MALWARE DETECTION WITH VIGOROUS ABILITY USING DEEP LEARNING

A Project Report
Submitted by

M.JAYA RAKESH VARMA (19JD1A0558)

S. PRABHU SAI KALYAN (19JD1A0594)

V.SASIDHAR (19JD1A05A3)

M.NITHIN KUMAR (19JD1A0564)

in partial fulfilment for the award of the degree of

BACHELOR OF TECHONOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Under the Esteemed guidance of

Mr. V. RAJESH BABU MTech(CSE)

Assistant Professor

ELURU COLLEGE OF ENGINEERING AND TECHNOLOGY

DUGGIRALA, A.P. (INDIA) – 534004

(AFFILIATED TO JNTU-KAKINADA, ANDHRA PRADESH)

April, 2023

DECLARATION

We here by declare that the project entitled "**MALWARE DETECTION WITH VIGOROUS ABILITY USING DEEP LEARNING**" submitted for the B. Tech. (CSE) degree is our original work and the project has not formed the basis for the award of any other degree, diploma, fellowship, or any other similar titles.

Signature of the student

M.JAYA RAKESH VARMA (19JD1A0558)

S.PRABHU SAI KALYAN (19JD1A0594)

V.SASIDHAR (19JD1A05A3)

M.NITHIN KUMAR (19JD1A0564)

Place:

Date:

CERTIFICATE

This is to certify that the project titled **MALWARE DETECTION WITH VIGOROUS ABILITY USING DEEP LEARNING**" is the bonafide work carried out V.SASIDHAR (19JD1A05A3), S.PRABHU SAI KALYAN (19JD1A0594), M.JAYA RAKESH VARMA (19JD1A0558), M.NITHIN KUMAR (19JD1A0564) students of B.Tech (CSE) of Eluru College of Engineering and Technology, affiliated to JNTU-Kakinada,AP(India) during the academic year 2022-23,in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Computer Science and Engineering) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Signature of the Guide

V.RAJESH BABU **MTech(CSE)**

Associate Professor

Signature of the HOD

Dr. S.SURESH **Ph.D(CSE)**

Professor

External Examiner

VISION – MISSION – PEOs

Vision/Mission/PEOs

Institute Vision	To Achieve Excellence in Engineering Education
Institute Mission	IM1: To deliver quality education through good infrastructure, facilities and committed staff. IM2: To train students as proficient, competent and socially responsible engineers. IM3: To promote research and development activities among faculty and students for betterment of the society.
Department Vision	Empower the students of the computer science and engineering department to be technologically strong, innovative and global citizens maintaining human values.
Department Mission	DM1: Inspire students to become self motivated and problem solving individuals. DM2: Furnish students for professional career with academic excellence and leadership skills. DM3: Create centre of excellence in Computer Science and Engineering. DM4: Empower the youth and rural communities with computer education.
Program Educational Objectives(PEOs)	Graduates of Computer Science & Engineering will: PEO1: Excel in professional career through knowledge in mathematics and engineering principles. PEO2: Able to pursue higher education and research. PEO3: Communicate effectively, recognize, and incorporate societal needs in their professional endeavors. PEO4: Adapt to technological advancements by continuous learning.

POs/PSOs

POs

1	**Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	**Design/development of solutions:** : Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.
12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

PSO1: Have proficiency in Algorithms, Networking's/W Engineering and Web Applications for systematic design of computer based systems of varying complexity.

PSO2: Qualifying National, International level competitive examinations for successful higher studies and employment.

Project Mappings

Batch No:	B-06
Project Title	MALWARE DETECTION WITH VIGOROUS ABILITY USING DEEP LEARNING
Project Domain	DEEP LEARNING
Type of the Project	APPLICATION ORIENTED
Guide Name	Mr. V. RAJESH BABU MTech(CSE)
Student Roll No	Student Name
19JD1A0558	JAYA RAKESH VARMA MANTENA
19JD1A0594	S PRABHU SAI KALYAN
19JD1A05A3	SASIDHAR VEERANKI
19JD1A0564	NITHIN KUMAR MENDEM

Project Title	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2
MALWARE DETECTION WITH VIGOROUS ABILITY USING DEEP LEARNING	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Mapping Level	Mapping Description
1	Low Level Mapping with PO & PSO
2	Moderate Mapping with PO & PSO
3	High Level Mapping with PO & PSO

Mapping Justifications:

Member 1: Jaya Rakesh Varma Mantena (19JD1A0558)

Member 2: Prabhu Sai Kalyan Singireddy (19JD1A0594)

Member 3: Sasidhar Veeranki (19JD1A05A3)

Member 4: Nithin Kumar Mendem (19JD1A0564)

Project Guide

Mr. V.RAJESH BABU MTech(CSE)

ACKNOWLEDGEMENT

First, we would like to express our deepest appreciation to my advisor and guide, **Mr. V.Rajesh Babu** for his sincere guidance, heart full encouragement, and continuous support during my B. Tech project at Eluru College of Engineering and Technology. His vision for insights into the problem resolution has greatly inspired us to put things in the right direction. It has been a great pleasure to conduct our project under his guidance

We must also express our sincere thanks to our B.Tech Project Co-Ordinator and Asst. Professor **Mr. G.Pranith,** The Head of the Department **Dr. S Suresh**, and to all of the faculty members of the Computer Science and Engineering of Eluru College of Engineering and Technology for their endeavors in our academic education.

We would like to express our gratitude to **Dr. P BalaKrishna Prasad**, Principal, and the Management of **ELURU COLLEGE OF ENGINEERING & TECHNOLOGY** for motivating and providing us adequate facilities and continuous improvements and suggestion without which this project would not have seen the light of the day

M.JAYA RAKESH VARMA (19JD1A0558)

S.PRABHU SAI KALYAN (19JD1A0594)

V.SASIDHAR (19JD1A05A3)

M.NITHIN KUMAR (19JD1A0564)

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.

LIST OF FIGURES

INDEX

1.INTRODUCTION

In this digital world of Industry 4.0, the rapid advancement of technologies has affected the daily activities in businesses as well as in personal lives. Internet of Things (IoT) and applications have led to the development of the modern concept of the information society. However, security concerns pose a major challenge in 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 and vulnerability of systems.

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. 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. 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, 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 using a 5step methodology incorporating machine learning algorithm (MLA) such as SVM with program features extracted from large samples of both the benign and malicious executables with 10-fold cross validations.

Later, in 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.

1.1 MOTIVATION:

The rise of technology has led to an increase in cyber threats, including malware attacks, which can cause severe damage to computer systems and compromise personal and sensitive data. Traditional antivirus software and signature-based detection methods are becoming less effective in detecting the latest malware variants, which are often designed to evade detection.

Deep learning, a subset of machine learning, has emerged as a powerful tool for detecting malware. By analyzing large amounts of data, deep learning algorithms can learn to identify patterns and characteristics of malware that are difficult to detect using traditional methods. This allows for more accurate and efficient detection of malware, leading to better protection for individuals and organizations.

The motivation for this project is to develop a malware detection system with a high level of accuracy and reliability, using deep learning techniques. By doing so, we can provide better protection against the growing threat of malware attacks and minimize the potential damage they can cause. This project has the potential to make a significant impact in the field of cybersecurity and improve the security of computer systems worldwide.

1.2 OBJECTIVE:

The objective of this project is to develop a malware detection system with a high level of accuracy and reliability, using deep learning techniques. Specifically, the project aims to:

1. Collect a large and diverse dataset of malware samples for training and testing the deep learning model.
2. Preprocess and feature engineer the dataset to extract relevant features and characteristics of malware samples.
3. Design and train a deep learning model, such as a convolutional neural network (CNN) or a recurrent neural network (RNN), using the preprocessed dataset.
4. Optimize the hyperparameters of the deep learning model to achieve the highest possible accuracy and reduce false positives and false negatives.
5. Evaluate the performance of the trained model on a separate test dataset, measuring metrics such as precision, recall, and F1 score.

2.LITERATURE SURVEY

1.Konrud Rieck,Carsten Willems,Thorsten Holz focused on Automatic Analysis of Malware Behavior Using Machine Learning (2016).

Recently, computer security faces the increase of security challenges. Because the static analysis is perceived as vulnerable to obfuscation and evasion attack, Rieck et al. tries to develop dynamic malware analysis. The primary challenge in using dynamic malware analysis is the time needed to perform the analysis. Furthermore, as the amount and diversity of malware increases, the time required to generate detection patterns is also longer. Therefore, Rieck et al. propose malware detection method to improve the performance of malware detector based on behavior analysis. In this experiment, Malheur datasets were used. These datasets were created by themselves using behavior reports of malware binaries from anti-malware vendors, Sunbelt Software. Com. Each sample was executed and monitored using CW Sandbox's analysis environment and generates 3,131 behavior reports. In this experiment ,Rieck used four main steps. First, malware binaries were executed and monitored in a sandbox environment. It would give system calls and arguments as the output. Then, in step 2, the sequential reports produced from the previous step were embedded into a high-dimensional vector space based on its behavioral pattern. By doing this, the vectorial representation geometrically could be analyzed, to design clustering and classification method. In step 3, the machine learning techniques were applied for clustering and classification to identify the class of malware. Finally, incremental analysis of malware's behavior was done by alternating between clustering and classification step. The result shows that the proposed method successfully reduces the run time and memory requirement by processing the behavior reports in a chunk. Incremental analysis needed 25 min for processing the data, while regular clustering took 100 min. Furthermore, the regular clustering required 5 Gigabytes of memory during computation while incremental analysis only needed less than 300 Megabytes. So, it can be concluded that incremental technique in behavior-based analysis gives better performance in time and memory requirement than regular clustering.

2.Bojan Kolosnjaji,Apostolis Zarras,George Webster,Claudra Eckert focused on the Deep Learning for Classification of Malware System Call Sequences (2017)

Nowadays, the number and variety of malware are kept increasing. As a result, malware detection and classification need to be improved to do safety prevention. This paper wants to model malware system call sequences and uses it to do classification using deep learning. The primary purpose of leveraging machine learning in this experiment is to find a fundamental pattern in a large dataset. They used malware sample dataset gathered from Virus Share, Maltrieve, and private collections. There are three main contributions in this paper. First, Kolosnjaji built DNN and implemented it to examine system call sequences. Then, in order to optimize malware classification process,

convolution neural networks and RNN were combined. Finally, the performance of their proposed method was analyzed by examining the activation pattern of the neural unit. During malware classification process, Kolosnjaji utilized malware collection from the dataset as the input of Cuckoo Sandbox. Then, the sandbox would give numerical feature vector as the output. After that, they used TensorFlow and Theano framework to construct and train the neural networks. It would give a list of malware families as the output of the NN. The neural network consisted of two parts, convolutional part and recurrent part. Convolutional part consisted of convolutional and pooling layer. At first, convolutional layer captured the correlation between neighboring input vectors and generated new features, resulting in feature vectors. Then, the result of the convolutional layer was forwarded to the input of recurrent layer. In recurrent layer, LSTM cells were used to model the resulting sequence and sort the importance based on mean pooling. Finally, dropout and a softmax layer were used to prevent overfitting to occur in the output. As the experiment result, it showed that the combination of convolutional network and LSTM gave better accuracy (89.4%) compared to feedforward network (79.8%) and convolutional network (89.2%).

3.Toshiki shibohra,Takeshi Yugi,Mit Suaki Akiyama,Daiki Chiba focused on the Efficient Dynamic Malware Analysis Based on Network Behavior Using Deep Learning(2018).

Nowadays, malware detection methods can be divided into three categories: static analysis, host behavior-based analysis, and network behavior analysis. Static analysis method can be evaded using packing techniques. Host behavior-based analysis can be deceived by using code injection. As a result, network behavior analysis becomes the spotlight because it does not have those vulnerabilities, and the need to communicate between the attacker and the infected host makes this method effective. One main challenge that hinders the use of network behavior analysis in malware detection is the analysis time. Various malware samples need to be collected and analyzed for a long period because the user does not know when the malwares start their activity. The main idea proposed by Shibahara et al. of this paper is aimed at two characteristics of malware communication, the change in communication purpose and common latent function. For the dataset, firstly they collected malware samples from Virus Total, which were detected as malware by antivirus program. Then, they used malware samples in Virus Total that have different sha1 hash compared to previously collected malware samples. They used 29,562 malware samples in total for training, validation, and classification. Their methodology consists of three main steps: feature extraction, neural network construction, and also training and classification label. First, Shibahara et al. extracted features from communications, which are collected with dynamic malware analysis. Then, these features are used as inputs of recurrent neural network (RNN). Then, in NN construction phase, the change in communication purpose was captured. During the training and classification, the feature vectors of root nodes were calculated according to Virus Total. Then, based on the vectors,

it gave the classification result. During the experiment, the analysis time and time reduction between the proposed method and the regular continuation method were compared. The result shows that the proposed method reduces 67.1% of analysis time and keeps the range of covered URL to 97.9% compared to full analysis method.

4.Ke He,Dong-Seong Kim focused on Malware Detection with Malware Images using Deep Learning Techniques (2019).

Driven by economic benefits, the number of malware attacks is increasing significantly on a daily basis. Malware Detection Systems (MDS) is the first line of defense against malicious attacks, thus it is important for malware detection systems to accurately and efficiently detect malware. Traditional MDS typically utilizes traditional machine learning algorithms that require feature selection and extraction, which are time-consuming and error- prone. Conventional deep learning based approaches typically use Recurrent Neural Network (RNN) which can be vulnerable to redundant API injection. Thus, we investigate the effectiveness of Convolutional Neural Networks (CNN) against redundant API injection. We designed a malware detection system that transforms malware files into image representations and classifies the image representation with CNN. The CNN is implemented with spatial pyramid pooling layers (SPP) to deal with varying size input. We evaluate the effectiveness of SPP and image color space (greyscale/RGB) by measuring the performance of our system on both unaltered data and adversarial data with redundant API injected. Results show that naive SPP implementation is impractical due to memory constraints and greyscale imaging is effective against redundant API injection.

5.Shun Tobiyama,Yukiko Yamagundi,Hajime Shimada,Tomonori Ikusu focused on the Malware Detection with Deep Neural Network Using Process Behavior (2020)

The background problem of this paper is detecting whether there is malware infection on a computer based on the data traffic. To analyze this, usually, expert knowledge is needed, and the amount of time consumed is not a little. As a result, the purpose of this paper is to propose a method to detect malware infection using traffic data by utilizing machine learning. Tobiyama et al. leveraged recurrent neural network (RNN) for feature extraction and CNN for classification. This paper is good because, during the training phase using RNN, they used LSTM. RNN is known with the error vanishing problem because of its sequential structure. Its output depends on the previous input. As a result, when the previous input is getting bigger over time, an error will occur. LSTM avoids the error problem by selecting the only required information for future output to reduce the number of data. In this paper, Tobiyama et al. used 81 malware log files and 69 benign process log files for training and validation. The dataset was generated by using Cuckoo Sandbox to run malware files in an emulated environment. Then, they traced malware process behavior to determine generated and injected processes. The methodology of this paper is as follows: generating log files during

behavior process monitoring; feature extraction using RNN, based on the log files from step 1; converting the extracted features into image features; training the CNN by using the image features; and evaluating the process of validation using trained model. The result shows that the proposed model achieved 92% of detection accuracy. The drawback is the dataset was too small; they performed 5 min logging for 10 times, so the proposed system is not tested for large-scale data.

6.Umme Zahoora,Muttu Krishnan Rajarajan,Asifullah Khan,Saddam Hessain Khanfocused on Ransomware detection using deep learning based unsupervised feature extraction and a cost sensitive Pareto Ensemble classifier (2021).

Ransomware attacks pose a serious threat to Internet resources due to their far-reaching effects. It's Zero-day variants are even more hazardous, as less is known about them. In this regard, when used for ransomware attack detection, conventional machine learning approaches may become datadependent, insensitive to error cost, and thus may not tackle zero-day ransomware attacks. Zero-day ransomware have normally unseen underlying data distribution. This paper presents a CostSensitive Pareto Ensemble strategy, CSPE-R to detect novel Ransomware attacks. Initially, the proposed framework exploits the unsupervised deep Contractive Auto Encoder (CAE) to transform the underlying varying feature space to a more uniform and core semantic feature space. To learn the robust features, the proposed CSPE-R ensemble technique explores different semantic spaces at various levels of detail. Heterogeneous base estimators are then trained over these extracted subspaces to find the core relevance between the various families of the ransomware attacks. Then, a novel Pareto Ensemble-based estimator selection strategy is implemented to achieve a costsensitive compromise between false positives and false negatives. Finally, the decision of selected estimators are aggregated to improve the detection against unknown ransomware attacks. The experimental results show that the proposed CSPE-R framework performs well against zero-day ransomware attacks.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

As of today, the topic of malware detection has received plenty of attention in the literature. However, few works focus on the ML methodologies employed and, to the best of our knowledge, none of them provides a clear classification of mobile malware detection systems based on the metrics and ML techniques used. Focusing on a period spanning from 2017 to 2021, this section chronologically identifies such literature contributions and places them the current work. Yan offered a thorough survey on dynamic mobile malware detection approaches, summarizing a number of criteria and performance evaluation metrics for mobile malware detection. Additionally, the authors analyzed and compared the until then existing mobile malware detection systems based on

the analysis methods and evaluation results. Finally, the authors pointed out open issues in the field and future research directions.

Odusami surveyed mobile malware detection techniques in an effort to identify gaps and provide insight for effective measures against unknown malware. Their work showed that approaches which rely on ML to detect malicious apps were more promising and produced higher detection accuracy as opposed to signature-based techniques.

Kouliaridis provided a holistic review of works on the topic of mobile malware detection and categorized each of them under a unique classification scheme. Precisely, the latter groups each work based on its target platform, feature selection method, and detection techniques, namely signaturebased or anomaly-based detection.

Liu presented a comprehensive survey of malware detection approaches that utilize ML techniques. The authors analyzed and summarized several key topics, including sample acquisition, data preprocessing, feature selection, ML models, algorithms, and detection performance. Finally, they elaborated on the limitations of ML approaches and offered insights for potential future directions.

Gibert surveyed popular ML techniques for malware detection and in particular, deep learning techniques. The authors explained research challenges and limitations of legacy ML techniques and scrutinized recent trends and developments in the field with a focus on deep learning schemes.

Disadvantages of Existing system:

- Not suitable multiple environment
- Can’t detect unknown malware

3.2 PROPOSED SYSTEM

The proposed scheme guides one in picking optimal ML techniques based on the dataset age and analysis method chosen in the first and second step, respectively. Namely, the first two steps associate the age of the dataset used for evaluation with one of the three analysis methods. The third step indicates the ML classification techniques to be used based on the choice made during the preceding steps. The final step depends on whether the dataset used is balanced in terms of malware and benign apps. This will determine if accuracy is indeed a trustworthy metric. In all cases, however, the AUC metric is preferable, as it constitutes a more conclusive and realistic evaluation of models, even when substantially imbalanced datasets are utilized. Generally, AUC quantifies the effectiveness of each examined approach for all possible score thresholds. As a rule, the value of AUC is extracted by examining the ranking of scores rather than their exact values produced when a method is applied to a dataset. On top of everything else, AUC does not depend on the equality of distribution between positive and negative classes.

Advantages of Proposed System:

- High accuracy of detection
- Multipath malware analysis
- More secure

3.3 SYSTEM REQUIREMENTS

Functional Requirements

- Graphical User interface with the User.

Software Requirements

For developing the application the following are the Software Requirements:

- Python
- Django

Operating Systems supported

- Windows 7 / 8 / 10 (64-bit)

Languages used to Develop & Debugger

- Python 3.X
- Any Browser

Hardware Requirements

For developing the application the following are the Hardware Requirements:

- Processor : Pentium IV or higher
- RAM : 4 GB (Min)
- Hard Disk : 500 GB

3.4 FUNCTIONAL REQUIREMENTS:

In software engineering and systems engineering, a **functional requirement** defines a function of a system or its component, where a function is described as a specification of behavior between outputs and inputs.

Functional requirements may involve calculations, technical details, data manipulation and processing, and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describe all the cases where the system uses the functional requirements, these are captured in use cases. Functional requirements are supported by non-functional

requirements (also known as "quality requirements"), which impose constraints on the design or implementation (such as performance requirements, security, or reliability). Generally, functional requirements are expressed in the form "system must do requirement," while non-functional requirements take the form "system shall be requirement." The plan for implementing functional requirements is detailed in the system design, whereas non-functional requirements are detailed in the system architecture. As defined in requirements engineering, functional requirements specify particular results of a system. This should be contrasted with non-functional requirements, which specify overall characteristics such as cost and reliability. Functional requirements drive the application architecture of a system, while non-functional requirements drive the technical architecture of a system. In some cases a requirements analyst generates use cases after gathering and validating a set of functional requirements. The hierarchy of functional requirements collection and change, broadly speaking, is: user/stakeholder request → analyze → use case → incorporate. Stakeholders make a request; systems engineers attempt to discuss, observe, and understand the aspects of the requirement; use cases, entity relationship diagrams, and other models are built to validate the requirement; and, if documented and approved, the requirement is implemented/incorporated. Each use case illustrates behavioral scenarios through one or more functional requirements. Often, though, an analyst will begin by eliciting a set of use cases, from which the analyst can derive the functional requirements that must be implemented to allow a user to perform each use case.

3.5 NON-FUNCTIONAL REQUIREMENTS

Nonfunctional Requirements (NFRs) define system attributes such as

- Security
- Capacity
- Compatibility
- Reliability and Availability
- Maintainability & Manageability
- Recoverability and Serviceability
- Scalability • Usability

These serve as constraints or restrictions on the design of the system across the different backlogs.

3.6 MODULES:

1.USER

1) A new proposal of a scalable and hybrid framework, namely Scale Mal Net which facilitates to collect malware samples from different sources in a distributed way and to apply preprocessing

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) Scale Mal Net follows two stages of 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**.**

2.MALWARE CLASSIFICATION

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 program approach is computationally expensive and the performance is considerably very low. 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. They adopted formatting of agnostic features such as raw byte histogram, byte entropy histogram which was taken from, and in addition employed string extraction.

3.DEEP NEURAL NETWORK (DNN)

A feed forward neural network (FFN) creates a directed graph in which a graph is composed of nodes and edges. 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:

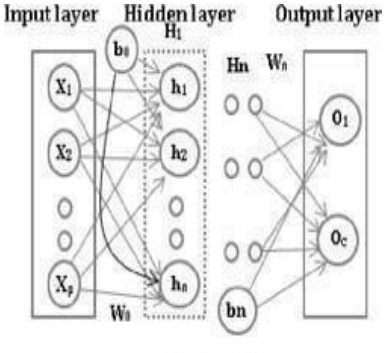


Fig.3.6.1 DNN

4.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. It is 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. 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.

4.SYSTEM DESIGN

4.1 SYSTEM ARICHITECTURE:

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

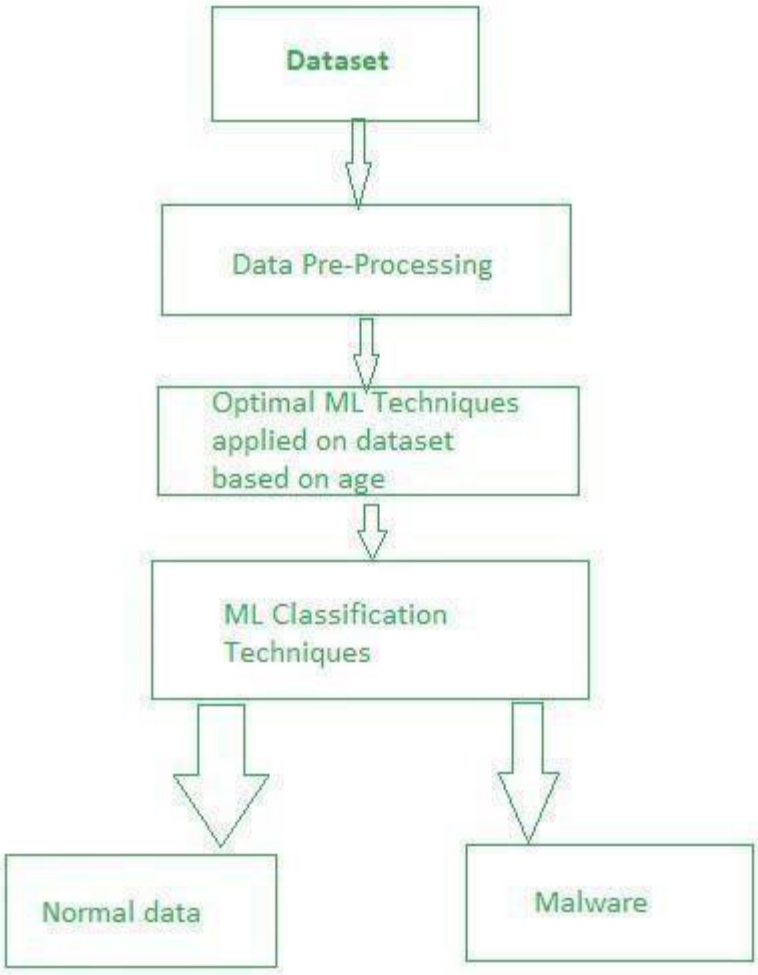


Fig.4.1.1 System Architecture

4.2 UML DIAGRAMS

4.2.1 CLASS DIAGRAM

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object- oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

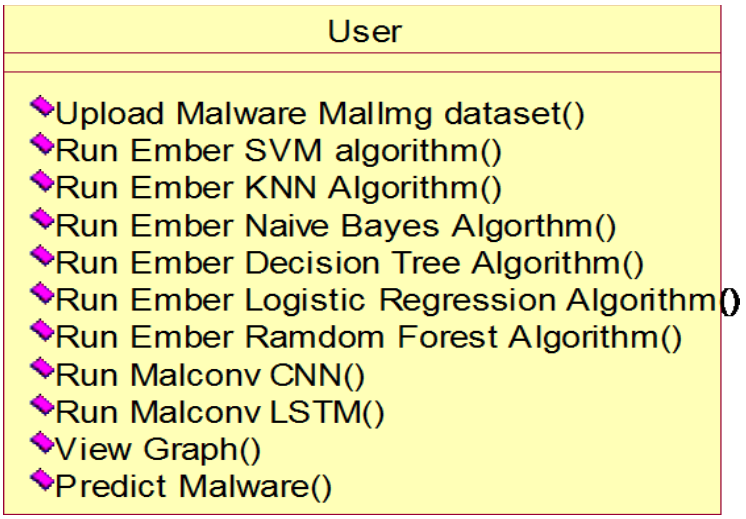


Fig.4.2.1 Class Diagram

4.2.2 USECASE DIAGRAM

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve the scope of your system

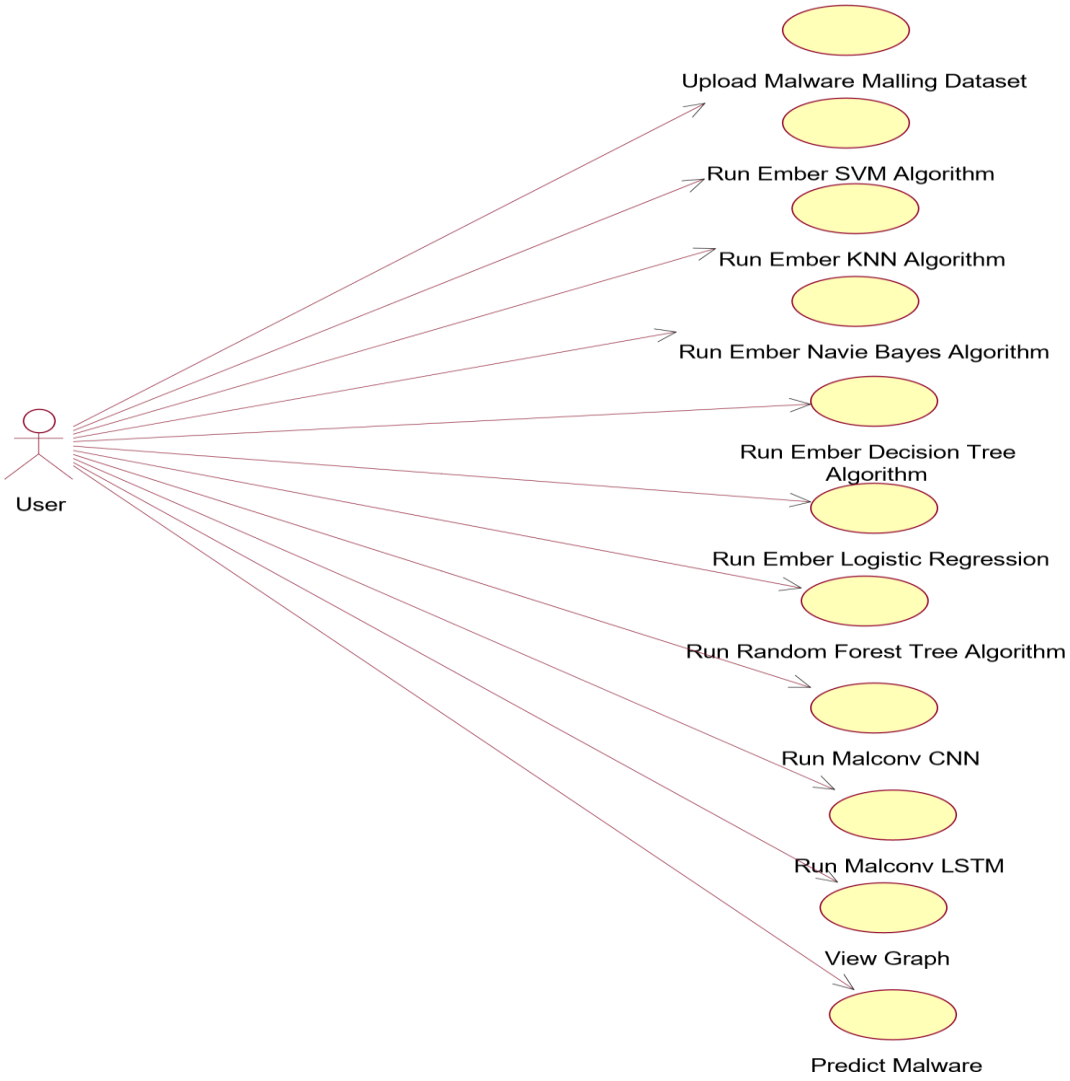


Fig4.2.2 Usecase Diagram

4.2.3 SEQUENCE DIAGRAM

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

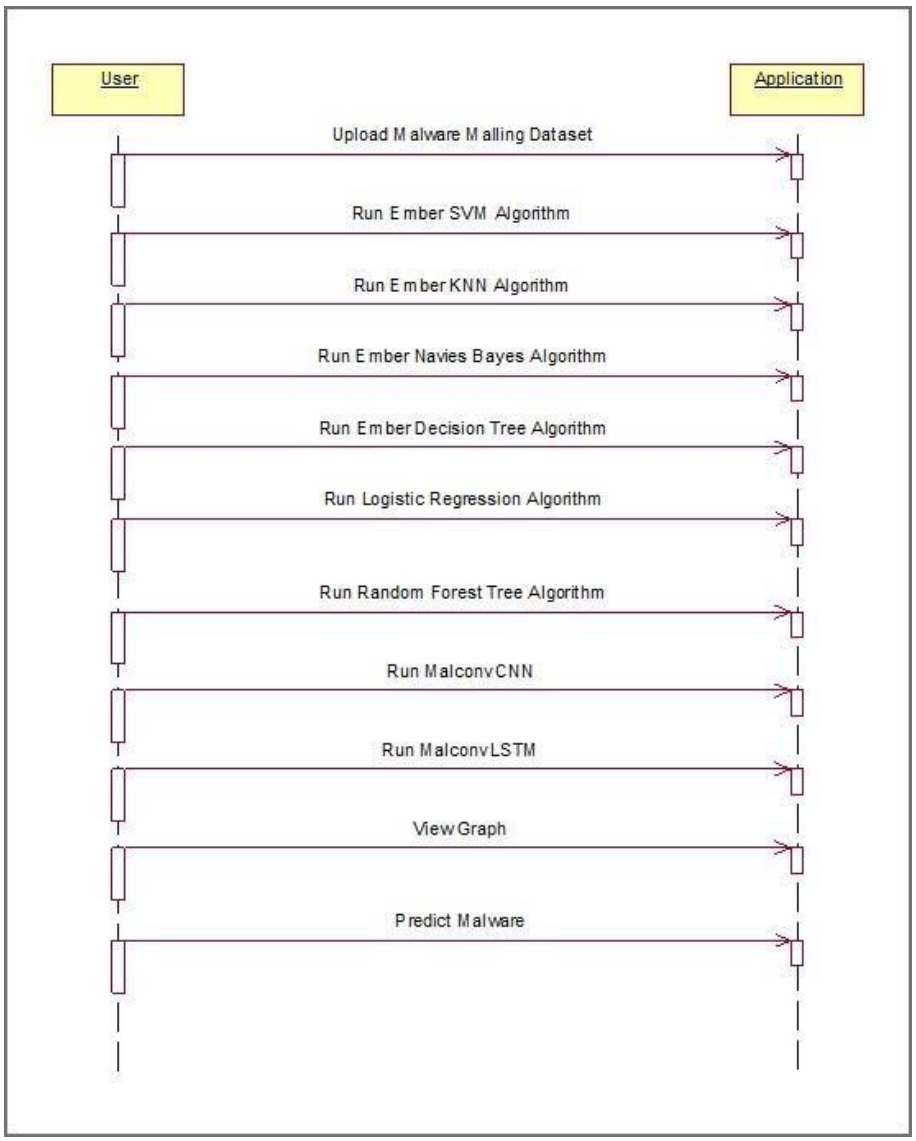


Fig.4.2.3 Sequence Diagram

4.2.4 COLLABORATION DIAGRAM

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

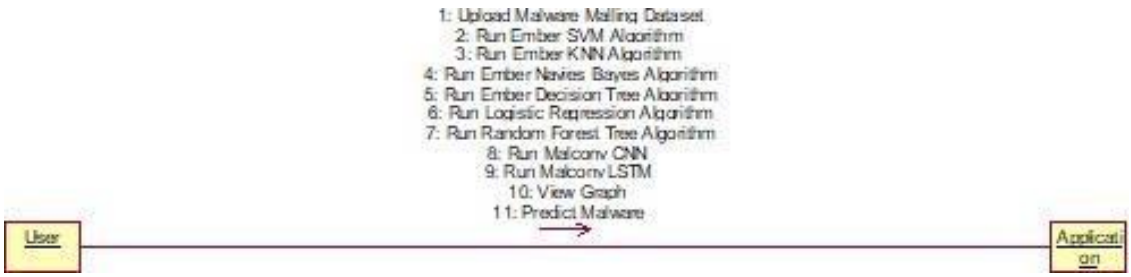


Fig.4.2.4 Collaboration Diagram

5.SYSTEM IMPLEMENTATION

5.1 SOURCE CODE:

```
from tkinter import messagebox
from tkinter import *
from tkinter.filedialog import askopenfilename
from tkinter import simpledialog import tkinter
import numpy as np from tkinter import
filedialog import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score  import
matplotlib.pyplot as plt from sklearn.naive_bayes
import BernoulliNB from sklearn.neighbors import
KNeighborsClassifier from sklearn.tree import
DecisionTreeClassifier from sklearn.metrics import
precision_score from sklearn.metrics import
recall_score from sklearn.metrics import f1_score from
sklearn.metrics import accuracy_score from
sklearn.ensemble import RandomForestClassifier from
sklearn.linear_model import LogisticRegression from
sklearn import svm from keras.models import
Sequential from keras.layers import Convolution2D
from keras.layers import MaxPooling2D from
keras.layers import Flatten
from keras.layers import Dense,Activation,BatchNormalization,Dropout
from sklearn.preprocessing import OneHotEncoder from keras.models
import model_from_json  #from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from sklearn.preprocessing import StandardScaler from
sklearn.preprocessing import Normalizer import
keras.layers
from keras.models import model_from_json  main
= tkinter.Tk()
main.title("Malware Detection with vigorous ability using Deep Learning")
main.geometry("1300x1200")
malware_name = ['Dialer Adialer.C','Backdoor Agent.FYI','Worm Allaple.A','Worm
Allaple.L','Trojan Alueron.gen','Worm:AutoIT Autorun.K',
'Trojan C2Lop.P','Trojan C2Lop.gen','Dialer Dialplatform.B','Trojan Downloader
Dontovo.A','Rogue Fakerean','Dialer Instantaccess',
'PWS Lolyda.AA 1','PWS Lolyda.AA 2','PWS Lolyda.AA 3','PWS Lolyda.AT','Trojan
Malex.gen','Trojan Downloader Obfuscator.AD',
'Backdoor Rbot!gen','Trojan Skintrim.N','Trojan Downloader Swizzor.gen!E','Trojan Downloader
Swizzor.gen!I','Worm VB.AT',
'Trojan Downloader Wintrim.BX','Worm Yuner.A'] global
filename global
knn_precision,nb_precision,tree_precision,svm_precision,random_precision,logistic_precision,cn
n_precision,lstm_precision global
knn_recall,nb_recall,tree_recall,svm_recall,random_recall,logistic_recall,cnn_recall,lstm_recall
global
knn_fmeasure,nb_fmeasure,tree_fmeasure,svm_fmeasure,random_fmeasure,logistic_fmeasure,cn
```

```
n_fmeasure,lstm_fmeasure
global knn_acc,nb_acc,tree_acc,svm_acc,random_acc,logistic_acc,cnn_acc,lstm_acc
global classifier global X_train, X_test, y_train, y_test def load_lstmcnn(dataset,
standardize=True):
    features = dataset['arr'][:, 0]
    features = np.array([feature for feature in features])     features = np.reshape(features,
(features.shape[0], features.shape[1] * features.shape[2]))     if standardize:         features =
StandardScaler().fit_transform(features)          labels  =  dataset['arr'][:,  1]          labels  =
np.array([label for label in labels])
print(labels.shape)     print(features.shape)     return
features, labels def load_data(dataset,
standardize=True):
    features = dataset['arr'][:, 0]
    features = np.array([feature for feature in features])     features = np.reshape(features,
(features.shape[0], features.shape[1] * features.shape[2]))     if standardize:         features =
StandardScaler().fit_transform(features)          labels  =  dataset['arr'][:,  1]          labels  =
np.array([label for label in labels])      feature = []      label = []      for i in range(0,4000):
feature.append(features[i])           label.append(labels[i])       feature = np.asarray(feature)
label = np.asarray(label)     print(labels.shape)     print(features.shape)     print(label.shape)
print(feature.shape)     return feature, label def upload():
    global filename
    filename = filedialog.askopenfilename(initialdir = "dataset")
pathlabel.config(text=filename)     text.delete('1.0', END)
text.insert(END,'MalImg dataset loaded\n') def prediction(X_test,
cls):
    y_pred    =    cls.predict(X_test)
for i in range(len(X_test)):
      print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
return y_pred  def KNN(): global knn_precision  global
    knn_recall global
    knn_fmeasure global
    knn_acc
    text.delete('1.0', END)
    cls = KNeighborsClassifier(n_neighbors = 10)  cls.fit(X_train,
    y_train)
    text.insert(END,"KNN Prediction Results\n\n")
    prediction_data = prediction(X_test, cls)     knn_precision =
precision_score(y_test, prediction_data,average='micro') * 100     knn_recall =
recall_score(y_test, prediction_data,average='micro') * 100     knn_fmeasure =
f1_score(y_test, prediction_data,average='micro') * 100     knn_acc =
accuracy_score(y_test,prediction_data)*100     text.insert(END,"KNN Precision :
"+str(knn_precision)+"\n")     text.insert(END,"KNN Recall : "+str(knn_recall)+"\n")
text.insert(END,"KNN FMeasure :
"+str(knn_fmeasure)+"\n")      text.insert(END,"KNN
Accuracy : "+str(knn_acc)+"\n")      #classifier = cls def
naivebayes():     global nb_precision     global nb_recall
global nb_fmeasure     global nb_acc     text.delete('1.0',
END)
    data, labels = load_data(np.load(filename,allow_pickle=True))
```

```
    X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2)
scaler = Normalizer().fit(X_train)     X_train = scaler.transform(X_train)
scaler = Normalizer().fit(X_test)     X_test = scaler.transform(X_test)     cls =
BernoulliNB(binarize=0.0)      cls.fit(X_train, y_train)
text.insert(END,"Naive Bayes Prediction Results\n\n")      prediction_data =
prediction(X_test, cls)      nb_precision = precision_score(y_test,
prediction_data,average='micro') * 100     nb_recall = recall_score(y_test,
prediction_data,average='micro') * 100     nb_fmeasure = f1_score(y_test,
prediction_data,average='micro') * 100     nb_acc =
accuracy_score(y_test,prediction_data)*100     text.insert(END,"Naive Bayes Precision
: "+str(nb_precision)+"\n")     text.insert(END,"Naive Bayes Recall :
"+str(nb_recall)+"\n")     text.insert(END,"Naive Bayes FMeasure :
"+str(nb_fmeasure)+"\n")     text.insert(END,"Naive Bayes Accuracy :
"+str(nb_acc)+"\n") def decisionTree():     text.delete('1.0', END)     global tree_acc
global tree_precision     global tree_recall     global tree_fmeasure
    rfc = DecisionTreeClassifier(criterion = "entropy", splitter = "random", max_depth = 20,
min_samples_split = 50, min_samples_leaf = 20, max_features = 5)     rfc.fit(X_train,
y_train) text.insert(END,"Decision Tree Prediction Results\n")  prediction_data =
prediction(X_test, rfc)   tree_precision = precision_score(y_test,
prediction_data,average='micro') * 100  tree_recall = recall_score(y_test,
prediction_data,average='micro') * 100
    tree_fmeasure = f1_score(y_test, prediction_data,average='micro') * 100
    tree_acc = accuracy_score(y_test,prediction_data)*100
    text.insert(END,"Decision Tree Precision : "+str(tree_precision)+"\n")
    text.insert(END,"Decision Tree Recall : "+str(tree_recall)+"\n")
    text.insert(END,"Decision Tree FMeasure : "+str(tree_fmeasure)+"\n")
    text.insert(END,"Decision Tree Accuracy : "+str(tree_acc)+"\n")
def randomForest():
   text.delete('1.0', END)     global
random_acc     global
random_precision     global
random_recall     global
random_fmeasure
   rfc   =   RandomForestClassifier(n_estimators=200,  random_state=0)   rfc.fit(X_train,
y_train)
   text.insert(END,"Random   Forest   Prediction   Results\n")
prediction_data = prediction(X_test, rfc)      random_precision =
precision_score(y_test, prediction_data,average='micro') * 100     random_recall =
recall_score(y_test, prediction_data,average='micro') * 100     random_fmeasure =
f1_score(y_test, prediction_data,average='micro') * 100     random_acc =
accuracy_score(y_test,prediction_data)*100     text.insert(END,"Random Forest
Precision : "+str(random_precision)+"\n")     text.insert(END,"Random Forest Recall
: "+str(random_recall)+"\n")     text.insert(END,"Random Forest FMeasure :
"+str(random_fmeasure)+"\n")     text.insert(END,"Random Forest Accuracy :
"+str(random_acc)+"\n") def logisticRegression():     text.delete('1.0', END)     global
logistic_acc     global logistic_precision     global logistic_recall     global
logistic_fmeasure
   rfc   =   LogisticRegression(penalty='l2',   dual=False,   tol=0.002,   C=2.0)
rfc.fit(X_train, y_train)
   text.insert(END,"Logistic  Regression  Prediction  Results\n")  prediction_data  =
prediction(X_test, rfc)
```

```
    logistic_precision = precision_score(y_test, prediction_data,average='micro') * 100
logistic_recall = recall_score(y_test, prediction_data,average='micro') * 100
logistic_fmeasure = f1_score(y_test, prediction_data,average='micro') * 100     logistic_acc
= accuracy_score(y_test,prediction_data)*100      text.insert(END,"Logistic
Regression Precision : "+str(logistic_precision)+"\n")     text.insert(END,"Logistic
Regression Recall : "+str(logistic_recall)+"\n")     text.insert(END,"Logistic
Regression FMeasure : "+str(logistic_fmeasure)+"\n")
text.insert(END,"Logistic Regression Accuracy : "+str(logistic_acc)+"\n") def
SVM():
    text.delete('1.0', END)
global svm_acc     global
svm_precision     global svm_recall
global
svm_fmeasure
    global   X_train,   X_test,   y_train,   y_test      data,   labels   =
    load_data(np.load(filename,allow_pickle=True))
    X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2) print("hello")  rfc
    = svm.SVC(C=2.0,gamma='scale',kernel = 'rbf', random_state = 2) rfc.fit(X_train,
    y_train)
    text.insert(END,"SVM Prediction Results\n")  prediction_data =
    prediction(X_test, rfc)  svm_precision = precision_score(y_test,
    prediction_data,average='micro') * 100 svm_recall = recall_score(y_test,
    prediction_data,average='micro') * 100 svm_fmeasure = f1_score(y_test,
    prediction_data,average='micro') * 100
    svm_acc = accuracy_score(y_test,prediction_data)*100
text.insert(END,"SVM Precision : "+str(svm_precision)+"\n")
text.insert(END,"SVM Recall : "+str(svm_recall)+"\n")
text.insert(END,"SVM FMeasure : "+str(svm_fmeasure)+"\n")
text.insert(END,"SVM Accuracy : "+str(svm_acc)+"\n") def LSTM():
    global lstm_acc
global lstm_precision
global lstm_recall     global
lstm_fmeasure
text.delete('1.0', END)
    data, labels = load_lstmcnn(np.load(filename,allow_pickle=True))
labels = labels.reshape((9339,1))     print(labels)
    data = data.reshape((9339, 32, 32))
    X_train1, X_test1, y_train1, y_test1 = train_test_split(data, labels, test_size=0.101)
print("X_test.shape before  = ",X_test1.shape)     X_test1 = X_test1.reshape((944, 32,
32))          print("X_test.shape after  = ",X_test1.shape)          print("y_test.shape  =
",y_test1.shape)                                                  model          =          Sequential()
model.add(keras.layers.LSTM(100,input_shape=(32, 32)))
model.add(Dropout(0.5))     model.add(Dense(100,
activation='relu'))     model.add(Dense(25, activation='softmax'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())                  model.fit(X_train1,   y_train1,   epochs=8,   batch_size=64)
prediction_data = model.predict(X_test1)     prediction_data =
np.argmax(prediction_data, axis=1)     y_test1 = np.argmax(y_test1, axis=1)
    lstm_precision = precision_score(y_test1, prediction_data,average='micro') * 100
```

```
lstm_recall = recall_score(y_test1, prediction_data,average='micro') * 100
lstm_fmeasure = f1_score(y_test1, prediction_data,average='micro') * 100
lstm_acc = accuracy_score(y_test1,prediction_data)*100     text.insert(END,"LSTM
Prediction Results\n")      text.insert(END,"LSTM Precision :
"+str(lstm_precision)+"\n")     text.insert(END,"LSTM Recall :
"+str(lstm_recall)+"\n")     text.insert(END,"LSTM FMeasure :
"+str(lstm_fmeasure)+"\n")     text.insert(END,"LSTM Accuracy :
"+str(lstm_acc)+"\n")
   #scores = model.evaluate(X_test, y_test, verbose=0)
#print("Accuracy: %.2f%%" % (scores[1]*100)) def CNN():
    global cnn_acc global
    cnn_precision global
    cnn_recall global
    cnn_fmeasure
    text.delete('1.0',
    END)                    data,          labels          =
    load_lstmcnn(np.load(filename,allow_pickle=True))
    labels = labels.reshape((9339,1)) print(labels)
    data = data.reshape((9339, 32, 32))
   X_train1, X_test1, y_train1, y_test1 = train_test_split(data, labels, test_size=0.101)
enc = OneHotEncoder()     enc.fit(y_train1)      print(y_train1.shape)     y_train1  =
enc.transform(y_train1)     y_test1   = enc.transform(y_test1)
   #rehsaping traing     print("X_train.shape before
= ",X_train1.shape)     X_train1 =
X_train1.reshape((8395,          32,          32,1))
print("X_train.shape after  = ",X_train1.shape)
   print("y_train.shape  = ",y_train1.shape)
   #rehsaping testing      print("X_test.shape before   =
",X_test1.shape)     X_test1 =
X_test1.reshape((944,       32,      32,1))       print("X_test.shape after
= ",X_test1.shape)
print("y_test.shape          =          ",y_test1.shape)
classifier = Sequential()
   classifier.add(Convolution2D(32, (3, 3), border_mode='valid', input_shape=(32, 32, 1)))
classifier.add(BatchNormalization())     classifier.add(Activation("relu"))
   classifier.add(Convolution2D(32, (3, 3),  border_mode='valid'))
classifier.add(BatchNormalization())     classifier.add(Activation("relu"))
   classifier.add(MaxPooling2D(pool_size=(2, 2)))
   classifier.add(Flatten())
classifier.add(Dense(128))
classifier.add(BatchNormalization())
classifier.add(Activation("relu"))     classifier.add(Dense(25))
classifier.add(BatchNormalization())
classifier.add(Activation("softmax"))
   classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
classifier.fit(X_train1, y_train1,epochs = 10,batch_size=64)     prediction_data =
classifier.predict(X_test1)     prediction_data = np.argmax(prediction_data, axis=1)
y_test1 = np.argmax(y_test1, axis=1)
   cnn_precision = precision_score(y_test1, prediction_data,average='micro') * 100
```

```
cnn_recall = recall_score(y_test1, prediction_data,average='micro') * 100
cnn_fmeasure = f1_score(y_test1, prediction_data,average='micro') * 100
cnn_acc = accuracy_score(y_test1,prediction_data)*100     text.insert(END,"CNN
Prediction Results\n")       text.insert(END,"CNN Precision :
"+str(cnn_precision)+"\n")     text.insert(END,"CNN Recall :
"+str(cnn_recall)+"\n")     text.insert(END,"CNN FMeasure :
"+str(cnn_fmeasure)+"\n")     text.insert(END,"CNN Accuracy :
"+str(cnn_acc)+"\n") def predict():
    filename = filedialog.askopenfilename(initialdir = "images")
   text.delete('1.0', END) text.insert(END,filename+" loaded\n\n") with open('model.json', "r")
as json_file:        loaded_model_json = json_file.read()        loaded_model =
model_from_json(loaded_model_json)     loaded_model.load_weights("model_weights.h5")
loaded_model._make_predict_function()        print(loaded_model.summary())     img =
np.load(filename)     im2arr = img.reshape(1,32,32,1)     preds =
loaded_model.predict(im2arr)          print(str(preds)+"  "+str(np.argmax(preds)))          predict  =
np.argmax(preds)     text.insert(END,'Uploaded file contains malware from family :
'+malware_name[predict]) def precisionGraph():     height =
[knn_precision,nb_precision,tree_precision,svm_precision,random_precision,logistic_precision,cn
n_precision,lstm_precision]
   bars = ('KNN Precision', 'NB Precision','DT Precision','SVM Precision','RF Precision','LR
Precision','CNN Precision','LSTM Precision')
y_pos = np.arange(len(bars))     plt.bar(y_pos,
height)     plt.xticks(y_pos, bars)     plt.show()
def recallGraph():
height =
[knn_recall,nb_recall,tree_recall,svm_recall,random_recall,logistic_recall,cnn_recall,lstm_recall]
bars = ('KNN Recall', 'NB Recall','DT Recall','SVM Recall','RF Recall','LR Recall','CNN
Recall','LSTM Recall')     y_pos = np.arange(len(bars))     plt.bar(y_pos, height)     plt.xticks(y_pos,
bars)     plt.show() def fscoreGraph():
   height =
[knn_fmeasure,nb_fmeasure,tree_fmeasure,svm_fmeasure,random_fmeasure,logistic_fmeasure,cn
n_fmeasure,lstm_fmeasure]
   bars = ('KNN FScore', 'NB FScore','DT FScore','SVM FScore','RF FScore','LR FScore','CNN
FScore','LSTM  FScore')          y_pos  =  np.arange(len(bars))          plt.bar(y_pos,  height)
plt.xticks(y_pos, bars)     plt.show()  def accuracyGraph():
   height = [knn_acc,nb_acc,tree_acc,svm_acc,random_acc,logistic_acc,cnn_acc,lstm_acc]
bars = ('KNN ACC', 'NB ACC','DT ACC','SVM ACC','RF ACC','LR ACC','CNN ACC','LSTM
ACC')     y_pos = np.arange(len(bars))
plt.bar(y_pos, height)
plt.xticks(y_pos, bars)     plt.show()
font = ('times', 16,
'bold')
title = Label(main, text='Malware Detection with vigorous ability using Deep Learning')
title.config(bg='dark goldenrod', fg='white')
title.config(font=font)          title.config(height=3,
width=120)       title.place(x=0,y=5) font1 =
('times', 14, 'bold')
upload = Button(main, text="Upload Malware MalImg Dataset", command=upload)
upload.place(x=700,y=100) upload.config(font=font1)   pathlabel = Label(main)
pathlabel.config(bg='DarkOrange1', fg='white')
```

```
pathlabel.config(font=font1)           pathlabel.place(x=700,y=150)  svmButton =
Button(main, text="Run Ember SVM Algorithm", command=SVM)
svmButton.place(x=700,y=200) svmButton.config(font=font1)
knnButton = Button(main, text="Run Ember KNN Algorithm", command=KNN)
knnButton.place(x=700,y=250) knnButton.config(font=font1)
nbButton = Button(main, text="Run Ember Naive Bayes Algorithm", command=naivebayes)
nbButton.place(x=700,y=300) nbButton.config(font=font1)
treeButton = Button(main, text="Run Ember Decision Tree Algorithm", command=decisionTree)
treeButton.place(x=700,y=350) treeButton.config(font=font1)
lrButton = Button(main, text="Run Ember Logistic Regression Algorithm",
command=logisticRegression) lrButton.place(x=700,y=400) lrButton.config(font=font1)
randomButton = Button(main, text="Run Ember Random Forest Algorithm",
command=randomForest) randomButton.place(x=700,y=450)
randomButton.config(font=font1)
cnnButton = Button(main, text="Run MalConv CNN", command=CNN)
cnnButton.place(x=700,y=500) cnnButton.config(font=font1) lstmButton =
Button(main, text="Run MalConv LSTM", command=LSTM)
lstmButton.place(x=900,y=500) lstmButton.config(font=font1)
graphButton = Button(main, text="Precision Graph", command=precisionGraph)
graphButton.place(x=700,y=550) graphButton.config(font=font1)
recallButton = Button(main, text="Recall Graph", command=recallGraph)
recallButton.place(x=900,y=550) recallButton.config(font=font1)
scoreButton = Button(main, text="Fscore Graph", command=fscoreGraph)
scoreButton.place(x=700,y=600) scoreButton.config(font=font1)
accButton = Button(main, text="Accuracy Graph", command=accuracyGraph)
accButton.place(x=900,y=600) accButton.config(font=font1)
predictButton = Button(main, text="Predict Malware Family", command=predict)
predictButton.place(x=700,y=650) predictButton.config(font=font1)
font1 = ('times', 12, 'bold')
text=Text(main,height=30,width=80)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=10,y=100) text.config(font=font1)
main.config(bg='turquoise') main.mainloop()
```

5.2 ALGORITHMS:

Support Vector Machines:

The support vector machines (SVM) algorithm is another well-known supervised machine learning model created by Vladimir Vapnik that is capable of both classification and regression problems. Although, it is more commonly used for classification problems rather than regression. The SVM algorithm is capable of splitting the given data points into different groups. This is done by having our algorithm plot the data and then drawing the most suitable line to split the data into multiple categories.

As you can see in the below figure, the drawn line perfectly splits the dataset into 2 different groups, blue and green.

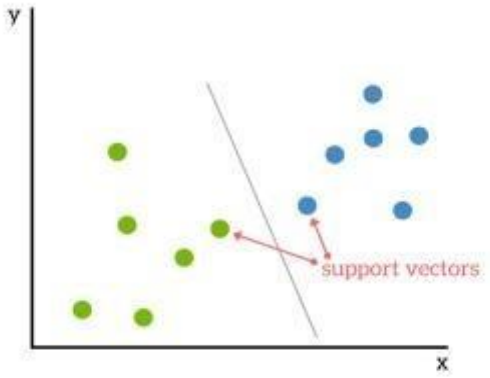


FIG 5.2.1 : SUPPORT VECTOR - A

The SVM model can draw lines or hyperplanes depending on the dimensions of the plotted graph. Lines can only work with 2-dimensional datasets meaning datasets with only 2 columns.

As multiple features may be used to predict the given dataset, higher dimensions will be necessary. In cases where our data set will result in more than 2 dimensions, the Support Vector Machine model will draw a more fitting hyperplane.

For our Support Vector Machine Python example, we will do a species classification for 3 different flower types. Our independent variable will include all the given features of a certain flower, while our dependent variable will be the specified species that the flower does belong to.

Our flower species include **Iris-setosa**, **Iris-versicolor**, and **Iris-virginica**.

To find the used dataset check the Iris Flower Dataset on Kaggle. Below is a screenshot of the given dataset.

STEPS FOR SVM ALGORITHM:

Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image**:**

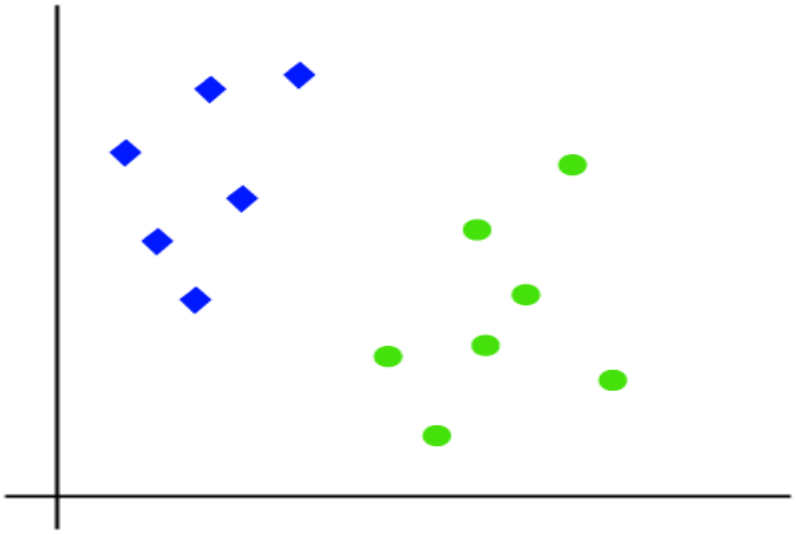


FIG 5.2.2 : Support Vector - B

So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

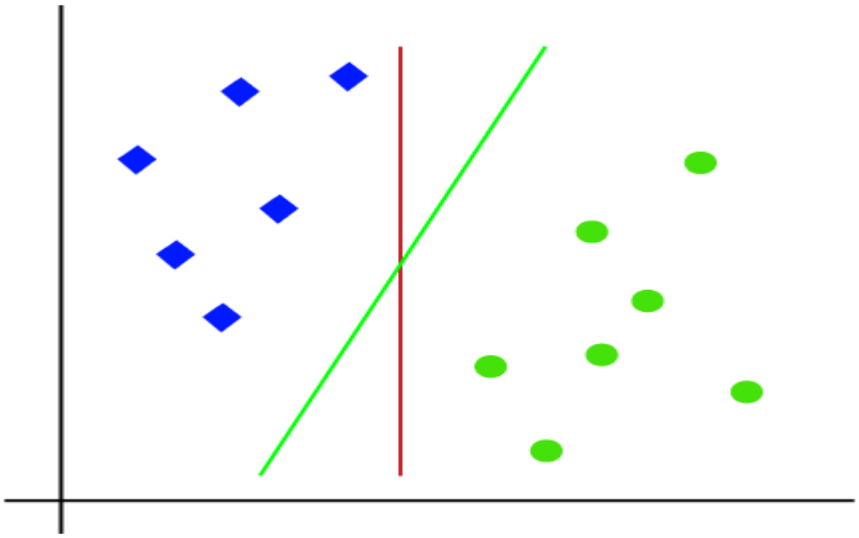


FIG 5.2.3 : Support Vector - C

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.

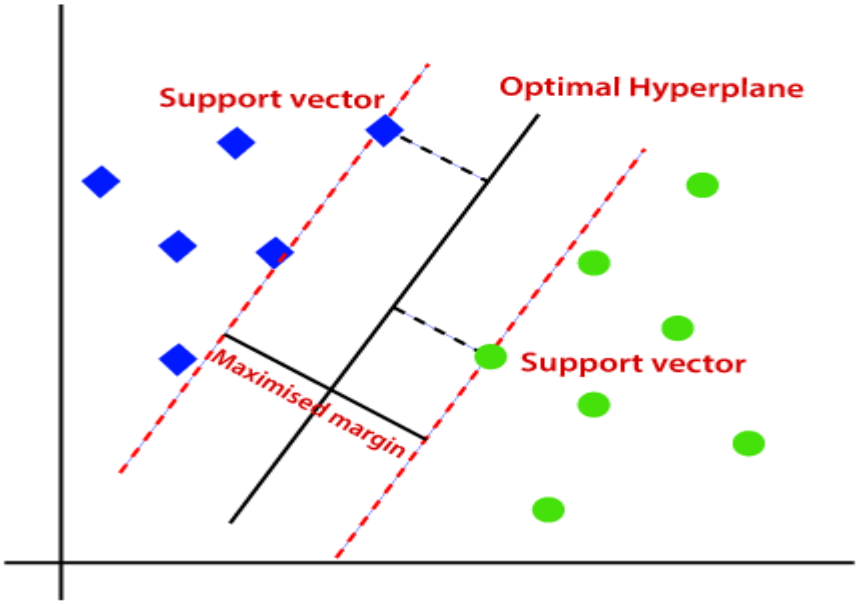


FIG 5.2.4 : SUPPORT VECTOR - D

Decision Tree Algorithm:

Decision tree algorithm is a supervised machine learning model that utilizes tree-like structures for decision making. Decision trees are usually used for classification problems in which the model can decide the group to which a given item in a dataset belongs.

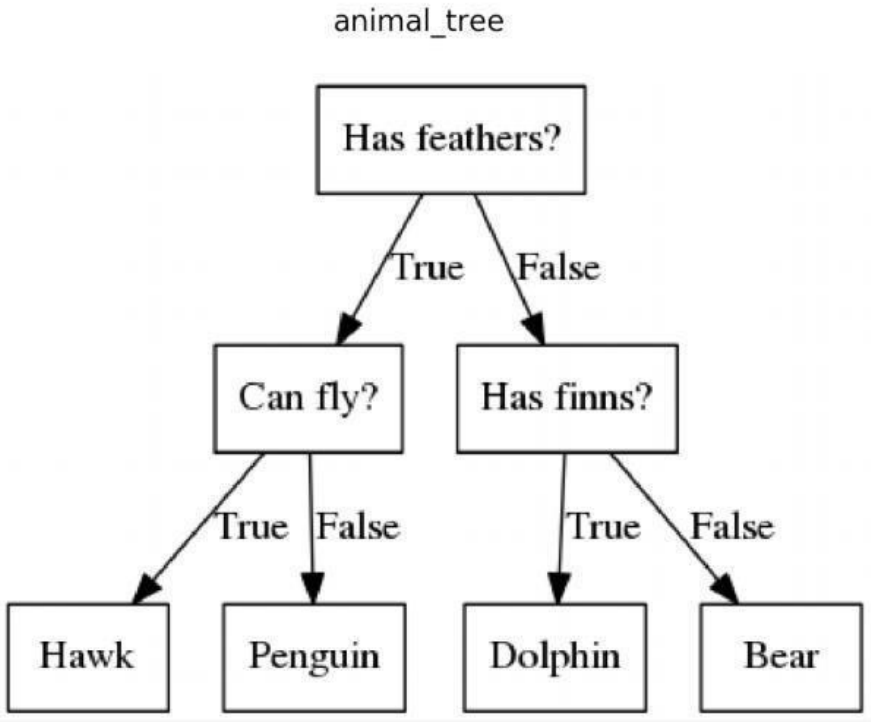


FIG 5.2.5 : DECISION TREE

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3 Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Random Forest:

Considered a more complicated algorithm, the Randomforest algorithm operates by constructing a multitude of decision trees to reach its final goal. Meaning that multiple decision trees are built simultaneously, each returning its own result, which is then combined for a better result. For classification problems, the random forest model would generate multiple decision trees and would classify a given object depending on the classification group predicted by most of the trees.This way the model can fix any overfitting caused by a single tree. Random forest algorithms can also be used for regression, although they may result in poor outcomes.

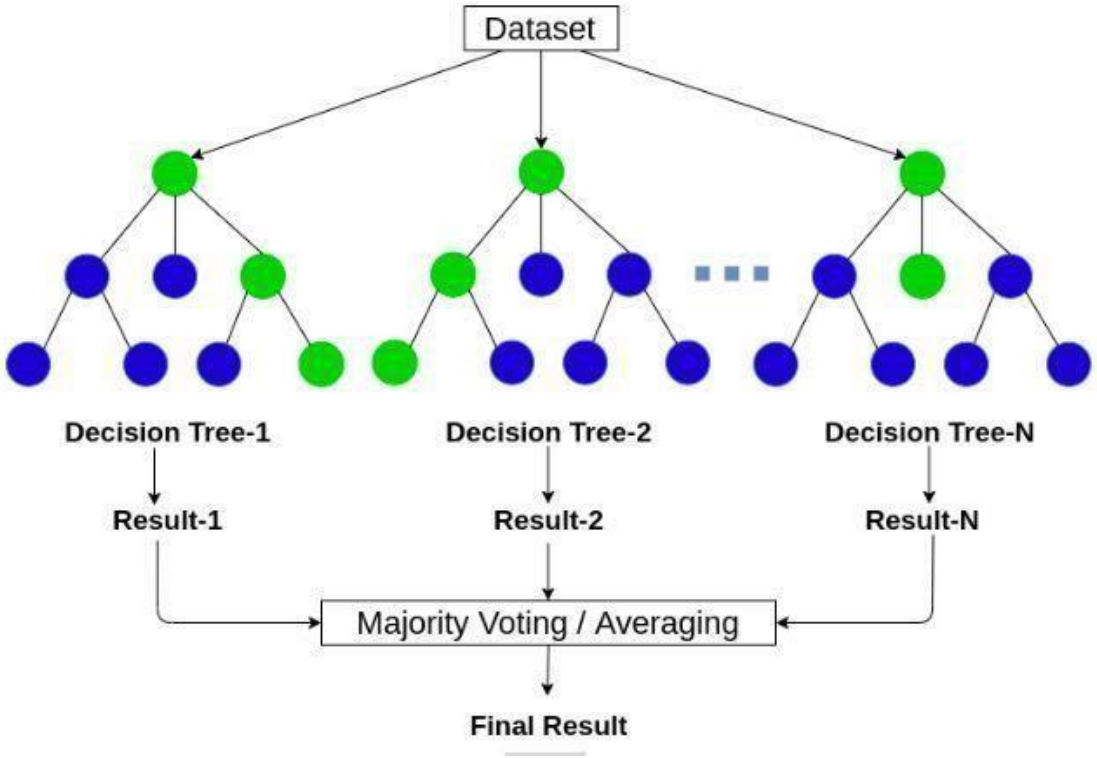


FIG 5.2.6 :RANDOM FOREST

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2. **k-nearest neighbors:**

The k-nearest neighbor (KNN) algorithm is a supervised machine learning method that groups all given data into separate groups.This grouping is based on shared common features

between different individuals. The KNN algorithm can be used for both classification and regression problems. An example of a KNN problem would be the classification of animal images into different groupsets. To review what we learned in this article, we started by defining supervised machine learning and the two types of problems that it can solve. Moving on we explained classification and regression problems and gave some examples of output data types for each.

Then we explained what a linear regression is and how it works and provided a concrete example in Python which tries to predict the Y value depending on the independent X variables. We also explained a similar model to the linear regression one, which is the logistic regression model. We stated what is logistic regression and gave a classification model example that classifies the given images into a given flower species. For our last algorithm, we had the Support Vector Machine algorithm, which we also used to predict the given flower species of 3 different flower species.

To end our article, we briefly explained other well-known supervised machine learning algorithms, such as Decision trees, Random Forests, and the K-nearest neighbor algorithms.

Whether you're reading this article for school, work, or fun, we think that starting with these primary algorithms may be a good way to start your new machine learning passion.

If you are interested and would like to know more about the machine learning wo,rld we would recommend learning more about how such algorithms work and how you can tune such models to improve their performance even further. Don't forget that there are tons of other supervised algorithms out there for you to learn more about.

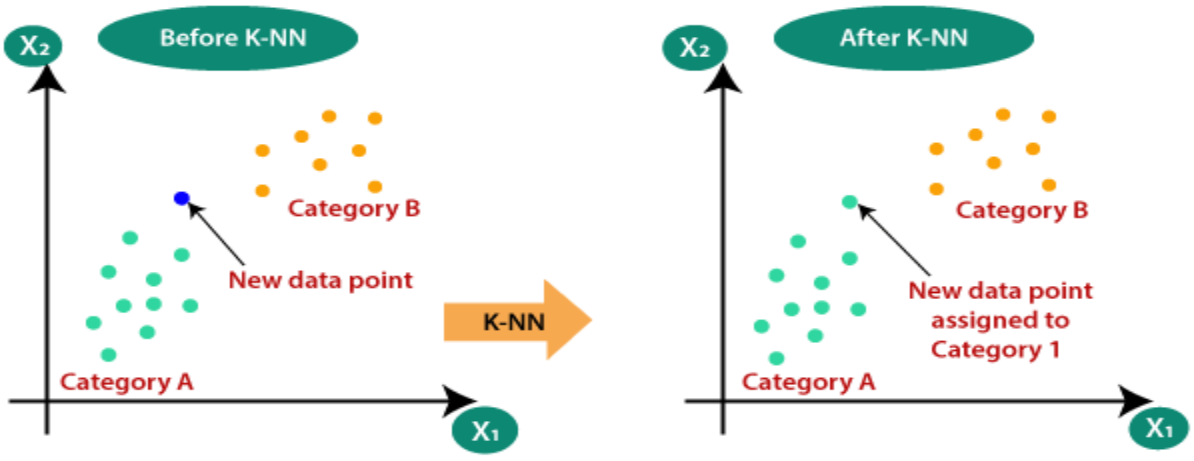


FIG 5.2.7:K-NN(A)

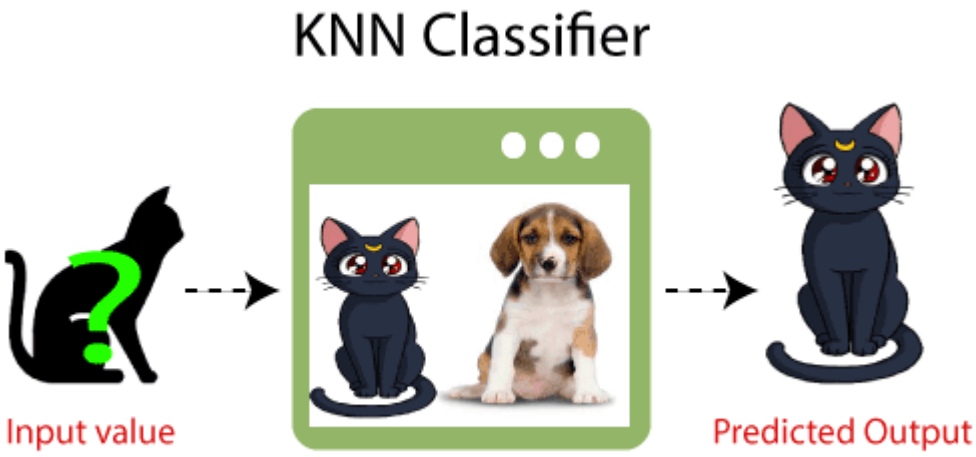


FIG 5.2.8 :K-NN(B)

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of K number of neighbors

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready**.**

Neural Networks:

At earlier times, the conventional computers incorporated algorithmic approach that is the computer used to follow a set of instructions to solve a problem unless those specific steps need that the computer need to follow are known the computer cannot solve a problem. So, obviously, a person is needed in order to solve the problems or someone who can provide instructions to the computer so as to how to solve that particular problem. It actually restricted the problem-solving capacity of conventional computers to problems that we already understand and know how to solve.

But what about those problems whose answers are not clear, so that is where our traditional approach face failure and so Neural Networks came into existence. Neural Networks processes information in a similar way the human brain does, and these networks actually learn from examples, you cannot program them to perform a specific task. They will learn only from past experiences as well as examples, which is why you don't need to provide all the information regarding any specific task. So, that was the main reason why neural networks came into existence.

A neural network comprises of three main layers, which are as follows;

Input layer: The input layer accepts all the inputs that are provided by the programmer.

Hidden layer: In between the input and output layer, there is a set of hidden layers on which computations are performed that further results in the output.

Output layer: After the input layer undergoes a series of transformations while passing through the hidden layer, it results in output that is delivered by the output layer.

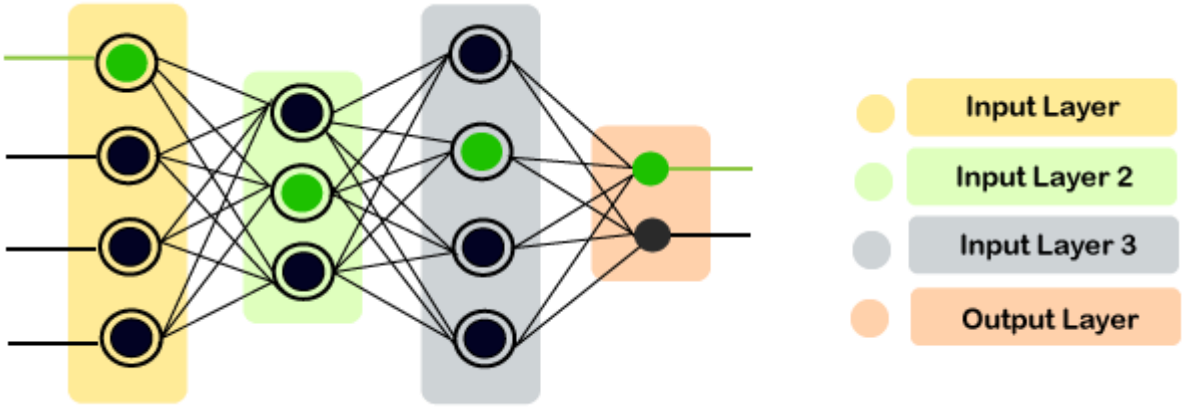


FIG 5.2.9 :NEURAL NETWORK STEPS IN

NEURAL NETWORKS:

Instead of directly getting into the working of Artificial Neural Networks, lets breakdown and try to understand Neural Network's basic unit, which is called a Perceptron. So, a perceptron can be defined as a neural network with a single layer that classifies the linear data. It further constitutes four major components, which are as follows:

Inputs

Weights and Bias

Summation Functions

Activation or transformation function

6. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS:

UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	:	Identified classes of valid input must be accepted.
Invalid Input	:	Identified classes of invalid input must be rejected.
Functions	:	Identified functions must be exercised.
Output	:	Identified classes of application outputs must be exercised.
Systems/Procedures	:	Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

SYSTEM TESTING:

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

WHITE BOX TESTING:

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

BLACK BOX TESTING:

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

UNIT TESTING:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach
Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link. • The entry screen, messages and responses must not be delayed. **Features to be tested**
- Verify that the entries are of the correct format.
- No duplicate entries should be allowed.
- All links should take the user to the correct page

.

INTEGRATION TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

7.RESULT

Now-a-days to detect cyber-attack we are using static and dynamic analysis of request data. Static analysis is based on signature which we will match existing attack signature with new request packet data to identify packet is normal or contains attack signature. Dynamic analysis will use dynamic execution of program to detect malware/attack but dynamic analysis is time consuming. To overcome from this problem and to increase detection accuracy with old and new malware attacks author is using machine learning algorithms and evaluating prediction performance of various machine learning algorithms such as Support Vector Machine, Random Forest, Decision Tree, Naïve Bayes, Logistic Regression, KNearest Neighbours and Deep Learning Algorithms such as Convolution Neural Networks (CNN) and LSTM (Long Short Term Memory). In all algorithms CNN and LSTM giving better performance.

To implement this paper and to evaluate machine learning algorithms performance author is using binary malware dataset called 'MALIMG'. This dataset contains 25 families of malware and application will convert this binary dataset into gray images to generate train and test models for machine learning algorithms. This algorithms converting binary data to images and then generating model so they are called as MalConv CNN and MalConv LSTM and other algorithm refers as EMBER. Application convert dataset into binary images and then used 80% dataset for training model and 20% dataset for testing. Whenever we upload new test malware binary data then application will apply new test data on train model to predict malware class. In dataset total 25 families of malware we can see and below are their names.'Dialer Adialer.C','Backdoor Agent.FYI','Worm Allaple.A','Worm Allaple.L','Trojan Alueron.gen','Worm:AutoIT Autorun.K', 'Trojan C2Lop.P','Trojan C2Lop.gen','Dialer Dialplatform.B','Trojan Downloader Dontovo.A','Rogue Fakerean','Dialer Instantaccess','PWS Lolyda.AA 1','PWS Lolyda.AA 2','PWS Lolyda.AA 3','PWS Lolyda.AT','Trojan Malex.gen','Trojan Downloader Obfuscator.AD', 'Backdoor Rbot!gen','Trojan Skintrim.N','Trojan Downloader Swizzor.gen!E','Trojan Downloader Swizzor.gen!I','Worm VB.AT','Trojan Downloader Wintrim.BX','Worm Yuner.A'

All above names are the malware families

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

To run this project double click on 'run.bat' file to get below screen

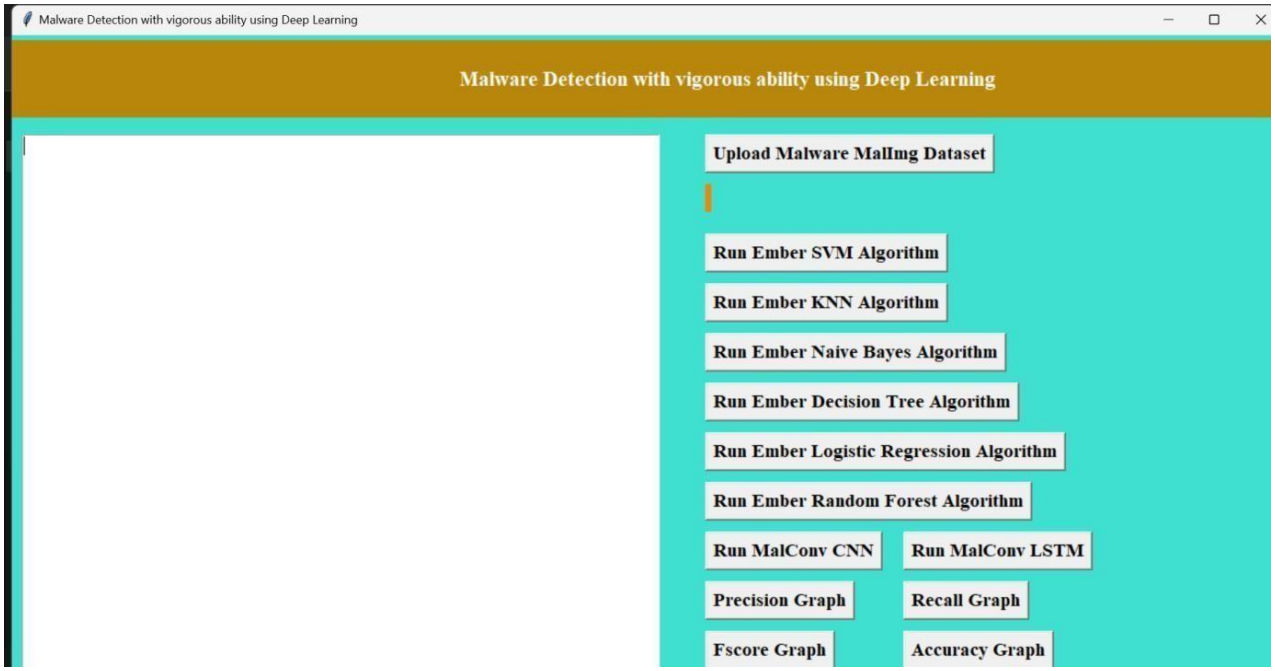


FIG:7.1 : RUN.BAT

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

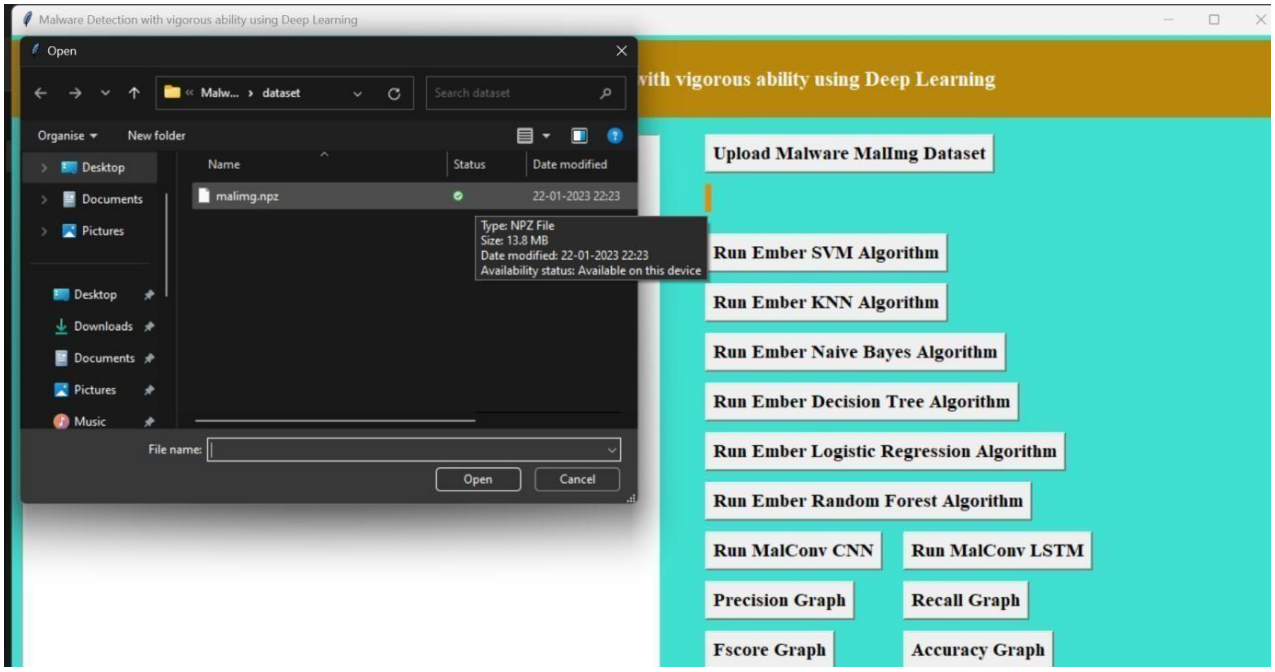


FIG:7.2 : UPLOAD MALIMG

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

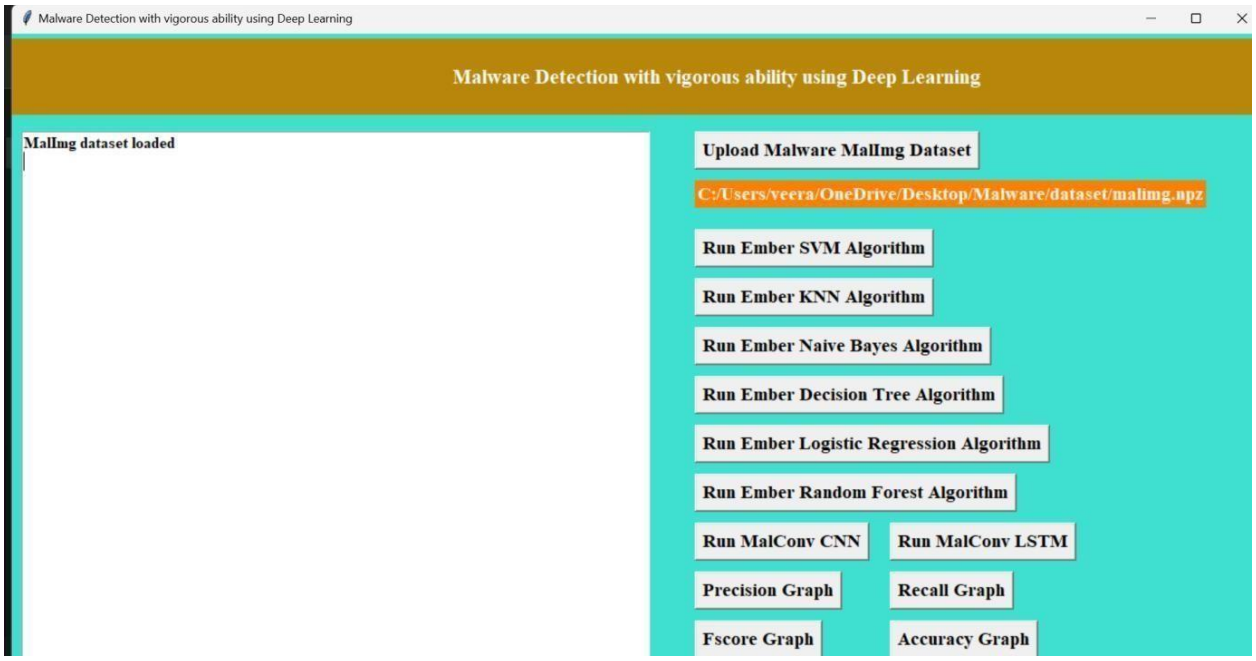


FIG:7.3 : MALIMG.NPZ

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

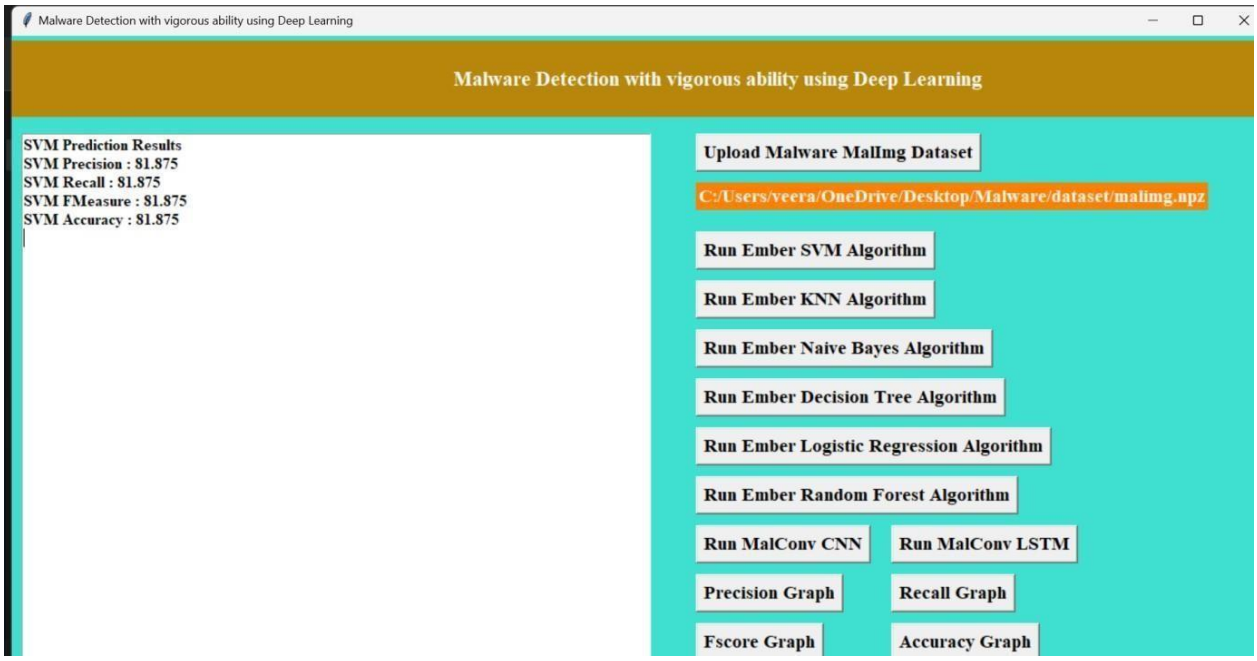


FIG:7.4 : RUN EMBER SVM ALG

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

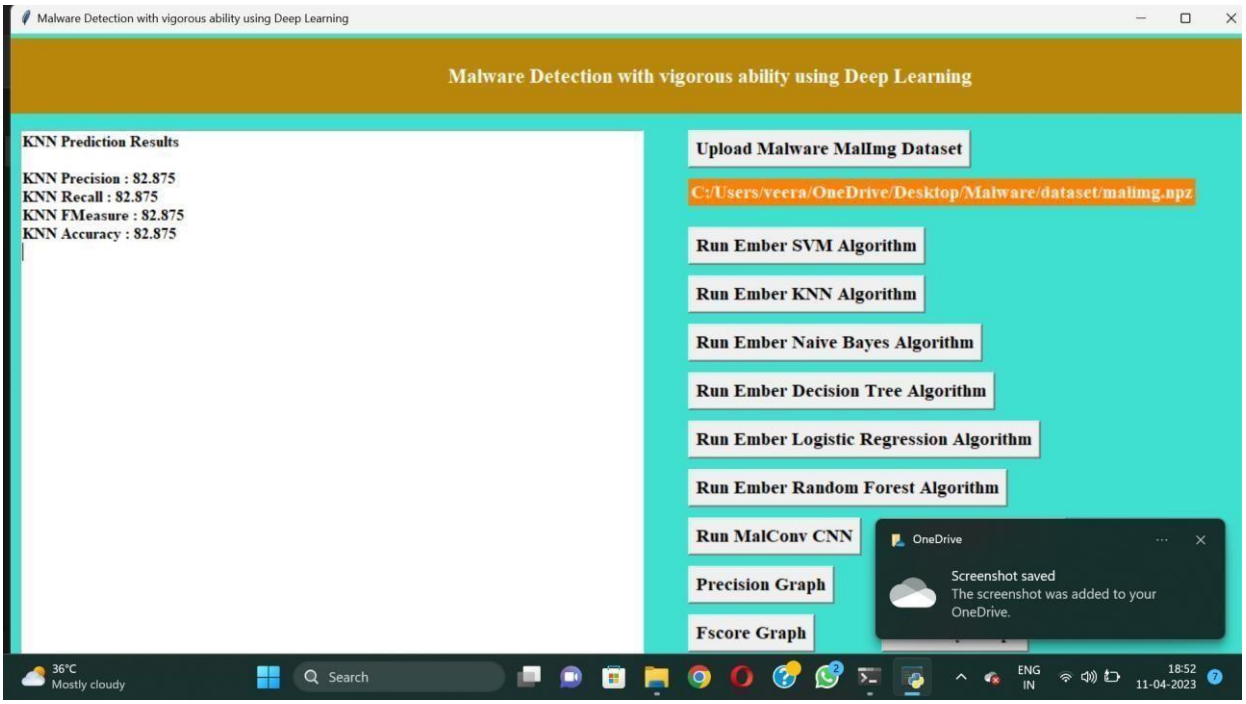


FIG:7.5 : RUN EMBER KNN ALG

In above screen we got KNN details and now click on ‘Naïve Bayes’ button to get its performance details

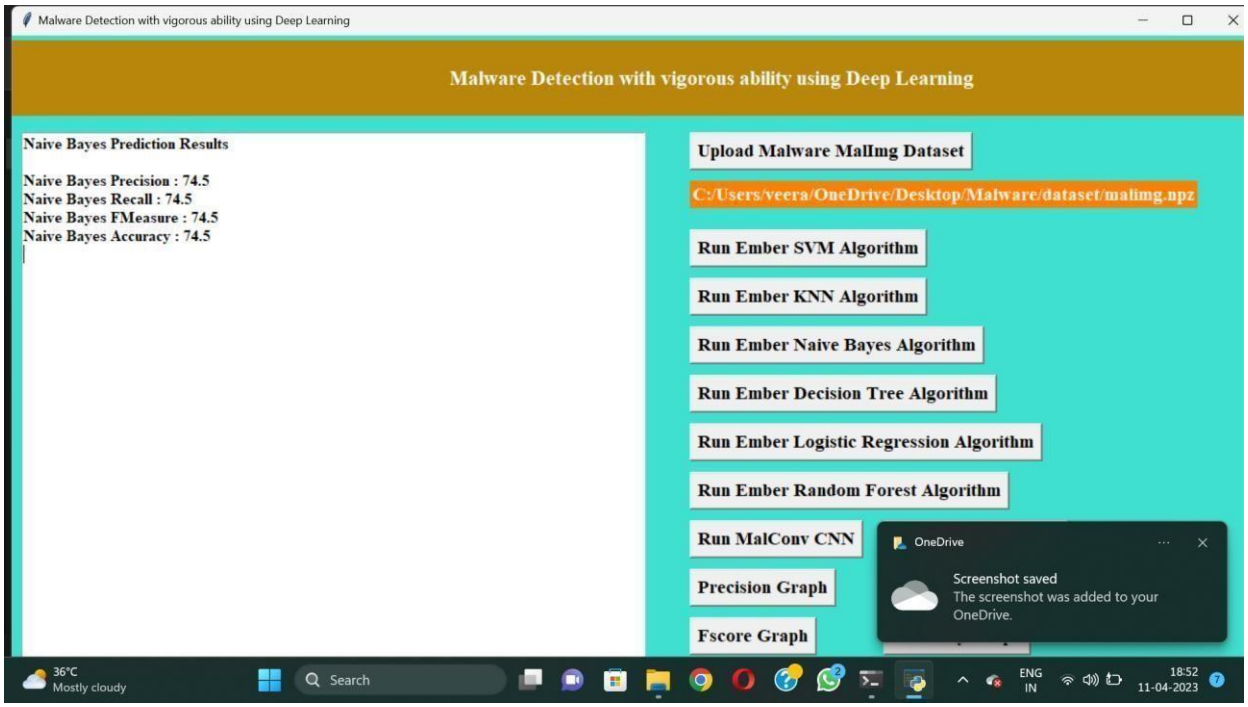


FIG:7.6 :RUN EMBER NB ALG

In above screen we got naïve bayes details and now click on ‘Decision Tree’ button to get its performance details

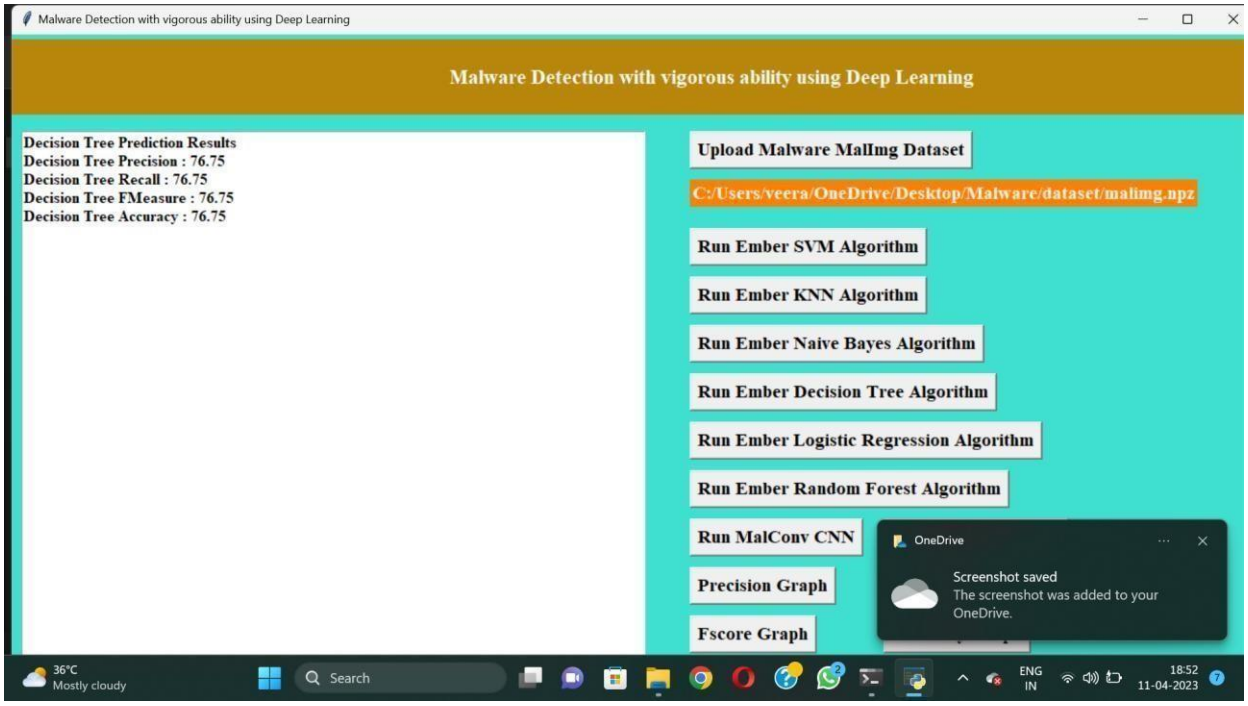


FIG:7.7 :RUN EMBER DT ALG

In above screen we got decision tree details and now click on ‘Logistic Regression’ button to get its details

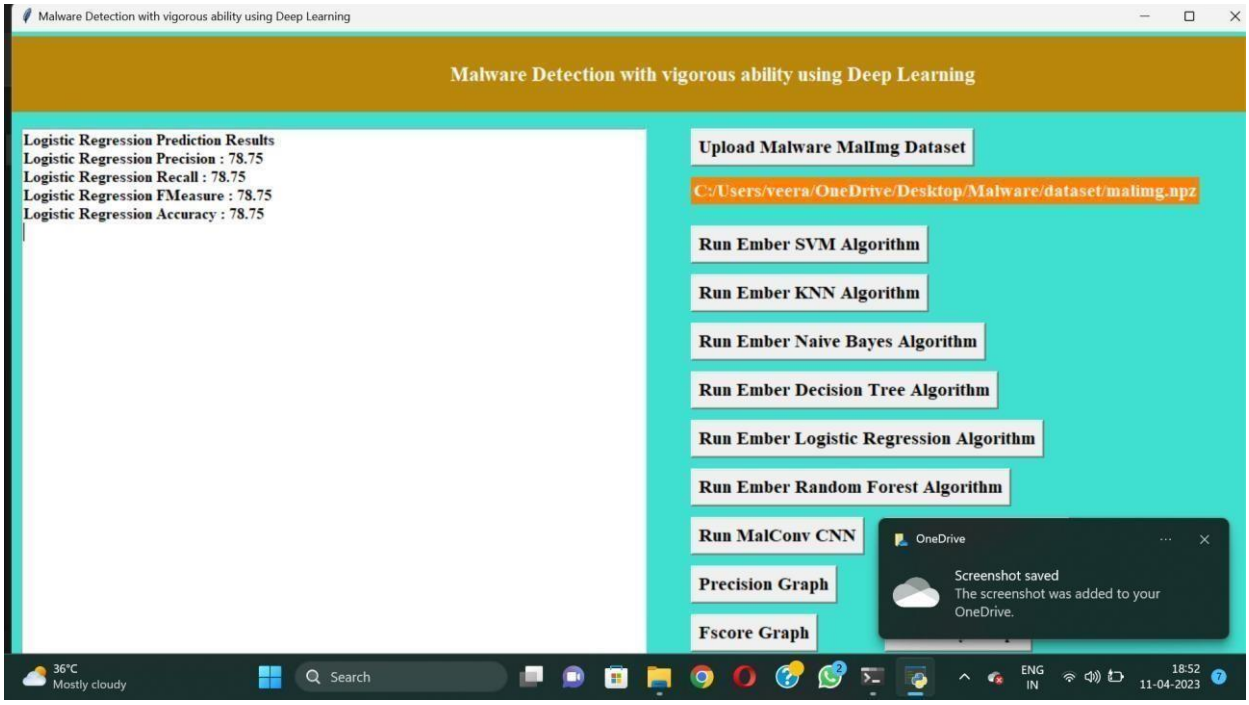


FIG:7.8 :RUN EMBER LR ALG

In above screen we got logistic regression details and now click on ‘Run Random Forest’ button to get its performance

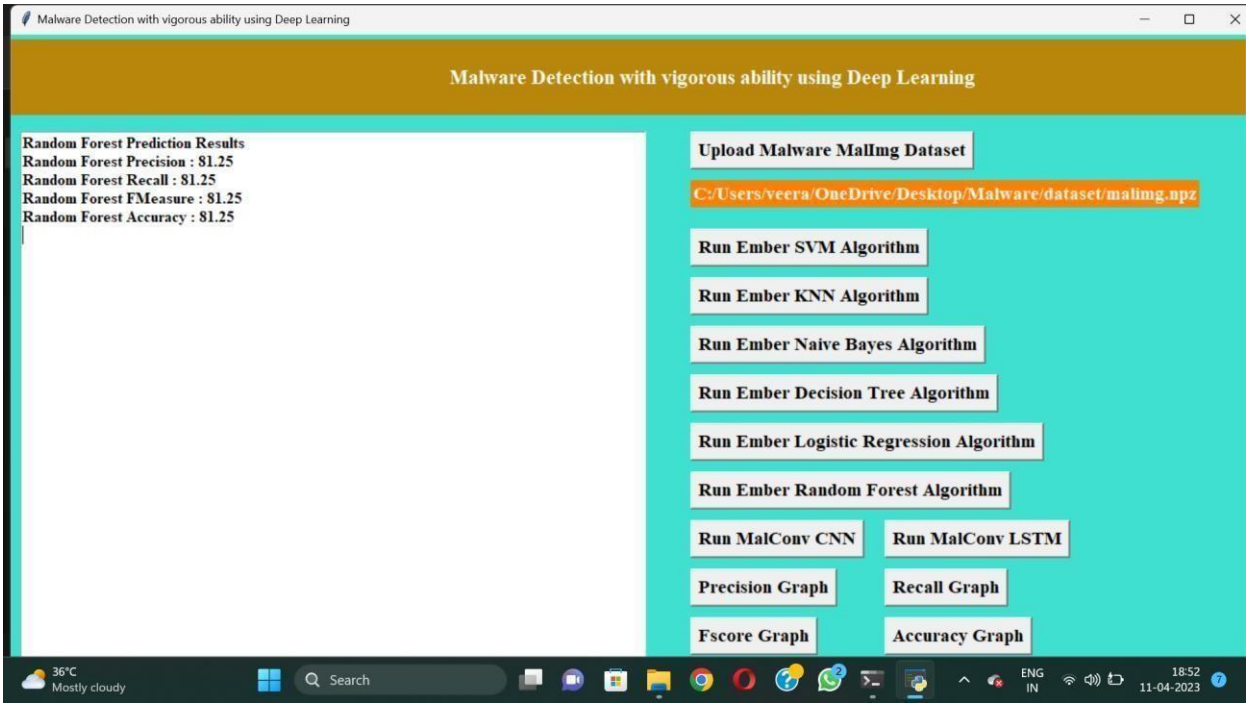


FIG: 7.9 :RUN EMBER RF ALG

In above screen we got random forest details and now click on 'Run MalConv CNN' button to get its performance details. CNN may take 10 minutes to complete execution and u can check its ongoing processing in black console

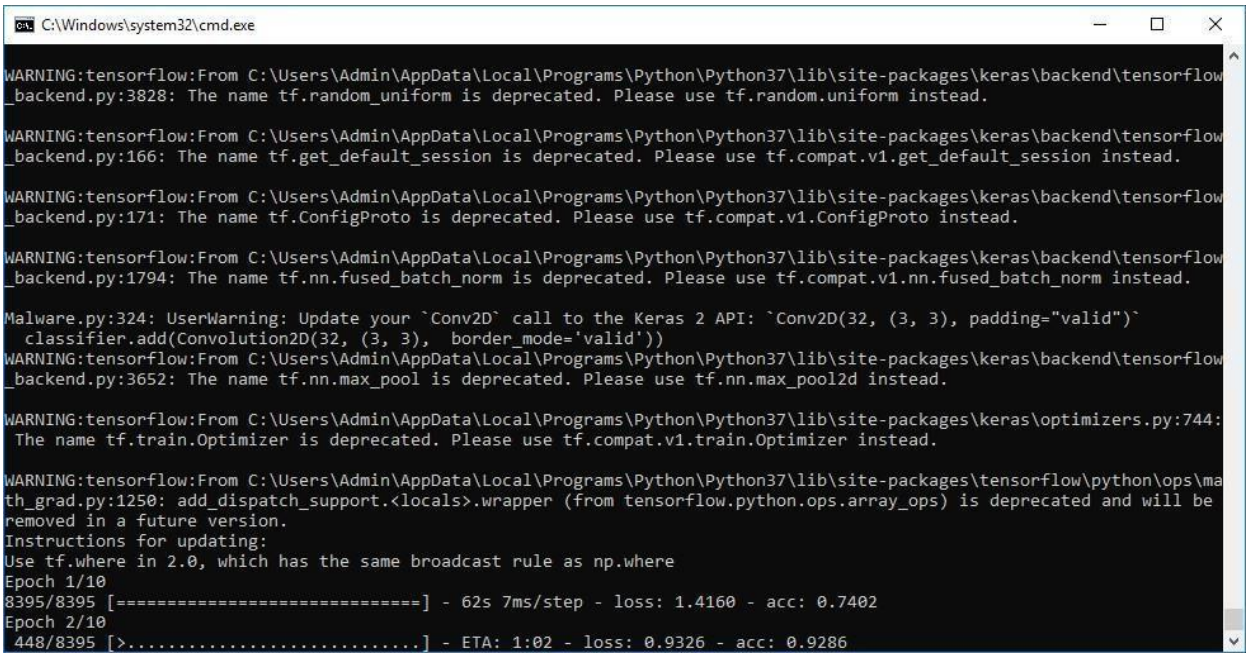


FIG:7.10 : RUN MALCONV CNN

In above console it will take 10 epochs iteration and for each iteration it calculate accuracy for 8395 malware data. So u need to wait till all 10 epochs completed then u will get its performance details

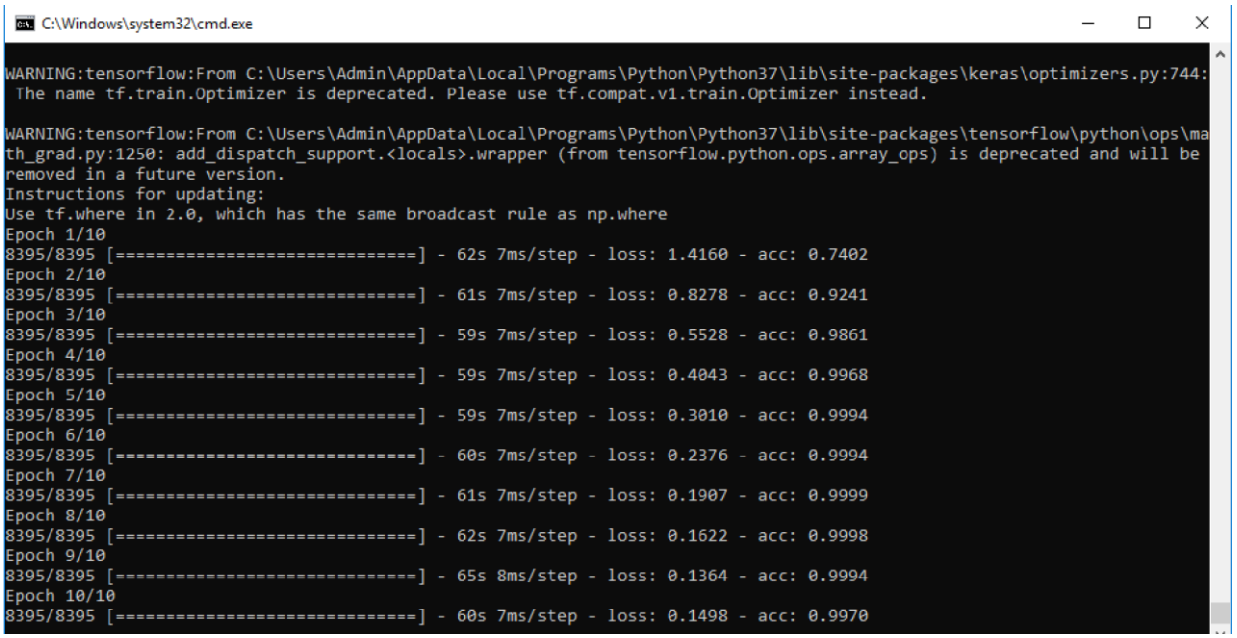


FIG:7.11 : CALCULATE ACCURACY

In above screen we can see CNN complete all 10 epochs and after that we will get accuracy details in main screen

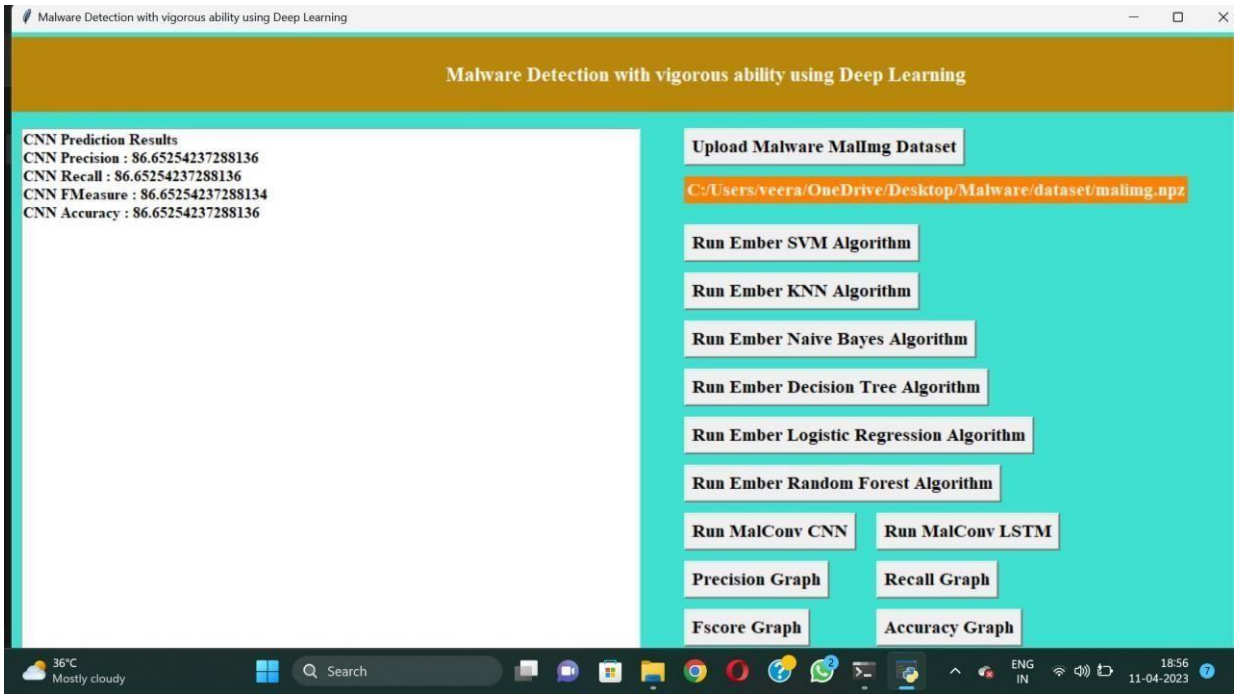


FIG:7.12 :RUN CNN ALG

In above screen we got CNN performance values and now click on ‘Run MalConv LSTM’ button to run LSTM algorithm. Similar to CNN LSTM also take 10 minutes and u can see ongoing process in below screen

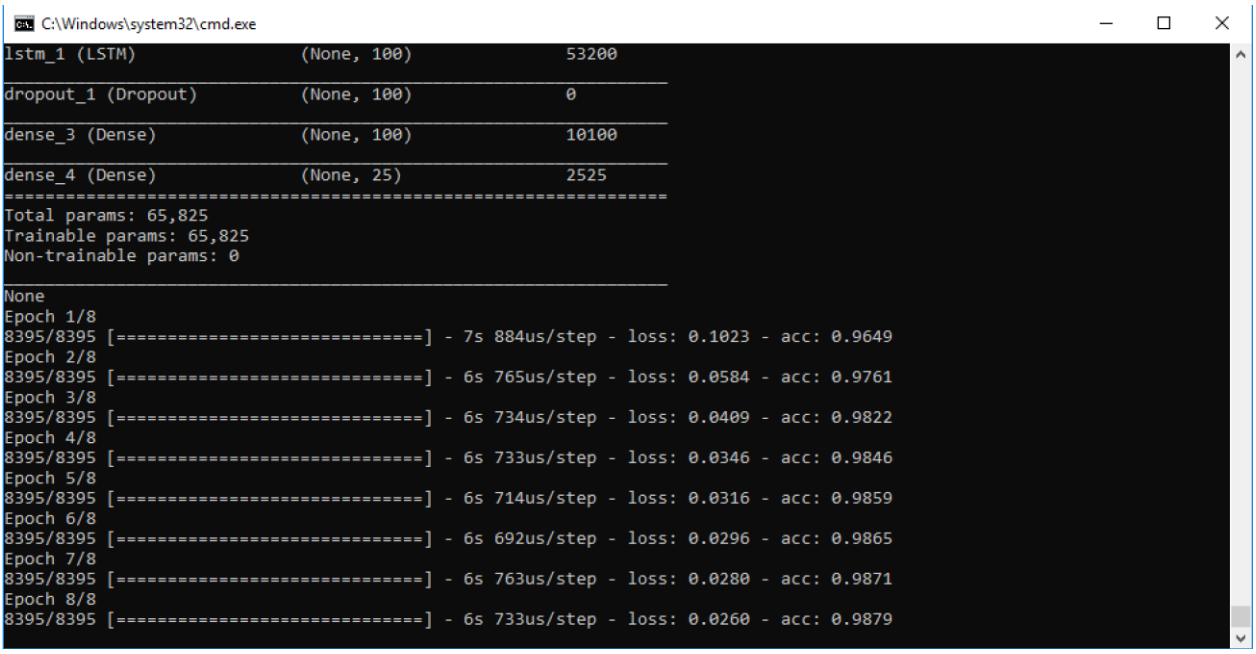


FIG:7.13 : RUN MALCONV LSTM

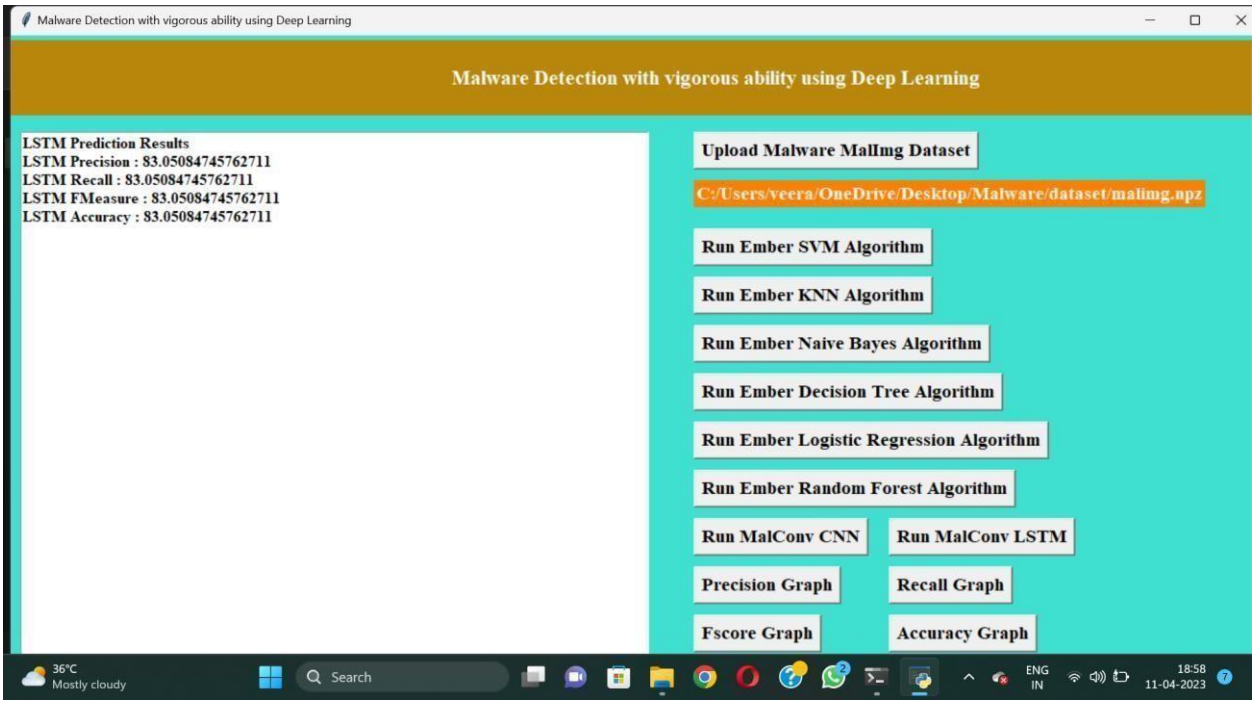


FIG:7.14 : RUN LSTM ALG

In above screen we can see LSTM details now click on ‘Precision, Recall & FScore’ button to get comparson graph for all metrics and all algorithms

FIG:7.15: PRECISION GRAPH

In above screen we can see precision graph for all algorithms and CNN get better performance. In above graph x-axis represents algorithm name and y-axis represents precision value and now close above graph to get recall graph

Now click on accuracy button to get accuracy graph

FIG:7.16 : FSCORE GRAPH

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

8.CONCLUSION

This project 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 Scale Mal Net 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 publicly 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.

9.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. AlNemrat, 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.

10.FUTURE SCOPE

1. **Improving the accuracy of malware detection:** One potential way to improve the accuracy of malware detection is to use a combination of deep learning models with other machine learning techniques, such as ensemble learning or transfer learning. Another approach is to incorporate additional features, such as metadata or contextual information, into the deep

learning models to improve their ability to distinguish between benign and malicious software.

2. **Real-time malware detection:** Real-time malware detection requires algorithms that can quickly analyze large amounts of data and make decisions in near real-time. One potential approach is to use hardware acceleration, such as GPUs or FPGAs, to speed up the deep learning computations. Another approach is to use distributed computing to distribute the workload across multiple machines.
3. **Detection of new and unknown malware:** Deep learning algorithms can be trained on large datasets of known malware to learn patterns that can be used to detect new and unknown malware. Researchers can also use techniques such as transfer learning, where a model trained on one dataset is adapted for use on a different dataset, to improve the model's ability to detect new and unknown malware.
4. **Scalability:** To handle large-scale datasets, deep learning algorithms need to be able to scale both vertically and horizontally. This requires careful optimization of the algorithms and hardware to ensure that they can handle the computational load. One potential approach is to use distributed computing, where the workload is divided across multiple machines, to improve scalability.
5. **Integration with existing security systems:** Deep learning algorithms can be integrated with existing security systems to provide an additional layer of security. This requires careful consideration of how the deep learning model will integrate with existing systems, as well as the potential impact on system performance and resource usage.
6. **Mobile device security:** Mobile devices pose unique challenges for malware detection, including limited resources, diverse operating systems, and a large number of potential attack vectors. Deep learning algorithms can be adapted to work on mobile devices by using lightweight models that consume minimal resources, and by incorporating additional features such as mobile device usage patterns into the models.
7. **Multimodal malware detection:** Multimodal deep learning algorithms can use multiple sources of information, such as file analysis, network traffic analysis, and behavioral analysis, to detect malware.

publication. Citation information: DOI
10.1109/ACCESS.2019.2906934, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.
Digital Object Identifier 10.1109/ACCESS.2017.DOI

Robust Intelligent Malware Detection Using Deep Learning

VINAYAKUMAR R[1], MAMOUN ALAZAB[2], (Senior Member, IEEE), SOMAN KP[1], PRABAHARAN POORNACHANDRAN[3], and SITALAKSHMI VENKATRAMAN[4]

[1]Center for Computational Engineering and Networking (CEN), Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India [2]Charles Darwin University, Australia
[3]Centre for Cyber Security Systems and Networks, Amrita School of Engineering, Amritapuri, Amrita Vishwa Vidyapeetham, India [4]Melbourne Polytechnic, Australia

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

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

ABSTRACT Malicious software or malware continues to pose a major security concern in this digital age as computer users, corporations, and governments witness an exponential growth in malware attacks. Current malware detection solutions adopt Static and Dynamic analysis of malware signatures and behaviour patterns that are time consuming and ineffective in identifying unknown malwares. Recent malwares use polymorphic, metamorphic and other evasive techniques to change the malware behaviours quickly and to generate large number of malwares. Since new malwares are predominantly variants of existing malwares, machine learning algorithms (MLAs) are being employed recently to conduct an effective malware analysis. This requires 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. Though some recent research studies exist in this direction, the performance of the algorithms is biased with the training data. There is a need to mitigate bias and evaluate these methods independently in order to arrive at new enhanced methods for effective zero day malware detection. To fill the gap in literature, this work evaluates classical MLAs and deep learning architectures for malware detection, classification and categorization with both public and private datasets. The train and test splits of public and private datasets used in the experimental analysis are disjoint to each others and collected in different timescales. In addition, we propose a novel image processing technique with optimal parameters for MLAs and deep learning architectures. A comprehensive experimental evaluation of these methods indicate that deep learning architectures outperform classical MLAs. Overall, this work proposes an effective visual detection of malware using a scalable and hybrid deep learning framework for real-time deployments. The visualization and deep learning architectures for static, dynamic and image processing based hybrid approach in a big data environment is a new enhanced method for effective zero-day malware detection.

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

PC's and networks for stealing confidential data for financial

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) applications have led to the development of the modern concept of the information society. However, security concerns pose a major challenge in realising the benefits of this industrial revolution as cyber criminals attack individual gains and causing denial of service to systems. Such attackers make use of malicious software or malware to cause serious threats and vulnerability of systems [1]. A malware is a computer program with the purpose of causing harm to the operating system (OS). A malware gets different names such as adware, spyware, virus, worm, trojan, rootkit, backdoor, ransomware and command and control (C&C) bot, based on its purpose and behaviour. Detection and mitigation of malware is an evolving problem in the cyber security field.

As researchers develop new techniques, malware authors improve their ability to evade detection.

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]. Signaturebased malware detection system requires extensive domain level knowledge to reverse engineer the malware using Static and Dynamic analysis and to assign a signature for that. Moreover, signature-based system requires larger time to reverse engineer the malware and during that time an attacker would encroach into the system. In addition, signature-based system fails to detect new types of malware.

Security researchers have identified that hackers predominantly use polymorphism and metamorphism as obfuscation techniques against signature-based detection. In order to address this problem, software tools are used to manually unpack the codes and analyse the application programming interface (API) calls. Since this process is a resource intensive task, [3] presented an automated system to extract API calls and analyse the malicious characteristics using a fourstep 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 5step 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 MinimumRedundancy 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 behaviour, during which malicious payload can get delivered. Malware detection methods based on Dynamic analysis are more robust to obfuscation methods when compared to statically collected data. Most commonly, the commercial anti-malware solutions use a hybrid of Static and Dynamic analysis approaches.

The major issue with the classical machine learning based malware detection system is that they rely on the feature engineering, feature learning and feature representation techniques that require an extensive domain level knowledge [11], [12], [13]. Moreover, once an attacker comes to know the features, the malware detector can be evaded easily [14].

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

In recent days, deep learning, which is an improved model of neural networks has outperformed the classical MLAs in many of the tasks which exist in the field of natural language processing (NLP), computer vision, speech processing and many others [16]. During the training process, it tries to capture higher level representation of features in deep hidden layers with the ability to learn from mistakes. MLAs experience diminishing outputs as they see more and more data whereas deep learning captures new patterns and establishes associations with the already captured pattern to enhance the performance of tasks. There exists few research studies towards the application of deep learning architectures for malware analysis to improve cyber security [13], [11], [12], [17], [18], [19], [20], [21], [22], [23], [24], [18]. 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 [46]. 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 ngram 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 realised, 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 [26], [11] 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 WindowsDynamic-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], [29], [30], [31], [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 famiy 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 analysed 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. To handle multiclass imbalanced issue, the bat algorithms are used with CNN architecture [47]. Along with the bat algorithm, the data augmentation methodology is followed to increase the number samples. This method performed well compared to the existing methods based on feature engineering. The case studies are discussed mainly to IoT environment of smart cities. However, instead of bat algorithms, cost sensitive approach i.e, cost items can be added to the samples during the backpropogation learning algorithm. In [48] proposed a methodology to represent binaries into image representation. This can preserve the sequential information of bytecodes and it is similar to [28]. The proposed method converts the byte code into byte streams and thereby this method is able to preserve the sequential order of binary code. Various deep learning architectures such as CNN and bidirectional LSTM and combination of CNN and bidirectional LSTM architectures are evaluated with sampling and as well as without sampling techniques to handle the samples equally across all the classes. All the architectures obtained better performance, particularly combination of CNN and bidirectional LSTM performed well. The proposed method is computationally expensive for larger binaries in preprocessing stage. A proper method is required to convert the large binaries into smaller without losing the semantic meaning. The memory dump for a malware binary is collected in a run-time environment [49]. Then these memory dumps are transformed into images and various feature engineering methods with classical machine learning algorithms are used for classification. In this work, the application of deep learning can be used that avoids the complete feature engineering and it can perform better than the proposed method. In [43] proposed a method which uses simhash algorithm for feature extraction and features are converted into image representation. These images are classified using CNN. The

phase of feature engineering can be avoided using deep learning. With the aim to enhance the interpretability while using RNN, along with the original data the opcode prediction was done using RNN [44]. The original code and the predicted codes are fused together and malwares are represented in the form of images using minhash. Then the classification is done using CNN and RNN architectures. This method performed well even in smaller datasets. The proposed method is computationally expensive in preprocessing stage particularly in opcode generation, fusion and then image representation using minhash method. In [17] proposed a transfer learning approach for malware classification. In preprocessing stage the malware images are resized into 1D representation by following flattening operation. Then various deep learning architectures are used for feature extractions and SVM for classification. However, the performances of the proposed method is considerably less. In this we work, we apply the deep learning architectures on the same dataset with the aim to enhance the malware identification performance. The detailed experimental analysis is done on the Malimg dataset to understand the characteristics of each malware families. Finally, 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.

To accurately detect malware in real-time environment, a multi-stage malware detection system is proposed [51]. The first stage is based on machine learning based behavioral malware identification system which classifies the malware based on the threshold to the uncertain stage. Later, using deep learning architectures the malwares are identified into intrusive or non-intrusive perturbations. The non-intrusive are considered as benign and the system continuously monitors this application. The intrusive application is passed into next stage where the system kills the application and removes completely from the system. With the aim to develop dynamic intrusion detection system, various deep earning architectures are evaluated for anomaly based intrusion detection system [52]. The performances of the deep learning architectures are evaluated using benchmark dataset and compared with the classical MLAs. The deep learning architecture showed better results in compared to classical MLAs in realworld application in anomaly intrusion detection system. A self-adaptive deep learning based anomaly detection system is proposed for identifying the unpredictable traffic fluctuation in the context of 5G mobile network architecture [53]. The deep learning model composed of two stages, in first stage the deep belief network is employed to identify the anomalous traffic conditions happening during a configurable short time period. The anomalous traffic are redirected to a central system and passed as input to LSTM with the aim to identify temporal patterns of cyber attacks. Two different streams of CNN network is employed to extract different scaled feature information from single lead short electrocardiogram (ECG) recordings [54]. Various experiments were run and different visualizations methods were adopted to show how the proposed method achieved the best results. Various feature sets are extracted and separately deep learning architecture is trained for each feature set and finally these models are merged [55]. The detailed case studies on how multimodal method implements online update method is discussed in detail. In [56] applied CNN for finger-vein based biometric identification. Various CNN architecture are evaluated to identify the optimal architecture. In [57] propose CNN based recolored image detection. This contains three different feature extraction phases and finally all these features are combined by following fusion approach. Following at last, we propose hybrid malware analysis system named as ScaleMalNet which composed of Windows-StaticBrainDroid (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 reallife 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:

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

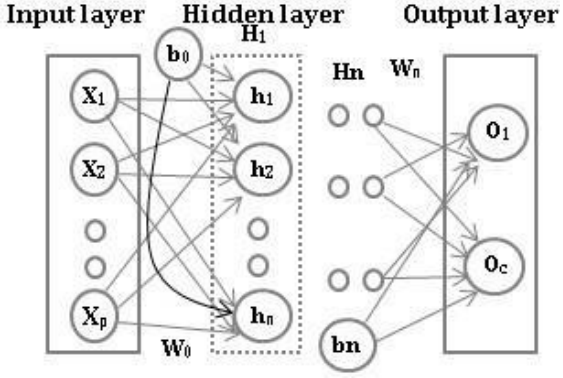


FIGURE 1: Architecture of DNN with n hidden layers.

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 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:

$$f(x) = \max(0, x) \quad (2)$$

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 [61]. 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) [58] and long short term memory (LSTM) [59]. 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:

$$h_t = \left\{ \begin{array}{ll} 0 & t = 0 \\ tf(h_{t-1}, x_t) & otherwise \end{array} \right\} \quad (3)$$

During backpropagation, the transition function tf goes computation of LSTM unit at time step t is mathematically

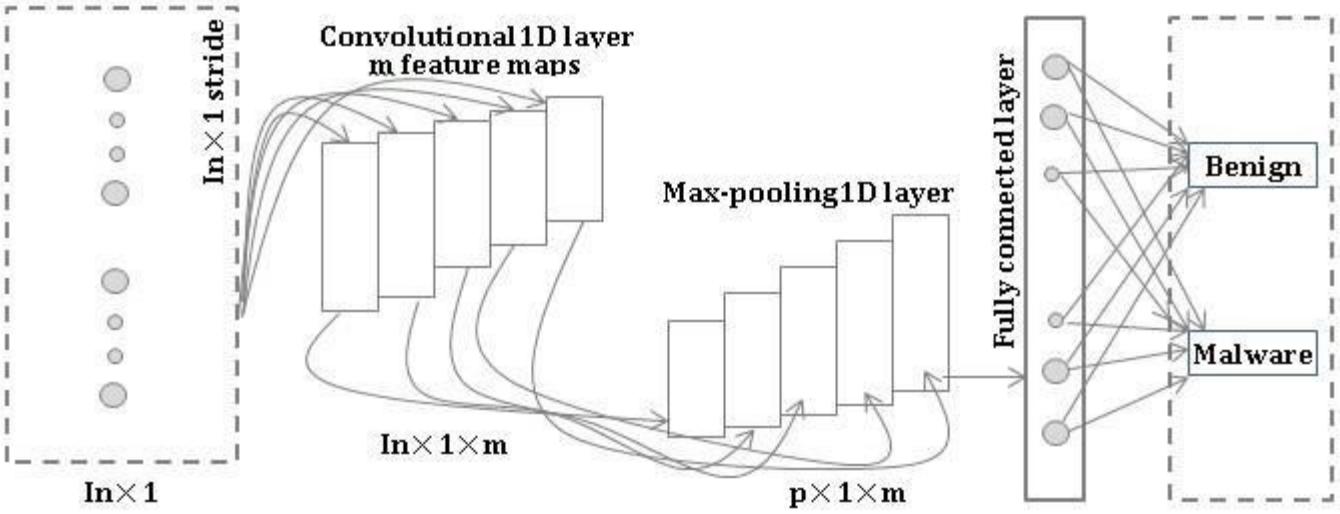


FIGURE 2: Architecture of CNN for malware detection.

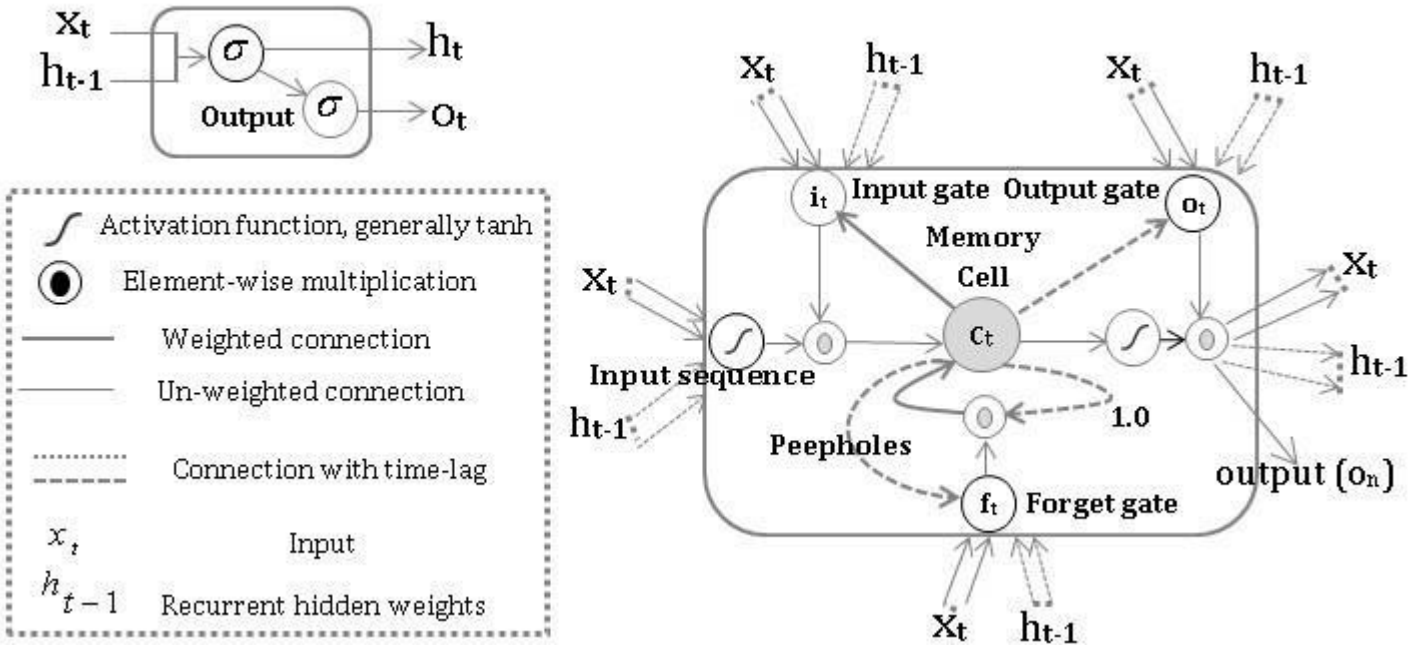


FIGURE 3: Architecture of RNN unit (left) and LSTM memory block (right).

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 [60]. 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 defined as follows:

$$f_t = \sigma(W_f.[h_{t-1},x_t] + b_f) \quad (4)$$

$$i_t = \sigma(W_i.[h_{t-1},x_t] + b_i)\ c_t \quad (5)$$

$$= \tanh(W_c.[h_{t-1},x_t] + b_c)$$

(6)

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (7)$$

$$h_t = o_t * \tanh(c_t) \quad (8)$$

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) nodes 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 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 F1score ($F1-score \in [0,1]$) 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

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN}, \quad (9)$$

$$Precision = \frac{\#TP}{\#TP + \#FP}, \quad (10)$$

$$Recall = \frac{\#TP}{\#TP + \#FN}, \quad (11)$$

and

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

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;

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

[1] https://spark.apache.org/

[2] http://hadoop.apache.org/

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

$$TPR = \frac{\#TP}{\#TP + \#FN}, \quad (14)$$

and

$$FPR = \frac{\#FP}{\#FP + \#TN}. \quad (15)$$

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-BrainDroid (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 realtime.

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 preprocessing 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

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

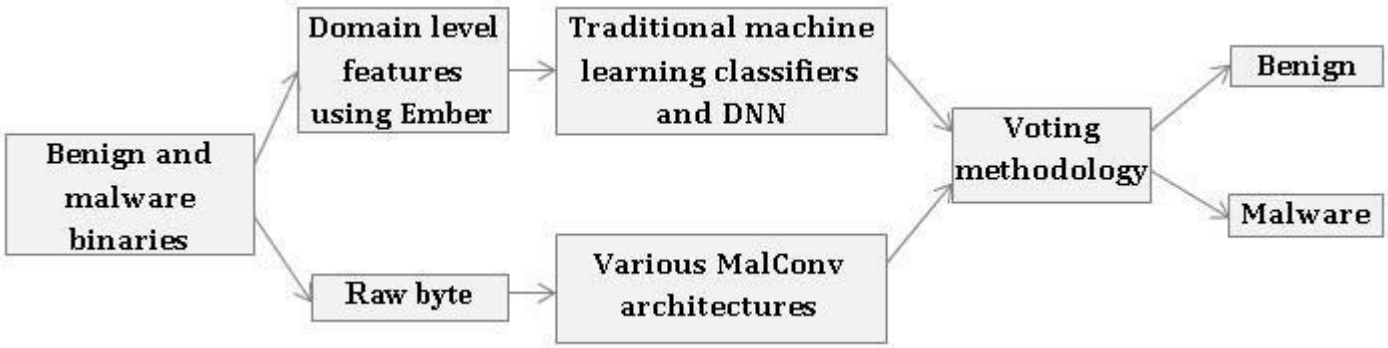
[4] https://virusshare.com/

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], [26], [12]. 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 corss 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]. 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 5120 and 5,632. Thus the number



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

FIGURE 4: Proposed
Deep learning

architecture based on Static Analysis - Windows-Static-Brain-Droid (WSBD)

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

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

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 hte 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 [0.010.5]. 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.

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 [464]. 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 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

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

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 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.

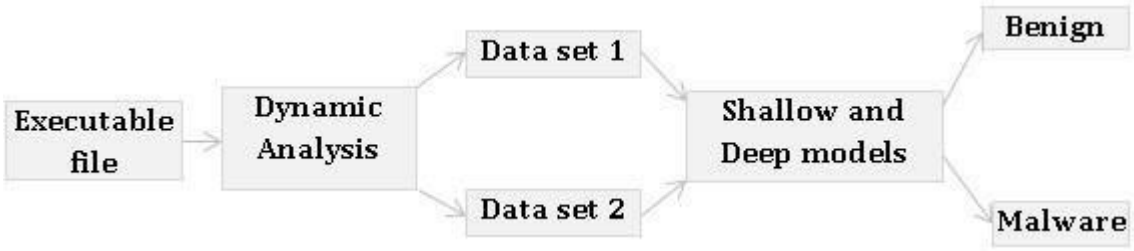


FIGURE 5: Proposed Deep learning architecture based on Dynamic Analysis - Windows-Dynamic-Brain-Droid (WDBD)

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

[6] https://cuckoosandbox.org/

[7] https://github.com/hyperic/sigar

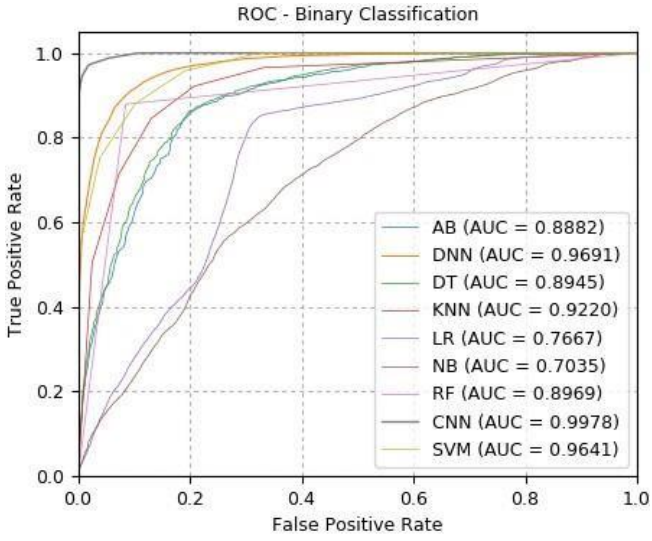
[8] https://pypi.org/project/psutil/

training time and the malware detection rate, the learning rate 0.01 is used for the rest of the experiments.

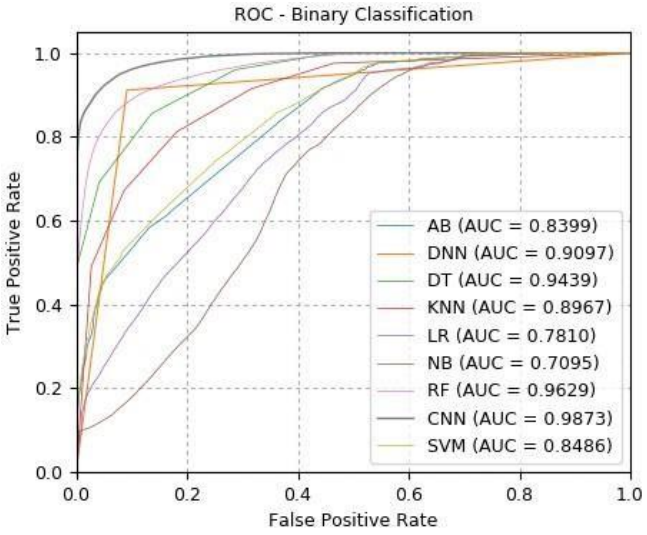
To find out the optimal network structure for DNN and CNN, DNN/CNN 1, 2, 3, 4, and 5 layers were used, and 3 trials of experiments were run for various network topologies for 100 epochs. DNN model with 4 layers and CNN with 1 layer performed well in comparison to the other network topologies. In DNN, 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:
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



(a)



(b)

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

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

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

where x defines input, pd is predicted probability, ed is the

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

to the existing data, and these explorations are devoted for future research.

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

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 x 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 [33], [17] and privately collected malware samples.

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

[9] http://www.virussign.com/

[10] https://virusshare.com/

[11] https://www.virustotal.com/

malware samples. These samples are grouped into 10 malware families. The detailed statistics of the dataset is given in Table 6.

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

B. DATA ANALYSIS AND RESULTS

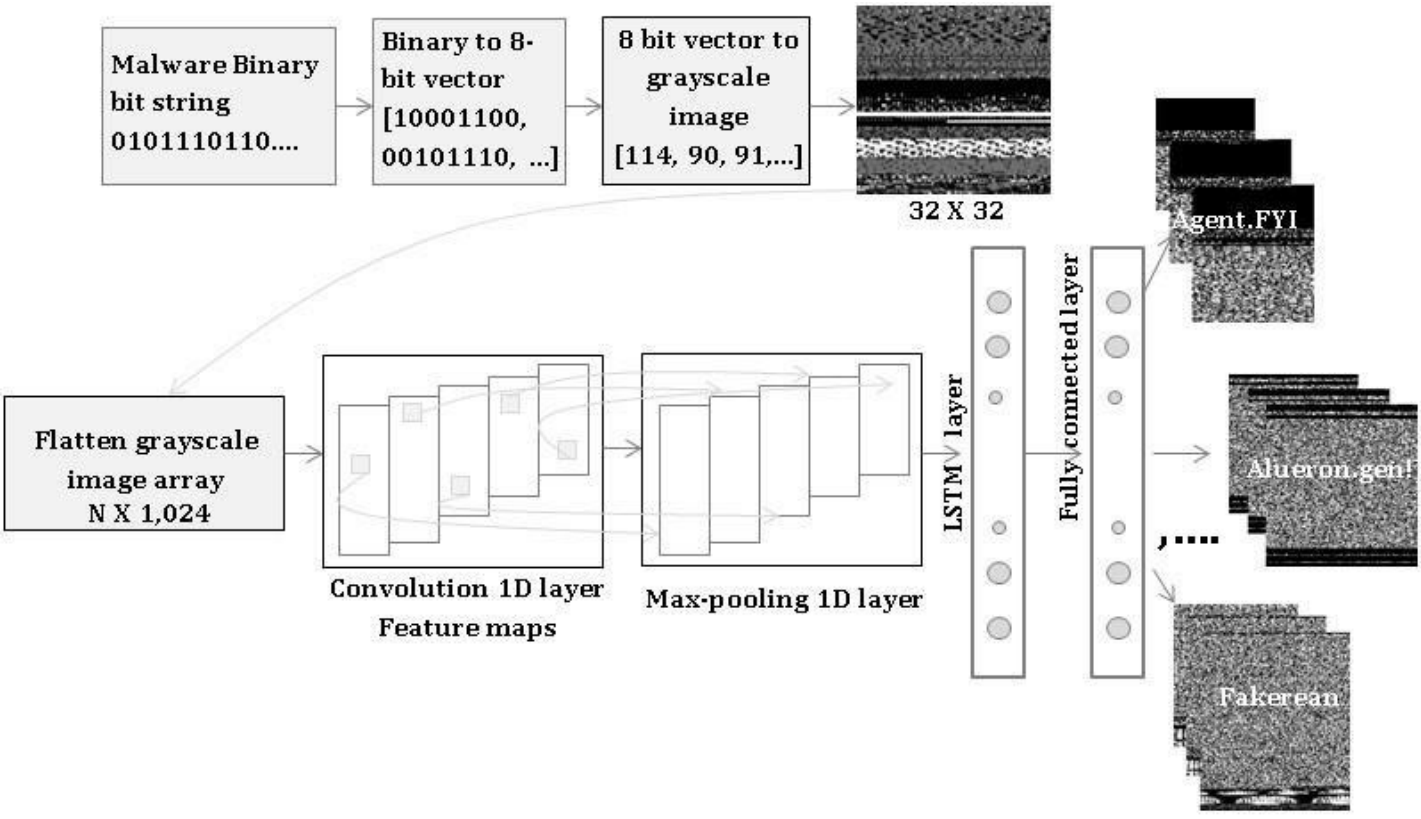


FIGURE 7: Proposed Deep learning architecture based on Image processing - DeepImageMalDetect (DIMD)

The proposed architecture, DeepImageMalDetect (DIMD) is shown in Figure 7. For both the datasets, the performance of various classical machine learning algorithms (MLAs) and 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

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,

   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.

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 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 using

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

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

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 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

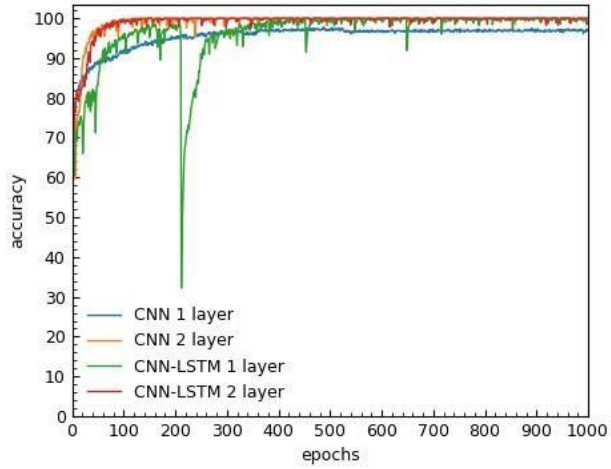


FIGURE 8: Training accuracy of various Deep learning architectures on malware catgorization

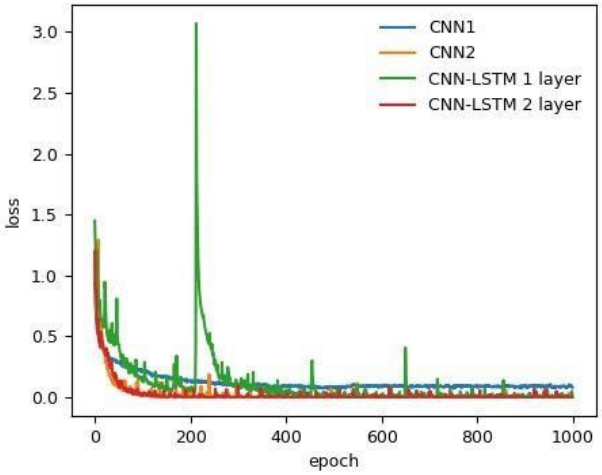


FIGURE 9: Training loss of various Deep learning architectures on malware catgorization

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 familes 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 costsensitive 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 undesrsatnd the performance, shown in Table 10 and computed the Error rate as

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

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

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

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

Malware Family	[redacted]	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 0	0
Agent.FYI	0 3[illegible] 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	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 4[illegible] 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 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 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	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 4[illegible] 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 1[illegible] 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 1[illegible] 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 0 118 0 0	3.279

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI

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

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

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	

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

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

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

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:

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

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. The efficacy of classical MLAs and deep learning architectures are evaluated on benchmark dataset and privately collected malware samples. Deep learning architectures 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

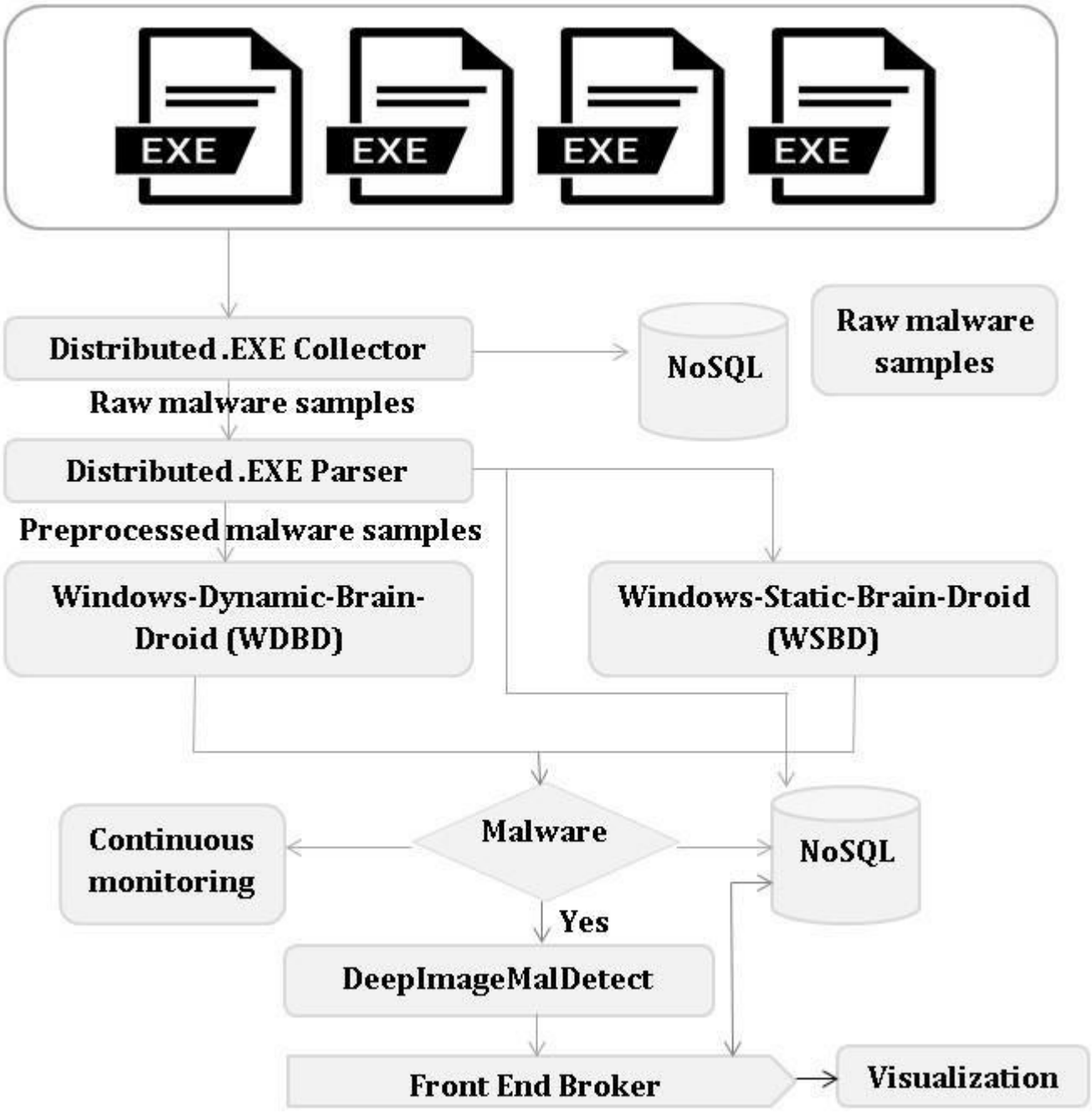


FIGURE 10: Proposed Deep learning architecture for real-time Malware Analysis

outperformed the classical MLAs. This method is agnostic to validation. The reported results can be further enhanced by packing and operating system. Moreover, it takes less time using highly complex deep learning architecture and as compared to Static and Dynamic analysis due to the fact 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 is explained in detail by [62]. Autoencoder is a generative model which learns the latent representation of different feature sets [63]. 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 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].

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 preprocessing in a distributed way. The framework incorporates self-learning techniques 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 zeroday malwares. This framework applies deep learning on the collected malwares from end user hosts and follows a twostage 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 stateof-the-art method, deep learning which detects the malware in first level and in second level the malware is categorized into their corresponding categories.

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

- 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 [50]. 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] Anderson, R., Barton, C., Böhme, R., Clayton, R., Van Eeten, M. J., Levi, M., ... & Savage, S. (2013). Measuring the cost of cybercrime. In The economics of information security and privacy (pp. 265-300). Springer, Berlin, Heidelberg.

[2] Li, B., Roundy, K., Gates, C., & Vorobeychik, Y. (2017, March). LargeScale Identification of Malicious Singleton Files. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (pp. 227238). ACM.

[3] Alazab, M., Venkataraman, S., & Watters, P. (2010, July). Towards understanding malware behaviour by the extraction of API calls. In 2010 Second Cybercrime and Trustworthy Computing Workshop (pp. 52-59). IEEE.

[4] Tang, M., Alazab, M., & Luo, Y. (2017). Big data for cybersecurity: vulnerability disclosure trends and dependencies. IEEE Transactions on Big Data.

[5] Alazab, M., Venkatraman, S., Watters, P., & Alazab, M. (2011, December). Zero-day malware detection based on supervised learning algorithms of API call signatures. In Proceedings of the Ninth Australasian Data Mining Conference-Volume 121 (pp. 171-182). Australian Computer Society, Inc..

[6] Alazab, M., Venkatraman, S., Watters, P., Alazab, M., & Alazab, A. (2011, January). Cybercrime: the case of obfuscated malware. In 7th ICGS3/4th e-Democracy Joint Conferences 2011: Proceedings of the International Conference in Global Security, Safety and Sustainability/International Conference on e-Democracy (pp. 1-8). [Springer].

[7] Alazab, M. (2015). Profiling and classifying the behavior of malicious codes. Journal of Systems and Software, 100, 91-102.

[8] Huda, S., Abawajy, J., Alazab, M., Abdollalihian, M., Islam, R., & Yearwood, J. (2016). Hybrids of support vector machine wrapper and filter based framework for malware detection. Future Generation Computer Systems, 55, 376-390.

[9] Raff, E., Sylvester, J., & Nicholas, C. (2017, November). Learning the PE Header, Malware Detection with Minimal Domain Knowledge. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (pp. 121-132). ACM.

[10] Rossow, C., Dietrich, C. J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., ... & Van Steen, M. (2012, May). Prudent practices for designing malware experiments: Status quo and outlook. In Security and Privacy (SP), 2012 IEEE Symposium on (pp. 65-79). IEEE.

[11] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., & Nicholas, C. (2017). Malware detection by eating a whole exe. arXiv preprint arXiv:1710.09435.

[12] Krcál, M.,ˇ Svec, O., Bálek, M., & Jaˇ sek, O. (2018). Deep Convolutionalˇ Malware Classifiers Can Learn from Raw Executables and Labels Only.

[13] Rhode, M., Burnap, P., & Jones, K. (2018). Early-stage malware prediction using recurrent neural networks. Computers & Security, 77, 578-594. [14] Anderson, H. S., Kharkar, A., Filar, B., & Roth, P. (2017). Evading machine learning malware detection. Black Hat.

[15] Verma, R. (2018, March). Security Analytics: Adapting Data Science for Security Challenges. In Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics (pp. 40-41). ACM. [16] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436.

[17] Agarap, A. F., & Pepito, F. J. H. (2017). Towards Building an Intelligent AntiMalware System: A Deep Learning Approach using Support Vector Machine (SVM) for Malware Classification. arXiv preprint arXiv:1801.00318.

[18] Rezende, E., Ruppert, G., Carvalho, T., Theophilo, A., Ramos, F., & de Geus, P. (2018). Malicious Software Classification Using VGG16 Deep Neural Network's Bottleneck Features. In Information Technology-New Generations (pp. 51-59). Springer, Cham.

[19] Saxe, J., & Berlin, K. (2015, October). Deep neural network based malware detection using two dimensional binary program features. In Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on (pp. 11-20). IEEE.

[20] Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., & Yagi, T. (2016, June). Malware detection with deep neural network using process behavior. In Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual (Vol. 2, pp. 577-582). IEEE.

[21] Huang, W., & Stokes, J. W. (2016, July). MtNet: a multi-task neural network for dynamic malware classification. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 399-418). Springer, Cham.

[22] Pascanu, R., Stokes, J. W., Sanossian, H., Marinescu, M., & Thomas, A. (2015, April). Malware classification with recurrent networks. In Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on (pp. 1916-1920). IEEE.

[23] Shibahara, T., Yagi, T., Akiyama, M., Chiba, D., & Yada, T. (2016, December). Efficient dynamic malware analysis based on network behavior using deep learning. In Global Communications Conference (GLOBECOM), 2016 IEEE (pp. 1-7). IEEE.

[24] Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016, December). Deep learning for classification of malware system call sequences. In

Australasian Joint Conference on Artificial Intelligence (pp. 137-149). Springer, Cham.

[25] Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., ... & Nicholas, C. (2018). An investigation of byte n-gram features for malware classification. Journal of Computer Virology and Hacking Techniques, 14(1), 1-20.

[26] Anderson, H. S., & Roth, P. (2018). EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv preprint arXiv:1804.04637.

[27] Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques, 13(1), 112. [28] Nataraj, L. (2015). A signal processing approach to malware analysis. University of California, Santa Barbara.

[29] Nataraj, L., Kirat, D., Manjunath, B. S., & Vigna, G. (2013, December). Sarvam: Search and retrieval of malware. In Proceedings of the Annual Computer Security Conference (ACSAC) Worshop on Next Generation Malware Attacks and Defense (NGMAD).

[30] Nataraj, L., Yegneswaran, V., Porras, P., & Zhang, J. (2011, October). A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (pp. 21-30). ACM.

[31] Nataraj, L., Jacob, G., & Manjunath, B. S. (2010). Detecting packed executables based on raw binary data. Technical report.

[32] Farrokhmanesh, M., & Hamzeh, A. (2016, April). A novel method for malware detection using audio signal processing techniques. In Artificial Intelligence and Robotics (IRANOPEN), 2016 (pp. 85-91). IEEE.

[33] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011, July). Malware images: visualization and automatic classification. In Proceedings of the 8th international symposium on visualization for cyber security (p. 4). ACM.

[34] Nataraj, L., & Manjunath, B. S. (2016). SPAM: signal processing to analyze malware. arXiv preprint arXiv:1605.05280.

[35] Kirat, D., Nataraj, L., Vigna, G., & Manjunath, B. S. (2013, December). Sigmal: A static signal processing based malware triage. In Proceedings of the 29th Annual Computer Security Applications Conference (pp. 89-98). ACM.

[36] Glorot, X., Bordes, A., & Bengio, Y. (2011, June). Deep sparse rectifier neural networks. In Proceedings of the fourteenth international conference on artificial intelligence and statistics (pp. 315-323).

[37] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). Tensorflow: a system for large-scale machine learning. In OSDI (Vol. 16, pp. 265-283).

[38] Chollet, F. (2015). Keras: Deep learning library for theano and tensorflow. URL: https://keras. io/k, 7(8).

[39] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[40] Conti, G., Dean, E., Sinda, M., & Sangster, B. (2008). Visual reverse engineering of binary and data files. In Visualization for Computer Security (pp. 1-17). Springer, Berlin, Heidelberg.

[41] Garcia, F. C. C., Muga, I. I., & Felix, P. (2016). Random forest for malware classification. arXiv preprint arXiv:1609.07770.

[42] Sebastián, M., Rivera, R., Kotzias, P., & Caballero, J. (2016, September). Avclass: A tool for massive malware labeling. In International Symposium on Research in Attacks, Intrusions, and Defenses (pp. 230-253). Springer, Cham.

[43] Ni, S., Qian, Q., & Zhang, R. (2018). Malware identification using visualization images and deep learning. Computers & Security.

[44] Sun, G., & Qian, Q. (2018). Deep Learning and Visualization for Identifying Malware Families. IEEE Transactions on Dependable and Secure Computing.

[45] Ebenuwa, S., Sharif, M. S., Alazab, M., & Al-Nemrat, A. (2019). Variance Ranking Attributes Selection Techniques for Binary Classification Problem in Imbalance Data. IEEE Access.

[46] Venkatraman, S., & Alazab, M. (2018). Use of Data Visualisation for ZeroDay Malware Detection. Security and Communication Networks, 2018.

[47] Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G. G., & Chen, J. (2018). Detection of malicious code variants based on deep learning. IEEE Transactions on Industrial Informatics, 14(7), 3187-3196.

[48] Le, Q., Boydell, O., Mac Namee, B., & Scanlon, M. (2018). Deep learning at the shallow end: Malware classification for non-domain experts. Digital Investigation, 26, S118-S126.

[49] Dai, Y., Li, H., Qian, Y., & Lu, X. (2018). A malware classification method based on memory dump grayscale image. Digital Investigation, 27, 30-37.

[50] Yuan, X., He, P., Zhu, Q., & Li, X. (2019). Adversarial examples: Attacks and defenses for deep learning. IEEE transactions on neural networks and learning systems.

[51] Yuan, X. (2017, May). PhD forum: deep learning-based real-time malware detection with multi-stage analysis. In 2017 IEEE International Conference on Smart Computing (SMARTCOMP) (pp. 1-2). IEEE.

[52] Naseer, S., Saleem, Y., Khalid, S., Bashir, M. K., Han, J., Iqbal, M. M., & Han, K. (2018). Enhanced network anomaly detection based on deep neural networks. IEEE Access, 6, 48231-48246.

[53] MaimÃs¸, L. F., GÃs¸mez, ÃA¸. L. P., Clemente, F. J. G., PÃl'rez, M. G., & PÃl'rez, G. M. (2018). A self-adaptive deep learning-based system for anomaly detection in 5G networks. IEEE Access, 6, 7700-7712.

[54] Fan, X., Yao, Q., Cai, Y., Miao, F., Sun, F., & Li, Y. (2018). Multiscaled Fusion of Deep Convolutional Neural Networks for Screening Atrial Fibrillation From Single Lead Short ECG Recordings. IEEE journal of biomedical and health informatics, 22(6), 1744-1753.

[55] Kim, T., Kang, B., Rho, M., Sezer, S., & Im, E. G. (2019). A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. IEEE Transactions on Information Forensics and Security, 14(3), 773-788.

[56] Das, R., Piciucco, E., Maiorana, E., & Campisi, P. (2019). Convolutional neural network for finger-vein-based biometric identification. IEEE Transactions on Information Forensics and Security, 14(2), 360-373.

[57] Yan, Y., Ren, W., & Cao, X. (2019). Recolored image detection via a deep discriminative model. IEEE Transactions on Information Forensics and Security, 14(1), 5-17.

[58] Elman, J. L. (1990). Finding structure in time. Cognitive science, 14(2), 179211.

[59] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[60] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

[61] Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

[62] Yousefi-Azar, M., Varadharajan, V., Hamey, L., & Tupakula, U. (2017, May). Autoencoder-based feature learning for cyber security applications. In 2017 International Joint Conference on Neural Networks (IJCNN) (pp. 3854-3861). IEEE.

[63] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. science, 313(5786), 504-507.

R. VINAYAKUMAR is a Ph.D. student in the Computational Engineering & Networking, Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India since July 2015. He has received BCA from JSS college of Arts, Commerce and Sciences, Ooty road, Mysore in 2011 and MCA from Amrita Vishwa Vidyapeetham, Mysore in 2014. He has several papers in Machine Learning applied to Cyber Security. His Ph.D. work centers on Application of Machine learning (some times Deep learning) for Cyber Security and discusses the importance of Natural language processing, Image processing and Big data analytics for 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'18 and ICACCI'18. More details available at https://vinayakumarr.github.io/

MAMOUN ALAZAB received his PhD degree in Computer Science from the Federation University of Australia, School of Science, Information Technology and Engineering. He is an Associate Professor in the College of Engineering, IT and Environment at Charles Darwin University, Australia. He is a cyber security researcher and practitioner with industry and academic experience. Alazab's research is multidisciplinary that focuses on cyber security and digital forensics of computer systems including current and emerging issues in the cyber environment like cyberphysical systems and internet of things with a focus on cybercrime detection and prevention. He has more than 100 research papers. He delivered many invited and keynote speeches, 22 events in 2018 alone. He convened and chaired more than 50 conferences and workshops. He works closely with government and industry on many projects. He is an editor on multiple editorial boards including Associate Editor of IEEE Access (2017 Impact Factor 3.5), Editor of the Security and Communication Networks Journal (2016 Impact Factor: 1.067) and Book Review Section Editor: Journal of Digital Forensics, Security and Law (JDFSL). He is a Senior Member of the IEEE.

K. P. SOMAN has 25 years of research and teaching experience at 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 authored books on "Insight into Wavelets", "Insight into Data mining", "Support Vector Machines and Other Kernel Methods" and "Signal and Image processing-the sparse way", published by Prentice Hall, New Delhi, and Elsevier. More details available at https://nlp.amrita.edu/somankp/ PRABAHARAN POORNACHANDRAN is a professor at Amrita Vishwa Vidyapeetham. He has more than two decades of experience in Computer Science and Security areas. His areas of interests are Malware, Critical Infrastructure security, Complex Binary analysis, AI and Machine Learning.

SITALAKSHMI VENKATRAMAN earned her PhD in Computer Science, with a doctoral thesis titled "Efficient Parallel Algorithms for Pattern Recognition", from National Institute of Industrial Engineering in 1993. Prior to that she was awarded MSc (Mathematics) in 1985 and MTech (Computer Science) in 1987, both from Indian Institute of Technology (Madras) and subsequently MEd from University of Sheffield in 2001.

Sita has more than 30 years of work experience both in 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 in Australia since 2007. She is currently the Discipline Leader and Senior Lecturer in Information Technology at Melbourne Polytechnic. She specialises in applying efficient computing models and data mining techniques for various industry problems and recently in the 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 Program Committee Member of several international conferences every year.