APPROVAL SHEET

Title of Thesis: Deep Convolutional Neural Networks for the Classification of the EMBER Malware Dataset

Name of Candidate: Anudeep Nallamothu
Master of Science, 2018

Thesis and Abstract Approved: [illegible]
Dr. Charles Nicholas
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: November 29, 2018

ABSTRACT

Title of Thesis: Deep Convolutional Neural Networks for the Classification of the EMBER Malware Dataset

Anudeep Nallamothu, Master of Science, 2018

Thesis directed by: Dr. Charles Nicholas, Professor
Department of Computer Science and
Electrical Engineering

With the growing number of computer users across the world, security issues are growing exponentially. There is an imbalance in the pace of growing security issues and companies coming up with solutions. In May 2017, more than 400,000 computer systems in Telefonia and UK's National Health System were attacked by WannaCry Malware. Attackers and malware developers are using advanced malware techniques and vulnerabilities in the operating system to gain control over the victim's computer. They are coming up with new techniques and strategies to hide the malicious code and infect the targets. Anti-Virus scanners help to solve the detection of malware to some extent, but they fail to function when a new class of malware is presented. Therefore, we need a method of automating malware detection. So we are trying to apply a machine learning technique called Convolutional Neural Networks (CNNs) to accomplish the goal of automating malware detection.

In recent years, applying machine learning to malware data has drawn much attention. In the past, researchers have used CNNs on malware binaries (Nataraj *et al.* 2011) and malware windows PE files. In this thesis, the CNN technique is applied to statistically extracted features from Windows Malware PE files. We use the EMBER labeled benchmark dataset in this work. Results show that our model outperforms the *LightGBM* and *MalConv* models.

Deep Convolutional Neural Networks for the Classification of the EMBER Malware Dataset

by

Anudeep Nallamothu

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2018

I dedicate my work to Parents and Friends

ACKNOWLEDGMENTS

I would like to first express my deepest gratitude to my thesis advisor, Dr. Charles Nicholas for supporting me through my master's study and research. I am gratefully indebted to his invaluable guidance, understanding, patience, and motivation for this thesis. I would also like to thank my parents N. Srinivasa Rao and Nirmala, my brother, Sandeep, for being my strength. I thank every one of my friends who are with me on this beautiful journey. This accomplishment would not have been possible without them. Thank you!

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

Malware attacks have become a growing concern in the modern world. A malware attack from a terrorist organization on a government network cripples the administration and could reveal military information (Wong & Solon 2017). A Cyberattack on banking systems impacts the economic and financial growth of the respective countries (Kitten 2017). ”Dyre Wolf” is one such malware used for an attack on banking systems. An attack on the hospital and medical institutions could lead to loss of life (Guardian 2017). ”WannaCry” is a malware used for an attack NHS in England. Attackers can also use Trojans to attack personal computer in large scale and steal personal data from individuals. To understand more about the impact of malware across the world, the Center for Internet Security publishes a report every six months about attacks and malware types (for Internet Security 2018).

Researchers and technology companies on the other side have developed different techniques for the detection and analysis of such viruses. Scientific methods and software systems have been designed for removal of detected malicious programs (Dehghantanha & Franke 2014). Some methods involve event log analysis (Yen *et al.* 2013), malware intrusion detection, network traffic analysis, and malware binary file analysis (Han, Lim, & Im 2013). Anti-Virus scanners remain one of the most widely used methods of malware analysis and removal.

Even though there are concrete methods of identifying and classifying malwares, none of them scale to the speed that malware developers are creating new malwares. This indicates a need for automating detection and classification process with large malware samples. Machine Learning algorithms are widely adopted for these kind of tasks.Machine Learning has been increasingly applied to various prediction, classification and detection problems. This has been the motivation behind my research.

Lot of work has been done in the area of Deep Learning. Researchers are coming up with new architectures specific to different types of problems. Some examples are VGG 16, Inception, and ResNet for Image Classification. Recently CNNs have been applied to Natural Language Processing (NLP) problems. Mostly all these networks consist of stacked convolutional layers. All architectures follow the same idea of successively applying convolutional layers to the input, periodically downsampling the dimensions while increasing the number of feature maps. We describe a new architecture of Convolutional Neural Networks to the specific problem of Malware Classification.

1.2 What is a Malware?

Software that deliberately fulfills the harmful intent of an attacker is referred to as malicious software or malware (Bayer *et al.* 2006). Malware analysis is a process of determining the functionality and nature of this malware. Malware is designed to achieve some goals such as collecting personal data, gaining control over computer systems, and taking the system down. Analysis of a malware sample leads to determining the class of it. This process is referred to as Malware Classification. Networm, Trojan, Worm, Backdoor, etc are some examples of malware classes. The most common way that malware reaches end users is downloading from the Internet, Once the malware is downloaded into the system, based on the behaviour of the malware code, it will start infecting systems. Malware may overload the system, affect performance, deny control to users or steal your information. Because of its malicious behaviour, it is critical to carry out an in-depth analysis to understand the malware. We will discuss various types of malware analysis methods followed by malware classification methods.

We come across many terms in Computer Security like malware, trojan, virus, etc. with which a software vulnerability is often confused. However, all the above terms have a different meaning and significance in the domain of computer security.

1. **Malware:** A Malware is a software which is specifically written with the intent to harm a system.

2. **Virus:** A computer virus can be defined as a malicious piece of code/ program which is designed to alter the way a computer works. Usually, when a computer virus enters a system, it gets attached to another program (usually known as the host program) without the user's knowledge. Once the host program executes it triggers the virus and then the virus then may self-replicate or perform malicious actions in the computer. These computer viruses are mostly programmed to perform specific tasks and are designed to spread from one computer to another.

3. **Worm:** A computer worm is a malicious program which is designed to spread across computer or network and perform malicious actions. A computer virus usually requires execution of host program which then triggers the execution of the virus. A worm, on the other hand, is independent (does not require a host program) and starts self-replicating as soon as it enters the computer.

4. **Trojan:** A Trojan (usually known as a trojan horse) is a piece of code or software that is designed to mimic a trustworthy software and may trick the user to take control

of the user's computer. Once executed it performs actions it is designed to do such as steal information, create a backdoor, delete data, disrupt the working of computer etc. Trojans can be employed by cyber-thieves and hackers trying to gain access to users' systems.

5. **Adware:** Adware can be defined as a computer program that displays advertisements on your web pages and then redirects you to those sites. It can also collect the data of your page's history and display relevant advertisements to the user. Most of the time it is not very harmful but is irritating. However, it may also collect the user's keystrokes, track their browsing habits, eats up Internet data, etc.

6. **Spyware:** Spyware can be defined as a program the penetrates the system without the user's knowledge and secretly monitors the user's computer. It may also secretly steal sensitive information on the computer or secretly disrupt the regular working of the computer.

7. **Botnet:** A bot is a computer program which runs automated scripts on the Internet. A botnet can be defined as a collection of Internet-connected systems in which each system is running one or more bots. These bots can be used to secretly infiltrate other systems and gain access to them. They can also collect data and sensitive information on the computer without user's knowledge. (Vinod *et al.*)

1.3 Basic Malware Types

There are three types of malware (Vinod *et al.*).

1. **Basic:** In basic malware, the program entry point is changed such that the control is transferred to malicious code embedded inside the file. This type of malware is

detected by finding the signature and is the most typical problem dealt with in Anti-Virus scanners.

2. **Polymorphic:** Polymorphic viruses mutate while keeping the original code intact. A polymorphic malware consists of encrypted malicious code along with the decryption module. Malicious code can call the polymorphic engine which is embedded somewhere in the code to enable this malware. The polymorphic engine generates new mutants each time it is executed. Signature-based detection failes to detect such malware. Robust static analysis based on API sequencing is used for polymorphic virus detection.

3. **Metamorphic:** Metamorphic malware uses advanced obfuscation techniques and generates new children by the forking process. This changes the signature to its children who are different from its parent.

1.4 Contribution and Thesis idea

In general, for image classification problems, it does not require sophisticated tools or skillset to identify cars, flowers or object in an image. However, using those methods need a human expert and someone with tool leveraging capabilities. Using an Anti-Virus(AV) scanners solves the problem to some extent. However, AV scanners fail to detect when there is a new malware class. Hackers are increasingly using more sophisticated methods in creating new malware classes. Some malware is changing their behavior depending on the current system state and processes. With such a complex problem at hand, using a machine learning approach enhances the detection rate of new malware classes. It also scales with the growing number of malware attacks.

Existing malware analysis methods cannot cope with variance as they cannot take statistical features into account. Also, retrieving information from such large sample is time-consuming, and computation was consuming. So, we need to solve the problem of automatic classification under the existing statistical variance on a large scale. Most of the researchers used malware binary files and Windows PE files. Following are the tasks considered for this paper,

1. A key idea for using the EMBER labeled dataset is by using statistical features of a file identifying malware. Using the entire executable everytime is a time-consuming task. It cannot be sized into memory in some situations. Our novel approach to this problem involves using different properties or metadata extracted from the malware code of executable files as a dataset to construct a Convolutional Neural Network model. By using this CNN, the model we classify a file as malicious/benign. Statistical features used in this model are discussed in 'Dataset' section in this paper.

2. Compare our results with MalConv algorithm results. Labeling common results from both algorithms to label 300K unlabelled malware samples and contributing them to research community.

We begin in Chapter 2 with relevant background about the Malware techniques, Dataset, Convolutional Neural Networks, and Evaluation metrics. Related work is also discussed in this chapter. In Chapter 3, we describe our experiment, model, results, and comparison to other models. Chapter 4 concludes the thesis followed by the scope for future work.

Chapter 2

BACKGROUND

2.1 Malware Analysis

The following section discusses malware classification methods in detail. There are two types of malware analysis methods static and dynamic malware analysis.

2.1.1 Malware Analysis Methods

Static Analysis Static Analysis involves analyzing software file/executable without running it but allowing to view actual code and instructions. Basic static analysis is simple and may be rapid, but it is mostly ineffective against malware that is complex, and it may miss important malicious behavior (M, A, & Mahmod 2013). A human expert may be required to analyze every dissected file. After analyzing the file, some technical indicators are collected like file name, MD5 checksums or hashes, file type, file size and recognition by antivirus detection tools. Advanced static analysis techniques involve usage of disassemblers, analyzers, and detectors. Some of them are listed below.(M, A, & Mahmod 2013)

- Diassemblers - Ollydbg, IDA
- Detectors - AnalyzePE, MultiScanner

- Deobfuscation - unpacker, PackerAttacker
- Online Scanners and Sandbox - Cuckoo Sandbox, Malwr

1. Verify the file type(file extension) of sample to know malware file extensions.
2. Generate Hash value(MD5 or SHA25) and compare value to the database of known malware hashes.
3. Use VirusTotal to compare with an existing database of malware sample worldwide.
4. A Strings program like BinText can be to used to display all the strings within sample. Usually programs contain print messages in the form of print functions. Some web applications may have URLs to open or copy commands. These strings can give us an idea of working of executable. However, if the executable is packed or obfuscated, no useful strings can be seen.
5. Deobfuscation tools like unpacker can used to find internals of the sample.
6. Using a dissembler(IDA or IDA Pro) to analyze functionality of malware sample.

These are the necessary steps of performing static malware analysis. In case, if all these steps did not result in the identification of malware, Dynamic analysis methods can be used.

Dynamic Analysis Dynamic Analysis involves running the malware file to observe its behavior and functionality. This is a behavior-based detection method. An exhaustive static analysis could theoretically answer any question, but it is slow and laborious. Following activities are observed while performing dynamic malware analysis (Distler & Hornat 2008).

1. Registry Activity
2. File Activity
3. Process Activity
4. Network Traffic

Tools like Process Monitor, Process Explorer are used to perform Dynamic Malware analysis.

1. Process Monitor - PM is a SysInternals tool that records information about the File System, Registry and Process/Thread activity.
2. WireShark - It is a protocol analyzer that captures and decodes network traffic.

In this process, specific registry, file system, network, processes, and thread activities are analyzed from a running malware executable. A closed and isolated environment is used to manage this process as running a malware file could potentially impact the host processes.

2.2 Using Anti-Virus Scanners

Using Anti-Virus scanners is the most common method of identifying a malware file. They are the most advanced security systems available in the market today. They are designed to detect, prevent and take action to quarantine or remove malicious software from your computer. There are three types of detection methods. Typically, antivirus software uses all three scanning detection processes:

1. **Signature-based detection:** Anti-Virus software works by checking computer files, calculate hashes of each file and comparing them to a database of known types of

malware. This method is the most popular mechanism of detection. Continuous update of Anti-Virus software by synchronizing databases with a remote server is essential in this type of detection, to keep updated with newly detected malware.

2. **Heuristic Scanning:** Anti-Virus programs use heuristics, by running susceptible programs or applications with the suspicious code on it, within a runtime virtual environment. This keeps the vulnerable code from infecting the real world environment.

3. **Behaviour-based detection:** In this method of detection, Anti-Virus will scan for cues like,

    - programs that slow down system performance.
    - programs that lock resources for indefinite time.
    - programs that save keyboards typing data.
    - programs that corrupt other files when opened.

    It will also scan your computer for behaviors that may indicate the presence of a new, unknown malware.

2.2.1 Disadvantages of Anti-Virus softwares

- The most critical downside for an Anti-Virus program is identifying new classes. When a malware developer comes up with a new type of malicious code, having different behavior and signature, it cannot be identified right away. Some human expert needs to analyze it using Static or Dynamic malware analysis methods. Then, it needs to be updated in a central database used by all Anti-Virus programs. This whole process is time-consuming and inefficient.

- Anti-Virus scanners themselves are not hacking proof. Advanced malware developers can produce code, that can hack Anti-Virus scans and infect computers.

2.3 Machine Learning

2.3.1 Introduction

”Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.” (V, M, & others 2010)

Instead of rule-based programming we have a generic algorithm to which we feed the data. The algorithm then analyses the data, identifies patterns, builds logic and then takes decisions based on the logic built. The goal of machine learning can be described as making a computer to learn from observations with minimal human intervention. Machine learning is shaping today’s world as we see. It has become one of the most exciting parts in the Tech industry. From Self driving car to SIRI, Cancer gene detection to Netflix suggestions machine learning is being used everywhere.

There are two major types of machine learning problems: supervised learning and unsupervised learning. There is also a third category called Semi-Supervised learning which is not discussed in this book.

2.3.2 Supervised Learning

Supervised learning is a type of machine learning approach where a generic algorithm is ”trained” with the data(usually called training data) that has predefined results. Once the algorithm analyses the data and builds the logic based on the data given, it predicts the results for the new data(called test data or data with unknown results) that is fed to the algorithm. It is similar to that of function approximation where we try to approximate a function that satisfies a given set of inputs and respective outputs. Once the function is approximated to the required criteria, the function is used to predict the output for future inputs. One of the major flaws for supervised learning is that it requires a large amount of training data for the classification/prediction to be accurate. Some of the most commonly used supervised learning algorithms are

- Support Vector Machines
- Decision Trees
- Neural Networks
- Linear & Logistic regression
- K nearest neighbors
- Naive Bayes

2.3.3 Unsupervised Learning

Unlike supervised learning, unsupervised learning is a machine learning technique where the data is divided or classified based on their properties and inferences made by the algorithm. In this approach, the data that is used for training the algorithm does not have any labeled/ predefined results. The algorithm has to analyze the data and look for similarities and differences using which it can group the data.

There is no right and wrong when it comes to unsupervised learning. It only comes to the point what are you considering the criteria for grouping of data. One of the major flaws with this type of learning is that sometimes it is entirely unpredictable and it becomes difficult for us to infer on what basis the data has to be grouped. Most commonly used algorithms for unsupervised learning are

- Clustering - K means
- Association rules
- Principal component analysis.
- Neural Networks - Auto Encoders, Belief Nets (Salakhutdinov 2016)

2.3.4 Artificial Neural Network

An Artificial Neural Network (commonly known as Neural network or a perceptron or ANN) is a computing system which is designed based on the structure and functionality of neurons present in the human brain. The purpose of ANN is to mimic the functionality of the biological neurons and 'learn' to identify or predict the output from the input without explicit programming. The human brain consists of a very complex network formed from approximately 100 billion neurons. It is this network that enables humans to learn new things and understand through observation. An ANN is simply a small-scale replica of this huge network. A simple single layer ANN typically consists of 3 layers namely input layer, hidden layer, and the output layer.

The input layer takes the input from the user, and the output layer gives the output which is obtained from training the input in the network from the input layer to the hidden layer to the output layer. ANNs are used to model the complex relations between input and outputs or identify patterns in large data. They work as a framework along with other algorithms for analysis, classification, and prediction.

There are different types of ANNs.

1. Multi-Layer Perceptron(MLP)
2. Convolutional Neural Network(CNN)

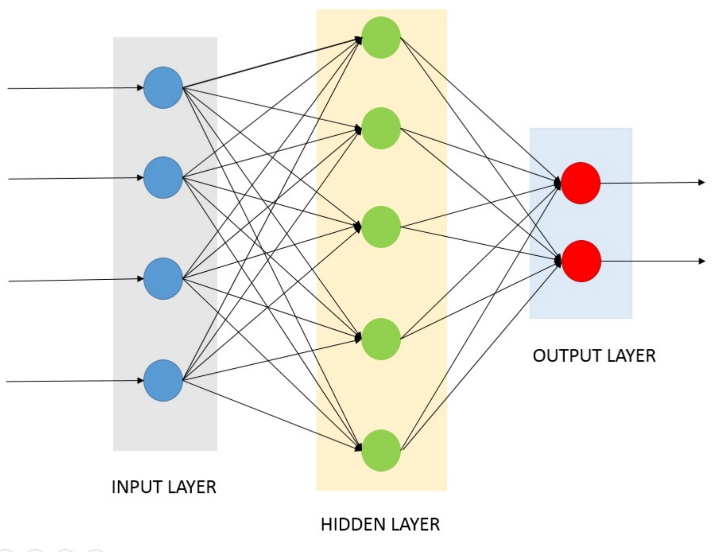


FIG. 2.1. Artificial Neural Network

3. Recurrent Neural Network(RNN)
4. Generative Adverserial Network(GAN)

2.3.5 Feature Engineering

Dr. Jason Brownlee says, “ *Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.*” (Lee 2014)

A feature is typically a specific representation on top of raw data, which is an individual, measurable attribute, typically depicted by a column in a dataset. Considering a generic two-dimensional dataset, each observation is depicted by a row and each feature by a column, which will have a specific value for an observation. Feature engineering is the most important step in machine learning process which can impact the accuracy of a model. In order to use machine learning access to a problem specific to a dataset, featuring is mandatory. Every record in the dataset contains information. Representation of this data is important for a machine learning model to perform effectively. The so-called Feature Engineering does this. A feature or an attribute of a data point is some specific information. For example, Humidity, Temperature, Wind speed, Pressure, etc are features of weather prediction data. Because machine learning algorithms require specific input, a conversion of this information is necessary. This can be done by transforming this information into feature vectors.

Features can be of two types: Raw Features and Derived Features.

- **Raw Features:** Inherent raw features are obtained directly from the dataset with no extra data manipulation or engineering.
- **Derived Features:** Derived features are usually obtained by the process of feature engineering on raw data and using existing data attributes. Including derived features in machine learning model is a way to inject expert knowledge into the training process, and so, to accelerate it.

Data can be two types. Continuous and Discrete data.

- **Discrete Data:** In weather prediction data, direction of wind, date, location name, etc represent discrete data. Categorical features could be represented by numbers. They cannot be increased boundlessly. The question is how to represent categorical data in an appropriate way.
- **Continuous Data:** In weather prediction data, temperature, pressure, wind speed are continuous data. These features can grow theoretically infinite. So we call it continuous data. A continuous data point is able to grow and can be represented for example by an integer or float.

One-Hot Encoding: Many machine learning algorithms cannot operate on labeled data directly. They require all input variables and output variables to be numeric. A popular way to represent categorical data is one-hot-encoding, where a categorical variable is replaced by features for their containing categories. They contain a numeric one for the chosen category, otherwise a zero. In our EMBER labeled dataset, there are three categories malicious (Infected file) represented by 1, benign(Uninfected file) represented by 0, unlabelled(Unknown file) represented by -1. Following is one-hot encoding for this data.

Benign	Malicious	Unlabelled
1	0	0
0	1	0
0	0	1

2.4 Convolutional Neural Network

In order to understand what Convolutional Neural Networks are, first, we need to understand what a convolution is. We can think of convolution as a sliding window function applied to a matrix. This can be illustrated better with the example below:

Lets imagine the matrix to the left is an image with pixel values 0(black) and 1(white). The sliding window (middle image) is called as a filter, kernel or a feature detector. Generally, a filter of size 3x3 is used as a sliding window, and we multiply the values of this filter with the original matrix on the left element-wise and then sum them up to get the values of a convolved feature on the right. For the full convolution, we do this for each element by sliding the filter over entire matrix.

So, what is the use of doing these convolutions? In the above example, we are multiplying the filter values and the matrix values in element-wise and summing them up. If we average the values rather than summing up, it blurs the image on the left. So we can also

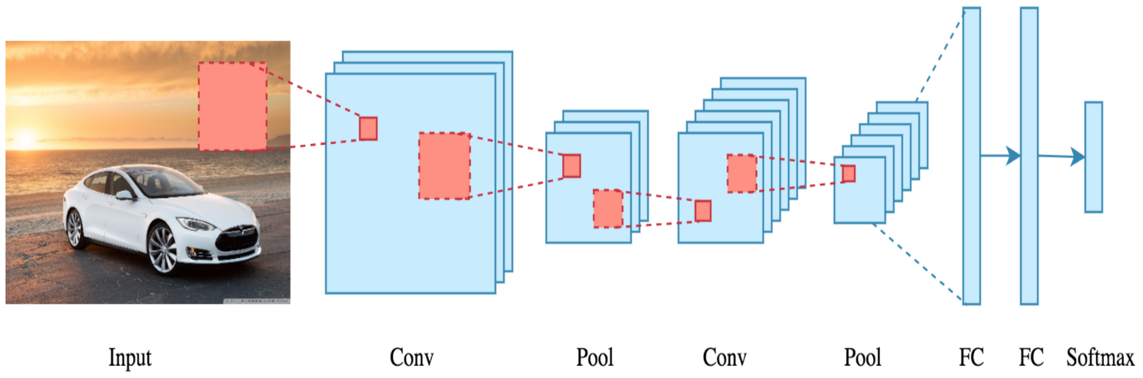


FIG. 2.2. Convolution Architecture Example

(Le 2018)

detect edges in an image if we take the difference between a pixel and its neighbors.

Convolutional Neural Networks are a bunch of layers of convolutions with nonlinear activation functions like ReLU or tanh applied to the outputs of convolutions. In Fully connected neural networks every input neuron is connected to every output neuron in the subsequent layers. That is also called a densely connected layer, or affine layer. In Convolutional Neural Networks we do not do that. Instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters, typically hundreds or thousands and combines their results.

During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform. For example, in Image Classification, a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features.

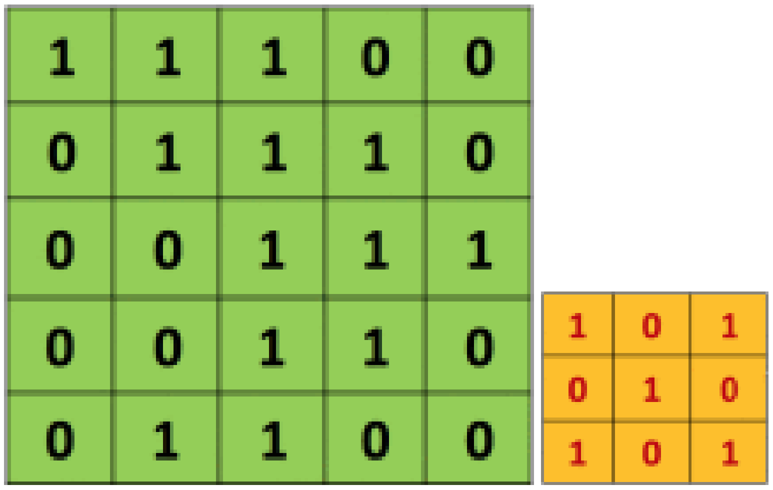


FIG. 2.3. Binary Image and Convolution matrix

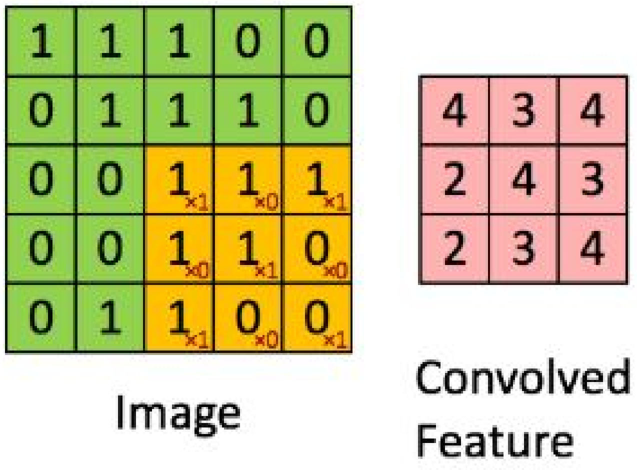


FIG. 2.4. Convolution Feature

(Kaparthy 2016)

2.4.1 Hyperparameters in CNNs

Narrow vs Wide convolution Applying a filter at the center of the image works fine. However, how do we apply the filter to the first element of the matrix that doesn't have adjacent elements to the left or the top? In this case, we apply zero padding. We consider all the elements that are falling outside of the matrix as zero. So by performing this, we can apply a filter to every element of our input matrix. Applying zero padding is called wide convolution and applying no padding is called narrow padding.

Stride Size One other important hyperparameter is the stride size which tells how much we want to shift the filter at each step. If the stride size is 1, the following applications of the filer will be overlapped. A larger stride leads to fewer applications of the filter and smaller output size.

Pooling Layers An important aspect of CNNs are the pooling layers, applied after the convolutional layers. Pooling reduces the size of the representation and to speed up the computation as well as to make some features it detects more robust. The most common way of applying pooling is the max pooling. We do not need to pool over the entire matrix, we could pool over a window. For example, the following shows max pooling for a 2x2 window.

One advantage of pooling is that it provides a fixed size output matrix. For example, if you have 500 filters and you apply max pooling to each, you will get a 500-dimensional output, regardless of the size of your filters, or the size of your input. This allows us to use variable size sentences, and variable size filters, but always get the same output dimensions to feed into a classifier.

Pooling also reduces the output dimensionality but keeps the essential information. We can think of each filter as detecting a specific feature, such as detecting if the sentence

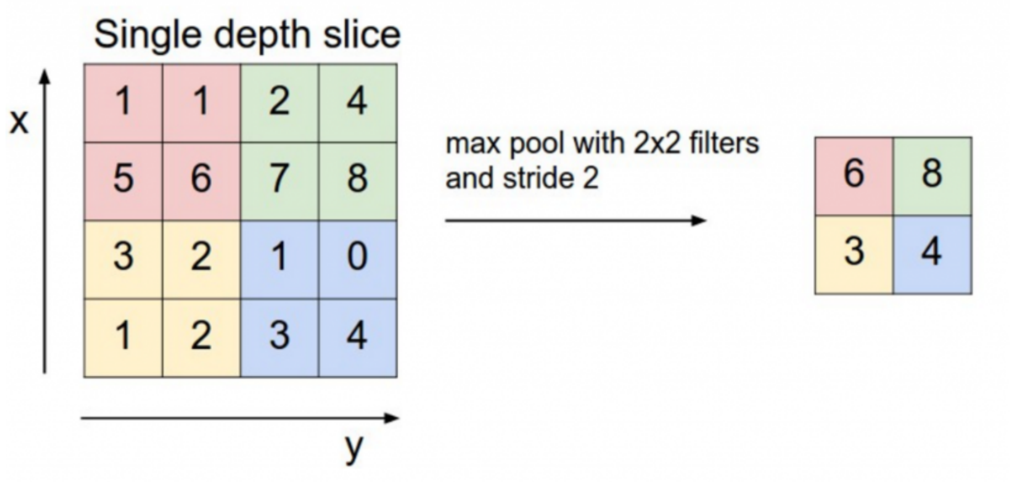


FIG. 2.5. Pooling

(Kaparthy 2016)

contains negations like ”not good” for example. If this phrase occurs somewhere in the sentence, the result of applying the filter to that region will yield a large value, but small value in other regions. By performing the max operation we are keeping information about whether or not the feature appeared in the sentence, but we are also losing information about where exactly it appeared.

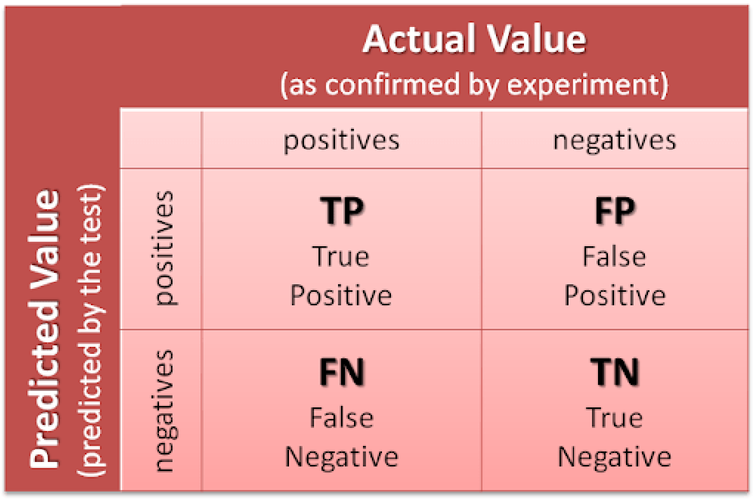
In image recognition, pooling also provides Translation, Invariance, and rotation. When you are pooling over a region, the output will stay approximately the same even if you shift/rotate the image by a few pixels, because the max operations will pick the same value regardless.

When we talk about CNNs, we mostly think about their applications in computer vision, but CNNs are also being applied in the area of NLP. In NLP, the most natural fit for CNNs seems to be classifications tasks such as Sentiment Analysis, Spam Detection or Topic Categorization. Convolutions and pooling operations lose information about the local order of words, so that sequence tagging as in, PoS Tagging or Entity Extraction is a bit harder to fit into a pure CNN architecture. (Yin *et al.* 2017)

2.5 Evaluation Metrics

2.5.1 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. Output can be 2 or more classes. The confusion matrix shows how the classification model is confused when it makes predictions. It is also called an Error matrix.



		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	**TP** True Positive	**FP** False Positive
	negatives	**FN** False Negative	**TN** True Negative

FIG. 2.6. Confusion Matrix

(Rohan 2018)

- **Positive (P)** Observation is positive. Considered as ’Yes’.
- **Negative (N):** Observation is not positive. Considered as ’No’.
- **True Positives (TP):** We predicted ’Yes’ and actual value is ’Yes’.

- **True Negatives (TN):** We predicted 'No', and actual value is 'No'.
- **False Positives (FP):** We predicted 'Yes', and actual value is 'No'. This is called a "Type I error."
- **False Negatives (FN):** We predicted 'No', and actual value is 'Yes'. This is called a "Type II error."

2.5.2 Recall

It is defined as the ratio of the total number of True positives to the total number of positive examples which is, out of all the positive classes, how much we predicted correctly. It should be as high as possible.

$$\boxed{Recall = \frac{TP}{TP + FN}} \tag{2.1}$$

2.5.3 Precision

It is defined as a ratio of the total number of True positives to the total number of predicted positives, which is out of all the classes, how much we predicted correctly. It should be as high as possible.

$$\boxed{Precision = \frac{TP}{TP + FP}} \tag{2.2}$$

2.5.4 F1-Score

Accuracy is affected mainly by True Negatives, and we do not focus on much whereas False Negative and False Positive usually. F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution. F1 score reaches its best value at 1 and worst at 0 (Wikipedia 2018).

$$\boxed{F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}} \tag{2.3}$$

2.5.5 AUROC

We use AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification models performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics)(Narkhede 2018). AUROC curve is used as a performance measurement for a classification problem. It can be used with various thresholds settings. ROC is a probability curve, and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. By analogy, higher the AUC, better the model is at distinguishing between malware and benign files.

When we decrease the threshold, we get more correctly identified values thus increasing the sensitivity. Meanwhile, this will decrease the specificity. Similarly, when we increase the threshold, we get more incorrectly identified values thus increasing the specificity and decreasing sensitivity.

2.6 Related Work

2.6.1 Application of Machine Learning to Malware Analysis

In the initial stages of applying machine learning to the malware classification problem, clustering algorithms were applied to the datasets. These approaches fail when there is a wide variety of malware classes available in the dataset. In the survey paper (Shalaginov *et al.* 2018), researchers discussed the different machine learning algorithms that can be applied to a Windows PE files dataset. They extracted byte n-gram, opcode n-gram, API calls and PE32 from Windows executable and applied Statistical based algorithms(Naive Bayes), SVM and K-nn algorithms. Despite having been able to extract all statistical features from an executable, none of the algorithms were able to show more than 90% accuracy.

In 2014, an attempt has been made to create a machine learning classifier that can detect zero-day attacks. They used various machine learning methods, finding that Decision Tree classifiers outperform Logistic Regression and Naive Bayes in this scenario. Also, they analyzed various features of the PE32 header and identified those most suitable for machine learning classifiers(Markel & Bilzor 2014).

In 2016, researchers from Technical University of Munich published their work on Application of Deep Neural Networks to a malware dataset. They used malware system call sequences extracted from Windows PE files as a dataset. Results show an accuracy of 89% when a hybrid model of the convolutional neural network and LSTM is used. Also, in the paper, they reported the application of SVM to the same model resulted in 71% accuracy. Although the accuracy achieved is not efficient for a real-world scenario, their research helped in understanding how data can be pre-processed in a way used by Neural Network models (Kolosnjaji *et al.* 2016).

2.6.2 Application of CNN to Malware Classification

In 2015, Microsoft released a Kaggle Challenge with a large dataset of Windows EXE files. Challenge is to classify malware into families based on file content and characteristics. Dataset has 35K malware samples of size 400GB and 10 malware families. We included this data in our dataset as well. This challenge opened up malware classification problem to deep learning researchers with hundreds of models submitted till date. Best performing model can achieve 99.3% on this dataset. A limitation to this dataset is, it represents an only small number of malware families that exist in the real world.

In 2017, researchers from Nvidia, Booz Allen Hamilton and UMBC collaborated to implement Malware detection model by using raw byte sequences of Windows EXE file

(Raff *et al.* 2017). They used a dataset from Microsoft Malware Classification Challenge. In this approach, malware file is converted to a raw byte sequence, which is then converted as embedded vector and fed into deep learning model. In my opinion, from the point of extracting raw byte sequence, this problem is dealt like an application of CNN to Natural Language Processing problem.

2.6.3 Malware Image Binaries

In 2011, Dr. Natarajan and Dr. Karthikeyan published a path-breaking idea of converting a malware executable into a grayscale image and applying image classification algorithms, as if it is any other real-world image (Nataraj *et al.* 2011). They used a dataset of 9K samples. Deep learning is not being actively used for classification problems at that time. So they used computer vision techniques to classify malware images and achieved 98% accuracy. However, the, in this case, is scaling. The same technique is hard to scale for millions of malware samples. Hence, it is difficult to prevent a zero-day attack.

In later years, the same idea of using malware images is considered for various research problems. Researchers started getting excellent results, with the usage of GPUs for the Deep Learning Algorithms. Researchers applied CNNs to the problem and achieved 98% accuracy but to a larger dataset(half a Terabyte) (Espoir K. Kabanga 2018). However, this approach is vulnerable to adversarial attack and produce erroneous results. A small change to the image could lead to misclassification. So attackers might leverage this drawback in the algorithm.

Chapter 3

METHODS

3.1 Implementation

In the coming sections, we've discussed software and hardware details used for this work.

3.1.1 Scikit-learn and SciPy

Scikit-learn is a machine learning library for the Python programming language. NumPy and SciPy are libraries which together provide MATLAB-like functionality in Python. It is open source software licensed under the BSD license [PVG+11]. Scikit-learn was initially started by David Cournapeau in 2007 as a Google Summer of Code project. In 2010 members of the French research institute INRIA 16 took leadership and made the first public release. Scikit-learn is continuously developed, with a solid and fast implementation and an outstanding documentation of the implemented algorithms. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. It provides classes for unsupervised and supervised machine learning approaches, like the described k-means, decision trees and random forests. It also provides efficient data structures for evaluation and model improvement. Often used alongside scikit-learn is the SciPy ecosystem 17. It provides several packages with different core features. For example NumPy 18 provides n-dimensional array structures with powerful linear algebra functionality. Matplotlib 19 is a 2D plotting library, which is able to provide fast and easy plotting results. It works seamlessly with the data structures of SciPy (Jones *et al.* 2001).

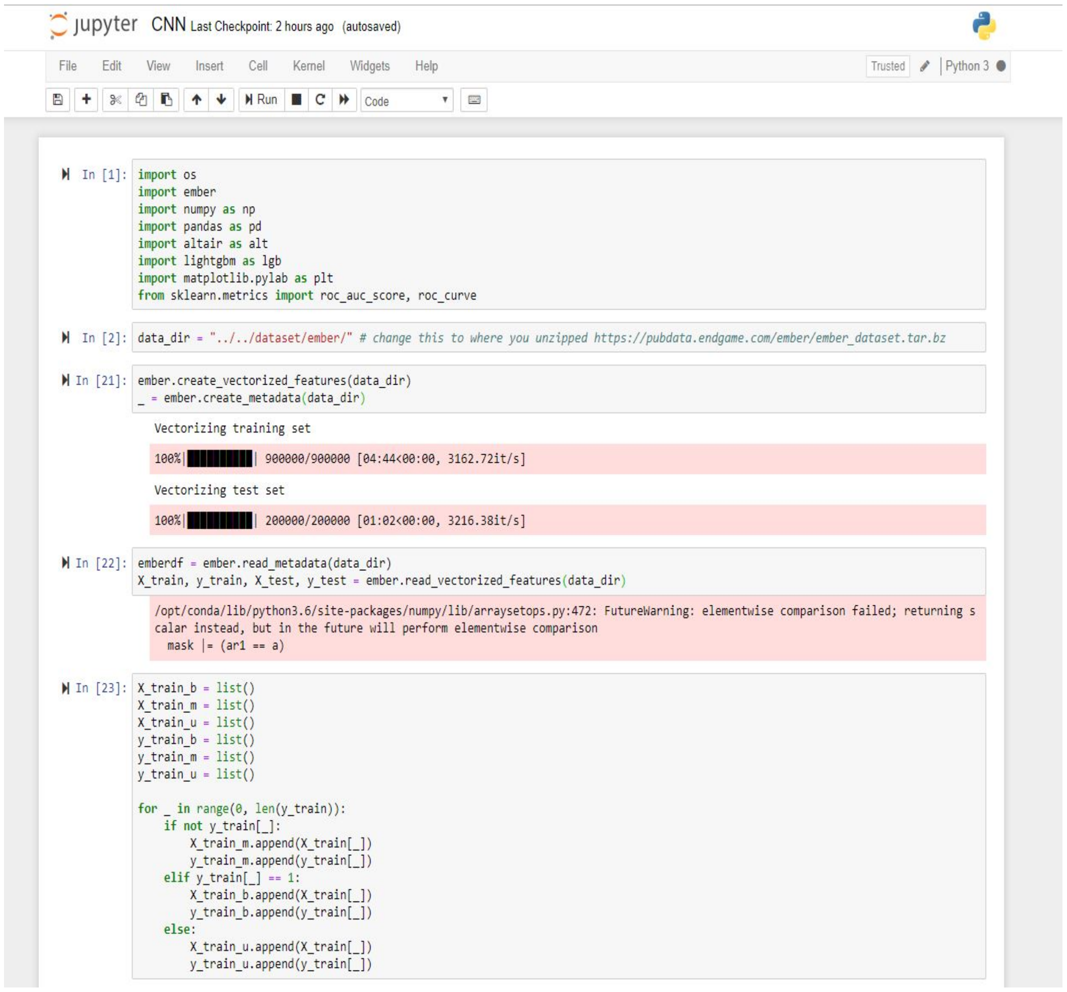
In our thesis, we used this for implementation of ROC curve, and graphs related to False Positive(FP), True Positive(TP), False Negative(FN) and True Negatives(TN) values. We also used this in calculating and generating Confusion matrix which is discussed later in this paper.

3.1.2 Keras Framework

Keras is a open source neural network framework created to facilitate rapid prototyping of deep learning models. It is a high-level neural networks API, written in Python. It is created on top of two leading deep learning frameworks, Theano and Tensorflow. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) (Chollet & others 2015) and its primary author and maintainer is Franois Chollet, a Google engineer. We used Keras with Tensorflow backend to implement the model. Keras facilitates APIs to implement, Multi-Layer Perceptron, Convolutional Neural Networks(1D, 2D, 3D), Long-Short Term Memory Cells(LSTMs). It also has support for various loss functions, optimizers and activation functions like Sigmoid, LeakyReLU, ReLU, softmax etc.

3.1.3 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. It can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning etc. Jupyter Notebook is backed by Project Jupyter, a spin off from IPython Notebook, authored by (Fernando Perez & others 2014). This is a revolutionary product which eased the use machine learning for rapid prototyping. Since it is a web based interface, we can access it from remote server which removes the overhead of installing in all our systems. Code written in individual cells are communicated to python server using kernel. A kernel is a program responsible for handling various types of request (code execution, code completions, inspection), and providing a reply. Kernels talks to the other components of Jupyter using ZeroMQ over the network, and thus can be on the same or remote machines. Following image shows sample jupyter notebook.


jupyter CNN Last Checkpoint: 2 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [1]: import os
        import ember
        import numpy as np
        import pandas as pd
        import altair as alt
        import lightgbm as lgb
        import matplotlib.pylab as plt
        from sklearn.metrics import roc_auc_score, roc_curve

In [2]: data_dir = "../../dataset/ember/" # change this to where you unzipped https://pubdata.endgame.com/ember/ember_dataset.tar.bz

In [21]: ember.create_vectorized_features(data_dir)
         _ = ember.create_metadata(data_dir)

  Vectorizing training set

  100%|██████████| 900000/900000 [04:44<00:00, 3162.72it/s]

  Vectorizing test set

  100%|██████████| 200000/200000 [01:02<00:00, 3216.38it/s]

In [22]: emberdf = ember.read_metadata(data_dir)
         X_train, y_train, X_test, y_test = ember.read_vectorized_features(data_dir)

  /opt/conda/lib/python3.6/site-packages/numpy/lib/arraysetops.py:472: FutureWarning: elementwise comparison failed; returning s
  calar instead, but in the future will perform elementwise comparison
    mask |= (ar1 == a)

In [23]: X_train_b = list()
         X_train_m = list()
         X_train_u = list()
         y_train_b = list()
         y_train_m = list()
         y_train_u = list()

         for _ in range(0, len(y_train)):
             if not y_train[_]:
                 X_train_m.append(X_train[_])
                 y_train_m.append(y_train[_])
             elif y_train[_] == 1:
                 X_train_b.append(X_train[_])
                 y_train_b.append(y_train[_])
             else:
                 X_train_u.append(X_train[_])
                 y_train_u.append(y_train[_])
```

FIG. 3.1. Jupyter Notebook

3.1.4 Docker

For initial testing of models, I used different GPU systems with varied computation power. Installing all the dependencies every other time became a bottleneck for quick experimentation. I used docker as a solution to this problem. Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system-level virtualization on Linux.

3.1.5 GPU - Nvidia GV100

Deep Learning is a compute intensive task involving millions of floating point operations and matrix multiplication operations. For this reason, researchers started using GPUs for implementation of Neural Networks models. Tensorflow runs on both GPUs and CPUs. CPUs are designed for more general computing workloads. GPUs in contrast are less flexible, however they are designed to compute in parallel for the same instructions. Also, MatMul(Matrix Multiplication) operations are performed in parallel. Deep Neural Networks (DNN) are structured in a very uniform manner such that at each layer of the network thousands of identical artificial neurons perform the same computation. Therefore, the structure of a DNN fits quite well with the kinds of computation that a GPU can efficiently perform. We used an Nvidia GV100 GPU with a performance of 118 Tensor Floating Point Operations(TFLOPs), 32 GB RAM and 250 W power consumption.

3.2 Dataset

3.2.1 Introduction

When coming up with new machine learning/deep learning models, using a well processed and structured dataset helps is getting fast results. Benchmark datasets are used for a systematic evaluation of machine learning models. Some examples of benchmark datasets,

- **MNIST dataset:** consists of images of hand written digits from 0 to 9. Each images is 28 x 28 dimension. A total of 60K gray scale images.
- **Fashion MNIST:** consists of images of 10 different classes. T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot. Each images is 28 x 28 dimension. A total of 60K gray scale images.
- **MovieLens:** Two datasets available. The first dataset has 100K ratings for 1682 movies by 943 users, subdivided into five disjoint subsets. The second dataset has about 1M ratings for 3900 movies by 6040 users.

These are some example datasets. They are used again and again by researchers to test the hypothesis quickly.

3.2.2 Challenges with Malware datasets

Gathering Malware datasets is a humongous task. We summarize some the challenges researchers face in collecting malware related datasets(Sarkar 2018).

1. **Variety:** Until Kaggle Microsoft challenge (Kaggle 2015) there is no pre-processed malware files are available publicly without licensing restrictions. Main disadvantage of using this dataset is it includes only windows PE files. But, the range of malwares developed by attackers is far larger than that. So, a dataset with samples from multiple data sources was never available until Endgame came up with the EMBER benchmark dataset. This is one prime reason for selecting EMBER benchmark as dataset for our thesis.

2. **Institutional Restrictions:** Information internal to organizations is bound to NDA(Non-Disclosure Agreement). So when there is an attack on organization's network, companies don't prefer to reveal attack related data to public for the fear revealing confidential information. This has become a major bottleneck for collecting data in large amounts.

3. **Data Labeling:** Labelling a file as malware/benign needs a human expert and time. New classes of malware cannot be identified solely based on SHA-256 which Anti-Virus scans work on. So, when there is dataset which is unlabelled, creating a labelled dataset out of it is a huge task.

4. **Infrastructure:** When there is a malware attack on personal computers used by individual, there is no defined pipeline to collect information without revealing personal information. Use of VirusTotal website is one solution. More awareness and infrastructure are needed to accomplish this task.

3.2.3 EMBER dataset

EMBER is a labeled benchmark dataset for training machine learning models to statically detect malicious Windows portable executable files. The dataset includes features extracted from 1.1M binary files. 900K training samples (300,000 malicious, 300K benign, 300K unlabeled) and 200K test samples (100K malicious, 100K benign). Following is an example data object from dataset.

```
{
   "appeared":"2006-12",
   "byteentropy":[0,619,493,......373615,372116,373375,373929,375883],
   "exports":[],
   "general":
    {
        "exports":0,
        "has_debug":0,
        "has_relocations":0,
        "has_resources":1,
        "has_signature":0,
        "has_tls":0,
        "imports":156,
        "size":3101705,
        "symbols":0,
        "vsize":380928
    },
"header":
    {
        "coff":
            {
                "characteristics":["CHARA_32BIT_MACHINE","RELOCS_STRIPPED",
                        "EXECUTABLE_IMAGE","LINE_NUMS_STRIPPED",
                        "LOCAL_SYMS_STRIPPED"],
                "machine":"I386",
                "timestamp":1124149349
            },
        "optional":
            {
                "dll_characteristics":[],
                "magic":"PE32",
                "major_image_version":0,
                "major_linker_version":7,
                "major_operating_system_version":4,
                "major_subsystem_version":4,
                "minor_image_version":0,
                "minor_linker_version":10,
                "minor_operating_system_version":0,
                "minor_subsystem_version":0,
                "sizeof_code":26624,
                "sizeof_headers":1024,
                "sizeof_heap_commit":4096,
                "subsystem":"WINDOWS_GUI"
             }
       },
    "histogram":[45521,13095,12167,.....12170,12596,22356],
    "imports":
       {
          "ADVAPI32.dll":["RegEnumValueA",.....“RegCreateKeyExA"],

          "COMCTL32.dll":["ImageList_AddMasked","ImageList_Create",
                "ImageList_Destroy",""],

          "GDI32.dll":["SetTextColor", .... "GetDeviceCaps"],

          "KERNEL32.dll":["SetFileTime" ......"MulDiv"],
          "SHELL32.dll":["SHFileOperationA",.....“ShellExecuteA"],

          "USER32.dll":["CloseClipboard",....“PostQuitMessage"],
        "VERSION.dll":["VerQueryValueA",....."GetFileVersionInfoA"],
        "ole32.dll":["OleInitialize",......"OleUninitialize"],

        "snmpapi.dll":["SnmpUtilOidCpy",....."SnmpUtilVarBindFree"]
     },
  "label":0,
  "section":
     {
        "entry":".text",
        "sections":[
           {
              "entropy":6.532239617101003,
              "name":".text",
              "props":["CNT_CODE","MEM_EXECUTE","MEM_READ"],
              "size":26624,
              "vsize":26134
           },
           {
              "entropy":5.433081641309689,
              "name":".rdata",
              "props":["CNT_INITIALIZED_DATA","MEM_READ"],
              "size":6656,
              "vsize":6216
           },
           {
              "entropy":1.7424160994148217,
            "name":".data",
            "props":["CNT_INITIALIZED_DATA","MEM_READ","MEM_WRITE"],
            "size":512,
            "vsize":172468
         },
         {
            "entropy":0,
            "name":".rsro",
            "props":["CNT_UNINITIALIZED_DATA","MEM_READ","MEM_WRITE"],
            "size":0,
            "vsize":135168
         },
         {
            "entropy":5.020929764194735,
            "name":".rsrc",
            "props":["CNT_INITIALIZED_DATA","MEM_READ"],
            "size":27648,
            "vsize":28672
         }]
   },
"sha256":"0abb4fda7d5b13801d63bee53e5e256be43e141faa077a6d149874242c3f02c
"strings":
   {
   "MZ":51,
   "avlength":5.972071639333013,
   "entropy":6.569897560341239,
   "numstrings":14573,
        "paths":3,
        "printabledist":[1046,817,877,..... 853,802,845,804,900],
        "printables":87031,
        "registry":0,
        "urls":0
        }
    }
```

Here is the summary of dataset,

1. **appeared** - Date on which malware is identified.

2. **entropy** - The average rate at which information is produced by a stochastic source of data. A stochastic data represents randomness. Entropy came into picture when Shannon wanted to solve a fundamental problem in data communications. *"Fundamental problem of communication is for the receiver to be able to identify what data was generated by the source, based on the signal it receives through the channel"*. The equation used by Shannon has a resulting value of something between zero (0) and eight (8). The closer the number is to zero, the more orderly or non-random the data is. The closer the data is to the value of eight, the more random or non-uniform the data is. The formula used by Shannon to represent binary data looks like this:

$$\boxed{H = \sum_{n=1}^{255} P_i \log_2 P_i} \tag{3.1}$$

3. **byteentropy** - Entropy calculation for each byte of executable file.

4. **exports** - All the export functions used inside executable.

5. **general** - This includes summary of meta data like, number of export functions,

number of imports used, size of file, number of resources used and signatures.

6. **header** - Header is a part of the data packet and contains transparent information about the file or the transmission. In file management, a header is a region at the beginning of each file where meta-data information is kept.

7. **imports** - List of imports used inside executable file. For windows OS based malware, these imports are used inside '.dll' files.

8. **label** - Malware category information. It can have three values. 1 for malicious file, 0 for benign file and -1 for unlabelled file.

9. **section** - Section includes different types of files like '.text', '.rdata', '.data' and its meta-data like entropy, size of file, name of file. These files are packed inside the executable.

10. **sha256** - SHA-256 stands for Secure Hashing Algorithm-256 bit. It is a cryptographic hash function created by NSA to facilitate one-way compression. A hash function is a set of mathematical computations performed to create a unique value to the input data. A key aspect of cryptographic hash functions is their collision resistance: nobody should be able to easily find two different input values that result in the same hash output. A hash cannot be decrypted back to its original contents. This is used to verify data loss in file transfer operations. SHA-256 is one of the strongest hash function available in SHA family.

11. **strings** - Strings section include information about number of printables, number of strings, entropy of the files, average string line length and count of URLs.

3.3 Data Processing

EMBER framework by Endgame has been used for doing data processing. Raw bytes are extracted to generate parsed features. Gross characteristics of each section(mentioned in 'Dataset' section) like hash, file size, entropy, inputs are used. Each of these characteristics are converted into a vector of 2351x1 dimensions ranging from -100000 to 100000.

3.4 Model

Following images reveals the structure of convolutional layers and dense layers used. Our model uses,

- **Input Vector** - Each input vector has a dimension of 2351 units. A total of 600K samples are in training data and 200K samples in validation data. So we have (600000, 2351, 1) and (200000, 2351, 1) vectors as input.
- **Convolutional Layers** - There are 4 convolutional layers. Each layer has on Convolutional unit(conv1d), a batch normalization unit(BatchNormaliztion()) and a Max pooling unit (MaxPooling1D). the convolutional filter helps to discover the higher-order local features that are invariant to small changes in data.
- **Filters** - A decremental size from 512 to 64 filters are used in each layer.
- **Kernal Size** - 32. This is constant in every layer.
- **MaxPooling** - In MaxPooling layer, a 3 strides are used with a pool size of 3.
- **Flatten Layer** is used to convert n-dimensional matrix into 1D matrix. In order to use, fully connected network, we need to input 1D array. So we need to convert output of convolutional layers into a 1D feature vector. This is called Flattening.

This layer flattens all its structure to create a single long vector which is given as feature vector to dense layers.

- **Dense Layers** - There are 5 dense layers. Each dense has one fully connected unit(Dense) and a dropout unit(Dropout) with Dropout value 0.5. We used a common dropout value all over the network. Each layer has decremental units from 1024 to 2. Final layer has 2 units because we have 2 classes in dataset.
- **Output** - Output layer gets a binary classified output.
- **Weights** - Keras maintains and updates weights of the model after each epoch.

3.4.1 Hyperparameters

One Hot Encoding: We used Keras in-built one-hot encoding function. It gives [1.0, 0.0] vector for malware class and [0.0, 1.0] vector for benign class.

Activation Function: We use two activation functions in our model. Convolutional layers and fully connected layers use **tanh** activation function. During data analysis, we observed each input has a range of [-100000, 100000]. **tanh** activation function has a range of -1 to 1. So to have wide range of values, we preferred 'tanh' over 'sigmoid' activation functions. Because of this reason, tanh converges faster than sigmoid.

$$\boxed{tanh = \frac{2}{1+e^{-2x}} - 1} \tag{3.2}$$

For final output, we used **sigmoid** activation function because we are dealing with a binary classification problem and output of sigmoid function range is [0, 1].

$$\boxed{sigmoid = \frac{1}{1+e^{x}}} \tag{3.3}$$

Loss Function: We need to compare training output in each iteration with actual labels and calculate error value. This helps us understand by how much margin predicted value missed actual value. We use to loss functions to calculate this. The output variable in classification problem is usually a probability value. With binary cross entropy, you can only classify two classes. It is just a special case of categorical cross entropy. The equation for binary cross entropy loss is the exact equation for categorical cross entropy loss with one output node (DiPietro 2016).

Optimizer: Gradient descent is one of the most popular algorithms to perform optimization in neural networks.We used ’rmsprop’ optimizer. ’ramprop’ is an adaptive learning rate method proposed by Geoff Hinton. RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients. This steps optimizes the weights so that loss can be minimized.

Learning rate in our experiment is 0.001 which is default in ’rmsprop’ optimizer.

Batch size in our experiment is 256. In our approach, we can use a maximum batch size of 512 due to the memory limitations of GPU. For us, 256 worked well.

L2 Regularization: To solve the problem of overfitting, we used L2 regularization as it is good in dealing correlated inputs and we don’t have any sparsity for the input vectors.

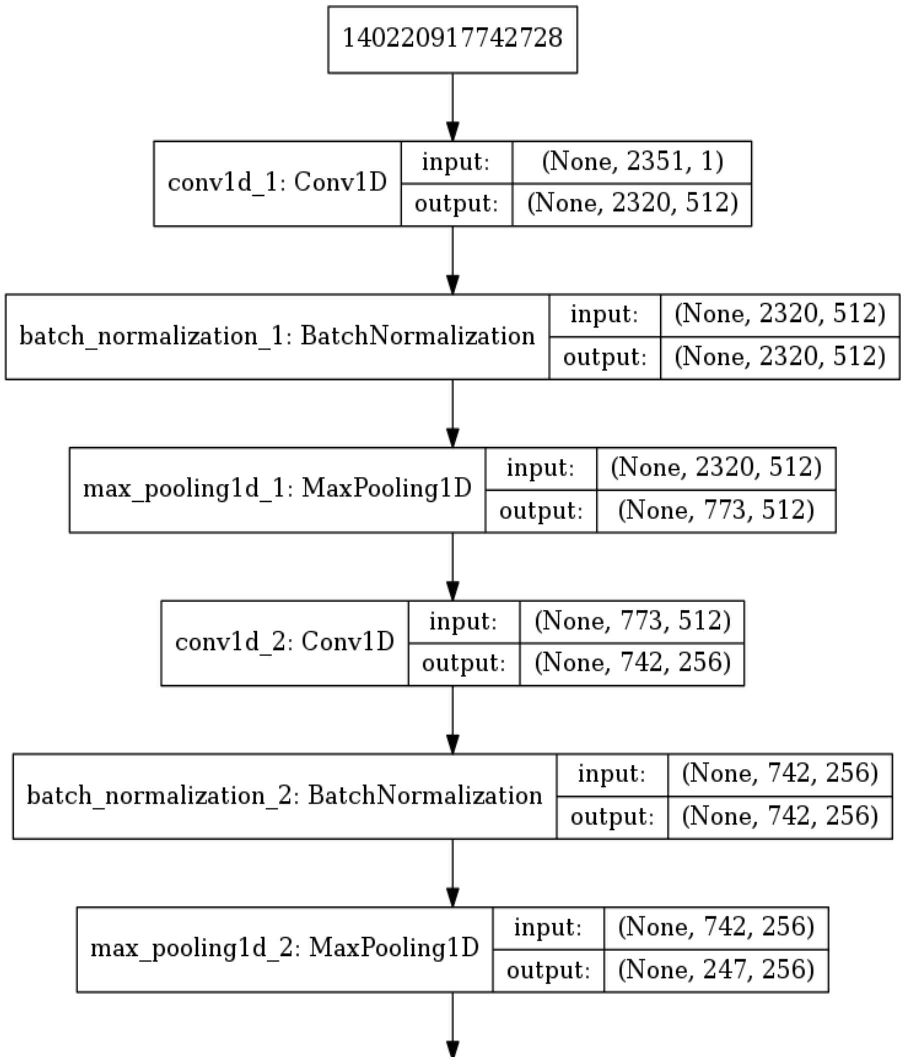


FIG. 3.2. Convolutional Layer 1 and 2

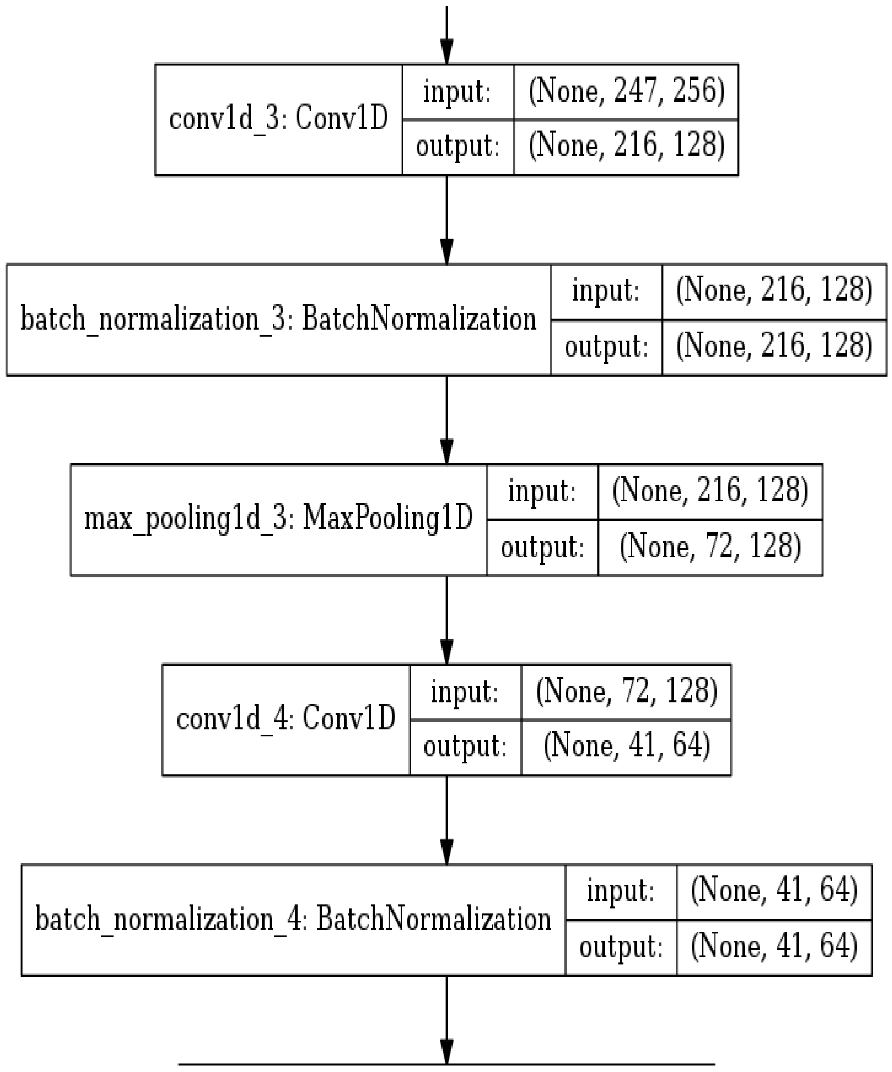


FIG. 3.3. Convolutional Layer 3 and 4

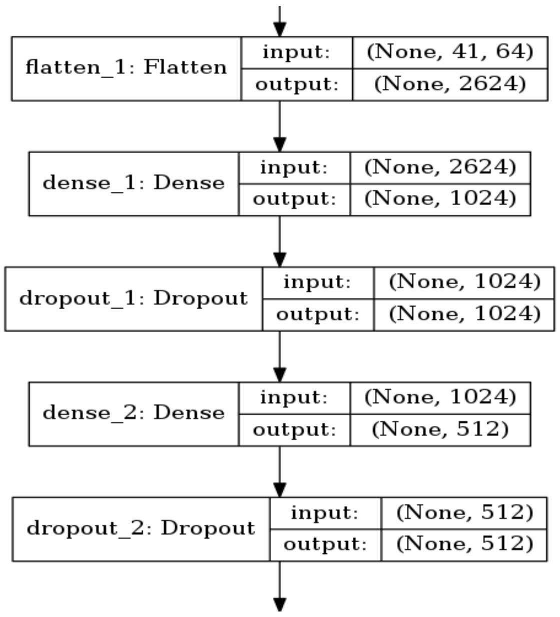


FIG. 3.4. Flatten, Dense Layer 1 and 2

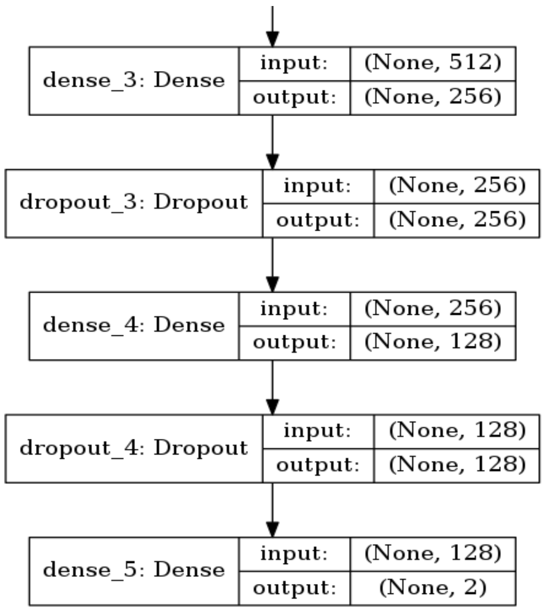


FIG. 3.5. Flatten, Dense Layer 3 and 4

Layer (type)	**Output Shape**	**Param #**
conv1d_1 (Conv1D)	**(None, 2320, 512)**	**16896**
batch_normalization_1 (Batch)	(None, 2320, 512)	2048
max_pooling1d_1 (MaxPooling1)	(None, 773, 512)	0
conv1d_2 (Conv1D)	**(None, 742, 256)**	**4194560**
batch_normalization_2 (Batch)	(None, 742, 256)	1024
max_pooling1d_2 (MaxPooling1)	(None, 247, 256)	0
conv1d_3 (Conv1D)	**(None, 216, 128)**	**1048704**
batch_normalization_3 (Batch)	(None, 216, 128)	512
max_pooling1d_3 (MaxPooling1)	(None, 72, 128)	0
conv1d_4 (Conv1D)	**(None, 41, 64)**	**262208**
batch_normalization_4 (Batch)	(None, 41, 64)	256
flatten_1 (Flatten)	**(None, 2624)**	**0**
dense_1 (Dense)	**(None, 1024)**	**2688000**
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	**(None, 512)**	**524800**
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	**(None, 256)**	**131328**
dropout_3 (Dropout)	(None, 256)	0
dense_4 (Dense)	**(None, 128)**	**32896**
dropout_4 (Dropout)	(None, 128)	0
dense_5 (Dense)	**(None, 2)**	**258**

Total params: 8,903,490

Trainable params: 8,901,570

Non-trainable params: 1,920

Table 3.1. Model Summary

3.5 Training

- We used Nvidia GV100 GPU which is supported by CUDA to train the model. More than 20 variations of Convolutional and Fully connected layer combinations are tried and tested.
- Current model took close to 80 minutes to train one epoch. We started training in the iterations of 5 to 25 epochs in the increments of 5 epochs.

3.6 Evaluation Metrics

3.6.1 Training Accuracy Vs Validation Accuracy

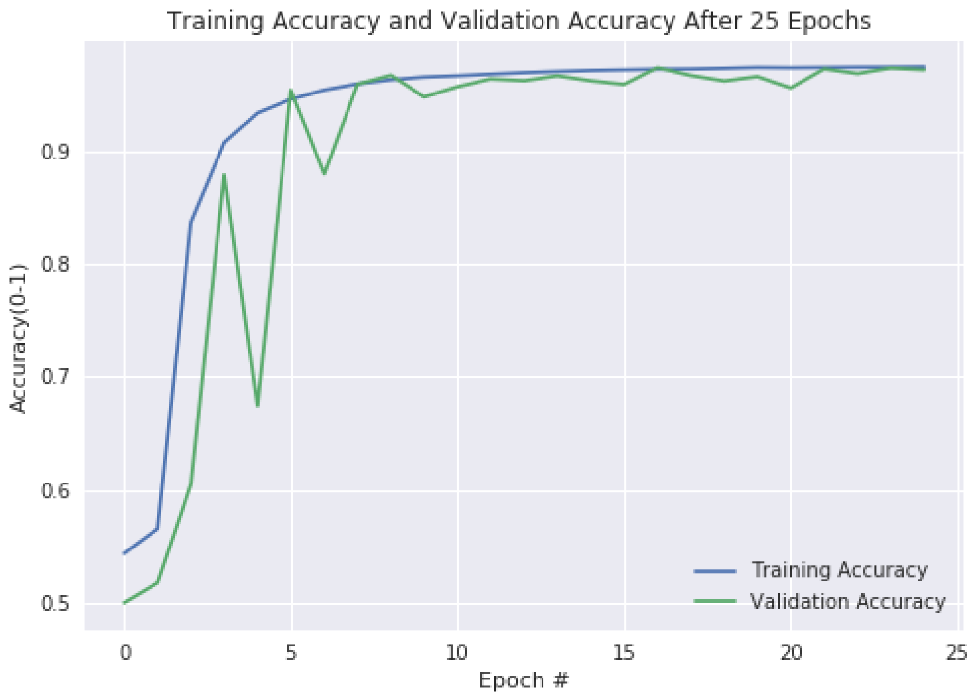


FIG. 3.6. Training Accuracy Vs Validation Accuracy

From Figure, 3.6 we can see that, Training Accuracy and Validation Accuracy converged, starting from epoch 5. Our model achieved a validation accuracy of 97.21% on 200k validation samples and a training accuracy of 97.4%

3.6.2 AUROC

ROC (Receiver Operating Characteristic) Curve tells us about how good the model can distinguish between two things. ROC value lies between 0-1 or 0%-100%. In Figure, 3.7, we can observe ROC curve from our experiment which is 0.97.

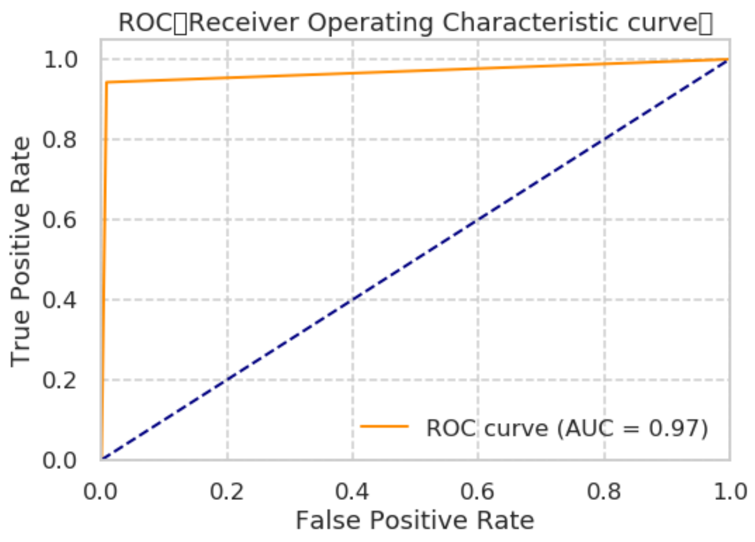


FIG. 3.7. AU-ROC Curve

3.6.3 Confusion Matrix

A confusion matrix is a technique used for summarizing the performance of a classification algorithm which can have 2 or more classes to classify. In Figure, 3.8, we can see summary of confusion matrix with actual metrics. In Figure, 3.9, we can see summary of confusion matrix with normalized values. This confusion matrix is used to calculate other evaluation metrics like Recall, Precision, Sensitivity which we discussed earlier.

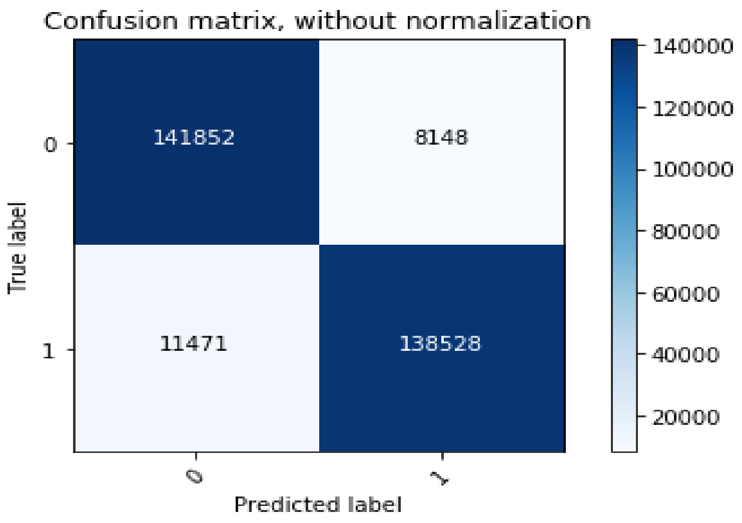


FIG. 3.8. Confusion Matrix without Normalization

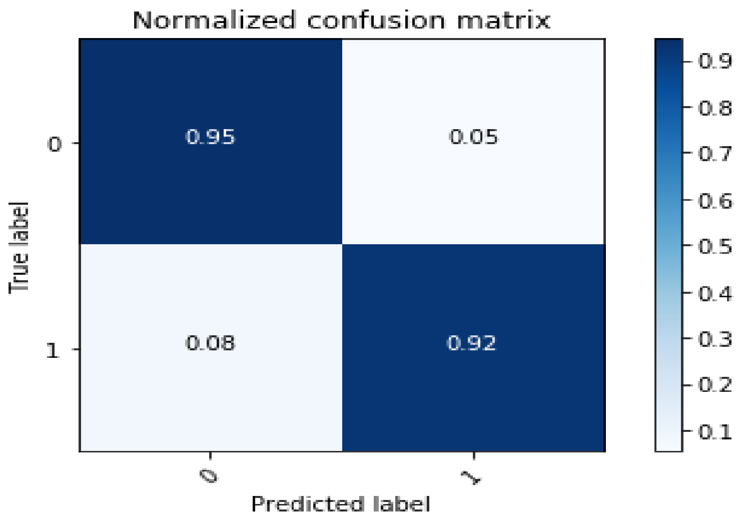


FIG. 3.9. Confusion Matrix With Normalization

Metric	**Value**
Recall(True Positive Rate)	94.5%
Specificity(True Negative Rate)	92.4%
Precision(Positive Predictive Value)	92.5%
Fall out(False Positive Rate)	7%
F1-score	93.5%
False Negative Rate	5%
False Discovery Rate	7%
Overall Accuracy	93.46%

Table 3.2. Evaluation Metrics Summary

3.7 Test Accuracy

As mentioned earlier, we used a test dataset od 300K files converted into vectorized features. We performed 5 fold cross validation experiment which will be discussed in next section. An average of 93.46% accuracy has been achieved on test dataset.

3.8 K-fold Cross Validation

Cross-Validation helps in avoiding underfitting and overfitting of data. By randomizing input data, we can make the model train properly without loosing any patterns in dataset. We conducted a 5 fold cross-validation experiments to estimate the results over new data. In each itertaion, we randomly split the dataset into three sets, 800K for Training, 200K for Validation and 300K for Testing. This process was repeated for five times. Finally, we computed the average results of the three tests to obtain a reliable measure of how well the proposed architecture performs over the entire dataset. Refer to Table 3.3

3.9 Results

To the best of our knowledge, this is the first work based on the EMBER dataset after its release in May 2018. Consequently, the effectiveness of the proposed approach can be assessed by comparing the reported results in the paper (Anderson & Roth 2018). Accuracy achieved for MalConv is 92.2% over the same dataset, whereas we registered 93.46% . While the performances are quite close, it is worth pointing out the differences between the method proposed in this paper and the one followed by MalConv are completely different. MalConv paper relies mostly upon raw byte sequences with a hybrid model whereas we created model based on vectorized inputs.

Iteration	**Validation Accuracy**	**Test Accuracy**
1	96.33	91.94
2	97.52	93.66
3	96.88	94.12
4	97.29	93.27
5	97.63	94.3
Average	**97.13**	**93.46**

Table 3.3. 5-fold Validation Summary

3.10 Unlabeled Data

In Dataset section, we mentioned about 300K unlabeled files. We used our model to vectorized the files and ran through the model. This experiment resulted in labelling 271K files accurately. This labelled data can be further used by future researchers to create better datasets.

3.11 Hyper parameter Tuning

1. In the initial model, training accuracy increased but validation accuracy calculated is very low(60%-70%). This is because of the overfitting problem. After some iterations, model will start memorizing inputs which will lead to the problem of overfitting data. So when it is encountered with new data, it fails. We used dropout layer with value 0.5 and added l2 regularization to address the overfitting problem.

2. We used a batch size of 256 and because of GPU memory limitations we can use a maximum value of 512.

Chapter 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

In this paper, we presented a novel convolutional neural network model for classification of malware based on vectorized statistical features from Windows PE files. It differentiates malware and benign files with 93.46% accuracy, 4% False-Positive rate and 1% False-Negative rate. This has been averaged over 5 fold cross-validation experiment. Evaluation results show that our model outperforms the MalConv (Raff *et al.* 2017) model which has 92.2% detection rate on the same dataset. We also, labeled 271K samples of unlabelled files, included in the EMBER dataset and contributed to the research community in an attempt to further add into datasets.

4.2 Future work

Malware developers can easily obfuscate the malware code to the point where it is impossible to retrieve any useful information. Using behavior-based approaches is not a reasonable solution. So, future researchers can find a way to include features about obfuscated code into the input data. We think, such a model will be robust in classifying malware in real time speed.

Currently, we are classifying a file as malware or benign. With more variety of classes and data sources in place now, classifying each sample into a malware class using Convolutional Neural Network could be a possible research problem. The scope of this problem will be huge because creating such a model can be used to detect any potential future malware classes that attackers come up with. Future research can be in the direction of creating models that can be inclusive of considering, Android-based, behavior-based, polymorphic and metamorphic malware classes.

Executable files are nothing but complexly packed assembly instructions. A sequence of instructions executed together will create malware behavior. So, Recurrent Neural Networks(RNNs) can be used because they are designed to deal with problems involving sequences or time series. RNNs are applied to Natural Language Processing problems since they involve a sequence of words. Another way forward is to create a hybrid model of RNNs and CNNs as architecture will be a potential research problem.

REFERENCES

[Anderson & Roth 2018] Anderson, H. S., and Roth, P. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*.

[Bayer *et al.* 2006] Bayer, U.; Moser, A.; Kruegel, C.; and Kirda, E. 2006. *Dynamic Analysis of Malicious Code*. *Journal in Computer Virology* 2(1):67–77.

[Chollet & others 2015] Chollet, F., et al. 2015. Keras. `https://github.com/fchollet/keras`.

[Dehghantanha & Franke 2014] Dehghantanha, A., and Franke, K. 2014. Privacy-respecting digital investigation. *2014 Twelfth Annual International Conference on Privacy, Security and Trust* 129–138.

[DiPietro 2016] DiPietro, R. 2016. Binary cross entropy. `https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/`.

[Distler & Hornat 2008] Distler, D., and Hornat, C. 2008. Malware analysis: An introduction. *Sans Reading Room*.

[Espoir K. Kabanga 2018] Espoir K. Kabanga, C. H. K. 2018. Malware Images Classification Using Convolutional Neural Network. *Computer Science & Communications* 6(1):153–158.

[Fernando Perez & others 2014] Fernando Perez, F., et al. 2014. Jupyter notebook. `https://jupyter.org/`.

[for Internet Security 2018] for Internet Security, C. 2018. Top 10 malware. `https://www.cisecurity.org/blog/top-10-malware-january-2018/`.

[Guardian 2017] Guardian, T. 2017. NHS seeks to recover from global cyber-attack as security concerns resurface. https://www.theguardian.com/society/2017/may/12/hospitals-across-england-hit-by-large-scale-cyber-attack.

[Han, Lim, & Im 2013] Han, K.; Lim, J. H.; and Im, E. G. 2013. Malware analysis method using visualization of binary files. *Proceedings of the 2013 Research in Adaptive and Convergent Systems* 317–321.

[Jones *et al.* 2001] Jones, E.; Oliphant, T.; Peterson, P.; et al. 2001–. SciPy: Open source scientific tools for Python. http://www.scipy.org/.

[Kaggle 2015] Kaggle. 2015. *Microsoft Malware Classification Challenge (BIG 2015).*

[Kaparthy 2016] Kaparthy, A. 2016. Cnn convolution matrix - cs231n. http://cs231n.github.io/convolutional-networks/.

[Kitten 2017] Kitten, T. 2017. New malware attacks prey on banks. https://www.bankinfosecurity.com/dyre-malware-a-8076.

[Kolosnjaji *et al.* 2016] Kolosnjaji, B.; Zarras, A.; Webster, G.; and Eckert, C. 2016. Deep learning for classification of malware system call sequences. 137–149.

[Le 2018] Le, J. 2018. CNN architecture. https://medium.com/analytics-vidhya/convolutional-neural-networks-the-\biologically-inspired-model-9b7e948f6987.

[Lee 2014] Lee, J. B. 2014. Discover feature engineering.

[M, A, & Mahmod 2013] M, D.; A, D.; and Mahmod. 2013. A survey on malware propagation, analysis, and detection. *International Journal of Cyber-Security and Digital Forensics* 2(4):10–.

[Markel & Bilzor 2014] Markel, Z., and Bilzor, M. 2014. Building a machine learning classifier for malware detection. *2014 Second Workshop on Anti-malware Testing Research (WATeR)* 1–4.

[Narkhede 2018] Narkhede, S. 2018. AUC-ROC curve. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5/.

[Nataraj *et al.* 2011] Nataraj, L.; Karthikeyan, S.; Jacob, G.; and Manjunath, B. S. 2011. Malware images: Visualization and automatic classification. *Proceedings of the 8th International Symposium on Visualization for Cyber Security* 4:1–4:7.

[Raff *et al.* 2017] Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; and Nicholas, C. 2017. Malware detection by eating a whole exe. *arXiv preprint arXiv:1710.09435*.

[Rohan 2018] Rohan. 2018. Confusion matrix.

[Salakhutdinov 2016] Salakhutdinov, R. 2016. Unsupervised learning. https://www.cs.cmu.edu/ rsalakhu/talk$_M LSS_part2.pdf$.

[Sarkar 2018] Sarkar, D. 2018. Understanding feature engineering. https://towardsdatascience.com/understanding-feature-engineering-part-1-continuous-numeric-data-da4e47099a7b.

[Shalaginov *et al.* 2018] Shalaginov, A.; Banin, S.; Dehghantanha, A.; and Franke, K. 2018. Machine Learning Aided Static Malware Analysis: A Survey and Tutorial. *Cyber Threat Intelligence. Advances in Information Security* 70:7–45.

[V, M, & others 2010] V, M.; M, D.; et al. 2010. What is machine learning? a definition. https://www.expertsystem.com/machine-learning-definition/.

[Vinod *et al.*] Vinod, P.; Jaipur, R.; Laxmi, V.; and Gaur, M. Survey on malware detection methods. *Proceedings of the 3rd Hackers Workshop on computer and internet security (IITKHACK09)* 74–79.

[Wikipedia 2018] Wikipedia. 2018. F1 score — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=F1%20score&oldid=861659585`. [Online; accessed 29-November-2018].

[Wong & Solon 2017] Wong, J. C., and Solon, O. 2017. Massive ransomware cyber-attack hits nearly 100 countries around the world. `https://www.theguardian.com/technology/2017/may/12/global-cyber-attack-ransomware-nsa-uk-nhs`.

[Yen *et al.* 2013] Yen, T.-F.; Oprea, A.; Onarlioglu, K.; Leetham, T.; Robertson, W.; Juels, A.; and Kirda, E. 2013. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. *Proceedings of the 29th Annual Computer Security Applications Conference* 199–208.

[Yin *et al.* 2017] Yin, W.; Kann, K.; Yu, M.; and Schütze, H. 2017. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*.