IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 27, 2018, accepted February 20, 2019, date of publication April 3, 2019, date of current version April 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2906934

Robust Intelligent Malware Detection Using Deep Learning

R. VINAYAKUMAR[1], MAMOUN ALAZAB[2], (Senior Member, IEEE), K. P. SOMAN[1], PRABAHARAN POORNACHANDRAN[3], AND SITALAKSHMI VENKATRAMAN[4]

[1]Center for Computational Engineering and Networking (CEN), Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore 641112, India
[2]College of Engineering, IT and Environment, Charles Darwin University, Casuarina, NT 0810, Australia
[3]Centre for Cyber Security Systems and Networks, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Amritapuri 690525, India
[4]Department of Information Technology, Melbourne Polytechnic, Prahran Campus, Melbourne, VIC 3181, Australia

Corresponding author: R. Vinayakumar (vinayakumarr77@gmail.com)

This work was supported by the Department of Corporate and Information Services, Northern Territory Government of Australia, the Paramount Computer Systems, and Lakhshya Cyber Security Labs.

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. However, such approaches are time consuming as they require extensive feature engineering, feature learning, and feature representation. By using the advanced MLAs such as deep learning, the feature engineering phase can be completely avoided. Recently reported research studies in this direction show the performance of their algorithms with a biased training data, which limits their practical use in real-time situations. There is a compelling need to mitigate bias and evaluate these methods independently in order to arrive at a new enhanced method for effective zero-day malware detection. To fill the gap in the literature, this paper, first, evaluates the classical MLAs and deep learning architectures for malware detection, classification, and categorization using different public and private datasets. Second, we remove all the dataset bias removed in the experimental analysis by having different splits of the public and private datasets to train and test the model in a disjoint way using different timescales. Third, our major contribution is in proposing a novel image processing technique with optimal parameters for MLAs and deep learning architectures to arrive at an effective zero-day malware detection model. A comprehensive comparative study of our model demonstrates that our proposed deep learning architectures outperform classical MLAs. Our novelty in combining visualization and deep learning architectures for static, dynamic, and image processing-based hybrid approach applied in a big data environment is the first of its kind toward achieving robust intelligent zero-day malware detection. Overall, this paper paves way for an effective visual detection of malware using a scalable and hybrid deep learning framework for real-time deployments.

INDEX TERMS Cyber security, cybercrime, malware detection, static and dynamic analysis, artificial intelligence, machine learning, deep learning, image processing, scalable and hybrid framework.

I. 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 realizing 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

The associate editor coordinating the review of this manuscript and approving it for publication was Corrado Mencar.

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 behavior. 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.

A. RESEARCH BACKGROUND

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 analyses the application programming interface (API) calls. Since this process is a resource intensive task, [3] presented an automated system to extract API calls and analyses 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 machine learning algorithm (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]. This forms the motivation of this research work.

B. NEED FOR THE STUDY

Machine learning algorithms (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 behavior, 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]–[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 malware 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 ScaleMalNet.

C. MAJOR CONTRIBUTIONS OF THE STUDY

To fill the gap in literature, in this paper, a scalable deep learning network architecture for malware detection called ScaleMalNet is proposed with the capability to leverage the application of Big Data techniques to handle vary large number of malware samples. Overall, the major contributions of the current research work are:

1) A new proposal of a scalable and hybrid framework, namely ScaleMalNet which facilitates to collect malware samples from different sources in a distributed way and to apply pre-processing in a distributed manner. The framework has the capability to process large number of malware samples both in real-time and on demand basis.
2) A proposal of a novel image processing technique for malware classification.
3) ScaleMalNet follows two stage approach, in the first stage the executables file is classified into malware or legitimate using Static and Dynamic analysis and in second stage the malware executables file is categorized into corresponding malware family.
4) An independent performance evaluation of classical MLAs and deep learning architectures, benchmarking various malware analysis models.

The rest of the paper is organized as follows. Section II reviews the key malware classification models considering various approaches traditionally employed for malware analysis. In Section III, deep learning architectures are introduced to get insights into this research background. Section IV presents the implementation architecture for deep learning in this research study and the statistical measures used to evaluate the performance of the classifier. Section V describes the experiments and observations of malware classification using deep learning based on Static analysis. Section VI discusses the experimental study and the results obtained for malware classification using deep learning based on Dynamic analysis. Section VII presents the experimental outcome of our deep learning architecture based on novel image processing technique used for malware categorization. Section VIII provides details of ScaleMalNet. Finally, Section IX provides the conclusion of this study and future work.

II. MALWARE CLASSIFICATION MODELS

In this section, we discuss the pros and cons of some of the popular classification models adopted for malware detection traditionally using static and dynamic analysis as well as their variations in recent years. However, with Big Data, it is more important to consider image processing techniques for enhanced data visualization and effective decision making. A good understanding of these methods form the basis of our research presented in this paper.

A. MALWARE CLASSIFICATION USING STATIC ANALYSIS

Several security researchers have applied domain level knowledge of portable executables (PE) for static malware detection. At present, analysis of byte n-grams and strings are the two most commonly used methods for static malware detection without domain level knowledge. However, the n-gram approach is computationally expensive and the performance is considerably very low [25]. It is often difficult to apply domain level knowledge to extract the necessary features when building a machine learning model to distinguish between the malware and benign files. This is due to the fact that the windows operating system does not consistently impose its own specifications and standards [9]. Due to constantly changing specifications and standards from time to time, the malware detection system warrants revisions to meet future security requirements. To address this, [26] has applied machine learning algorithms (MLAs) with the features obtained from parsed information of PE file. They adopted formatting of agnostic features such as raw byte histogram, byte entropy histogram which was taken from [19], and in addition employed string extraction. MalConv [11] compared these classical machine learning models with the deep learning approach. They have made the dataset with features as well as raw files and the associated code publicly available since deep learning models require more exploration and require further research.

A classical fully connected network and recurrent neural network (RNN) model of deep learning was traditionally employed to detect malware with 300 bytes information from the PE header file [9]. Subsequently, [12] has employed convolutional neural network (CNN) on a large number of byte long executables and obtained consistent results across 2 different tests based on a previous study [25]. Using domain level knowledge, [26] has extracted several features and showed that its performance is comparable to the MalConv [11] deep learning approach. The performance of Malconv model was improved by making modification to the existing architecture [12]. We believe that the deep learning capabilities have not been fully realized, and this

work proposes the application of Windows-StaticBrain-Droid (WSBD) model for incorporating deep learning.

In this work, by using WSBD, we evaluate the performance of benchmarked models [11], [26] and [12] on the publicly available dataset from [26] along with privately collected samples of benign and malwares. We introduce several variants of the existing deep learning architectures from [11] and [12]. In addition, we compare the performance of various classical machine learning classifiers on the domain level features obtained from [26] using deep learning techniques.

B. MALWARE CLASSIFICATION USING DYNAMIC ANALYSIS

Malware analysis methods based on Dynamic analysis are more robust to obfuscation methods as compared to Static analysis. In [20], features from 5 minutes API calls were extracted and passed on to CNN for classification using Dynamic analysis. They used around 170 samples and obtained 0.96 for Area under Curve (AUC) as the quality measure. In [21], shallow feed forward network with feature sets of API calls were obtained from a large number of samples of benign and malware that were collected privately. It performs well as compared to the existing approaches but it lacks the study on the speed of execution, which is important for real-time deployments. In [22], experiments with echo state networks (ESNs) and RNN were conducted to learn the language of malwares. In most of the experiments, the ESNs performed well in comparison to RNN. In [23], experiments were conducted to determine when to stop the malware execution with respect to the network communication. This method has reduced the total time taken by 67% compared with conventional methods. In [24] the application of RNN and its variant long short term memory (LSTM) and CNN were employed for malware classification with API call long sequences as features. The major problem with the existing methods are that they take more time to analyze the behaviors during execution. In [24], a hybrid of CNN and RNN was employed for malware classification using system call sequences. These system calls were obtained from Dynamic analysis, and their method was reported to outperform previously used algorithms such as SVM and hidden markov model (HMM). However, we identify the main drawback as the failure to discuss the importance of execution time towards detection of malware in real-time. In [13] has proposed a method based on RNN with two different datasets. They also evaluated the performance of other classical machine learning classifiers. They had reported 94% accuracy with 5s execution time.

Many research studies have compared malware detection techniques based on Static, Dynamic, and Hybdrid analysis. In [27], use of HMM on both static and Dynamic analysis of feature sets and a comparative study on detection rates was conducted over a substantial number of malware families. They reported that Dynamic analysis generally yielded the best detection rates. In this work, we propose Windows-Dynamic-Brain-Droid (WDBD) model that evaluates the efficacy of the various classical machine learning algorithms (MLAs) and deep learning architectures to know which algorithm is most appropriate for windows malware classification. We employ two different datasets that contain different numbers of malware and benign samples that are captured during different execution time.

C. MALWARE CLASSIFICATION USING IMAGE PROCESSING TECHNIQUES

Malware attacks are on the rise and in recent days, new malwares are easily generated as variants of existing malware from a known malware family. To overcome this problem, it is important to learn the similar characteristics of malware that can help to classify it into its family. Several studies conducted in [28]–[32], [34], [35] have taken advantage of the fact that most malware variants are similar in structure, with digital signal and image processing techniques used for malware categorization. They have transformed the malware binaries into gray scale images and report that malwares from the same malware family seem to be quite similar in layout and texture. Since image processing techniques require neither disassembly nor code execution, it is faster in comparison to the Static and Dynamic analysis. The main advantage of such an approach is that it can handle packed malware, and can work on various malwares irrespective of the operating system. Experimental results have shown 98% classification accuracy on a huge malware database and it is also resilient to popular obfuscation techniques namely, encryption. They have made benchmarked data, Malimg as public for further research. They also presented Search and RetrieVAl of Malware (SARVAM), an online malware search and retrieval system where binary executable can be analyzed by utilizing similarity metrics. They also presented SigMal, a malware similarity detection framework which was based on signal processing. It has the capability to handle both packed and unpacked samples, avoiding unpacking process which uses resources intensively. Heuristics based on the information about the PE structure were used to augment the accuracy of the signal processing based features. Experimental outcomes exhibit that SigMal's performance out wings all other static malware detection methods in terms of accuracy.

In recent days, the Malimg dataset is used to evaluate the efficacy of advanced machine learning algorithms (MLAs) over classical MLAs. Instead of following various signal and image processing techniques, the applications of deep learning algorithms are transformed into malware categorization using Malimg dataset [17], [18]. In [17], they have applied the combination of SVM and deep learning architectures such CNN and RNN variations. They have divided the dataset randomly into 80% for training and 20% for testing and claimed that the combination of GRU and SVM performed well in comparison to the other methods. Recently, [18] did the detailed analysis of different CNN architectures such as ResNet-50, VGG16, VGG-19 and the transfer learning applied on both the Malimg and privately collected dataset. In this work, we propose DeepImageMalDetect (DIMD) that

leverages deep learning with image processing approach for malware categorization. The performance of the proposed architecture is compared with the other deep learning architectures and classical machine learning classifiers. All these methods are evaluated on the benchmark dataset and the also the performance of those methods are shown on the recently collected private malware samples.

At last, we propose hybrid malware analysis system named as ScaleMalNet which composed of Windows-Static-Brain-Droid (WSBD), Windows-Dynamic-Brain-Droid (WDBD) and DeepImageMalDetect (DIMD). This can detect malware in real-time more accurately. The details of the deep learning architectures, the implementation architecture and the analysis of different approaches are presented as follows.

III. DEEP LEARNING ARCHITECTURES

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. The concepts behind the various deep learning architectures are discussed in a mathematical way. All the mathematical equations are taken from [16].

A. DEEP NEURAL NETWORK (DNN)

A feed forward neural network (FFN) creates a directed graph in which a graph is composed of nodes and edges [16]. FFN passes information along edges from one node to another without formation of a cycle. Multi-layer perceptron (MLP) is a type of FFN that contains 3 or more layers, specifically one input layer, one or more hidden layer and an output layer in which each layer has many neurons, called as units in mathematical notation. The number of hidden layers is selected by following a hyper parameter tuning approach. The information is transformed from one layer to another layer in forward direction without considering the past values. Moreover, neurons in each layer are fully connected. An MLP with n hidden layers can be mathematically formulated as given below:

$$H(x) = H_n(H_{n-1}(H_{n-2}(\cdots(H_1(x))))) \tag{1}$$

H defines hidden layer. This way of stacking hidden layers is typically called as deep neural networks (DNNs). Figure 1 shows a pictorial representation of DNN architecture with n hidden layers. It takes input $x = x_1, x_2, \cdots, x_{p-1}, x_p$ and outputs $o = o_1, o_2, \cdots, o_{c-1}, o_c$. Each hidden layer uses Rectified linear units ($ReLU$) as the non-linear activation function. This helps to reduce the state of vanishing and error

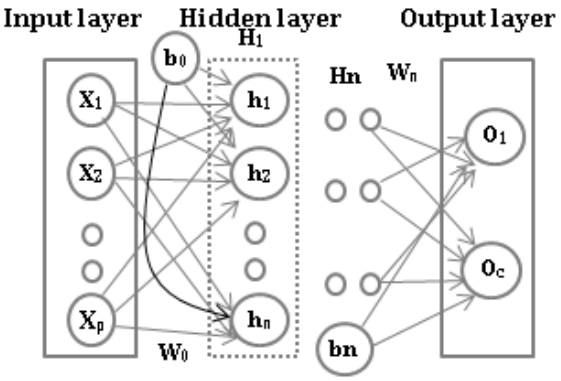

FIGURE 1. Architecture of DNN with *n* hidden layers.

gradient issue [36]. *ReLU* has been turned out to be more proficient and capable of accelerating the entire training process altogether. *ReLU* is defined mathematically as follows:

$$f(x) = \max(0, x) \tag{2}$$

where x denotes input.

B. CONVOLUTIONAL NEURAL NETWORK (CNN)

Convolutional network or convolutional neural network or CNN is supplement to the classical feed forward network (FFN), primarily used in the field of image processing [16]. It is shown in Figure 2, where all connections and hidden layers and its units are not shown. Here, m denotes number of filters, ln denotes number of input features and p denotes reduced feature dimension, it depends on pooling length.

In this work, CNN network composed of convolution 1D layer, pooling 1D layer and fully connected layer. A CNN network can have more than one convolution 1D layer, pooling 1D layer and fully connected layer. In convolutional 1D layer, the filters slide over the 1D sequence data and extracts optimal features. The features that are extracted from each filter are grouped into a new feature set called as feature map. The number of filters and the length are chosen by following a hyper parameter tuning method. This in turn uses non-linear activation function, *ReLU* on each element. The dimensions of the optimal features are reduced using pooling 1D layer using either max pooling, min pooling or average pooling. Since the maximum output within a selected region is selected in max pooling, we adopt max pooling in this work. Finally, the CNN network contains fully connected layer for classification. In fully connected layer, each neuron contains a connection to every other neuron. Instead of passing the pooling 1D layer features into fully connected layer, it can also be given to recurrent layer, LSTM to capture the sequence related information. Finally, the LSTM features are passed into fully connected layer for classification.

C. RECURRENT STRUCTURES

Recurrent structures have the capability to learn the sequence information in the data. The well-known recurrent structures are recurrent neural network (RNN) and long short term

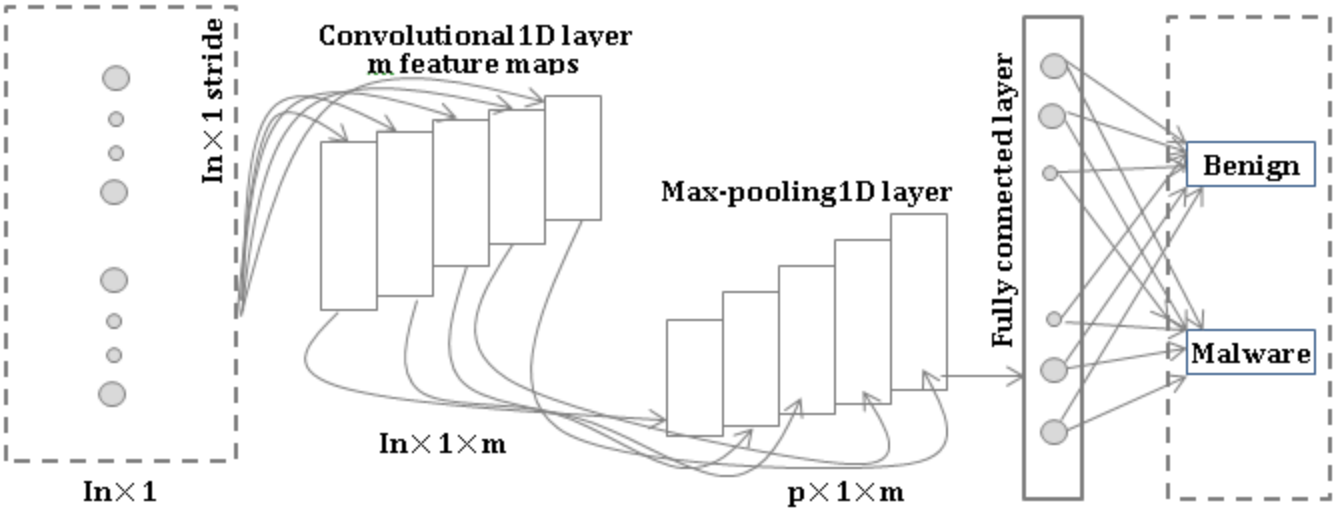

FIGURE 2. Architecture of CNN for malware detection.

memory (LSTM) [16]. Figure 3 shows the architecture of CNN depicting an RNN unit and an LSTM memory block. RNN is not like a feedforward network where the signals flow only in one direction from input to output. When data is represented as a sequence, RNN is used to handle the positional memory. In this network, the output of a layer is added to the next input and then fed into the same layer. It controls the information of signal with respect to the time. RNNs are the best suited network if the pattern of the data changes with time.

The transition function of RNN is denoted by tf. At each time step t, the hidden state vector h_t is estimated as follows:

$$h_t = \begin{cases} 0 & t = 0 \\ tf(h_{t-1}, x_t) & otherwise \end{cases} \tag{3}$$

During backpropagation, the transition function tf goes into the problem of vanishing gradient when the model propagates through multiple steps. This can lead to the decay of information through time. To overcome this, gradient clipping and gating mechanisms are introduced [16]. An LSTM network is a type of RNN which helps in handling long-term dependencies with memory blocks and gating functions. A memory cell is a part of memory block which acts like a memory and this is controlled using several gating functions such as input, output and forget gates. The computation of LSTM unit at time step t is mathematically defined as follows:

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{4}$$
$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{5}$$
$$c_t = \tanh(W_c.[h_{t-1}, x_t] + b_c) \tag{6}$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{7}$$
$$h_t = o_t * \tanh(c_t) \tag{8}$$

where x_t denotes an input vector, h_t denotes hidden state vector, c_t denotes cell state vector, o_t denotes output vector, i_t denotes input vector and f_t denotes forget state vector and terms of w and b denotes weights and biases respectively.

IV. IMPLEMENTATION ARCHITECTURE AND STATISTICAL MEASURES

The implementation architecture adopted for our experimental analysis is a real-time distributed Apache Spark cluster computing platform. A prototype model is developed for this research and to protect the confidentiality, we provide only a summary of the scalable framework implemented for this purpose. The Apache spark.[1] cluster computing framework is set up over Apache Hadoop Yet Another Resource Negotiator (YARN)[2] This framework facilitates to efficiently distribute, execute and harvest tasks. Each system has specifications (32 GB RAM, 2 TB hard disk, Intel(R) Xeon(R) CPU E3-1220 v3 @ 3.10GHz) running over 1 Gbps Ethernet network. We define the basic unit of our Apache Spark cluster as a node, which is a machine. The developed framework has 3 kinds of nodes, master node, slave node and data storage node. The master node (Cm) controls all the nodes in the framework. The user can communicate to the system through master node interface. It retrieves data from data storage node and performs preprocessing and segmentation. This distributes workloads to other slave nodes and finally aggregates the output from all other slave nodes. The slave nodes (Cs) retrieve preprocessed data from the master node. There might be a big quantity of them to investigate data in parallel. The master-slave node framework keeps computation very fast and also adjusts according to the size of the data. The data storage node (Cds) is used for storing data. It also acts as a slave node. This keeps track of data on daily basis and aggregates the data to daily, weekly and monthly basis. The proposed scalable architecture can be

[1] https://spark.apache.org/
[2] http://hadoop.apache.org/

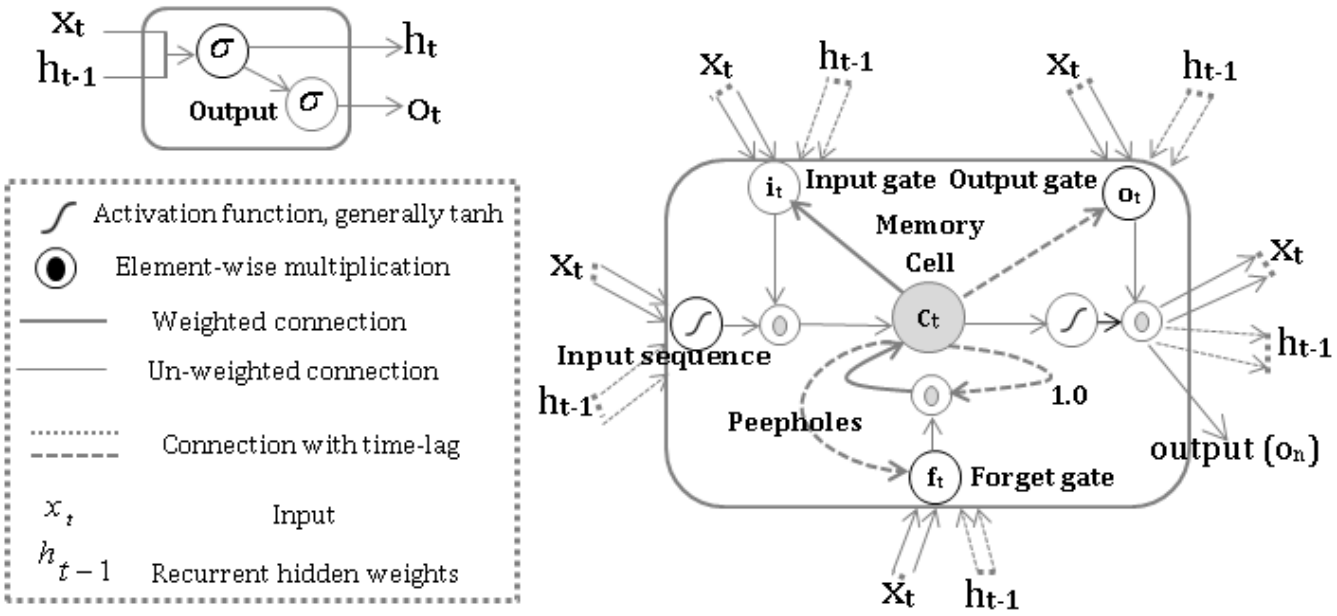

FIGURE 3. Architecture of RNN unit (left) and LSTM memory block (right).

scaled out to break down much bigger volumes of network event information. All deep learning models are implemented using TensorFlow [37] with Keras [38]. All classical machine learning algorithms (MLAs) are implemented using Scikit-learn [39]. All experiments related to deep learning architectures are run on GPU enabled TensorFlow machines.

In this study, to evaluate the performance of classifiers, we have considered Accuracy ($Accuracy \in [0, 1]$), Precision ($Precision \in [0, 1]$), Recall ($Recall \in [0, 1]$) and F1-score ($F1 - score \in [0, 1]$) standard metrics. These metrics are estimated based on True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). TP represent the number of malware application samples correctly identified as malware application, TN represent the number of benign application samples correctly identified as benign application samples, FP represent the number of benign application samples misclassified as malware application samples, FN represent the number of malware application samples misclassified as benign application samples. The metrics such as Accuracy, Precision, Recall and F1-score are defined as follows

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN}, \tag{9}$$

$$Precision = \frac{\#TP}{\#TP + \#FP}, \tag{10}$$

$$Recall = \frac{\#TP}{\#TP + \#FN}, \tag{11}$$

and

$$F1{-}score = 2 \times \left(\frac{\Pr ecision \times Recall}{\Pr ecision + Recall} \right). \tag{12}$$

One of the most commonly used diagnostic tool to identify the interpretation of the binary classifier is Receiver Operating Characteristic (ROC) curve. Primarily, the ROC curves are used when the samples of each class are balanced. Most commonly, area under the curve (AUC) is used to compare the ROC curves. AUC, as the name indicates, just the area under the ROC curve. It specifically measures the amount of separation between classes. AUC is defined as;

$$AUC = \int_0^1 \frac{\#TP}{\#TP + \#FN} d \frac{\#FP}{\#TN + \#FP} \tag{13}$$

Higher the AUC indicates that the model predicts classes accurately. To generate ROC, we estimated the trade-off between the true positive rate ($TPR \in [0, 1]$) on the Y axis to false positive rate ($FPR \in [0, 1]$) on the X axis across varying threshold in the range of [0, 1], where

$$TPR = \frac{\#TP}{\#TP + \#FN}, \tag{14}$$

and

$$FPR = \frac{\#FP}{\#FP + \#TN}. \tag{15}$$

V. MALWARE DETECTION USING DEEP LEARNING BASED ON STATIC ANALYSIS

We adopt an evaluation sub module to benchmark the deep learning architectures based on Static analysis. The performance of various classical machine learning and deep learning for static portable executable (PE) malware detection and classification are evaluated on publicly available dataset called Ember along with privately collected samples of benign and malwares. The variants of deep learning architectures are proposed by carefully following a hyper parameter tuning approach. Various trials of experiments are run for different classical machine learning algorithms (MLAs)

and deep learning architectures. Experiments related to deep learning architecture are run till 1,000 epochs with varied learning rate [0.01-0.5]. All of the models of classical machine learning and deep learning have marginal difference in their performances. Thus, the performance of the malware detection can be enhanced by incorporating a hybrid system pipeline typically called as Windows-Static-Brain-Droid (WSBD), which is composed of both classical machine learning and deep learning models. WSBD can be deployed at an organization level to detect malware effectively in real-time.

A. DESCRIPTION OF DATASET

To evaluate the effectiveness of classical machine learning and deep learning architectures, it is required to create a large dataset with a variety of different samples. The publicly available datasets for possible research in cyber security for malware detection are very limited due to the privacy preserving policies of the individuals and organizations. Over time, as malware have grown it has become increasingly difficult to have one source having all types of malware families. Many researchers try to collaborate their findings but still there is not a single dataset or repository to acquire all the required samples. In this research, the publicly available dataset Ember is used with a subset containing 70,140 benign and 69,860 malicious files. This dataset is randomly divided into 60% training and 40% testing using Scikit-learn. The training dataset contains 42,140 benign files and 41,860 malicious files. The testing dataset contains 28,000 benign files and 28,000 malicious files. These samples were obtained from VirusTotal[3], VirusShare[4] and privately collected samples of benign and malware samples.

We prepare the datasets for conducting the experimental analysis using the following pre-processing stages:

1) **Ember:** Using domain level knowledge, various features from parsed PE file as well as format-agnostic features such as raw byte histogram, byte entropy histogram are taken from [26], and strings are extracted and passed into the LightGBM model. Since the performance of LightGBM model is good as compared to MalConv model, they use gradient boosed decision tree (GBDT) in LightGBM with default parameters consisting of 100 trees and 31 leaves per tree. Following, in this work we evaluate the performance of classical MLAs and DNNs for malware classification using the Ember dataset.
2) **MalConv:** MalConv is an architecture proposed in [11] for malware detection which composed of 3 different sections are undergone, namely (1) pre-processing (2) convolution and (3) fully-connected. In the pre-processing section, the raw byte sequences from the binary files are passed into embedding layer. The embedding layer contains 257 as the size of the dictionary of embeddings and 8 as the embedding dimension. Embedding layer maps bytes into fixed length feature vector representation. In convolution section, MalConv contains two convolution 1D layers. Each convolution 1D layer contains 512 (kernel size 4, 128 filters) units and 500 strides. These convolution layers follow the gated convolution approach. Convolution layer follows a temporal maxpooling which uses 4000 as pooling length to reduce the dimension and to handle the information sparsity issue. Fully connected section is composed of 2 fully connected layers: the first fully connected layer contains 128 units, and the second fully connected layer contains 1 unit with *sigmoid* non-linear activation function. SVM is used at the last layer for classification with LSTM.
3) **Variants of MalConv:** The slight variation to the strides, SELU nonlinear activation function of the MalConv model and removed the DeConv regularization by [12]. The convolution section contains two convolution layers, a maxpooling followed by another two convolution layers. The first two convolution layers contain 32 units with strides 4 and the next two convolution layers contain 16 units with strides of 8. The last two layers follow the global average pooling with 4 fully connected layers.
4) **Other variants of MalConv:** There are four different deep learning architectures adopted here. Two deep learning architectures are variants of the MalConv [11] and other two deep learning architectures are variants to the variants of MalConv [12]. We introduce the modifications in the LSTM layer with 30 memory blocks added into MalConv as well as variants of MalConv after the Global maxpooling and Global average are respectively incorporated. Finally, the features of LSTM layer are passed into SVM for classification. In SVM, c and kernel function value is set into 1.0 and rbf respectively.

B. DATA ANALYSIS AND RESULTS

We present the data analysis and results obtained from various experiments conducted on the variants of the existing deep learning architecture mentioned above [11], [12], [26]. In order to evaluate the performance of various classical machine learning classifiers such as Logistic Regression (LR), Navie Bayes (NB), K-Nearest Neighbor (KNN), Decision Tree (DT), Ada Boost (AB), Random Forest (RF) and Support Vector Machine (SVM) and deep neural network (DNN) on the domain level features, we conducted various experiments using the Ember dataset. All classical MLAs used the default parameters provided by scikit-learn machine learning library. Initially, two trails of experiments were run for the DNN to find out the optimal parameters for the number of units till 200 epochs. The experiments were used *adam* optimizer and binary cross entropy as loss function. This DNN contains an input layer, output layer and a fully connected layer with units in the range [32-5,120].

[3]https://www.virustotal.com/
[4]https://virusshare.com/

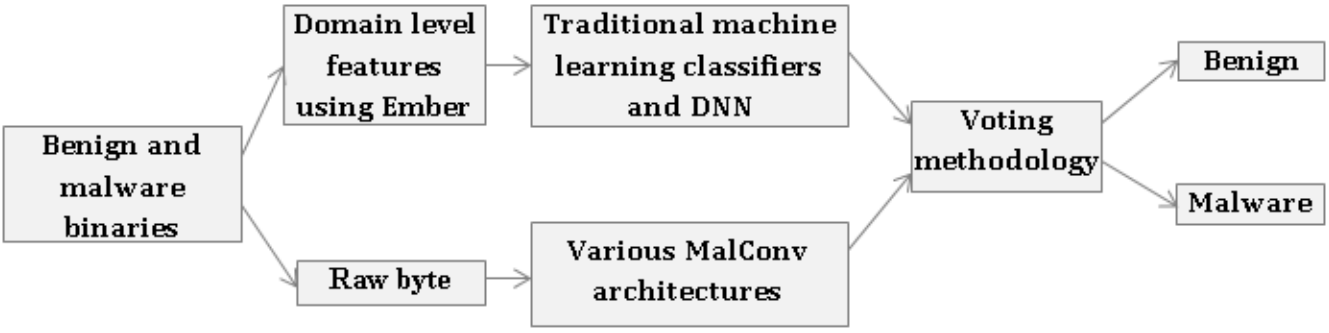

FIGURE 4. Proposed deep learning architecture based on static analysis - windows-static-brain-droid (WSBD).

A fully connected layer has used *ReLU* activation function which helps to prevent from vanishing and exploding gradient issue. DNN with 4,608 achieved the best performance and the performance deteriorated when the number of hidden units increases from 4,608 to 5,120 and 5,632. Thus the number hidden units is set into 4,608. Later, to identify an optimal learning rate, two trails of experiments were run for DNN with learning rate in the range [0.01-0.05] till 100 epochs. The DNN network with learning rate 0.01 performed well in compared to other learning rates. Later to identify the DNN network structure, we run two trails of experiments with DNN 1 to 12 layers till 100 epochs. Initially, all the fully connected layers have used the 4,608 hidden units and the performance with DNN 10 layer was good compared to other DNN networks. Later, we followed decreasing the number of units along with increasing the number of layers. Finally, the neurons in the 10 fully connected layers are set to 4,608, 4,096, 3,584, 3,072, 2,560, 2,048, 1,536, 1,024, 512, 128 hidden units respectively. Dropout of 0.01 and Batch normalization is placed in between the fully connected layers which helps to prevent from overfitting and increases the speed of learning during training respectively. Generally, dropout removes the neurons and its connections randomly. When the experiments with DNN were run without the dropout and Batch normalization, the DNN models results in overfitting and took larger time for training. *Sigmoid* is used in output layer which results 0 or 1 where 0 indicates benign and 1 indicates malware. The detailed parameter details of DNN network is reported in Table 1.

Table 2 gives the detailed results of all the models having marginal difference in terms of accuracy. Among all models, variants of MalConv performed better. The performance of Ember datasets with domain level features have outperformed the Malconv, which can be enhanced by following a hyper parameter tuning method. More importantly, DNN outperformed other classical MLAs and as well as the Malconv architecture. The dataset used in this study is a sub set of Ember and the dataset is balanced. However, in most of the cases the benign and malware classes have imbalanced data samples distribution. This can be controlled using data mining techniques [45]. The performances of variants to the existing deep learning architectures are closer to the Ember and Malconv.

Finally, this work suggests that the hybrid of domain level knowledge with classical machine learning models and deep learning architectures on the entire byte sequences can be used in real-time to detect the malware effectively, shown in Figure 4.

The limitation of this work is that a detailed analysis on the hyper parameter tuning method has not been adopted for the variants of the existing deep learning architectures. Thus, this remains scope for future enhancement towards improving the performance of malware detection.

VI. MALWARE DETECTION USING DEEP LEARNING BASED ON DYNAMIC ANALYSIS

We present an evaluation sub module to compare classical machine learning algorithms (MLAs) and deep learning architectures based on Dynamic analysis for windows malware detection. All the models are examined on the behavioral data that are collected via Dynamic analysis [13]. The parameters for deep networks are selected by following a hyper parameter selection approach with various trials of experiments conducted upto 1,000 epochs with varied learning rate [0.01-0.5]. Deep learning architectures outperformed the classical MLAs in all types of experiments. This is due to the fact that those deep models are able to learn the optimal, high level and abstract feature representations by passing them into more than one hidden layers. The result of best performed model is not directly comparable to [13], due to the splitting methodology used for training and testing which is entirely different. Within the first 5 seconds of execution, both classical MLAs and deep learning architectures have the capability to detect whether the executable file is benign or malicious.

A. DESCRIPTION OF DATASET

We have employed two types of datasets from previous research works [13]. Dataset 1 was collected using VirtualBox[5] virtual machine using Cuckoo Sandbox[6] with a custom

[5]https://www.virtualbox.org/
[6]https://cuckoosandbox.org/

TABLE 1. Detailed configuration details of deep neural network (DNN).

Layers	Type	Output shape	Number of units	Activation function	Parameters 73,924,865
0-1	Fully-connected	(None, 4,608)	4,608	ReLU	10,833,408
1-2	Batch Normalization	(None, 4,608)			18,432
2-3	Dropout (0.01)	(None, 4,608)			0
3-4	Fully-connected	(None, 4,096)	4,096	ReLU	18,878,464
4-5	Batch Normalization	(None, 4,096)			16,384
5-6	Dropout (0.01)	(None, 4,096)			0
6-7	Fully-connected	(None, 3,584)	3,584	ReLU	14,683,648
7-8	Batch Normalization	(None, 3,584)			14,336
8-9	Dropout (0.01)	(None, 3,584)			0
9-10	Fully-connected	(None, 3,072)	3,072	ReLU	11,013,120
10-11	Batch Normalization	(None, 3,072)			12,288
11-12	Dropout (0.01)	(None, 3,072)			0
12-13	Fully-connected	(None, 2,560)	2,560	ReLU	7,866,880
13-14	Batch Normalization	(None, 2,560)			10,240
14-15	Dropout (0.01)	(None, 2,560)			0
16-17	Fully-connected	(None, 2,048)	2,048	ReLU	5,244,928
17-18	Batch Normalization	(None, 2,048)			8,192
18-19	Dropout (0.01)	(None, 2,048)			0
20-21	Fully-connected	(None, 1,536)	1,536	ReLU	3,147,264
21-22	Batch Normalization	(None, 1,536)			6,144
22-23	Dropout (0.01)	(None, 1,536)			0
23-24	Fully-connected	(None, 1,024)	1,024	ReLU	1,573,888
24-25	Batch Normalization	(None, 1,024)			4,096
25-26	Dropout (0.01)	(None, 1,024)			0
26-27	Fully-connected	(None, 512)	512	ReLU	524,800
27-28	Batch Normalization	(None, 512)			2,048
28-29	Dropout (0.01)	(None, 512)			0
29-30	Fully-connected	(None, 128)	128	ReLU	656,644
30-31	Batch Normalization	(None, 128)			512
31-32	Dropout (0.01)	(None, 128)			0
32-33	Fully-connected	(None, 1)			129
33-34	Activation	(None, 1)	1	Sigmoid	0

package written in the Java library, Sigar[7] to collect the machine activity data. The virtual machine has the capacity of 2GB RAM, 25GB storage, and a single CPU core running 64-bit Windows 7. Dataset 2 was collected in a VirtualBox virtual machine using Cuckoo Sandbox with a custom package written in the Python library, Psutil[8] to collect the machine activity data. The virtual machine has the capacity of 8GB RAM, 25 GB storage, and a single CPU core running 64-bit Windows operating system. The detailed statistics of Dataset 1 and Dataset 2 is reported in Table 3.

B. DATA ANALYSIS AND RESULTS

We adopt a hyper parameter technique to identify the optimal parameters for deep learning models so that the malware detection rate is enhanced. Initially, the training dataset is randomly split into 70% training and 30% validation. The validation data helped to observe the training accuracy across different epochs. Finally, the performance of the trained model is evaluated on the test dataset. For network parameters, three trials of experiments are run for the hidden units to enhance the learning rate with the basic CNN and DNN model. Both the CNN and DNN models experiments have used *adam* as optimizer and binary cross entropy as loss function. Both the models are composed of 3 layers such as input layer, hidden layer and an output layer. In input layer, the two models contain 10 neurons for 10 different features and the output layer contains 1 neuron with *sigmoid* activation function. To find out the hidden units for DNN, various experiments are run for the neurons in the range [4-128]. In the experiments with 64 neurons, DNN performed well in comparison to the other neurons. To find out the number of filters in CNN, 3 trials of experiments are run for the filters in the range [4-64]. CNN network with filters 32 performed well in comparison to the other filters. These parameters are set for the rest of the experiments were conducted to

[7]https://github.com/hyperic/sigar
[8]https://pypi.org/project/psutil/

TABLE 2. Detailed test results.

Model	Algorithm	Accuracy(%)	Precision(%)	Recall(%)	F1-score(%)
MalConv [11]	CNN	98.8	99.7	97.9	98.8
Ember [26]	LightGBM	97.5	99	96.2	97.1
Ember - proposed	LR	54.9	52.6	99.2	68.7
Ember - proposed	NB	53.8	52.0	99.3	68.3
Ember - proposed	KNN	95.1	95.5	94.6	95.1
Ember - proposed	DT	96.9	97.1	96.7	96.9
Ember - proposed	AB	83.0	86.1	78.8	82.3
Ember - proposed	RF	97.0	98.6	95.3	96.9
Ember - proposed	SVM	96.1	96.4	95.7	96.1
Ember - proposed	**DNN**	**98.9**	**99.7**	**98.1**	**98.9**
Variants of MalConv [12]	Modified CNN [11]	99.9	99.7	1.00	99.9
LSTM with MalConv 1 - proposed	-	98.8	99.8	97.8	98.8
LSTM with Variants of MalConv 2 - proposed	-	98.8	99.8	97.9	98.8
LSTM and SVM with MalConv 1 - proposed	-	97.0	98.5	.95.5	97.0
LSTM and SVM with Variants of MalConv 2 - proposed	-	97.1	98.5	95.6	97.0

TABLE 3. Statistics of datasets.

Data set	Benign	Malicious	Total
Data set 1	1,21,701	1,18,717	2,40,418
Data set 2	52,245	50,792	1,03,037

identify the optimal parameter for learning rate and various configurations of experiments for network parameters were made with learning rate within the limit [0.01-0.5]. In most of the cases, performance of experiments associated with lower learning rate was found to be good in identifying the executable as either benign or malware. By reviewing the training time and the malware detection rate, the learning rate 0.01 is used for the rest of the experiments.

To find out the optimal network structure for DNN and CNN, DNN/CNN 1, 2, 3, 4, and 5 layers were used, and 3 trials of experiments were run for various network topologies for 100 epochs. DNN model with 4 layers and CNN with 1 layer performed well in comparison to the other network topologies. In DNN, to reduce overfitting and increase the training speed, the concept of dropout 0.01 and Batch normalization were employed. In CNN, dropout 0.3 was used before the penultimate layer. When the numbers of layers were increased in the DNN and CNN model, the malware detection rate decreased. This is attributed to over fitting. DNN and CNN models with less number of parameters attained good malware detection rate with up to 100 epochs, but the more complex deep learning models attained the highest malware detection rate when the experiments were run up to 1,000 epochs. The functional block diagram of the proposed deep learning architecture based on Dynamic analysis for windows malware detection is shown in Figure 5. The executable files are passed into the Dynamic analysis phase which extracts different features. These features are passed into various classical MLAs and deep learning architectures to learn the characteristics of legitimate and malware files. Both CNN and DNN models have used *adam* optimizer, *sigmoid* nonlinear activation function and binary-cross entropy loss function. *ReLU* is used as activation function in convolution and fully connected layers. The *sigmoid* and binary-cross entropy are mathematically defined as follows:

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{16}$$

$$loss(pd, ed) = -\frac{1}{N}\sum_{i=1}^{N}[ed_i \log pd_i + (1-ed_i)\log(1-pd_i)] \tag{17}$$

where x defines input, pd is predicted probability, ed is the expected class label.

The detailed test results are reported in Table 4 and ROC curve for Data set 1 and Data set 2 is shown in Figure 6a and Figure 6b respectively. For a comparative study, various classical MLAs are evaluated on the Dataset 1 and Dataset 2. For these algorithms, the default parameters of Scikit-learn were used and not the hyper parameter tuning method. Thus the performance of various classical MLAs can be further enhanced by following a hyper parameter tuning method. In both the datasets the deep models outperformed the classical MLAs. Moreover, CNN model outperformed the DNN model.

In this sub module, a comparative study of various classical MLAs and deep learning architectures for windows malware detection is done. Deep learning architectures outperformed the classical MLAs in all types of experiments conducted with 2 different datasets. These models are capable of detecting the executable as malicious or benign within the first 5 seconds of execution. The reported results could be improved further as future research work by promoting training or stacking a few more layers to the existing architectures. Further, new features could be added to the existing data, and these explorations are devoted for future research.

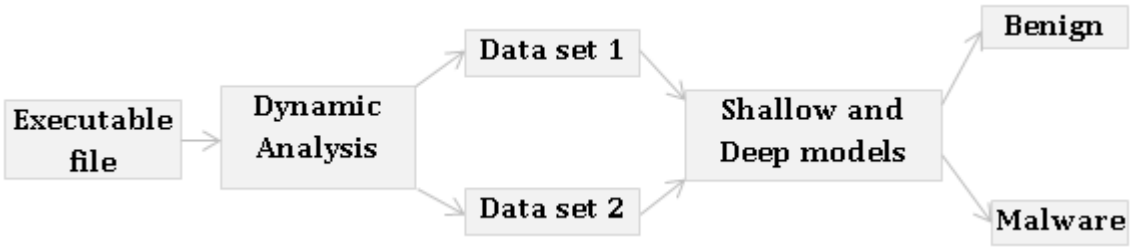

FIGURE 5. Proposed deep learning architecture based on dynamic analysis - windows-dynamic-brain-droid (WDBD).

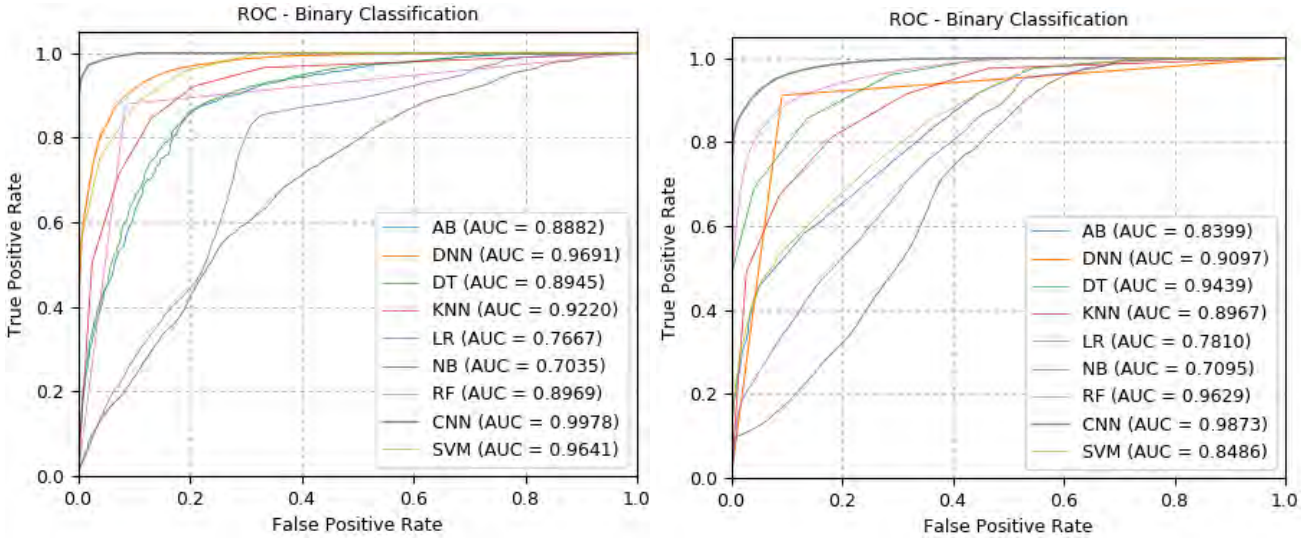

FIGURE 6. ROC curve for (a) Data set 1, (b) Data set 2.

VII. MALWARE FAMILY CATEGORIZATION USING DEEP LEARNING BASED ON IMAGE PROCESSING

We propose a novel DeepImageMalDetect (DIMD) as a deep learning model based on image processing techniques with convolutional neural network (CNN) and long short term memory (LSTM) hybrid pipeline for malware categorization. The proposed method uses visualization with deep learning for malware family categorization. This method completely avoids the feature engineering which was followed in the existing methods [43] and [44] to convert the malware files into images. The proposed method is fast compared to Static and Dynamic analysis as it works on the raw bytes and completely avoids disassembly or execution. The second advantage compared to the existing methods [43] and [44] is that the proposed method is agnostic to packed malware. In other words, packed malware variants from the unpacked malware can contain visual similarity.

Most commonly, similar techniques are followed separately to develop malware detection for different operating system (OS). The proposed method has the capability to work on malwares from different OS such as Windows, Android Linux etc.

The method behind malware image generation was initially proposed by [40]. A binary object's data can be represented as gray scale images, where each byte associated to the image pixel color with a value of zero for black, and a value of 255 for white and all other values are intermediate shades of gray. They also reported that malware image analysis facilitates to differentiate the different parts of the data. This approach can be applied towards various tasks such as fragment classification, file type identification, methods which require an understating of primitive data types and identifying the contents of location of regions. In [33], the malware was transformed into gray scale images by reading 8-bit unsigned integers. The width of image is defined by file size and height is allowed to vary depending on the width and file size. By following a method proposed by [41], the malware images were resized to two-dimensional (2D) matrix of 32×32 and mapped into 1x1024 size array. Each feature array is normalized using $L2$ normalization.

We performed a comparative study by employing classical MLAs and deep learning algorithms on the benchmark datasets, Malimg [17], [33] and privately collected malware samples.

TABLE 4. Test results.

Model	Accuracy(%)	Precision(%)	Recall(%)	F1-score(%)
Data set 1				
LR	67.4	60.6	96.4	74.4
NB	54.6	76.3	11.2	19.5
KNN	81.5	81.2	81.2	81.2
DT	86.0	85.9	85.6	85.7
AB	73.3	67.2	89.5	76.7
RF	89.5	89.9	88.6	89.2
SVM	74.5	70.0	84.4	76.5
DNN	**91.0**	**90.6**	**91.1**	**90.9**
CNN	**93.6**	**94.8**	**92.0**	**93.4**
Data set 2				
LR	57.3	54.0	0.3	0.5
NB	50.5	46.3	97.6	62.8
KNN	85.9	82.8	84.5	83.6
DT	82.5	77.6	83.1	80.2
AB	82.1	77.4	82.1	79.7
RF	89.9	88.5	87.9	88.2
SVM	89.0	86.6	87.9	87.2
DNN	**90.4**	**86.1**	**92.5**	**89.2**
CNN	**96.6**	**94.0**	**98.4**	**96.2**

A. DESCRIPTION OF DATASET

The two types of datasets used here are: Malimg (Dataset 1) and privately collected samples (Dataset 2). Malimg dataset contains 9,339 malware samples from 25 various malware families. The detailed statistics of the dataset is reported in Table 5. The dataset was formed by transforming malware binaries into a matrix. This matrix has 8-bit unsigned integer. This matrix can be visualized as a grayscale image which contains values in the range of [0, 255], 0 represents black and 255 represents white. We converted the 2D matrix into 1D vector form, resulting in a 1x1024 size array. $L2$ normalization is employed for newly formed data. Next, the dataset was randomly divided into 70% training and 30% testing dataset with both these datasets containing samples for each malware family.

Dataset 2 was crawled from VirusSign[9] and VirusShare[10] over one year period. About 15,512 malware samples and antivirus labels for malware samples were obtained by using VirusTotal.[11] AVclass [42] tool was used for labeling the malware samples. These samples are grouped into 10 malware families. The detailed statistics of the dataset is given in Table 6.

B. DATA ANALYSIS AND RESULTS

The proposed architecture, DeepImageMalDetect (DIMD) is shown in Figure 7. For both the datasets, the performance of various classical machine learning algorithms (MLAs) and deep learning architectures were evaluated for malware data analysis. In most of the cases, the deep learning architectures outperformed the classical MLAs. For all the deep learning architectures, we adopted *adam* optimizer and *softmax* activation function with categorical-cross entropy loss function. The detailed configuration of optimal deep learning architecture are reported in Table 7. The experimental design and the results obtained are described below:

1) Experiments on Dataset 1, Malimg: To select optimal values for parameters and structures of the deep learning architecture, various experiments were conducted using Dataset 1. Initially the 70% training was randomly divided into 50% training and 20% validation. To find out suitable parameter for the number of filters,

[9]http://www.virussign.com/
[10]https://virusshare.com/
[11]https://www.virustotal.com/

TABLE 5. Description of Data set 1, Malimg.

No.	Family	Family Name	No. of. Variants
01	Dialer	Adialer.C	122
02	Backdoor	Agent.FYI	166
03	Worm	Allaple.A	2949
04	Worm	Allaple.L	1591
05	Trojan	Alueron.gen!J	198
06	Worm:AutoIT	Autorun.K	106
07	Trojan	C2Lop.P	146
08	Trojan	C2Lop.gen!G	200
09	Dialer	Dialplatform.B	177
10	Trojan Downloader	Dontovo.A	162
11	Rogue	Fakerean	381
12	Dialer	Instantaccess	431
13	PWS	Lolyda.AA 1	213
14	PWS	Lolyda.AA 2	184
15	PWS	Lolyda.AA 3	123
16	PWS	Lolyda.AT	159
17	Trojan	Malex.gen!J	136
18	Trojan Downloader	Obfuscator.AD	142
19	Backdoor	Rbot!gen	158
20	Trojan	Skintrim.N	80
21	Trojan Downloader	Swizzor.gen!E	128
22	Trojan Downloader	Swizzor.gen!I	132
23	Worm	VB.AT	408
24	Trojan Downloader	Wintrim.BX	97
25	Worm	Yuner.A	800

TABLE 6. Description of Data set 2.

Malware Family	Malware samples
allaple	1,000
delf	1,050
gamarue	1,100
loring	970
mydoom	990
quakart	1,010
softpulse	1,070
ramnit	1,080
zbot	980
wapomi	1,100

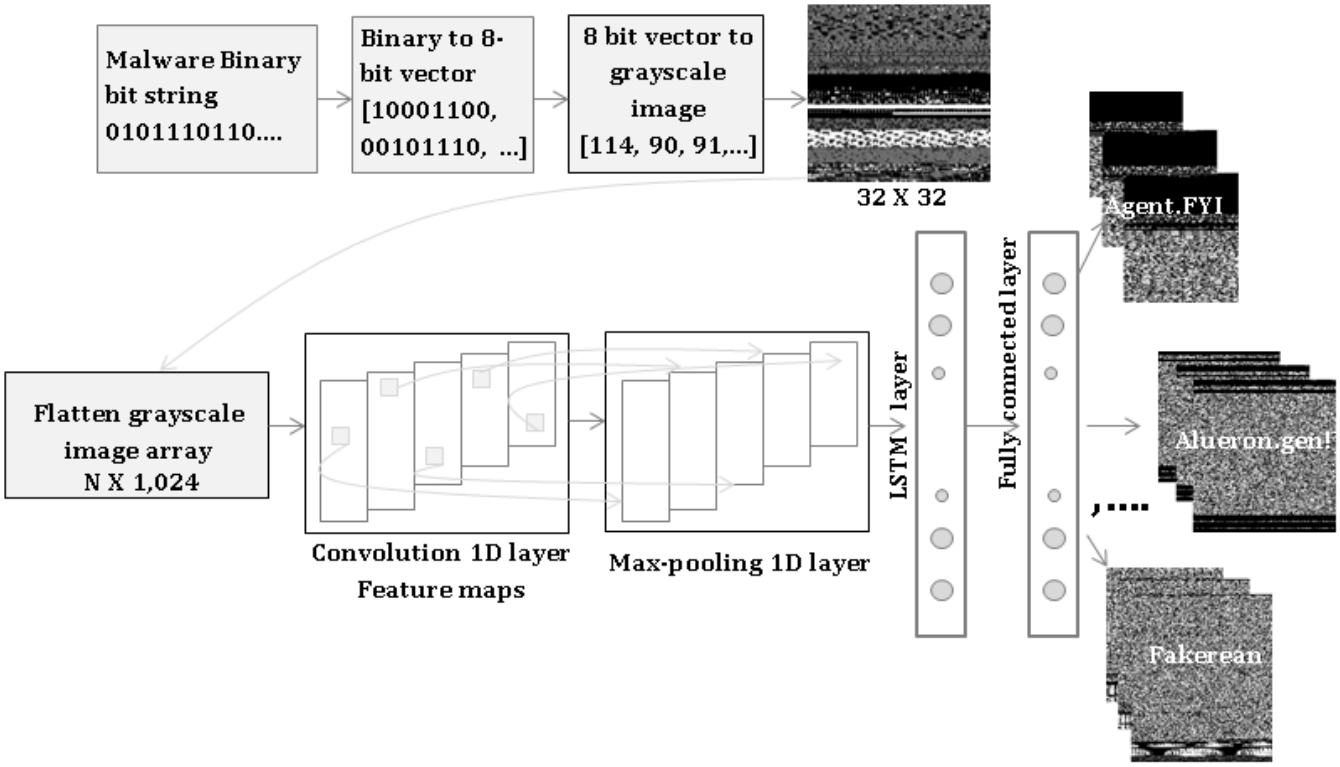

FIGURE 7. Proposed deep learning architecture based on Image processing - deep image mal detect (DIMD).

3 trials of experiments were run for filters 16, 32 and 64, and with filter length 3 for a CNN network with one layer CNN, maxpooling with pooling length 2 and followed by fully connected layers.

The fully connected layer contains 25 neurons with *softmax* activation function. These experiments were run till 100 epochs with batch size 32. After it was tested on the validation dataset, the CNN network with 64 filters and filter length 3 showed the best accuracy. In the same experiment, dropout of 0.5 was placed before the fully connected layer. This facilitated to avoid over fitting. Without dropout, the experiments with CNN ended up in overfitting. To find out suitable learning rate, 2 trials of experiments were run for varied learning rate in the range 0.01-0.5. Experiments with learning rate 0.01 performed well. Experiments with lower learning rate showed less accuracy in comparison to higher learning rate when the experiments were run till 50 epochs. When it was run till 200 epochs, the lower learning rates performed well. Based on computational time and accuracy, the learning rate was set to 0.01 for the rest of the experiments.

In order to determine the network structure, 3 types of CNN networks were used with 1, 2 and 3 layers. Four trials of experiments were run for all network topologies of CNN. These experiments were run till 150 epochs. CNN network with 2 CNN layers had performed well. The performance of CNN

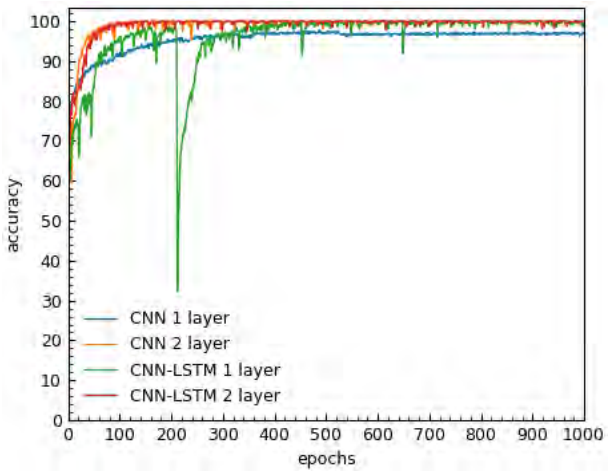

FIGURE 8. Training accuracy of various deep learning architectures on malware categorization.

3 layer remained same as CNN 2 layer even when the experiments were run till 300 epochs. In second set of experiments, the outputs of the CNN layer were passed into recurrent layer, LSTM. In order to determine the number of memory blocks, 3 trials of experiments were run for memory blocks 18, 36, 70 and 100. The experiment with 70 memory blocks performed well in comparison to the others.

There are two phases in both machine learning and deep learning models. All these models are trained

TABLE 7. Detailed configuration details of CNN 1, CNN 2, CNN 1 + LSTM and CNN 2 + LSTM.

Layer (type)	Output Shape	Param #
CNN 1		
conv1d_1 (Conv1D)	(None, 1024, 64)	256
max_pooling1d_1	(None, 512, 64)	0
flatten_1 (Flatten)	(None, 32768)	0
dense_1 (Dense)	(None, 128)	4194432
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 25)	3225
Trainable params: 4,197,913		
CNN 2		
conv1d_1 (Conv1D)	(None, 1024, 64)	256
max_pooling1d_1	(None, 512, 64)	0
conv1d_2 (Conv1D)	(None, 512, 128)	24704
max_pooling1d_2	(None, 256, 128)	0
flatten_1 (Flatten)	(None, 32768)	0
dense_1 (Dense)	(None, 128)	4194432
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 25)	3225
Trainable params: 4,222,617		
CNN 1 + LSTM		
conv1d_1 (Conv1D)	(None, 1024, 64)	256
max_pooling1d_1	(None, 512, 64)	0
lstm_1 (LSTM)	(None, 70)	37800
dropout_1 (Dropout)	(None, 70)	0
dense_1 (Dense)	(None, 25)	1775
Trainable params: 39,831		
CNN 2 + LSTM		
conv1d_1 (Conv1D)	(None, 1024, 64)	256
max_pooling1d_1	(None, 512, 64)	0
conv1d_2 (Conv1D)	(None, 512, 128)	24704
max_pooling1d_2	(None, 256, 128)	0
lstm_1 (LSTM)	(None, 70)	55720
dropout_1 (Dropout)	(None, 70)	0
dense_1 (Dense)	(None, 25)	1775
Trainable params: 82,455		

using training data and evaluated on the testing dataset. The training loss function is monitored during training and if $c + 1$ achieves improvement in loss function than c, then that epoch model is saved. The training accuracy of all deep learning architectures for 1,000 epochs are shown in Figure 8. The training loss of all deep learning architectures are shown in Figure 9. We observe that CNN 2 and CNN-LSTM 2 architectures have achieved optimal accuracy for 150 epochs. CNN 1 followed improvement in accuracy till 1,000 epochs. This shows that the network with less number of parameters require more epochs to converge or attain optimal performance. CNN 2 achieved optimal performance for epochs 210 and after that due to overfitting the network has shown sudden decrease. Finally, CNN 2 performed well in comparison to all

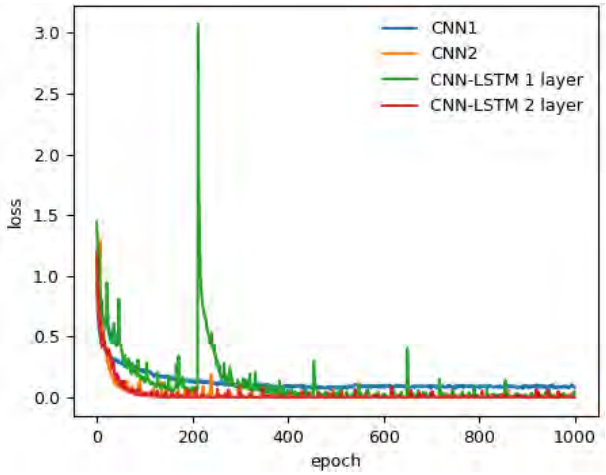

FIGURE 9. Training loss of various deep learning architectures on malware categorization.

other architectures for 1,000 epochs. During testing, the test data was passed to all the saved models and we estimated the accuracy, precision, recall and f1-score. The results are reported in Table 8. Tables 9, 11 contains the detailed test results in terms of *TPR* and *FPR* of each classes for classical MLAs and deep learning architectures respectively. Among all classical MLAs, SBM performed well, particularly showed higher *TPR* and lower *FPR* for the malware families C2Lop.P, Lolyda.AT, Swizzor.gen!I and VB.AT. More importantly, it showed better *TPR* and *FPR* for the malware family VB.AT compared to deep learning architectures. Thus the application of transfer learning in deep learning architectures can enhance the performance in malware detection and categorization. The performance of CNN 2 layer with LSTM showed higher *TPR* and lower *FPR* except the malware families Malex.gen!J, Obfuscator.AD, Rbot!gen, Skintrim.N, Swizzor.gen!E,VB.AT, and Yuner.A. This is mainly due to the reason that these classes contains less number of samples and due to overfitting the performance of CNN 2 layer with LSTM architecture reduced compared to other architectures. Another significant reason is that these malware families contains less number of families compared to other classes. Thus, application of cost-sensitive approach in deep learning can easily handle imbalanced data during training [45].

The accuracy measure hides the detail of the performance of the classification model. Since the Malimg dataset is highly imbalanced, we used the confusion matrix to understand the performance, shown in Table 10 and computed the Error rate as

$$Error\ rate = \left(1 - \left(\frac{corrected\ predictions}{total\ predictions}\right)\right) \times 100. \tag{18}$$

The error rate for malwares Adialer.C, Agent.FYI, Alueron.gen!J, Autorun.K, Dontovo.A, Fakerean,

TABLE 8. Detailed test results for Data set 1.

Model	Accuracy(%)	Precision(%)	Recall(%)	F1-score(%)
LR	78.6	78.0	78.6	78.2
NB	80.5	84.3	80.5	80.8
KNN	41.8	73.4	41.8	45.4
DT	79.5	79.5	79.5	79.4
AB	33.0	11.4	33.0	16.7
RF	84.3	85.3	84.3	83.6
SVM rbf	83.7	82.9	83.7	82.5
SVM linear	82.8	82.6	82.8	82.6
CNN + SVM [17]	77.2	84	77	79
GRU + SVM [17]	84.92	85	85	85
MLP + SVM [17]	80.4	83	80	81
CNN 1	**90.0**	**90.0**	**89.7**	**89.5**
CNN 2	**93.6**	**93.6**	**93.3**	**93.2**
CNN 1 + LSTM	**95.0**	**95.0**	**94.8**	**94.8**
CNN 2 + LSTM	**96.3**	**96.3**	**96.2**	**96.2**

Instantaccess, Lolyda.AA 2, Lolyda.AA 3, Lolyda.AT, Obfuscator.AD, Rbot!gen, Yuner.A is 0. It indicates that the model learnt the complete behaviors of these malwares. In Allaple.A malware, the classification model correctly classified 871 samples out of 885 and 8 samples are misclassified as Allaple.L malware. This indicates that both of these malwares have a lot of similarity and most importantly both are belongs to 'worm' malware family. The classification model achieved highest error rate for the malwares C2Lop.P, C2Lop.gen!G, Swizzor.gen!E and Swizzor.gen!I. The 10 samples of C2Lop.P is misclassified into C2Lop.gen!G and Swizzor.gen!E equally. More interestingly the C2Lop.P and C2Lop.gen!G belongs to 'Trozan' malware family and Swizzor.gen!E is belongs to 'Trojan Downloader' malware family. Few samples of Swizzor.gen!E and Swizzor.gen!I malwares are misclassified each other's. This shows that both of the malwares have a lot of similar characteristics and both of them are from same malware family 'Trozan Downloader'. More interestingly few samples in both of the malware families are misclassified into C2Lop.P and C2Lop.gen!G malwares. This indicates that the classification model may require few more additional samples from these malware families to accurately learn the hidden characteristics between the C2Lop.P and C2Lop.gen!G and Swizzor.gen!E and Swizzor.gen!I malwares.

2) Experiments on Dataset 2, privately collected samples: It is a common practice to adopt cross-validation as a statistical method to assess learning algorithms. It splits data into training and testing. Learning algorithms are trained on training data and evaluated on the testing data. The fundamental form is k-fold cross validation. It splits the data into k groups with the same length. The $k-1$ groups used for training and the other fold is used for testing and this process is repeated for k learnings. The detailed results of 10-fold cross validation are reported in Table 12. The optimal parameters and structures for deep learning architectures are selected through hyper parameter tuning. The detailed parameter details of all deep learning architectures are available at. In order to select an optimal parameter for the number of filters for CNN, initially two trials of experiments are run for filters 16,32,64 of CNN. This architecture is composed of an input layer, convolution, pooling and fully connected layer. All experiments were run for 200 epochs. Experiments with 64 filters performed well and when we increased the filters into 128, the performance deteriorated. Thus 64 filters are set for the rest of the CNN experiments. To select optimal learning rate, 3 trials of experiments are run for learning rate in the range [0.01-0.5]. The experiments with learning rate 0.05 performed well. To select the network structure for CNN, we run three trials of experiments with the CNN 1, CNN 2 and CNN 3 layers network. CNN 2 layers network performed well in comparison to other networks. Additionally, the CNN features are passed into LSTM layer instead of fully connected layer for classification. LSTM contains 50 memory blocks. This in turn obtains the sequence related information and passes onto fully connected layer for classification.

C. DEEPIMAGEMALDETECT (DIMD)

An overview of our proposed DeepImageMalDetect (DIMD) model is shown in Figure 7. This uses CNN-LSTM pipeline which helps to extract temporal and spatial features. The architecture is composed of 3 layers. In input layer, the malwares are converted into image format [33], and these images are transformed into 1D vector [17]. These 1D vectors of length 1024 form an input to the CNN layer composed of 64 filters with filter length 3, max-pooling with pooling length 2, a convolution layer with 128 filters of filter length 3, max-pooling with pooling length 2, an LSTM layer with 70 memory blocks and dropout 0.1 and a fully connected layer. The dropout is used to alleviate the overfitting. It randomly removes the units along with connections. The fully connected layer contains 25 units with activation function *softmax*. The *softmax* is defined as follows

$$SF(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{19}$$

where SF defines *softmax* activation function, x defines input.

The fully connected layer uses categorical-cross entropy as loss function and is estimated as follows:

$$loss(pd, ed) = -\sum_{x} pd(x) \log(ed(x)) \tag{20}$$

TABLE 9. Detailed Data set 1 test results of various classical machine learning classifiers.

Family Name	LR		NB		KNN		DT		AB		RF		SVM rbf		SVM linear	
	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
Adialer.C	1.0	0.0	1.0	0.0	1.0	0.004	1.0	0.0007	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Agent.FYI	1.0	0.0	0.971	0.0	1.0	0.0	0.971	0.0007	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Allaple.A	0.757	0.12	0.618	0.041	0.164	0.0	0.759	0.088	0.995	0.975	0.713	0.055	0.746	0.084	0.811	0.114
Allaple.L	0.688	0.083	0.141	0.933	0.0	0.0	0.689	0.066	0.0	0.0	0.992	0.112	0.937	0.099	0.683	0.069
Alueron.gen!J	0.847	0.003	0.881	0.0	0.034	0.0	0.915	0.005	0.0	0.0	0.915	0.0	0.881	0.0	0.949	0.0
Autorun.K	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0007	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
C2Lop.P	0.167	0.014	0.6	0.013	0.0	0.0	0.133	0.015	0.0	0.0	0.4	0.013	**0.467**	**0.014**	**0.45**	**0.01**
C2Lop.gen!G	0.091	0.009	0.432	0.013	0.0	0.0	0.25	0.016	0.0	0.0	0.068	0.002	0.091	0.003	0.272	0.008
Dialplatform.B	0.962	0.0	0.962	0.0	0.962	0.0	0.962	0.001	0.0	0.0	0.962	0.0	0.962	0.0	0.962	0.0
Dontovo.A	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Fakerean	0.982	0.001	0.965	0.0	0.947	0.0	0.982	0.003	0.0	0.0	0.982	0.0	0.982	0.0	0.982	0.0
Instantaccess	1.0	0.0	0.977	0.0	1.0	0.0	1.0	0.002	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Lolyda.AA 1	0.906	0.002	0.875	0.0004	0.844	0.0	1.0	0.0007	0.0	0.0	0.922	0.0004	0.922	0.0	0.922	0.0
Lolyda.AA 2	0.982	0.0004	0.855	0.001	0.927	0.0	1.0	0.0	0.0	0.0	1.0	0.0007	1.0	0.0	1.0	0.0004
Lolyda.AA 3	1.0	0.002	0.946	0.0	1.0	0.586	1.0	0.003	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0004
Lolyda.AT	0.833	0.002	1.0	0.005	0.021	0.0	0.875	0.003	0.0	0.0	0.938	0.0004	**0.958**	**0.0**	**1.0**	**0.001**
Malex.gen!J	0.439	0.006	0.61	0.0004	0.0	0.0	0.854	0.003	0.0	0.0	0.829	0.0	0.0	0.0	0.415	0.0025
Obfuscator.AD	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.003	1.0	0.0	1.0	0.0	1.0	0.0
Rbot!gen	0.872	0.003	0.957	0.0	0.489	0.0	0.9149	0.005	0.0	0.0	1.0	0.001	1.0	0.0	1.0	0.0007
Skintrim.N	0.917	0.0	1.0	0.001	0.708	0.0	0.75	0.001	0.0	0.0	0.917	0.0	0.917	0.0	1.0	0.0
Swizzor.gen!E	0.263	0.009	0.477	0.008	0.0	0.0	0.132	0.01	0.0	0.0	0.004	0.237	0.237	0.003	0.342	0.006
Swizzor.gen!I	0.15	0.007	0.35	0.008	0.0	0.0	0.175	0.012	0.0	0.0	0.225	0.0029	**0.25**	**0.0025**	**0.375**	**0.006**
VB.AT	0.861	0.004	0.869	0.0008	0.844	0.0	0.803	0.002	0.0	0.0	0.902	0.002	**0.992**	**0.001**	**0.967**	**0.0003**
Wintrim.BX	0.687	0.0004	0.552	0.0	0.517	0.0	0.758	0.001	0.0	0.0	0.793	0.0	0.655	0.0	0.689	0.0
Yuner.A	1.0	0.0004	1.0	0.0	1.0	0.0	1.0	0.0004	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Accuracy(%)	**78.6**		**80.5**		**41.8**		**79.5**		**33.0**		**84.3**		**83.7**		**82.8**	

TABLE 10. Confusion matrix for CNN 2 + LSTM architecture.

Malware Family	Adialer.C	Agent.FYI	Allaple.A	Allaple.L	Alueron.gen!J	Autorun.K	C2Lop.P	C2Lop.gen!G	Dialplatform.B	Dontovo.A	Fakerean	Instantaccess	Lolyda.AA 1	Lolyda.AA 2	Lolyda.AA 3	Lolyda.AT	Malex.gen!J	Obfuscator.AD	Rbot!gen	Skintrim.N	Swizzor.gen!E	Swizzor.gen!I	VB.AT	Wintrim.BX	Yuner.A	Error rate (%)
Adialer.C	**37**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
Agent.FYI	0	**35**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
Allaple.A	0	0	**871**	8	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0	3	0	0	0	0	1.582
Allaple.L	0	0	2	**475**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.419
Alueron.gen!J	0	0	0	0	**59**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
Autorun.K	0	0	0	0	0	**32**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
C2Lop.P	0	0	2	0	0	0	**41**	5	0	0	1	1	0	0	0	0	1	0	0	0	5	2	0	2	0	31.667
C2Lop.gen!G	0	0	1	0	0	0	9	**33**	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	25
Dialplatform.B	0	0	0	0	0	0	0	0	**52**	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1.887
Dontovo.A	0	0	0	0	0	0	0	0	0	**49**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
Fakerean	0	0	0	0	0	0	0	0	0	0	**114**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
Instantaccess	0	0	0	0	0	0	0	0	0	0	0	**129**	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
Lolyda.AA 1	0	0	1	0	0	0	0	0	0	0	0	0	**63**	0	0	0	0	0	0	0	0	0	0	0	0	1.563
Lolyda.AA 2	0	0	0	0	0	0	0	0	0	0	0	0	0	**55**	0	0	0	0	0	0	0	0	0	0	0	**0**
Lolyda.AA 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**37**	0	0	0	0	0	0	0	0	0	0	**0**
Lolyda.AT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**48**	0	0	0	0	0	0	0	0	0	**0**
Malex.gen!J	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	**36**	0	0	0	0	0	0	0	0	12.195
Obfuscator.AD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**43**	0	0	0	0	0	0	0	**0**
Rbot!gen	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**47**	0	0	0	0	0	0	**0**
Skintrim.N	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**23**	0	0	0	0	0	4.167
Swizzor.gen!E	0	0	0	0	0	0	8	1	0	0	0	0	0	0	0	0	0	0	0	0	**14**	14	0	1	0	63.158
Swizzor.gen!I	0	0	0	0	0	0	4	4	0	0	0	0	0	0	0	0	0	0	0	0	11	**21**	0	0	0	47.5
VB.AT	0	0	1	0	0	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	0	0	**118**	0	0	3.279
Wintrim.BX	0	0	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**25**	0	13.793
Yuner.A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**240**	**0**

where *pd* is true probability distribution, *ed* is predicted probability distribution. We have used *adam* as an optimizer to minimize the loss of categorical-cross entropy.

In this sub module, the application of image processing techniques along with classical MLAs and deep learning architectures are used for malware categorization.

TABLE 11. Detailed Data set 1 test results of CNN 1, CNN 2, CNN 1 + LSTM and CNN 2 + LSTM architectures.

Family Name	CNN 1		CNN 2		CNN 1 + LSTM		CNN 2 + LSTM	
	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
Adialer.C	1.0	0.0	1.0	0.0	1.0	0.0	**1.0**	**0.0**
Agent.FYI	1.0	0.0004	1.0	0.0	1.0	0.0	**1.0**	**0.0**
Allaple.A	0.941	0.071	0.966	0.045	0.981	0.013	**0.984**	**0.008**
Allaple.L	0.855	0.023	0.954	0.015	0.981	0.005	**0.996**	**0.003**
Alueron.gen!J	0.915	0.001	0.983	0.0	0.983	0.0	**1.0**	**0.0**
Autorun.K	1.0	0.0	1.0	0.0	1.0	0.0	**1.0**	**0.0**
C2Lop.P	0.45	0.005	0.533	0.01	0.6	0.012	**0.683**	**0.009**
C2Lop.gen!G	0.386	0.006	0.386	0.003	0.432	0.005	**0.75**	**0.004**
Dialplatform.B	0.962	0.0	0.962	0.0	0.962	0.0	**0.981**	**0.0**
Dontovo.A	1.0	0.0	1.0	0.0	1.0	0.0004	**1.0**	**0.0007**
Fakerean	0.982	0.0	0.991	0.0	1.0	0.003	**1.0**	**0.0004**
Instantaccess	1.0	0.0	1.0	0.0	1.0	0.0007	**1.0**	**0.0004**
Lolyda.AA 1	0.922	0.0	0.953	0.0	0.9688	0.001	**0.984**	**0.0**
Lolyda.AA 2	1.0	0.0	1.0	0.0	0.982	0.0	**1.0**	**0.0004**
Lolyda.AA 3	1.0	0.0	1.0	0.0	1.0	0.0	**1.0**	**0.0**
Lolyda.AT	0.958	0.0004	1.0	0.0	1.0	0.0	**1.0**	**0.0**
Malex.gen!J	0.512	0.0	0.732	0.0	0.951	0.0004	0.878	0.001
Obfuscator.AD	1.0	0.0	1.0	0.0	1.0	0.0004	1.0	0.0
Rbot!gen	0.978	0.0004	1.0	0.0	1.0	0.0	1.0	0.0
Skintrim.N	1.0	0.0	1.0	0.0	0.875	0.0	0.958	0.0
Swizzor.gen!E	0.368	0.004	0.368	0.003	0.368	0.007	0.368	0.007
Swizzor.gen!I	0.3	0.008	0.5	0.004	0.375	0.007	**0.525**	**0.006**
VB.AT	0.983	0.008	0.992	0.0007	0.984	0.001	0.967	0.0004
Wintrim.BX	0.689	0.0	0.69	0.0	0.862	0.0004	**0.862**	**0.001**
Yuner.A	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Accuracy(%)	**90.0**		**93.6**		**95.0**		**96.3**	

The efficacy of classical MLAs and deep learning architectures are evaluated on benchmark dataset and privately collected malware samples. Deep learning architectures outperformed the classical MLAs. This method is agnostic to packing and operating system. Moreover, it takes less time as compared to Static and Dynamic analysis due to the fact that it does not require disassembly or execution inside virtual environment. Additionally, the sub module proposed DeepImageMalDetect (DIMD) which contains convolutional neural network and long short term memory (CNN-LSTM) pipeline for malware categorization has achieved highest accuracy of 96.3% on Dataset 1 with a 10-fold cross validation. The reported results can be further enhanced by using highly complex deep learning architecture and carefully following a hyper parameter technique.

Optimal parameters are set for both the classical machine learning and deep learning algorithms by following a hyper parameter selection technique. The deep learning algorithms performed well in comparison to the classical MLAs. Moreover, the hybrid network CNN-LSTM performed well in comparison to all other algorithms. This has showed accuracy of 96.3% which outperforms the existing methods. The primary reason is that the CNN-LSTM is able to capture both the spatial and temporal features. DIMD is an effective method for malware categorization in comparison to the existing methods which completely avoids disassembly, decompiling, deobfuscation or execution of the binary. Since the hyperparameters plays an important role in achieving better performance, the reported results can be further enhanced by finding optimal parameters.

The rapidly evolving technologies particularly ICT systems generates huge amount of data, typically called as big data. Due to the large volume, large variety, large velocity and large veracity as the big data characteristics, big data causes many challenging issues in applying machine learning algorithms and deep learning architectures. This requires the concepts of data mining and information processing. In deep learning, Autoencoder is the most commonly used method for dimensionality reduction and in classical machine learning most commonly used classical methods for dimensionality reduction are principal component analysis (PCA) and singular value decomposition (SVD). Recently, the application of Autoencoder for cyber security applications is employed [16]. Autoencoder is a generative model which learns the latent representation of different feature sets. It learns significant features in an unsupervised way and found to be suitable method for network traffic analysis because as the amount of data generated by

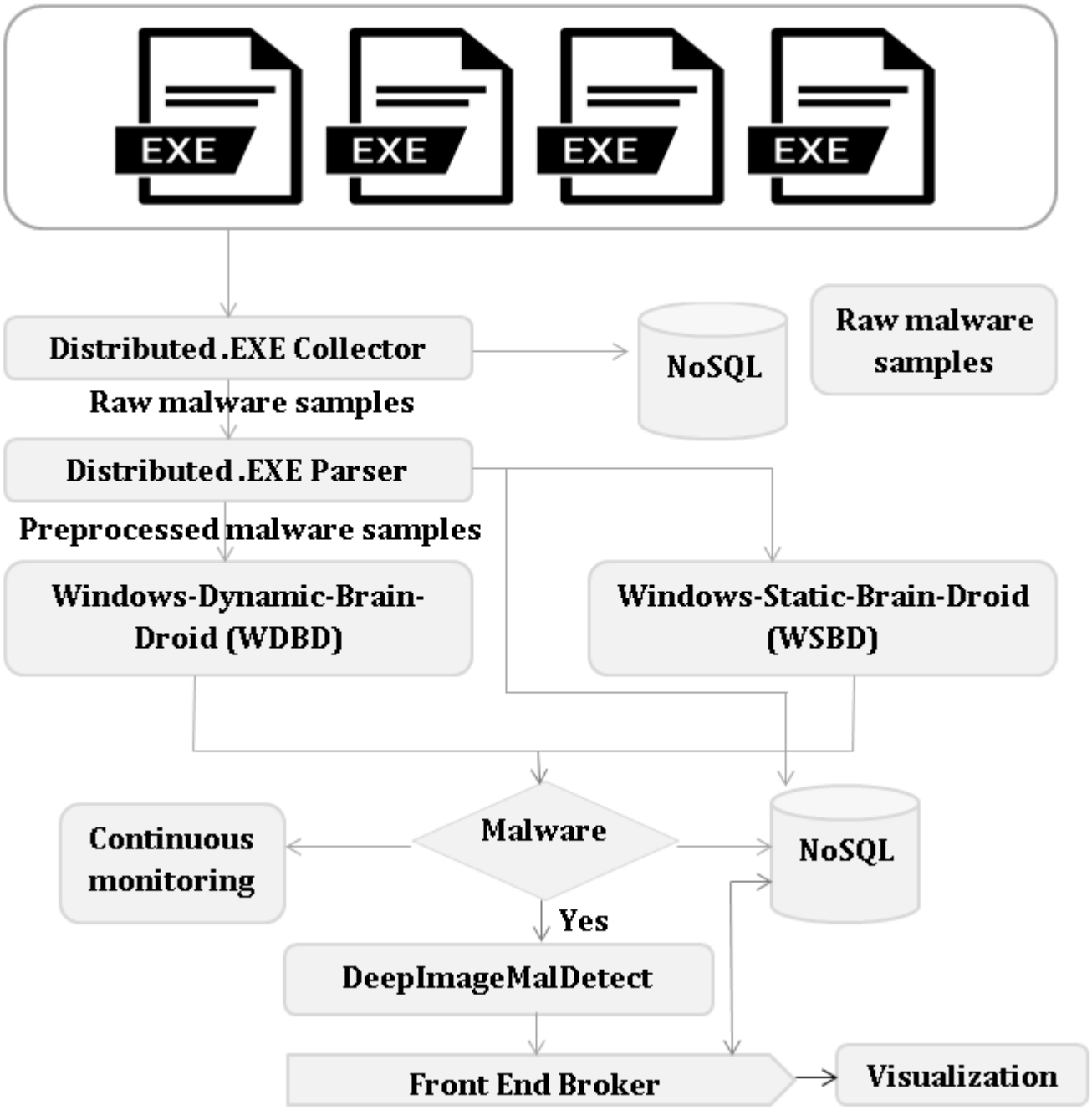

FIGURE 10. Proposed deep learning architecture for real-time malware analysis.

ICT systems is very large in a fraction of system and within this time the data has to be preprocessed without losing any information to attain significant performance. Autoencoder can also be used as dimensionality reduction techniques. Dimensionality reduction technique to get better classification rate can be thoroughly discussed to enhance the performance of the proposed method in this study remained as future work. The source code and the trained models for all the experiments are made publically available for further research.[12]

[12]https://github.com/vinayakumarr/DeepImageMalDetect-DIMD

VIII. PROPOSED ARCHITECTURE - SCALEMALNET

The results obtained from the rigorous experiments conducted in this research work has aided in proposing ScaleMalNet, a malware analysis system that follows a systematic process to collect data internally from various data sources and uses self-learning techniques such as classical machine learning, deep learning and image processing techniques to detect, classify and categorize malware to their corresponding malware family accurately. The framework is highly scalable which facilitates collection of malware samples from different sources and applies pre-processing in a distributed way. The framework incorporates self-learning techniques

TABLE 12. Detailed results for Data set 2.

Methods	10-fold cross validation accuracy(%)
LR	85.2
NB	83.7
KNN	75.8
DT	84.2
AB	81.7
RF	88.4
SVM rbf	89.2
SVM linear	88.3
CNN + SVM [17]	84.2
GRU + SVM [17]	89.8
MLP + SVM [17]	86.7
CNN 1 layer	**90.9**
CNN 2 layer	**94.2**
CNN 1 layer + LSTM	**93.7**
CNN 2 layer + LSTM	**98.8**

to malware analysis such as malware detection, classification and categorization. The performance of deep learning architectures are evaluated over classical machine learning algorithms (MLAs) and an improvement in performance is observed consistently. The framework has malware detection system which uses Static and Dynamic analysis in the first stage. In the second stage the detected malwares of the first stage are passed into second stage to categorize into corresponding malware family. ScaleMalNet architecture is shown in Figure 10.

IX. CONCLUSION

This paper evaluated classical machine learning algorithms (MLAs) and deep learning architectures based on Static analysis, Dynamic analysis and image processing techniques for malware detection and designed a highly scalable framework called ScaleMalNet to detect, classify and categorize zero-day malwares. This framework applies deep learning 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 both the publically available benchmark datasets and privately collected datasets in this study indicated that deep learning based methodologies outperformed classical MLAs. The developed framework is capable of analyzing large number of malwares in real-time, and scaled out to analyze 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. The major finding of this work, weakness and future scope can be summarized as follows:

- Two-stage process scalable malware detection framework is proposed. The proposed framework uses state-of-the-art method, deep learning which detects the malware in first level and in second level the malware is categorized into their corresponding categories.
- The performances obtained by deep learning architectures outperformed classical MLAs in static, dynamic and image processing based malware detection and categorization. However, in the dynamic analysis based malware detection study, the deep learning architectures are applied on the domain knowledge extracted features. This can be avoided by collecting memory dumps for binary files at run time and then memory dump file can be mapped into grayscale image.
- In image processing with deep learning based malware identification study; the malwares were transformed into fixed-sized images and then were flattened. In future work, the spatial pyramid pooling (SPP) layer can be used to allow images of any size to be used as input. This learns features at variable scales and it can be put in between the sub sampling layer and the fully connected layer to improve our models flexibility.
- The malware families in Malimg dataset are highly imbalanced. To handle the multiclass malware families imbalanced issue, cost sensitive approach can be followed. This facilitates to introduce the cost items into the backpropogation learning methodology of deep learning architectures. Primarily the cost item represents the classification importance which provides lower value for the classes that has large number of samples and higher value for the classes that has smaller number of samples.
- The deep learning architectures are vulnerable in an adversarial environment [16]. The method generative adversarial network can be used to generate samples during testing or deployment stage can easily the deep learning architectures can fooled. In the proposed work, the robustness of the deep learning architectures is not discussed. This is one of the significant directions towards future work since the malware defection is an important application in safety-critical environment. A single misclassification can cause several damages to the organization.

ACKNOWLEDGEMENT

The authors would like to thank NVIDIA India, for the GPU hardware support to research grant. They would also like to thank Computational Engineering and Networking (CEN) department for encouraging the research.

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] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Proc. Australas. Joint Conf. Artif. Intell.* Cham, Switzerland: Springer, Dec. 2016, pp. 137–149.

[25] E. Raff *et al.*, "An investigation of byte n-gram features for malware classification," *J. Comput. Virology Hacking Techn.*, vol. 14, no. 1, pp. 1–20, 2018.

[26] 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

[27] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *J. Comput. Virology Hacking Techn.*, vol. 13, no. 1, pp. 1–12, 2017.

[28] L. Nataraj, *A Signal Processing Approach To malware Analysis*. Santa Barbara, CA, USA: Univ. California, 2015.

[29] L. Nataraj, D. Kirat, B. S. Manjunath, and G. Vigna, "Sarvam: Search and retrieVAL of malware," in *Proc. Annu. Comput. Secur. Conf. (ACSAC) Worshop Next Gener. Malware Attacks Defense (NGMAD)*, Dec. 2013, pp. 1–9.

[30] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proc. 4th ACM Workshop Secur. Artif. Intell.* New York, NY, USA: ACM, pp. 21–30.

[31] L. Nataraj, G. Jacob, and B. S. Manjunath, "Detecting packed executables based on raw binary data," Univ. California, Santa Barbara, CA, USA, Tech. Rep., 2010.

[32] M. Farrokhmanesh and A. Hamzeh, "A novel method for malware detection using audio signal processing techniques," in *Proc. Artif. Intell. Robot. (IRANOPEN)*, Apr. 2016, pp. 85–91.

[33] 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.

[34] L. Nataraj and B. S. Manjunath. (2016). "SPAM: Signal processing to analyze malware." [Online]. Available: https://arxiv.org/abs/1605.05280

[35] D. Kirat, L. Nataraj, G. Vigna, and B. S. Manjunath "SigMal: A static signal processing based malware triage," in *Proc. 29th Annu. Comput. Secur. Appl. Conf.* New York, NY, USA: ACM, Dec. 2013, pp. 89–98.

[36] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, Jun. 2011, pp. 315–323.

[37] M. Abadi *et al.* "Tensorflow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16, Nov. 2016, pp. 265–283.

[38] F. Chollet. (2015). *Keras: Deep Learning Library for Theano and Tensorflow*. [Online]. Available: https://keras.io/k,7(8)

[39] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[40] G. Conti, E. Dean, M. Sinda, and B. Sangster, "Visual reverse engineering of binary and data files," in *Visualization for Computer Security*. Berlin, Germany: Springer, 2008, pp. 1–17.

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

[42] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Cham, Switzerland: Springer, 2016, pp. 230–253.

[43] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Comput. Secur.*, vol. 77, pp. 871–885, Aug. 2018.

[44] G. Sun and Q. Qian, "Deep learning and visualization for identifying malware families," *IEEE Trans. Dependable Secure Comput.* to be published.

[45] 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.

R. VINAYAKUMAR received the B.C.A. degree from the JSS college of Arts, Commerce and Sciences, Mysore, in 2011, and the M.C.A. degree from Amrita Vishwa Vidyapeetham, Mysore, in 2014. He is currently pursuing the Ph.D. degree with the Computational Engineering and Networking, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, India. His Ph.D. work centers on the application of machine learning (sometimes deep learning) for cyber security and discusses the importance of natural language processing, image processing, and big data analytics for cyber Security. He has published several papers in machine learning applied to cyber security. He has participated in several international shared tasks and organized a shared task on detecting malicious domain names (DMD 2018) as part of SSCC 2018 and ICACCI 2018.

MAMOUN ALAZAB received the Ph.D. degree in computer science from the School of Science, Information Technology and Engineering, Federation University of Australia. He is currently an Associate Professor with the College of Engineering, IT and Environment, Charles Darwin University, Australia. He is also a Cyber Security Researcher and a Practitioner with industry and academic experience. He works closely with government and industry on many projects. He has published more than 100 research papers. He delivered many invited and keynote speeches, 22 events in 2018 alone. His research interests include the cyber security and digital forensics of computer systems, including current and emerging issues in the cyber environment such as cyber-physical systems and the Internet of Things with a focus on cybercrime detection and prevention. He is a Senior Member of the IEEE. He convened and chaired more than 50 conferences and workshops. He is an Editor on multiple editorial boards, including an Associate Editor of the IEEE Access, in 2017 (impact factor: 3.5), an Editor of the *Security and Communication Networks* Journal, in 2016 (impact factor: 1.067), and a Book Review Section Editor of the *Journal of Digital Forensics, Security and Law* (JDFSL).

K. P. SOMAN has 25 years of research and teaching experience at the Amrita School of Engineering, Coimbatore. He has around 150 publications in national and international journals and conference proceedings. He has organized a series of workshops and summer schools in advanced signal processing using wavelets, Kernel methods for pattern classification, deep learning, and big-data analytics for industry and academia. He has authored books *Insight Into Wavelets* (New Delhi, India: Prentice Hall), *Insight Into Data Mining* (New Delhi, India: Prentice Hall), *Support Vector Machines and Other Kernel Methods* (New Delhi, India: Prentice Hall), and *Signal and Image Processing: The Sparse Way* (Elsevier).

PRABAHARAN POORNACHANDRAN is currently a Professor with Amrita Vishwa Vidyapeetham. He has more than two decades of experience in computer science and security areas. His research interests include malware, critical infrastructure security, complex binary analysis, AI, and machine learning.

SITALAKSHMI VENKATRAMAN received the M.Sc. degree in mathematics and the M.Tech. degree in computer science from IIT Madras, in 1985 and 1987, respectively, the M.Ed. degree from The University of Sheffield, in 2001, and the Ph.D. degree in computer science, with a doctoral thesis titled Efficient Parallel Algorithms for Pattern Recognition, from the National Institute of Industrial Engineering, in 1993.

She has more than 30 years of work experience in both industry and academics–developing turnkey projects for IT industry and teaching a variety of IT courses for tertiary institutions, in India, Singapore, New Zealand, and Australia, since 2007. She is currently a Discipline Leader and a Senior Lecturer in information technology with Melbourne Polytechnic. She specializes in applying efficient computing models and data mining techniques for various industry problems and recently in e-health, e-security, and e-business domains through collaborations with industry and universities in Australia. She has published seven book chapters and more than 130 research papers in internationally well-known refereed journals and conferences. She is a Senior Member of professional societies and editorial boards of international journals and serves as a Program Committee Member of several international conferences every year.