calls are not refined
2017	Using a code-heterogeneity-analysis framework to classify Android repackaged malware by Smali code intermediate representation	Manifest Analysis for Intents, Permissions and API calls	Genome, Virus-Share, Benign App	RF, KNN, DT, SVM	RF with custom model proposed	FNR-0.35%, FPR-2.96%	Provide in-depth and fine-grained behavioural analysis and classification on programs	Detection issues can happen when the malware use coding techniques like reflection and cannot handle if the encryption techniques used in DEX
2018	Extracting features and transforming into binary vectors and training using ML with RanDroid Framework	Manifest Analysis for Permissions Code Analysis for API calls, opcode and native calls	Drebin	SVM, DT, RF NBs	DT	97.7%	Highly accurate to analyse permission, API calls, opcode an native calls toward malware detection	Broadcast receivers, filtered intend, Control Flow Graph analysis, deep native code analysis were not considered
2018	Creating the binary vector, apply ML models, evaluate performance of the features and their	Manifest analysis for permissions, code analysis for API calls and system	Google Play, AnZhi, LenovoMM, Wandoujia	SVM, KNN, RF	SVM	98.4%	Characterises the static behaviours of apps with ensemble of string and	Mechanism will fail if the malware contains encryption, anti-disassembly, or kernel-level features to evade the detection

Year	Detection Approach	Feature Extraction Method	Used Datasets	ML Algorithms/ Models	Selected ML Algorithms/ Models	Model Accuracy	Strengths	Limitations/Drawbacks
	ensemble using DroidEnsemble	calls analysis					structural features.	
2019	Extracting applications features from manifest while decompiling classes.dex into jar file and applying ML models	Manifest Analysis for permissions, activities and Code Analysis for Opcode	Drebin, playstore, Genome	KNN, SVM, BayesNet, NB, LR, J48, RT, RF, AB	RF with 1000 decision trees	98.7%	High efficiency, Lightweight analysis and fully automated approach	Did not consider about the API calls and other important features when analysing the DEX.
2019	Using FlowDroid for static analysis and proposing TFDroid framework to detect malware using sensitive data flow analysis	Manifest Analysis for permission and Code Analysis for information flow	Drebin, Google Play	SVM	SVM	93.7%	Analysed the functions of applications by their descriptions to check the data flow.	Did not consider the improving clustering techniques and applicability of other ML models

Therefore, based on the collected results, the proposed technique demonstrates satisfactory outcomes in predicting malware families compared to other current mechanisms in use".

VI. CONCLUSION

This study presents a novel deep learning model that integrates clustering and classification methodologies to effectively classify the predominant types of malware in a recently published dataset. This dataset has a significant compilation of both benign and malicious patterns. This research provides a strategy for preparing data that works with a wide variety of dynamic analysis tools. Features with significant error margins were disregarded during outlier management and feature ranking. Using fuzzy based clustering, the features are further categorized in a more compact manner. The accuracy increased by 0.99% and complexity was lowered by reducing the number of characteristics. Our suggested opcode highway memory network paradigm outperforms all other conventional architectures in terms of performance metrics. In the future, we'll use this technique on more newly available datasets to evaluate its effectiveness.

REFERENCES

[1] K. Khariwal, J. Singh, and A. Arora, "IPDroid: Android malware detection using intents and permissions," in 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), 2020, pp. 197-202.

[2] J. Wang, Q. Jing, J. Gao, and X. Qiu, "SEdroid: A robust Android malware detector using selective ensemble learning," in 2020 IEEE wireless communications and networking conference (WCNC), 2020, pp. 1-5.

[3] A. Mahindru and A. Sangal, "MLDroid—framework for Android malware detection using machine learning techniques," Neural Computing and Applications, vol. 33, pp. 5183-5240, 2021.

[4] L. Sun, Z. Li, Q. Yan, W. Srisa-an, and Y. Pan, "SigPID: significant permission identification for android malware detection," in 2016 11th international conference on malicious and unwanted software (MALWARE), 2016, pp. 1-8.

[5] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "DroidEnsemble: Detecting Android malicious applications with ensemble of string and structural static features," IEEE Access, vol. 6, pp. 31798-31807, 2018.

[6] N. Al Sarah, F. Y. Rifat, M. S. Hossain, and H. S. Narman, "An efficient android malware prediction using Ensemble machine learning algorithms," Procedia Computer Science, vol. 191, pp. 184-191, 2021.

[7] A. T. Kabakus, "DroidMalwareDetector: A novel Android malware detection framework based on convolutional neural network," Expert Systems with Applications, vol. 206, p. 117833, 2022.

[8] D. V. Nguyen, G. L. Nguyen, T. T. Nguyen, A. H. Ngo, and G. T. Pham, "Minad: Multi-inputs neural network based on application structure for android malware detection," Peer-to-Peer Networking and Applications, pp. 1-15, 2022.

[9] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," IEEE access, vol. 7, pp. 21235-21245, 2019.

[10] X. Liu, X. Du, X. Zhang, Q. Zhu, H. Wang, and M. Guizani, "Adversarial samples on android malware detection systems for IoT systems," Sensors, vol. 19, p. 974, 2019.

[11] R. S. Arslan, İ. A. Doğru, and N. Barişçi, "Permission-based malware detection system for android using machine learning techniques," International journal of software engineering and knowledge engineering, vol. 29, pp. 43-61, 2019.

[12] S. Lou, S. Cheng, J. Huang, and F. Jiang, "TFDroid: Android malware detection by topics and sensitive data flows using machine learning techniques," in 2019 IEEE 2Nd international conference on information and computer technologies (ICICT), 2019, pp. 30-36.

[13] S. R. Tiwari and R. U. Shukla, "An android malware detection technique based on optimized permissions and API," in 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), 2018, pp. 258-263.

[14] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient Android malware detection system based on method-level behavioral semantic analysis," IEEE Access, vol. 7, pp. 69246-69256, 2019.

[15] E. Mariconti, O. Lucky, and P. Andriotis, "Detecting Android Malware by Building Markov Chains of Behavioural Models," in NDDS'17, ed, 2017.

[16] S. Y. Yerima and S. Khan, "Longitudinal performance analysis of machine learning based Android malware detectors," in 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), 2019, pp. 1-8.

[17] M. Fan, X. Luo, J. Liu, M. Wang, C. Nong, Q. Zheng, et al., "Graph embedding based familial analysis of android malware using unsupervised learning," in 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 771-782.

[18] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," computers & security, vol. 65, pp. 121-134, 2017.

[19] K. Xu, Y. Li, and R. H. Deng, "Iccdetector: Icc-based malware detection on android," IEEE Transactions on Information Forensics and Security, vol. 11, pp. 1252-1264, 2016.

[20] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of android malware," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 26, pp. 1-29, 2018.

[21] H. Li, S. Zhou, W. Yuan, J. Li, and H. Leung, "Adversarial-example attacks toward android malware detection system," IEEE Systems Journal, vol. 14, pp. 653-656, 2019.

[22] C. Zhang, Q. Zhou, Y. Huang, K. Tang, H. Gui, and F. Liu, "Automatic detection of Android malware via hybrid graph neural network," Wireless Communications and Mobile Computing, vol. 2022, 2022.

[23] Islam, R., Sayed, M. I., Saha, S., Hossain, M. J., & Masud, M. A. (2023). Android malware classification using optimum feature selection and ensemble machine learning. Internet of Things and Cyber-Physical Systems, 3, 100-111.

[24] Amin, M., Shah, B., Sharif, A., Ali, T., Kim, K. I., & Anwar, S. (2022). Android malware detection through generative adversarial networks. Transactions on Emerging Telecommunications Technologies, 33(2), e3675.

[25] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pp. 1506–1515, Copenhagen, Denmark, 2017.

[26] [I. U. Haq, T. A. Khan, and A. Akhunzada, "A dynamic robust DL-based model for android malware detection," IEEE Access, vol. 9, pp. 74510–74521, 2021.

[27] O. Mirzaei, G. Suarez-Tangil, J. M. de Fuentes, J. Tapiador, and G. Stringhini, "Andrensemble: leveraging api ensembles to characterize android malware families," in Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, pp. 307–314, Auckland, New Zealand, 2019.
[28] X. Pei, L. Yu, and S. Tian, "AMalNet: a deep learning framework based on graph convolutional networks for malware detection," Computers & Security, vol. 93, article 101792, 2020.
[29] Elkabbash, E. T., Mostafa, R. R., & Barakat, S. I. (2021). Android malware classification based on random vector functional link and artificial Jellyfish Search optimizer. PloS one, 16(11), e0260232.
[30] Senanayake, J., Kalutarage, H., & Al-Kadri, M. O. (2021). Android mobile malware detection using machine learning: A systematic review. Electronics, 10(13), 1606.