ELMNT: INTELLIGENT MALWARE DETECTION AND CLASSIFICATION USING EXTREME LEARNING MACHINE CLASSIFIER

D Naresh[1], P. Abhinandh[2], A. Sloka[2], P. Krishna[2], P. Rithvik Goud[2]

[1,2]Department of Computer Science and Engineering

[1,2] Sree Dattha Group of Institutions, Sheriguda, Telangana.

ABSTRACT

Security breaches due to attacks by malicious software (malware) continue to escalate posing a major security concern in this digital age. With many computer users, corporations, and governments affected due to an exponential growth in malware attacks, malware detection continues to be a hot research topic. Current malware detection solutions that adopt the static and dynamic analysis of malware signatures and behavior patterns are time consuming and have proven to be ineffective in identifying unknown malwares in real-time. Recent malwares use polymorphic, metamorphic, and other evasive techniques to change the malware behaviors quickly and to generate a large number of new malwares. Such new malwares are predominantly variants of existing malwares, and machine learning algorithms (MLAs) are being employed recently to conduct an effective malware analysis. Therefore, this work proposes the combined visualization and deep learning architectures for static, dynamic, and image processing based hybrid approach applied in a big data environment, which is the first of its kind toward achieving robust intelligent zero-day malware detection. Overall, this work paves way for an effective visual detection of malware using a scalable and hybrid extreme learning machine model named as ELMNet for real-time deployments.

Keywords: Malicious software, ELMNet, Machine learning.

1. INTRODUCTION

In this digital world of Industry 4.0, the rapid advancement of technologies has affected the daily activities in businesses as well as in personal lives. Internet of Things (IoT) and applications have led to the development of the modern concept of the information society. However, security concerns pose a major challenge in realising the benefits of this industrial revolution as cyber criminals attack individual PC's and networks for stealing confidential data for financial gains and causing denial of service to systems. Such attackers make use of malicious software or malware to cause serious threats and vulnerability of systems [1]. A malware is a computer program with the purpose of causing harm to the operating system (OS). A malware gets different names such as adware, spyware, virus, worm, trojan, rootkit, backdoor, ransomware and command and control (C&C) bot, based on its purpose and behaviour. Detection and mitigation of malware is an evolving problem in the cyber security field. As researchers develop new techniques, malware authors improve their ability to evade detection. When Morris worm made its appearance as the first ever computer virus in 1988-89, antivirus software programs were designed to detect the existence of such a malware by finding a match with the virus definition database updated from time to time. This is called signature-based malware detection, which can also perform a heuristic search to identify the behavior of malware.

However, the major challenge in such classical approaches is that new variants of malware use antivirus evasion techniques such as code obfuscation and hence such signature-based approaches are unable to detect zero-day malwares [2]. Signature-based malware detection system requires extensive domain level knowledge to reverse engineer the malware using Static and Dynamic analysis and to assign a signature for that. Moreover, signature-based system requires larger time to reverse

engineer the malware and during that time an attacker would encroach into the system. In addition, signature-based system fails to detect new types of malware. Security researchers have identified that hackers predominantly use polymorphism and metamorphism as obfuscation techniques against signature-based detection. In order to address this problem, software tools are used to manually unpack the codes and analyse the application programming interface (API) calls. Since this process is a resource intensive task, [3] presented an automated system to extract API calls and analyse the malicious characteristics using a four-step methodology. In step 1, the malware is unpacked. In step 2, the binary executable is disassembled. Step 3 involves API call extraction. Step 4 involves API call mapping and statistical feature analysis. This was enhanced in [4] using a 5- step methodology incorporating MLA such as SVM with n-gram features extracted from large samples of both the benign and malicious executables with 10-fold cross validations. Later, in [5] a comparative study of various classical machine learning classifiers for malware detection was performed, and a framework for zero-day malware detection was proposed. To handle malicious code variants, the sequence of API calls and their frequency of appearance of API calls passed into similarity-based mining and machine learning methods [7]. The detailed experimental analysis was done on very large data set and to extract the features from malware binaries a unified framework proposed. In [8], API calls features and a hybrid of support vector machine (SVM) and Maximum-Relevance Minimum Redundancy Filter (MRMRF) heuristics were employed to present novel feature selection approaches for enhanced malware detection. Recently, with the increase in unknown malware attacks, the detailed information on obfuscated malware is discussed by [6] and many researchers are improving the MLAs for malware detection [9].

However, the major issue with the classical machine learning based malware detection system is that they rely on the feature engineering, feature learning and feature representation techniques that require an extensive domain level knowledge. Moreover, once an attacker comes to know the features, the malware detector can be evaded easily. Therefore, this paper implemented a scalable deep learning network architecture for malware detection called ELMNet with the capability to leverage the application of Big Data techniques to handle vary large number of malware samples.

2. LITERATURE SURVEY

MLAs rely on the feature engineering, feature selection and feature representation methods. The set of features with a corresponding class is used to train a model in order to create a separating plane between the benign and malwares. This separating plane helps to detect a malware and categorize it into its corresponding malware family. Both feature engineering and feature selection methods require domain level knowledge. The various features can be obtained through Static and Dynamic analysis. Static analysis is a method that captures the information from the binary program without executing. Dynamic analysis is the process of monitoring malware behavior at run time in an isolated environment. The complexities and various issues of Dynamic analysis are discussed in detail by [10]. Dynamic analysis can be an efficient long-term solution for malware detection system. The Dynamic analysis cannot be deployed in end-point real time malware detection due to the reason that it takes much time to analyze its behaviour, during which malicious payload can get delivered. Malware detection methods based on Dynamic analysis are more robust to obfuscation methods when compared to statically collected data. Most commonly, the commercial anti-malware solutions use a hybrid of Static and Dynamic analysis approaches. The major issue with the classical machine learning based malware detection system is that they rely on the feature engineering, feature learning and feature representation techniques that require an extensive domain level knowledge [11], [12], [13]. Moreover, once an attacker comes to know the features, the malware detector can be evaded easily [14]. To be successful, MLAs require data with a variety of patterns of malware. The publicly available benchmark data for malware analysis research is very less due to the security and privacy

concerns. Though few datasets exist, each of them has their own harsh criticisms as most of them are outdated. Many of the published results of machine learning based malware analysis have used their own datasets. Even though publicly available sources exist to crawl the malware datasets, preparing a proper dataset for research is a daunting task. These issues are the main drawbacks behind developing generic machine learning based malware analysis system that can be deployed in real time. More importantly, the compelling issues in applying data science techniques were discussed in detail by [15].

In recent days, deep learning, which is an improved model of neural networks has outperformed the classical MLAs in many of the tasks which exist in the field of natural language processing (NLP), computer vision, speech processing and many others [16]. During the training process, it tries to capture higher level representation of features in deep hidden layers with the ability to learn from mistakes. MLAs experience diminishing outputs as they see more and more data whereas deep learning captures new patterns and establishes associations with the already captured pattern to enhance the performance of tasks. There exists few research studies towards the application of deep learning architectures for malware analysis to improve cyber security [13], [11], [12], [17], [18], [18]–[24]. However, with Industry 4.0, the number of malwares is rapidly increasing in recent times. Since the continuous collection of malwares in real time results in Big Data, the existing approaches are not scalable with very high requirements for storage and time in making efficient decisions. The absence of scalable and distributed architectures in solving malware analysis motivated the current research to investigate the algorithms and develop a scalable architecture, namely ELMNet.

3. PROPOSED METHODOLOGY

Deep learning or deep neural networks (DNNs) takes inspiration from how the brain works and forms a sub module of artificial intelligence. The main strength of deep learning architectures is the capability to understand the meaning of data when it is in large amounts and to automatically tune the derived meaning with new data without the need for a domain expert knowledge. Convolutional neural networks (CNNs) and Recurrent neural networks (RNNs) are two types of deep learning architectures predominantly applied in real-life scenarios. Generally, CNN architectures are used for spatial data and RNN architectures are used for temporal data. The combination of CNN and LSTM is used for spatial and temporal data analysis.

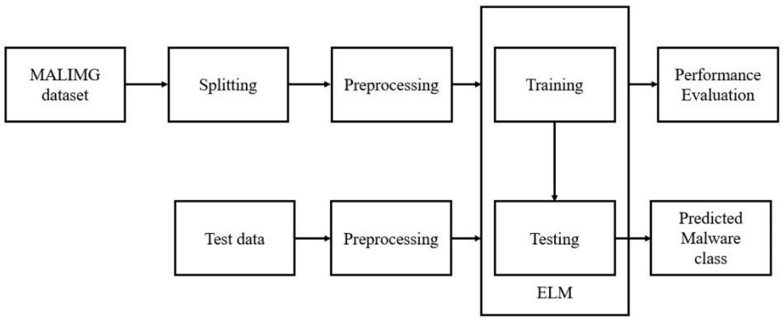


Fig. 1: Proposed block diagram.

Fig. 1 shows the block diagram of proposed method. Initially, MALIMG dataset is spitted into 80% for training and 20% for testing. Then, dataset preprocessing operation is performed to normalize the entire dataset. Further, DLCNN classifier is used for prediction of malware attack from test sample. The performance evaluation is carried out to show supremacy of proposed method.

3.1 MALIMG dataset

CICDDoS2019 contains benign and the most up-to-date common DDoS attacks, which resembles the true real-world data (PCAPs). It also includes the results of the network traffic analysis using CICFlowMeter-V3 with labeled flows based on the time stamp, source, and destination IPs, source and destination ports, protocols and attack (CSV files). Generating realistic background traffic was our top priority in building this dataset. We have used our proposed B-Profile system to profile the abstract behaviour of human interactions and generates naturalistic benign background traffic in the proposed testbed. For this dataset, we built the abstract behaviour of 25 users based on the HTTP, HTTPS, FTP, SSH and email protocols.

3.2 Preprocessing

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data preprocessing task.

Need of Data Preprocessing: A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling

3.3 Splitting the Dataset

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model. Suppose if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models. If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So, we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

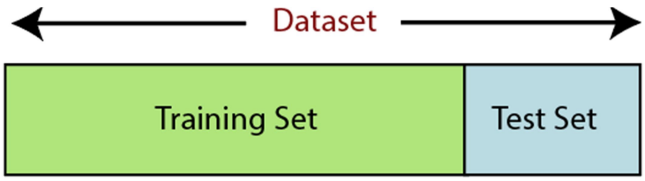


Fig. 2: Splitting the dataset.

Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)

Explanation

- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are
- x_train: features for the training data
- x_test: features for testing data
- y_train: Dependent variables for training data
- y_test: Independent variable for testing data
- In train_test_split() function, we have passed four parameters in which first two are for arrays of data, and test_size is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
- The last parameter random_state is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

3.4 ELM Prediction

ELM is a kind of advanced neural network, consists of three layers such as input layer, hidden layer (number of neurons) and an output layer. The input layer captures the input variable, hidden layers make a linear relationship among the variables and the output layer presents the predicted value. The following principle that differentiates ELM from other traditional NN is based on the parameters of the feed-forward network, inputs weights and biases provided to the hidden layer. In ELM, the bias of the hidden layer and input weight are randomly generated and the output is calculated by the Moore–Penrose generalized inverse of the hidden layer output matrix. The randomly chosen input weight and hidden layer biases learn the training samples with minimum error. After randomly choosing the input weights and the hidden layer biases, SLFNs can be simply considered as a linear system. The main advantage of ELM is its structure does not depend on network parameters which produce stability. Hence it is useful for classification, regression, and clustering.

Figure 3.5 is a representation of the MLP structure of HELM. This structure has m number of output nodes, N number of input nodes, and N number of hidden nodes, and it is subdivided into three layers. Think of the input features that were extracted from the GLCM as $x = (x_1, x_2, \dots, x_N)^T$, where N is the number of features, and it is going to be used on the input layer. A collection of weights denoted by the notation w_i=$[w_{i1}, w_{i2}, \dots, w_{iN}]^T$ is the connection that is made between each of the nodes in the input layer and the nodes in the hidden layers, where N is the total number of weights. Further, bias weights B_j =$(B_{j1}, B_{j2}, \dots, B_{jN})^T$are used in the process of interconnecting nodes of the output layer with nodes of the hidden layer.

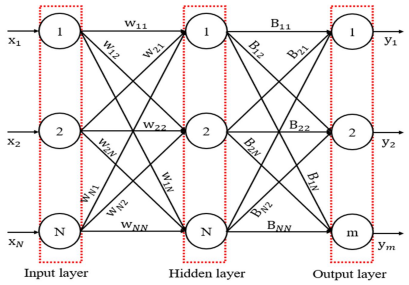


Fig. 3: MLP structure of ELM.

The process is carried out by the hidden layer using the hidden matrix H, and it is described in the following way:

$$H = g\left(w_i^T x + b_i\right) \tag{1}$$

In this context, the activation function of ELM is denoted by $g(.)$, and the bias function of ELM is denoted by b_i. Then, the convolution operation is carried out between B_j and H, which results in the generation of the anticipated vector $\bar{y} = (\bar{y}_{1,}\bar{y}_{2,} \ldots., \bar{y}_m)^T$. and the explanation behind this is as follows:

$$\bar{y} = \sum_{j=1}^{\bar{N}} B_j * H \tag{2}$$

The typical ELM uses a selection procedure in which the values for B_j and w_i are picked at random from a pool of potential values. These values are not created by a static training process. However, this led to a decrease in performance; as a consequence, this study changed a characteristic of ELM and developed ELM. ELM produces the B_j and w_i weights from new training set (s) by using the technique of reinforcement learning.

$$s = [(x_k, y_k)|x_k \in R^k, y_k \in R^m, \text{k} = [1, 2, ..., \text{K}]] \tag{3}$$

In this case, the output and input vectors for the kth training instance are denoted as $y_k = [y_{k1}, y_{k2}, \ldots., y_{kk}]$ and $x_k = [x_{k1}, x_{k2}, \ldots., x_{kk}]$, respectively. In addition, the B_j and w_i weights are developed via the process of optimising the training set. The goal function of the optimization process is represented by Equation 3.22, which has to be solved and minimized in order to provide an effective conclusion.

$$L(B, \zeta) = \frac{1}{2} \left\| B^2 \right\| + \frac{C}{2} \sum_{k=1}^{K} \left\| \zeta_k^2 \right\| \tag{4}$$

$$H(w_k) = L(yk - \zeta_k) \tag{5}$$

Here, $L()$ represents the feedback process, which is feedback to hidden layer from output layer, C represents the regularization parameter, ζ_k represents the predicted error of instance k, $h(w_k)$ represents the hyperparameter of w_i.

$$H(w_k) = \begin{bmatrix} g(w_1^T x_k + b_1) \\ g(w_2^T x_k + b_2) \\ . \\ . \\ g(w_{\bar{N}}^T x_k + b_N) \end{bmatrix} \tag{6}$$

Applying the Kuhn–Tucker conditions, such as Lagrange multipliers, serves the aim of resolving the optimization constraint that was mentioned before, and the solution that is obtained as a consequence is as follows:

$$\vartheta_{Bw} = \left(\frac{I_{\bar{N}\times\bar{N}}}{C} + \left(H(w_k)\right)^T H(w_k)\right)^{-1} \left(H(w_k)\right)^T \Upsilon \quad (7)$$

In this case, ϑ_{Bw} represents the value that has been optimised for the ϑ_{Bw}weights, $I_{\bar{N}\times\bar{N}}$ stands for an identity matrix, and Υ is the output feedback constant. Finally, the layers of ELM were updated with optimized ϑ_{Bw} weights, and the output vector y_k was produced.

$$y_k = \begin{cases} class\ A: & \rho^{class\ A} > \rho^{class\ B} \\ class\ B: & \text{else} \end{cases} \quad (8)$$

Here, $classA$ to $classB$ represents different classes of malware families. Here, $\rho^{class\ A}$ represents probability of $class\ A$ type and $\rho^{class\ B}$ represents probability of $class\ B$ type.

4. RESULTS AND DISCUSSION

To implement this project and to evaluate machine learning algorithms performance author is using binary malware dataset called 'MALIMG'. This dataset contains 25 families of malware and application will convert this binary dataset into gray images to generate train and test models for machine learning algorithms. These algorithms converting binary data to images and then generating model, so they are called as MalConv CNN and MalConv LSTM and other algorithm refers as EMBER. Application convert dataset into binary images and then used 80% dataset for training model and 20% dataset for testing. Whenever we upload new test malware binary data then application will apply new test data on train model to predict malware class. In dataset total 25 families of malware, we can see and below are their names.

'Dialer Adialer.C','Backdoor Agent.FYI','Worm Allaple.A','Worm Allaple.L','Trojan Alueron.gen','Worm:AutoIT Autorun.K',

'Trojan C2Lop.P','Trojan C2Lop.gen','Dialer Dialplatform.B','Trojan Downloader Dontovo.A','Rogue Fakerean','Dialer Instantaccess',

'PWS Lolyda.AA 1','PWS Lolyda.AA 2','PWS Lolyda.AA 3','PWS Lolyda.AT','Trojan Malex.gen','Trojan Downloader Obfuscator.AD',

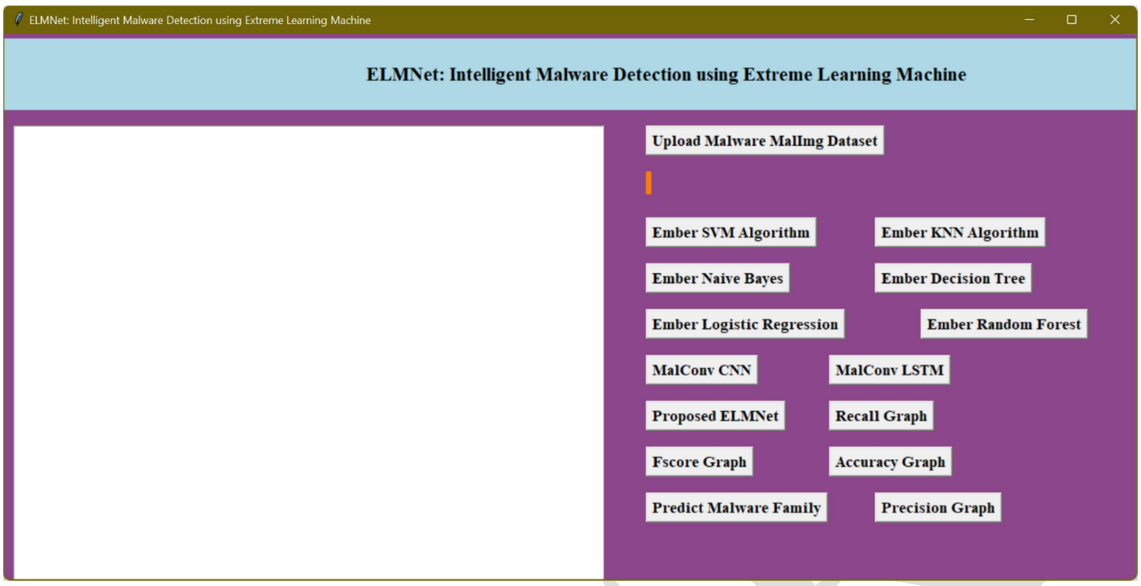
'Backdoor Rbot!gen','Trojan Skintrim.N','Trojan Downloader Swizzor.gen!E','Trojan Downloader Swizzor.gen!I','Worm VB.AT',

'Trojan Downloader Wintrim.BX','Worm Yuner.A'

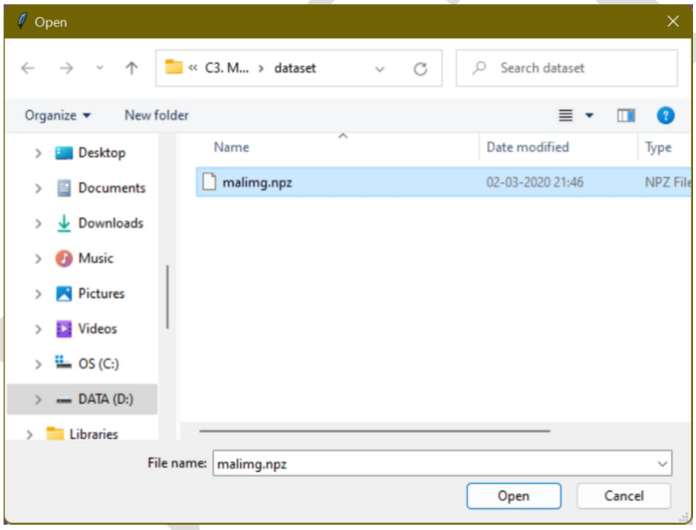
All above names are the malware families

All new malware test files I saved inside images folder, and you can upload this files to predict it class. All algorithms detail you can read from paper.

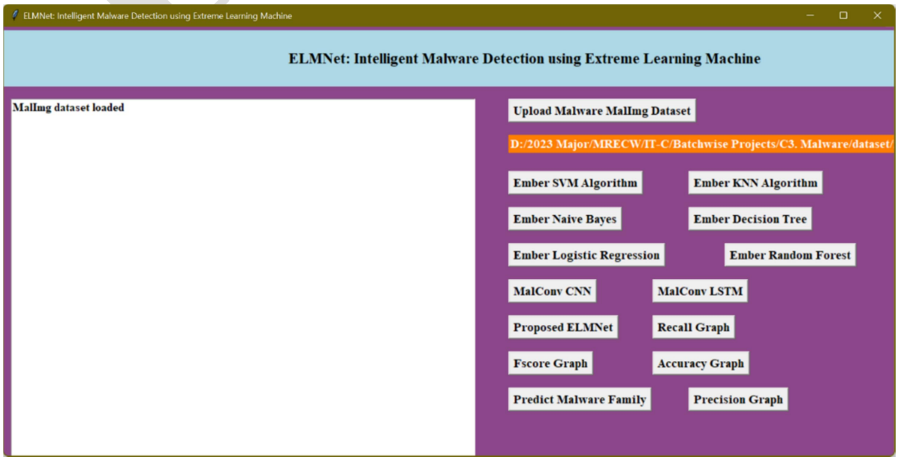
SCREEN SHOTS



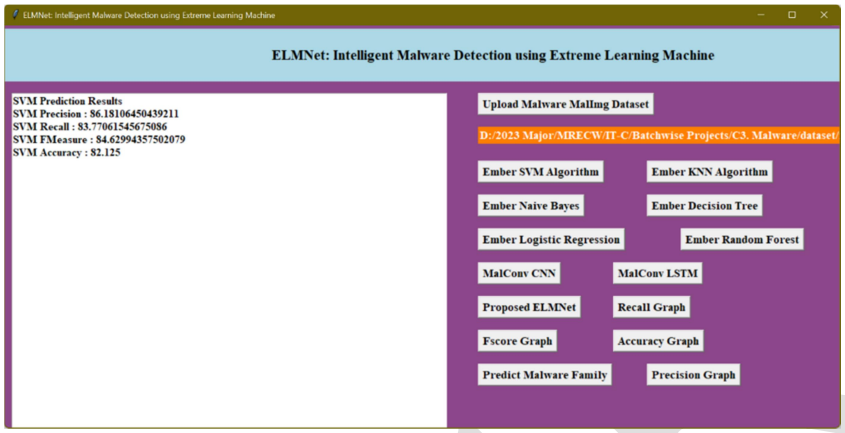
In above screen click on 'Upload Malware MalImg dataset' button to upload dataset.



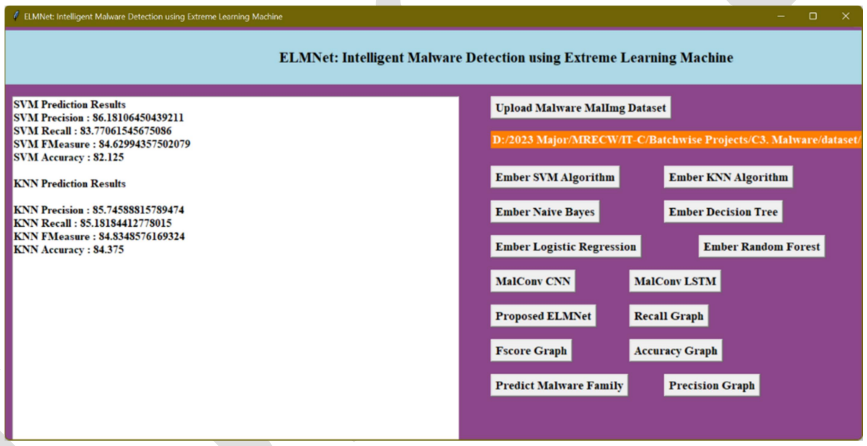
In above screen I am uploading 'malimg.npz' binary malware dataset and after uploading dataset will get below screen.



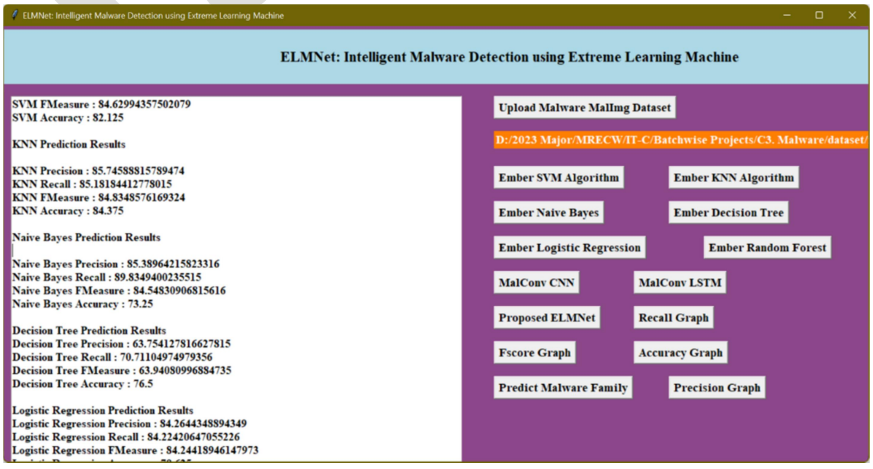
Now click on 'Ember SVM algorithm' button to read malware dataset and generate train and test model and then apply SVM algorithm to calculate its prediction accuracy, FSCORE, Precision and Recall. If algorithm performance is good then its accuracy, precision or recall values will be closer to 100.



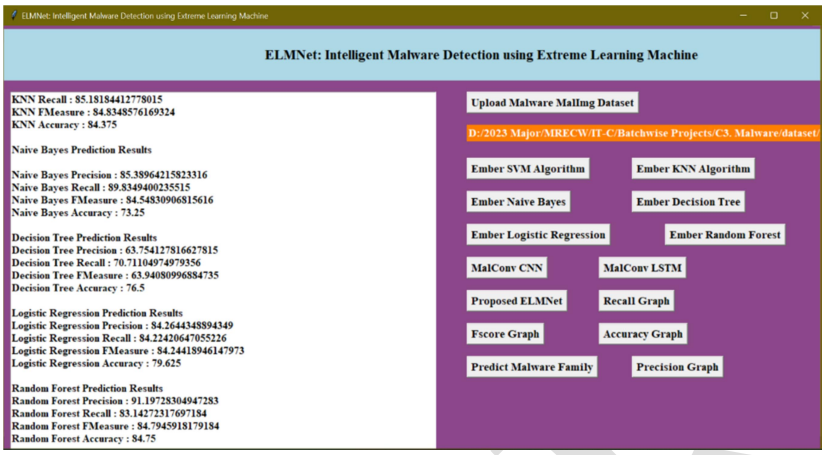
In above screen we got SVM precision, recall and F-Measure. Now click on 'Ember KNN Algorithm' button to get its performance



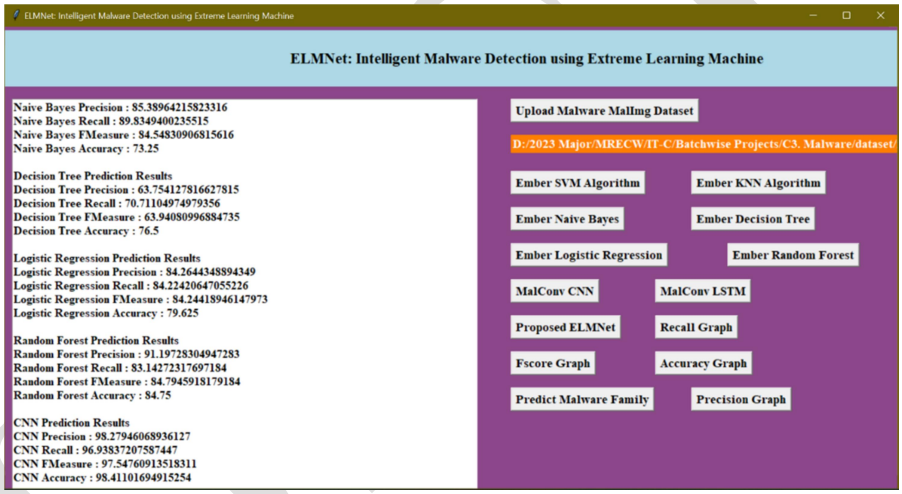
In above screen we got KNN details and now click on 'Naïve Bayes', Decision Tree and Logistic Regression buttons to get its performance details



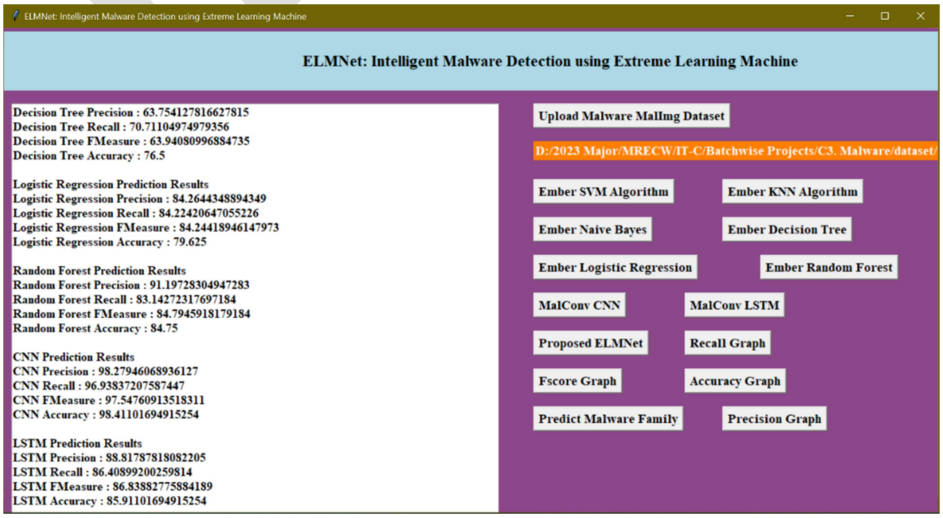
In above screen we got Naïve Bayes, Decision Tree and logistic regression details and now click on 'Random Forest' button to get its performance



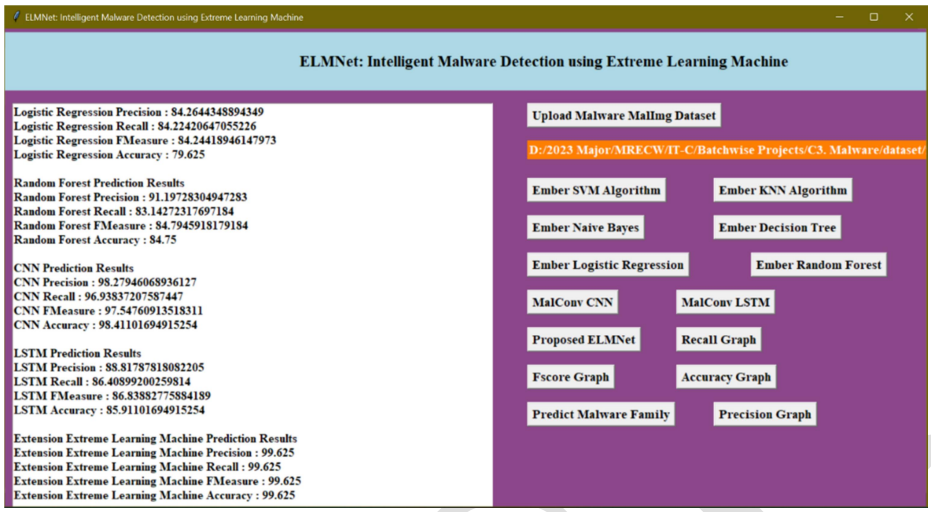
In above screen we got random forest details and now click on 'MalConv CNN' button to get its performance details.



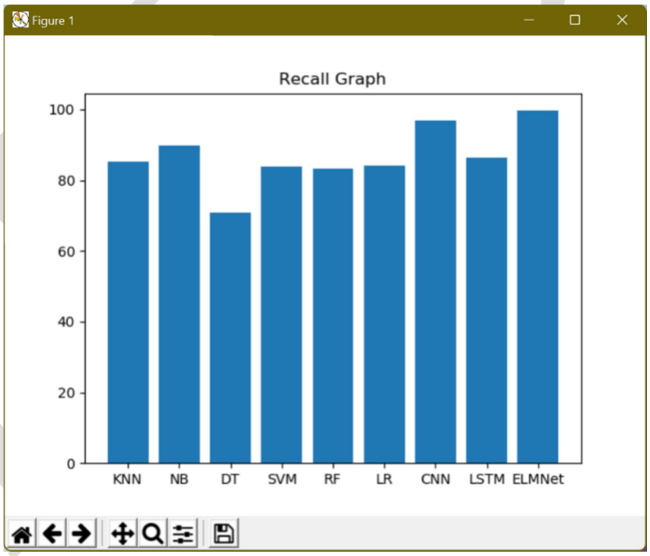
In above screen we got CNN performance values and now click on 'MalConv LSTM' button to run LSTM algorithm.

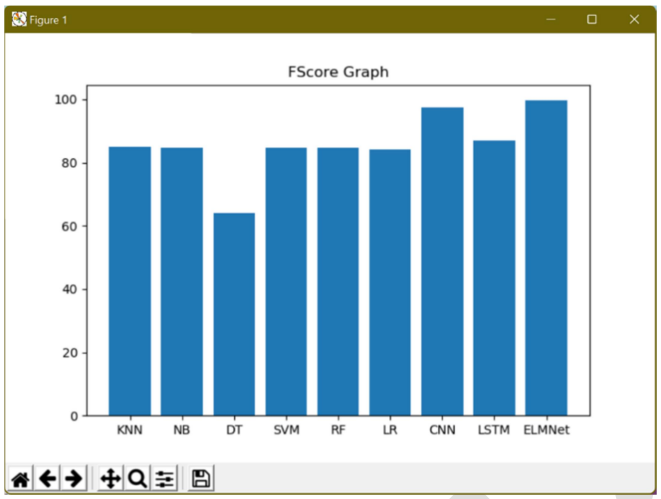


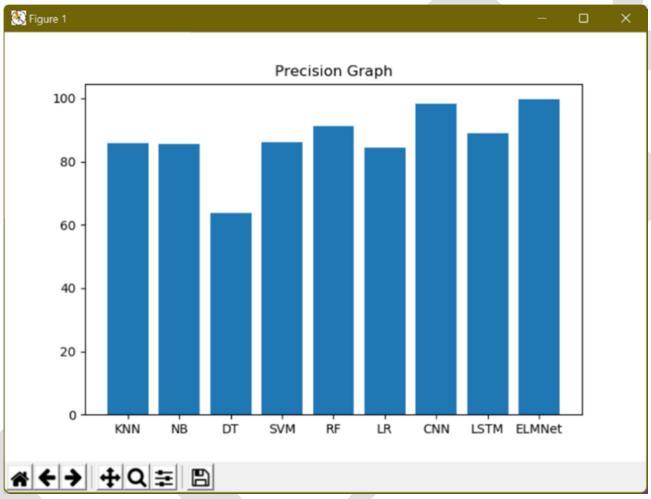
In above screen we can see LSTM details. In above screen we can see accuracy and other metrics from various algorithms where CNN got 98.41% accuracy which is higher as compared all the above algorithms. Now click on proposed ELMNet to get below output.

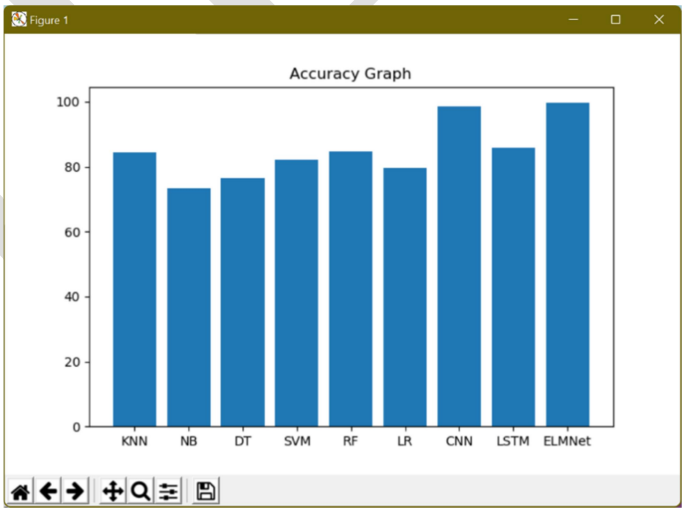


In above screen CNN got 98.41% accuracy and proposed ELMNet got 99.62% accuracy which is higher than any other algorithm. So, by employing ELM we can further increase malware prediction capability of the application. Now click on 'Precision, Recall & F-Measure' button to get comparison graph for all metrics and all algorithms

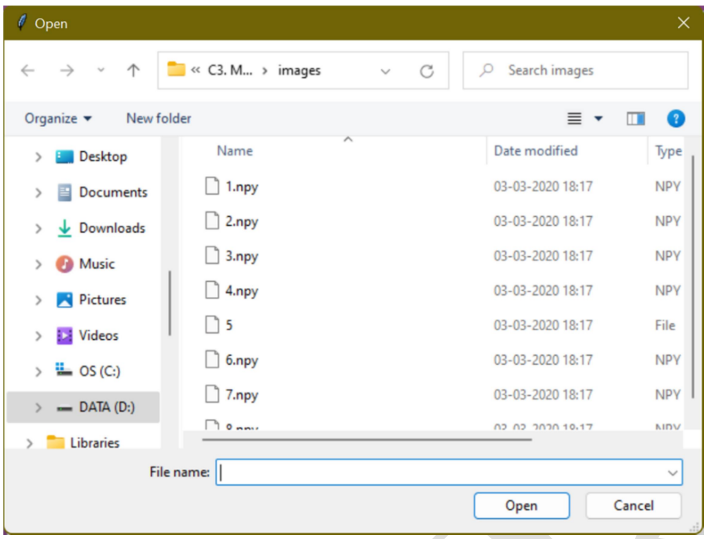




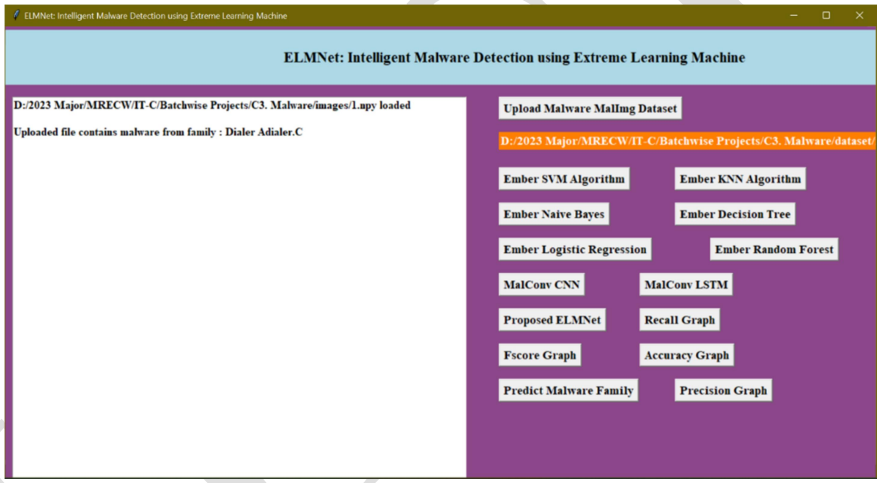




Now click on 'Predict Malware Family' button and upload binary file to get or predict class of malware.



In above graph I am uploading one binary file called 1.npy and below is the malware prediction of that file.



In above screen we can see uploaded test file contains 'Dialer Adialer.C' malware attack. Similarly, we can upload other files and predict class.

5. CONCLUSION AND FUTURE SCOPE

This project proposed an efficient malware detection and designed a highly scalable framework to detect, classify and categorize zero-day malwares. This framework applies neural network on the collected malwares from end user hosts and follows a two-stage process for malware analysis. In the first stage, a hybrid of static and dynamic analysis was applied for malware classification. In the second stage, malwares were grouped into corresponding malware categories using image processing approaches. Various experimental analysis conducted by applying variations in the models on publicly available benchmark dataset and indicated the proposed model outperformed classical MLAs. The developed framework is capable of analyzing large number of malwares in real-time and scaled out to analyse even larger number of malwares by stacking a few more layers to the existing architectures. Future research entails exploration of these variations with new features that could be added to the existing data.

REFERENCES

[1] R. Anderson et al., ‘‘Measuring the cost of cybercrime,’’ in The Economics of Information Security and Privacy. Berlin, Germany: Springer, 2013, pp. 265–300.
[2] B. Li, K. Roundy, C. Gates, and Y. Vorobeychik, ‘‘Large-scale identification of malicious singleton files,’’ in Proc. 7th ACM Conf. Data Appl. Secur. Privacy. New York, NY, USA: ACM, Mar. 2017, pp. 227–238.
[3] M. Alazab, S. Venkataraman, and P. Watters, ‘‘Towards understanding malware behaviour by the extraction of API calls,’’ in Proc. 2nd Cybercrime Trustworthy Comput. Workshop, Jul. 2010, pp. 52–59.
[4] M. Tang, M. Alazab, and Y. Luo, ‘‘Big data for cybersecurity: Vulnerability disclosure trends and dependencies,’’ IEEE Trans. Big Data, to be published.
[5] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, ‘‘Zero-day malware detection based on supervised learning algorithms of API call signatures,’’ in Proc. 9th Australas. Data Mining Conf., vol. 121. Ballarat, Australia: Australian Computer Society, Dec. 2011, pp. 171–182.
[6] M. Alazab, S. Venkatraman, P. Watters, M. Alazab, and A. Alazab, ‘‘Cybercrime: The case of obfuscated malware,’’ in Global Security, Safety and Sustainability & e-Democracy (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 99, C. K. Georgiadis, H. Jahankhani, E. Pimenidis, R. Bashroush, and A. Al-Nemrat, Eds. Berlin, Germany: Springer, 2012.
[7] M. Alazab, ‘‘Profiling and classifying the behavior of malicious codes,’’ J. Syst. Softw., vol. 100, pp. 91–102, Feb. 2015.
[8] S. Huda, J. Abawajy, M. Alazab, M. Abdollalihian, R. Islam, and J. Yearwood, ‘‘Hybrids of support vector machine wrapper and filter based framework for malware detection,’’ Future Gener. Comput. Syst., vol. 55, pp. 376–390, Feb. 2016.
[9] E. Raff, J. Sylvester, and C. Nicholas, ‘‘Learning the PE header, malware detection with minimal domain knowledge,’’ in Proc. 10th ACM Workshop Artif. Intell. Secur. New York, NY, USA: ACM, Nov. 2017, pp. 121–132.
[10] C. Rossow, et al., ‘‘Prudent practices for designing malware experiments: Status quo and outlook,’’ in Proc. IEEE Symp. Secur. Privacy (SP), Mar. 2012, pp. 65–79.
[11] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas. (2017). ‘‘Malware detection by eating a whole exe.’’ [Online]. Available: https://arxiv.org/abs/1710.09435
[12] M. Krcál, O. Švec, M. Bálek, and O. Jašek. (2018). Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only. [Online]. Available: https://openreview.net/forum?id=HkHrmM1PM
[13] M. Rhode, P. Burnap, and K. Jones, ‘‘Early-stage malware prediction using recurrent neural networks,’’ Comput. Secur., vol. 77, pp. 578–594, Aug. 2018.
[14] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, Evading Machine Learning malware Detection. New York, NY, USA: Black Hat, 2017.
[15] R. Verma, ‘‘Security analytics: Adapting data science for security challenges,’’ in Proc. 4th ACM Int. Workshop Secur. Privacy Anal. New York, NY, USA: ACM, Mar. 2018, pp. 40–41.
[16] Y. LeCun, Y. Bengio, and G. Hinton, ‘‘Deep learning,’’ Nature, vol. 521, no. 7553, pp. 436–444, 2015.
[17] A. F. Agarap and F. J. H. Pepito. (2017). ‘‘Towards building an intelligent anti-malware system: A deep learning approach using support vector machine (SVM) for malware classification.’’ [Online]. Available: https://arxiv.org/abs/1801.00318

[18] E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, and P. de Geus, ''Malicious software classification using VGG16 deep neural network's bottleneck features,'' in Information Technology-New Generations. Cham, Switzerland: Springer, 2018, pp. 51–59.

[19] J. Saxe and K. Berlin, ''Deep neural network based malware detection using two dimensional binary program features,'' in Proc. 10th Int. Conf. Malicious Unwanted Softw. (Malware), Oct. 2015, pp. 11–20.

[20] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, ''Malware detection with deep neural network using process behavior,'' in Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC), vol. 2, Jun. 2016, pp. 577–582.

[21] W. Huang, J. W. Stokes, ''Mtnet: A multi-task neural network for dynamic malware classification,'' in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment, Cham, Switzerland: Springer, Jul. 2016, pp. 399–418.

[22] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, ''Malware classification with recurrent networks,'' in Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP), Apr. 2015, pp. 1916–1920.

[23] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, ''Efficient dynamic malware analysis based on network behavior using deep learning,'' in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2016, pp. 1–7.

[24] S. H. Ebenuwa, M. S. Sharif, M. Alazab, and A. Al-Nemrat, ''Variance ranking attributes selection techniques for binary classification problem in imbalance data,'' IEEE Access, vol. 7, pp. 24649–24666, 2019.

[25] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, ''Malware images: Visualization and automatic classification,'' in Proc. 8th Int. Symp. Vis. Cyber Secur. New York, NY, USA: ACM, Jul. 2011, p. 4.

[26] F. C. C. Garcia, and F. P. Muga, II, (2016). ''Random forest for malware classification.'' [Online]. Available: https://arxiv.org/abs/arXiv: 1609.07770

[27] H. S. Anderson and P. Roth. (2018). ''EMBER: An open dataset for training static PE malware machine learning models.'' https://arxiv.org/ abs/1804.04637