Spring 5-20-2019

Classification of Malware Models

Akriti Sethi
San Jose State University

Recommended Citation
Sethi, Akriti, "Classification of Malware Models" (2019). *Master's Projects*. 703.
DOI: https://doi.org/10.31979/etd.mrqp-sur9
https://scholarworks.sjsu.edu/etd_projects/703

Classification of Malware Models

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Akriti Sethi

May 2019

The Designated Project Committee Approves the Project Titled

Classification of Malware Models

by

Akriti Sethi

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2019

Dr. Mark Stamp	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science
Dr. Philip Heller	Department of Computer Science

ABSTRACT

Classification of Malware Models

by Akriti Sethi

Automatically classifying similar malware families is a challenging problem. In this research, we attempt to classify malware families by applying machine learning to machine learning models. Specifically, we train hidden Markov models (HMM) for each malware family in our dataset. The resulting models are then compared in two ways. First, we treat the HMM matrices as images and experiment with convolutional neural networks (CNN) for image classification. Second, we apply support vector machines (SVM) to classify the HMMs. We analyze the results and discuss the relative advantages and disadvantages of each approach.

ACKNOWLEDGMENTS

I would first like to express my gratitude to my project advisor, Dr. Mark Stamp who constantly guided and supported me through the entire project. He always listened to me patiently and came up with suggestions for making the project what it is today. He constantly guided me through all the issues which I had during the course of the project. He was also very helpful in getting the dataset required for the experiments.

A special thanks to my committee member Dr. Thomas Austin for his valuable inputs and suggestions.

I would also like to thank my committee member Dr. Philip Heller for his time and guidance.

A very special thanks to Professor Fabio Di Troia for providing me the dataset for the experiments and also helping me with the script for the extraction of opcodes.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malware is software that is developed for the purpose of causing damage to a computer system, client, server or computer network, without the consent of the user [9]. The damage is done by the malware once it reaches the target's computer. The malware can take the form of scripts, executable codes and so on. The malware perform tasks with purpose of stealing important information from the target computer or it could spread from one file to the other and infect the entire computer [10]. It can also ask users critical information, including financial specifics, or even crash the hard disk. Malware not only focuses on a narrow scope but includes worms, ransomware, trojan horses, spyware and computer viruses. An important step in protecting the users and preventing any malicious activities from happening is to recognize the malware and eliminate it.

In May of 2017, a worldwide cyber attack took place known as the WannaCry ransomware attack. It was caused by the WannaCry ransomware cryptoworm [11]. The main victims of this attack were computers running the Microsoft Windows operating system. It attacked the target system by encrypting the data and in turn demanded a huge amount of payment in the form of bitcoins from its user. The attack circulated via EternalBlue, which was an exploit, a piece of software which took advantage of a bug to cause unanticipated behavior of computer software, created by the US National Security Agency (NSA) for systems using older versions of Windows. This was released by The Shadow Brokers, a hacker group, some months prior to the attack [11]. The impact of the attack was huge with its major victim being the National Health Service hospitals in Scotland and England as nearly 70,000 devices, comprising of MRI scanners, refrigerators storing blood and computers were affected.

Another computer worm which targeted users was the multi-platform worm by the name Koobface. It targets users of social networking websites such as Twitter and Facebook. Although it does not gather any important financial data, it gathers login information for these websites. The Koobface worm actively infects the PC and then it transmits itself intentionally via social networking sites. It uses the internet cookies in order to track what sites a user is a member of [12]. Once known, it accesses the users social network account and sends messages pertaining to Koobface on the website. Its threat consists of several component malware files which work in union to create and sustain the Koobface botnet [12]. Each unit performs a specific task to help the Koobface botnet function. Such a model enables Koobface to modernize its existing components on an already affected system, stop improving non-working modules or add new ones.

Due to the examples presented above, we know that malware pose a significant threat and it is growing every day at a very fast speed. New methods to escape the detection of malware are becoming the prime focus of malware developers. The creation of malware has become a very simple task these days because of the abundance of malware development kits and generation of viruses using metamorphic generators. Thus, different ways of analysis of malware is an important research area with a focus on reducing the problems caused by malware.

Malware samples belonging to a specific family will exhibit similar characteristics and behavior. The standard method of detecting malware relies on signatures, which are typically sequences of bits found in a specific malware sample. However, such an approach is ineffective against advanced forms of malware, or zero-day malware, i.e., malware that has not been previously detected, and for which no signature currently exists. And, in any case, signature detection is inefficient, due to the vast number of malware samples in existence. Hence, there is considerable recent research focused on trying to find efficient and effective means of detecting all samples belonging to a particular malware family. Machine learning techniques have proven extremely useful in this research. For example, classification based on hidden Markov model (HMM) scores [13] has shown to be useful for classifying malware samples into their respective families.

In this research, we will analyze a class of machine learning models to determine whether they can be used to directly provide information about the underlying malware families, as opposed to simply classifying samples based on scores generated by the models [14]. Specifically, we will train an HMM for each family in a large and diverse malware dataset. Then we will classify the resulting models and analyze these to determine whether the groupings represent families with common characteristics. We will also extend this analysis to the context of zero-day malware, i.e., we will train HMMs from new malware samples and try to predict the malware family to which they most closely match. If successful, this technique will greatly speed up the process of analyzing new malware families.

The report is structured in the following manner: Chapter 2 gives an overview of malware and the various types of malware, along with their detection techniques. It also gives an overview about the previous work done in the detection of malware. A brief introduction about Convolutional Neural Networks (CNN) and Support Vector Machine (SVM) is also given in this chapter. Chapter 3 talks about the implementation and the steps followed to complete this project. Chapter 4 highlights the experiments performed and discusses what the results imply. Chapter 5 gives the conclusion and provides pointers towards related future work.

CHAPTER 2

Background

2.1 Malware

As mentioned in Chapter 1, malware is a software developed with the intent of performing malicious activities on a computer. The malware cause disruption of the normal functioning of the target's computer. It may be because of some code or script which may be written with the purpose of stealing confidential information or to cause the hard drive of the target to crash. Malware include virus, worms, adware, trojan horse and spyware.

Figure 1 depicts at what rates are malware increasing day by day. The x axis depicts the year in which the number of malware is recorded and the y axis depicts the number of malware generated in that particular year. As we can see in the figure, the number of malware increased by nearly 110 million from 2017 to 2018.

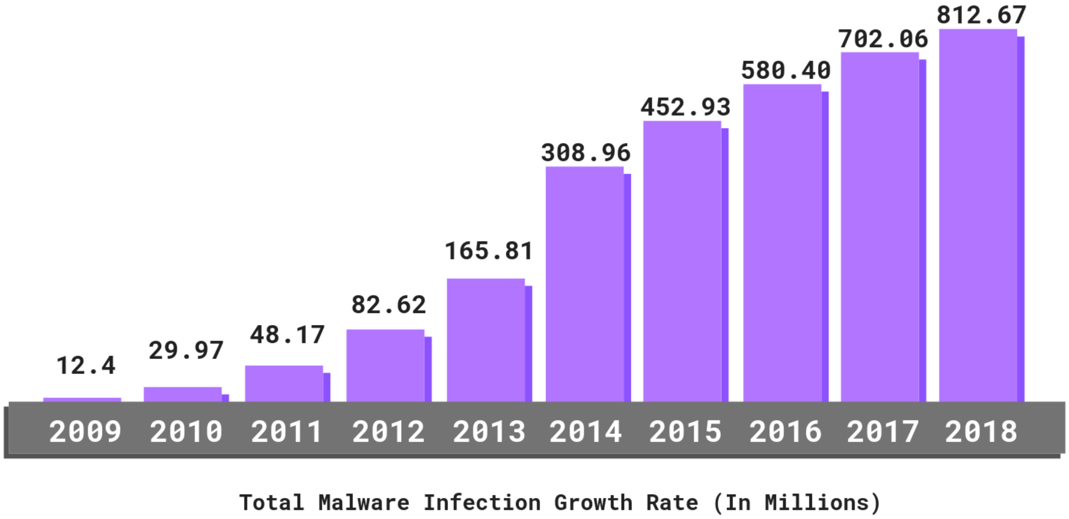


Figure 1: Growth Rate of Malware [3]

2.1.1 Virus

A virus is one of the most typical and commonly found type of malware. It spreads by executing itself and infecting other files on the target's computer. The virus spreads by duplicating, i.e., it joins itself to various other executables to spread the infection. It can reside anywhere and hide itself in hidden folders, files containing data or even the boot sector [9]. A virus can prove to be very harmful when it resides on the boot sector as it becomes activated at the time of the target's computer starting. Due to this, all antivirus starting after the system boots, may prove to be ineffective. Also, since it starts even before any operating system security is enabled, it can prove to be more harmful [15]. The virus can also block the antivirus from running on the system. Other than living in the boot sector, it can also live in memory. The writers of various virus keep updating their software in order to escape the various techniques for virus detection. Some examples of computer viruses are Concept virus, Melissa virus, Stuxnet and Pathogen [16].

2.1.1.1 Encrypted Virus

In most cases, antivirus software identify virus by looking up signatures of the virus. A signature is a unique string of binary patterns or bits of a virus. The signature of a virus is like a fingerprint which can be used in identifying and detecting a particular virus. An approach to mask the virus signature is to inscribe it with various other encryption keys [17]. Even though the encrypted virus makes the virus difficult to detect, it is still possible to detect the encrypted signature. This technique attempts to dodge the signature detection used by antivirus, but it is not a guaranteed way. Even though there are so many changes to the virus, the decryptor loop is the part that remains constant. Since the decryptor loop remains the same, the antivirus exploits this fact for the detection of the virus [9]. Hence the next step for the virus writers is to change this constant part so that they can evade the signature based detection of the antivirus.

2.1.1.2 Polymorphic Virus

As mentioned in Section 2.1.1.1, the antivirus software exploit the fact that the decryptor loop remains the same. Hence polymorphism solves the issue of non-decrypted decryptor loop. In each phase, the decryptor loop is morphed [9]. A polymorphic virus has a never ending number of variations of the decryptor loop. Detecting polymorphic virus is very difficult since it is not possible to try all combinations of the decryptor loop. Hence techniques such as emulation could be used for the detection of polymorphic virus [18]. An example of a polymorphic virus is Tremor, which has nearly as high as six billion combinations of decryptor loops [19].

2.1.1.3 Metamorphic Virus

In the case of a metamorphic virus, the virus writers try to make the sample not detectable by the antivirus, by changing the appearance of the virus before it affects any system. Due to even more changes to the virus, it becomes difficult to detect this type. The morphed virus has a very different structure as compared to the original virus.

2.1.2 Trojan

A Trojan is a malware which which performs a malicious task in the background, but it appears to do an approved task to the user. A virus may appear similar to a trojan horse but the major difference between the two is that Trojan does not duplicate itself [14]. They appear as authentic programs that users may use mistaking them for genuine programs. But once the user mistakens them for genuine programs and executes it, the malware file performs unapproved activities in the background and installs programs which can be harmful. These types of malware have the ability to destroy information on the hard drive, delete files or open a backdoor to the security systems. Examples of trojan horse are Rootkit, Backdoor, Exploit, Trojan-dropper and Netbus [20].

2.1.3 Worm

Another type of malware is the worm, which is very similar to virus as they also reproduce themselves for spreading to other computers. The factor distinguishing virus from worm is that unlike virus, worm have no requirement of a host file whereas virus require a host file which is infected to spread [21]. Often worms are referred to as standalone programs as they have no dependency on any other host file, unlike virus, which need an infected host file to spread. The most common way by which the worm spreads is on a network. It affects systems in the network on its way and exhausts the network which in turn causes it to collapse. Examples of major virus are Sobig, Storm worm, Code Red and Morris worm [22].

2.1.4 Spyware

It is a type of malware which once installed in the targets system can collect the credentials of the user. This activity happens with the user not knowing. The malware dissembles itself and, in the process, collects personal information, details about the users credit card, data related to keystrokes and patterns related to the users browsing [23]. Spywares do not spread by reproducing themselves like virus or worms. Examples of spyware malware are BlazeFind, CoolWebSearch, Zwangi and HuntBar [24].

2.2 Malware Detection Techniques

Malware have been known to mankind since a very long time and antivirus focus on static detection techniques for the detection of malware. In static detection techniques, the antivirus extracts fascinating features from the file, but it does not execute the file. Dynamic analysis runs the file in a sandbox because of which it extracts the actual behavior of the file. These days malware writers are improving their software in order to evade the standard signature detection techniques of antivirus. Similarly, antivirus companies are also revamping their techniques for detection. This section focuses on some of the static methods of malware detection which include signature based detection, anomaly based detection and Hidden Markov Model based detection.

2.2.1 Signature Based Detection

This is the most common and popular detection technique in antivirus software. It is very common because the detection is simple and precise. It is a form of pattern matching and is used to identify known malware. In this, the antivirus scanner looks through each sample and searches for a particular pattern of bits. It depends on looking out for a sequence of bits that in particular identifies any specific malware [25]. It is an approach which considers the attack patterns as signatures and further compares the signatures of known attacks to incoming attacks for detection. All the antivirus programs have a database of signatures and this database is updated for new signatures on a regular basis. The antivirus looks for a signature matching that in the database and once the signature is found, that particular file is marked as a malware.

This detection technique works well but only on malware which are already known. Because of this, a new signature cannot be detected. Known malware which are morphed also cannot be detected as the morphed version of that malware would not have been stored in the database. Also, the database holding the malware signatures should be updated regularly as this would determine the accuracy of the antivirus. A very easy way to dodge the signature based detection of malware is to use techniques for code obfuscation [26].

2.2.2 Anomaly Based Detection

Signature based detection techniques work well with known malware but the problem of detecting unknown malware can be solved using anomaly based detection. It is formed on the structure of the contents of the file [27]. The attributes are taken to be the organization details of the file. In order to detect anomalous behavior heuristic methods are implemented. It takes place in two phases: the training and detection phase. In the first phase, the training phase, the model is trained with normal behavior. Other than normal behavior, any behavior is considered malicious behavior. The structural organization of the file or attributes are used as features which are given to a machine learning algorithm to classify file structures into normal or anomalous [28].

The main disadvantage of anomaly detection is knowing what is unusual and what is normal. This technique could have many false positives and false negatives depending on our definition of normal and abnormal behavior. Once a file is recognized as malware, it is assessed and reviewed thoroughly in order to reduce the number of false positives by determining if the file is actually malicious [27]. Anomaly detection is usually combined with other detection techniques like signature based techniques, in order to reduce the number of false positives [25].

2.2.3 Hidden Markov Based Detection

Hidden Markov Models (HMM) can prove to be a functional tool for the detection of malware [19]. It is usually used for statistical pattern analysis. HMMs can not only be applied to detection of malware, but also in recognition of human activity [29], prediction of genes or speech recognition [30]. In the case of detection of malware from benign samples, we train HMM models [31] and decide a threshold value of the score which separates the malware samples from the benign ones. This section covers a short overview of HMMs and the problems which the HMM can solve. It also outlines the solutions for the problems.

2.2.3.1 Introduction to HMM

HMM is a statistical model which has various states and transition probabilities from and to various states. This is known as a Markov model [1]. In a Markov model, the states and transitions are known to the user. Unlike Markov models, a Hidden Markov Model has states which are not observable directly to the user. Hence it has been coined with the name of Hidden Markov Model. HMM is a machine learning technique and acts as a state machine. These can be used in protein modeling and to know about the various types of software piracy detection [32]. Each state has a probability distribution for the observation of a set of observation symbols. Each transition between the various states have the same probabilities. In order to represent a set of data, we can train the HMM using the observation sequence. If we want to determine the probability of seeing such a sequence, we can match the observation sequence against a trained HMM. When we get a high probability for the same, it means that the observation sequence is very similar to the training sequences. Table 1 below highlights the standard notations used for HMM.

In order to represent a Hidden Markov Model, we need the A, B and π matrices. Each of these matrices are row stochastic in nature. A matrix is said to be row stochastic if the elements in each row satisfy the requirements of a discrete probability distribution, that is, each element is between 0 and 1, and each row sums to 1. An HMM is denoted as

$$\lambda = (A, B, \pi).$$

Figure 2 gives a generic overview of the Hidden Markov Model.

We can solve the following three major problem using HMMs.

Table 1: Standard HMM Notation [1]

Notation	Description
T	Length of the observation sequence
N	Number of states in the model
M	Number of distinct observation symbols
Q	Distinct states of the Markov Model
V	Set of possible observations
A	State transition probability matrix, $N \times N$
B	Observation probability matrix, $N \times M$
π	Initial state distribution, $1 \times N$
O	Observation sequence

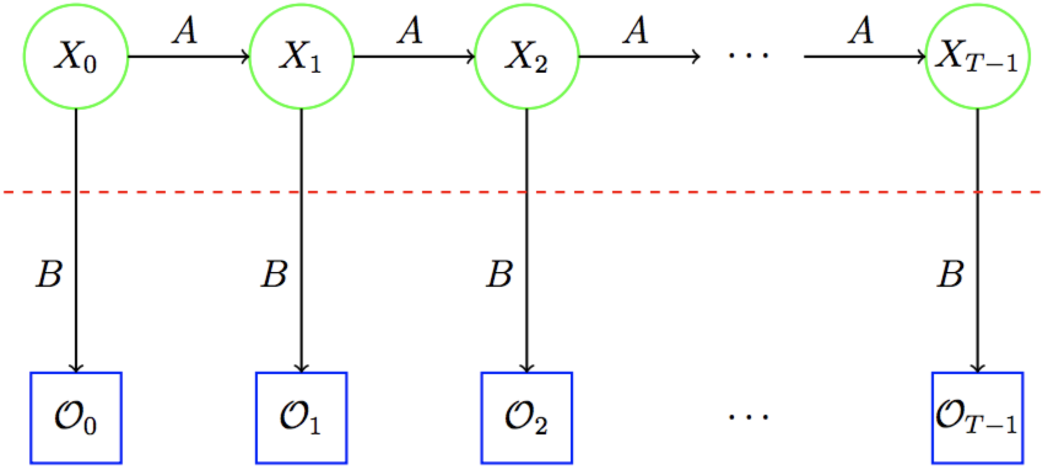


Figure 2: Generic View of Hidden Markov Model [1]

Problem 1: Given a model

$$\lambda = (A, B, \pi)$$

and an observation sequence O, we can find $P(O|\lambda)$. In this case, we score an observation sequence to see how well it fits a given model [1].

Problem 2: We are given a model

$$\lambda = (A, B, \pi)$$

and an observation sequence O, we could determine an optimal state sequence for the Markov model. This means we can find out the most likely hidden state sequence of the model [1].

Problem 3: If we are given O, N, M, we can find a model λ which gives the maximum probability of that particular observation sequence O. This is the training phase of the model to best fit an observation sequence [1].

In this paper, we have trained various HMM models for different malware families, which correspond to the above mentioned problem 3. These trained HMM values, which represent a set of data, give us values of A, B which we use further for the Convolutional Neural Network and Support Vector Machine. Please refer Section 2.4 for more information about Convolutional Neural Networks and Section 2.5 for more information about Support Vector Machine.

The solution to the three HMM problems mentioned above can be solved as follows.

Solution to Problem 1: In this, we find $P(O|\lambda)$ with respect to the model

$$\lambda = (A, B, \pi)$$

which indicates the probability of an observation sequence at a time t in a state q^i. To calculate $P(O|\lambda)$, we use the alpha pass or forward algorithm [1].

$$\alpha_t(i) = P(O_0, O_1, \ldots, O_t, x_t = q_i|\lambda) \tag{1}$$

$$P(O|\lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i) \tag{2}$$

<u>Solution to Problem 2</u>: The solution to problem 2 is achieved by the beta pass or backward algorithm where our aim is to find the most likely state sequence. For $i = 0, 1, \ldots, N-1$ and $t = 0, 1, \ldots, T-1$, we have [1]

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_{T-1} | x_t = q_i, \lambda) \tag{3}$$

<u>Solution to Problem 3</u>: In problem 3, our aim is to train a model such that it best fits the observation sequence. We follow the Baum-Welch algorithm. In this, we start by initializing the A, B and π matrix by random initialization and maintaining the row stochastic nature of the matrix. The Baum-Welch algorithm can be summarized as below [1]

1. Initialize the model
$$\lambda = (A, B, \pi)$$
2. Compute the values of $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i, j)$
3. Reestimate the model
$$\lambda = (A, B, \pi)$$

   as below

   For $i = 0, 1, \ldots, N-1$, let

$$\pi_i = \gamma_0(i) \tag{4}$$

   For $i = 0, 1, \ldots, N-1$ and $j = 0, 1, \ldots, N-1$, calculate

$$a_{i,j} = \sum_{t=0}^{T-2} \gamma_t(i, j) \bigg/ \sum_{t=0}^{T-2} \gamma_t(i) \tag{5}$$

For $j = 0, 1, \ldots, N-1$ and $k = 0, 1, \ldots, M-1$, calculate

$$b_j(k) = \sum_{t \in (0,1,\ldots,T-2), O_t = k} \gamma_t(j) \Bigg/ \sum_{t=0}^{T-2} \gamma_t(j) \tag{6}$$

4. If the value of $P(O|\lambda)$ increases, go to step 2.

2.3 Deep Learning

Deep Learning or Deep Structured Learning or Hierarchical Learning is a subclass of machine learning methods based on learning the representation of data [33]. The learning of data representation can be either unsupervised or supervised. Deep learning finds its applications in the field of speech recognition, image classification, audio recognition and natural language processing. This is a section of machine learning algorithms which use a combination of multiple layers of nonlinear units for processing for the extraction of features and transformation [34]. In this, the input for each layer is the output from the previous layer and the model learns various levels of representation, each corresponding to varied levels of abstraction, hence forming a hierarchy of concepts [34]. In this section, we discuss the basic components of deep learning.

2.3.1 Loss Function

The loss function gives us a measure of the dissimilarity between the prediction given by the algorithm and the actual label. Several cost functions can be used but one of the most simple and commonly used in neural networks is the mean squared error (MSE). The mean squared error can be defined as [4]

$$L(W, b) = \frac{1}{m}\Big(\sum_{i=1}^{m} ||h(x_i) - y_i||^2\Big) \tag{7}$$

where

- m is the number of training examples
- x_i is the i^{th} training sample
- y_i is the class label for the i^{th} training sample
- $h(x_i)$ is the prediction of the algorithm for the i^{th} training sample

The main aim of the training phase is to find the values of weight and bias for which we get a minimum value for the cost or loss function L [4].

2.3.2 Gradient Descent Algorithm

This is an algorithm used to minimize the loss function as discussed in Section 2.3.1. The gradient descent algorithm is as discussed below [4]:

1. In the neural network, we start with an initialization of the weight and bias. The main step is to initialize all the parameters randomly since
    (a) this helps in breaking the symmetry
    (b) if that does not happen, all the parameters would start with similar values and in turn, all the hidden layer units will learn the same function of the input.
2. Iteration is repeated to update the weight and bias until we reach a minimum.

The selection of the correct learning rate (α) is very crucial as it plays an important role to control how big a step should be taken. In case α is very large, it can go beyond the minimum whereas if α is very small, the gradient descent could be slow [4]. Figure 3 demonstrates the effects of various different learning rates on the loss function.

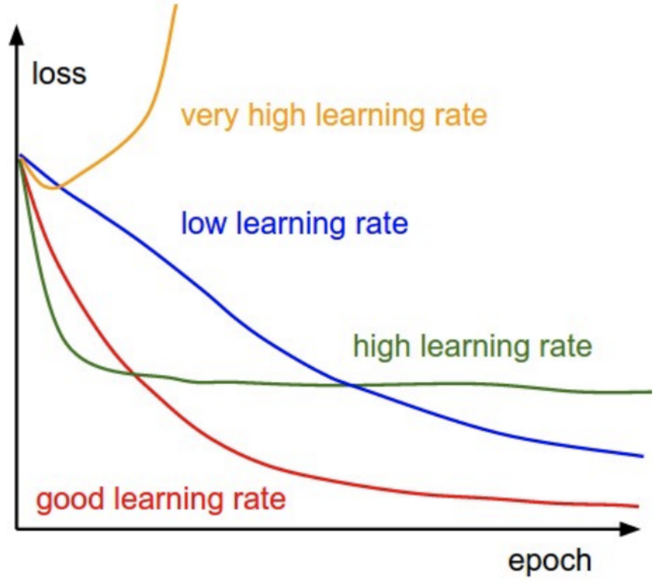


Figure 3: Effects of Different Learning Rates [4]

2.3.3 Backpropagation

This is an algorithm which, with the use of gradient descent, finds its applications in supervised learning of neural networks. The general concept behind backpropagation is that if we have a training sample, it first runs a forward pass in which it calculates all the activation and output values. Once the activation and output values are calculated, an error term is computed for every node i in layer l. This error term tells the contribution of the node i for causing errors in the output. In the case of an output node, we can directly measure the error term by checking the difference between the value of the true target and the value obtained using the network's activation [4].

2.4 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a feed forward neural network. CNNs are made up of neurons and they have weights and bias which are learnt in the training phase. It is currently the latest neural network architecture for the problem of classification of images.

In the case of CNN for image classification, it takes an image as the input, it assigns learnable weights and biases to various objects in the image in order to differentiate images. The pre-processing required in order to classify images using CNN is very low in comparison to the various other algorithms for classification. In earlier techniques, the filters were engineered by hand, but now CNNs have the capability to learn these filters, if the model is trained well [35].

The various versions of the implementation of CNN can be generally described by the below process [4]:

1. Consider the input image and convolve various small filters
2. Subsample this space
3. Repeat steps 1 and 2 till we have enough high level features
4. To the high level features, apply a standard feed forward neural network

CNNs are composed of three main components [35]:

1. Convolutional layer
2. Pooling layer
3. Fully connected layer

Figure 4 shows the basic architecture for the classification of images. The Section 2.4.1 - 2.4.3 explains the three main components of CNN.

2.4.1 Convolutional Layer

This is the most important part of the CNN. It consists of a collection of kernels which can be learnt and are convolved during the forward pass. This convolution is done across the height and width of the input features, which in turn produces a

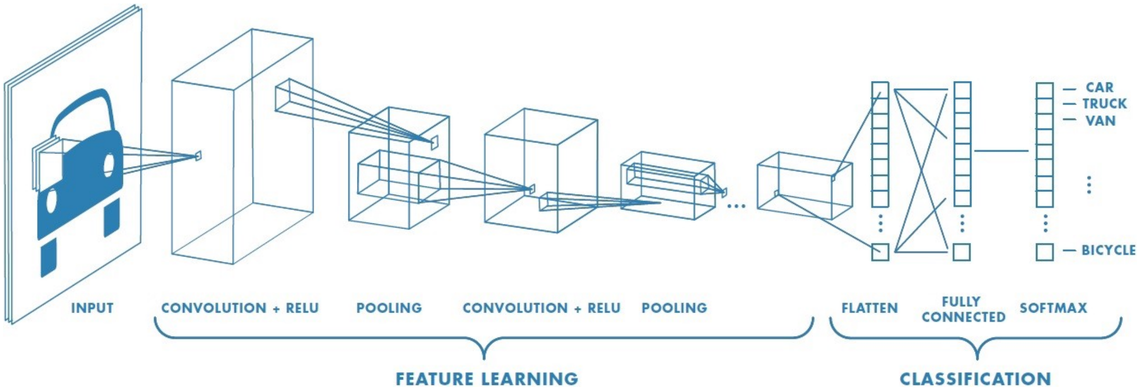


Figure 4: CNN Architecture [5]

2-dimensional map of the kernel [4]. In short, a kernel is a set of a layer of weights in which the size of the input is that of a small 2D patch while the output is in the form of a single unit [4]. The convolution layers apply convolution operation to the input and then pass the result to the next layer. The main aim of this layer is to learn the representation of the input in the form of features.

This consists of various feature maps. Within the same feature map, every neuron is used to get local characteristics of points in the previous layer, whereas for single neurons, the extraction is local characteristics of same points in previous different feature maps. To get a new feature, we take a learned kernel and convolve it with the input feature maps, after which the results are passed to an activation function, which is generally non-linear [36]. There are various kernels applying which we get different feature maps [37]. The activation functions generally used are Tanh, Sigmoid and Relu.

Figure 5 shows how convolution works. We take into account an image of size 5×5 pixels, as shown in the figure, where 255 represents white and 0 represents black. Towards the center of the figure, a kernel has been defined of size 3×3 pixels, where except one white set, all the others are 0. In this case, we get the output by calculating the kernel for every possible position in the image.

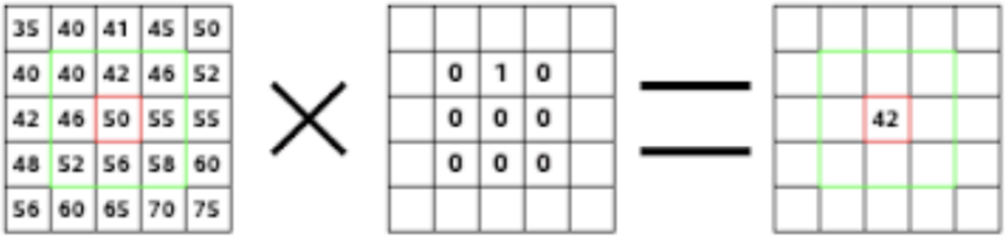


Figure 5: Convolution [4]

The stride determines if the kernel is convolved through all places. For example, if the stride is 1, the output is the normal convolution, whereas if the stride is 2 the number of convolutions reduce by half. The size of the output image after a kernel of size Z convolves over an image N with stride S is given as [4]

$$\text{output} = \frac{N - Z}{S} + 1 \tag{8}$$

2.4.2 Pooling Layer

This layer is helpful for down-sampling the data non-linearly, which is produced from the convolution layers, i.e., decreasing the spatial resolution of the input layers [35]. Due to this, the time required for processing is reduced and also the scale of the data can be handled by the computational resource. Pooling can be implemented by various several non-linear functions, for example, maximum, minimum and average. The most common pooling function is the maximum [4]. In max pooling, the image is partitioned into several smaller rectangles, which do not occupy the same area. Then for each small rectangular region the maximum value is presented as the output. Figure 6 represents the concept of max pooling.

2.4.3 Fully Connected Layer

Generally, the classifier of CNN consists of one or several fully connected layers. In this, the spatial information is not preserved. This performs classification based

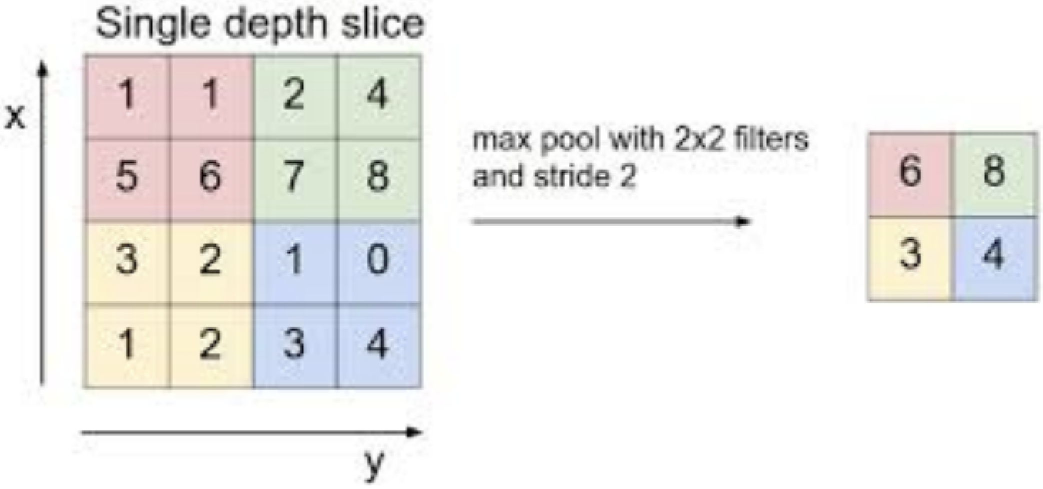


Figure 6: Max Pooling [4]

on the output which is a result of various convolution and pooling layers. The last such layer results in an output layer [35]. In order to classify the images, the flattened matrix is passed through a fully connected layer. For many cases of classification, a well performed probability distribution of the outputs is generated in the case of softmax regression.

2.4.4 Softmax

The softmax function finds the probability distribution of the event over n various events. Generally speaking, the function will calculate the probabilities of every target class over all the possible target classes. After that, the probabilities which are calculated would be helpful for finding out the target class for the given inputs [4].

2.5 Support Vector Machine (SVM)

One of the most popular and commonly used algorithm for classification problems is SVM. It is also used for regression analysis. Since classification is a form of supervised learning, we must assign labels to the dataset. In this, the algorithm aims to find an optimal hyperplane in such a way that the hyperplane acts as a dividing parameter for the class labels, i.e., every data point on one side of the hyperplane belongs to one class [38].

Figure 7 shows an optimal hyperplane. We can see from the figure the main aim of the hyperplane is to maximize the margin, which means the distance between the nearest data points of the different classes should be maximum.

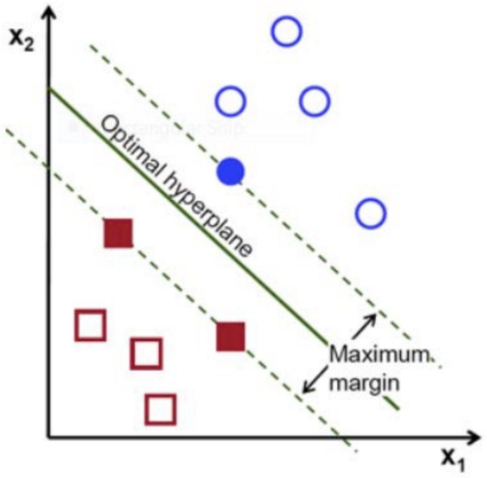


Figure 7: Example of Optimal Hyperplane [6]

There are many kernels which can be used in SVM. The most commonly used kernel is the linear kernel, but in cases where the training data is not easily linearly separable, we have to use kernels like Radial Basis Function (RBF) or Gaussian. Using the kernel, we can transform the data into a higher dimensional space [25]. Figure 8 shows how the training set is transformed into a higher dimensional space.

2.6 *k*-Means Clustering

k-Means Clustering is a form of unsupervised learning algorithm, where the data is not labelled and the technique aims to find patterns in the data. The aim of this algorithm is to group data having similar features. In the end, each cluster will represent groups of similar data. The term *k* refers to the number of clusters or groups

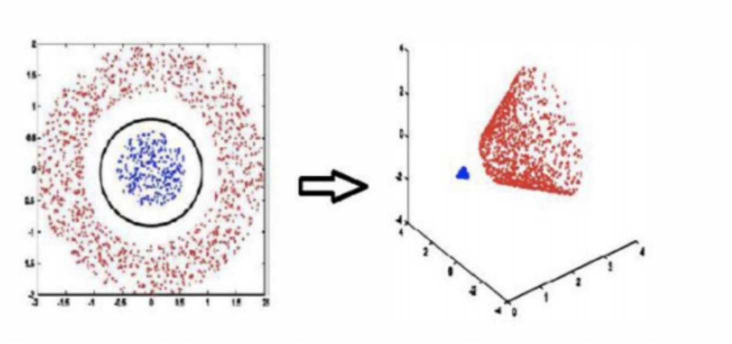


Figure 8: Mapping Data to Higher Dimensionality [7]

in which the data has to be divided. A point is considered in any particular cluster when the distance between that point and the cluster's centroid is lesser than any other centroid. Figure 9 shows how the data points are clustered into 3 different clusters.

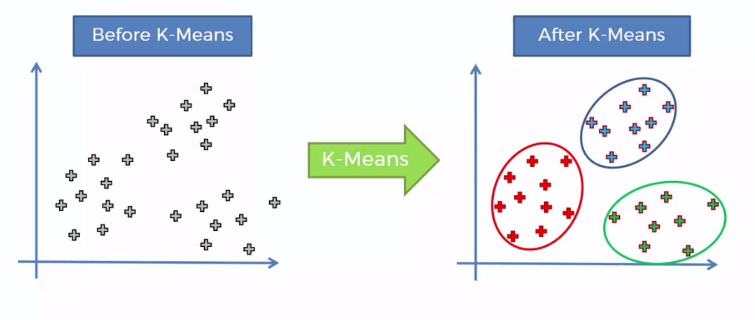


Figure 9: k-Means Clustering [8]

2.7 Related Work

Many people have worked on classifying samples into malware or benign. Research is also done to classify a sample as binary or malware using HMMs [39]. Also, some research is done on clustering malware families using HMM. This section gives an overview of the work previously done in this domain.

2.7.1 Behavior Based Malware Detection

Many antiviruses used today identify malware based on the signature of the malware [40]. But such a method proves less helpful as the detection happens after the malware has caused the damage and also the malware signature is stored in the database maintaining the known malware signature [19]. Most methods deal with static features for malware detection but in this case in order to detect the malware, the dynamic behavior of the files is observed. This dynamic behavior can be observed by running the executable in a virtual environment. The virtual environment permits the user to follow the program execution step by step. The approach used in [41], uses machine learning techniques on the features extracted from the dynamic analysis of the malware samples. Since it is a machine learning based approach, the research in [41] has 2 phases, the training phase and the testing phase. The training phase comprises of the data samples being monitored for behavioral patterns. In order to observe the behavioral patterns, the samples (both benign and malware) are sent via an automatic dynamic analysis tool [41] that executes the samples in a virtual environment.

Once the file is run through the virtual environment, a report showing the behavior of the input file is generated which are then further processed for the selection of features. In this step, the most important characteristics are selected which are then stored as vectors. Next in this research [41], machine learning techniques are applied to these feature vectors for the classification of malware samples from benign. This research [41] presents a comparison of the five different classification algorithms such as Naive Baye's, k-Nearest Neighbours, Decision Tress, SVM and Multilayer Perceptron neural network and compares the performance of these five classifiers.

2.7.2 Malware Classification Using Structured Control Flow

Many techniques have been used for malware classification and in the research paper [42] the authors have proposed an approach using control flow. The control flow depicts the flow of the program, i.e., the execution path which the program will follow. Control information appears in two forms. The call graph shows the inter-procedural flow of control while the intra-procedural control flow is depicted as a series of control flow graphs which have a single graph per procedure [42].

The research paper in [42] shows that the malware can be characterized by the control flow and the authors have suggested a classification system of malware which uses approximate matching of the control flow graphs [42]. The calculation of the string edit distances can be done by the edit distance between the structured graphs of the malware in the database and the control flow signatures [42]. A threshold is decided by the author, above which a sample can be labeled as a malware, otherwise it is a benign. In cases of metamorphic malware, the control flow does not change much. Hence using the research in [42], it is possible to identify variations of the malware, i.e., in cases of metamorphic malware.

2.7.3 Hidden Markov Model Based Malware Families Detection

Attempts have been made in previous researches to cluster similar malware families based on HMM results. The approach used in [19] is that HMM models for different malware families were run for different compilers and then these scores were used as dimensions for clustering the malware families. In this, HMM models were created for 7 cases which included 4 compilers, hand-written assembly, virus construction kit and for metamorphic code [19]. The author performed 5-fold cross validation and created models for N varying from 2 to 6. Every malware sample is scored against each of the trained models in the 7 cases and then the author considered the average score for all the models is taken as the final score for that case [19]. Hence each sample is depicted as a 7-tuple score which are provided to the k-means clustering algorithm.

2.7.4 Malware Classification Using Images

Many techniques have been applied for the purpose of classification of malware. People have tried to show the malware as an image in order to classify them. In the research paper in [43], the authors have tried to follow a similar approach. They have converted the malware samples into images and then have applied classification algorithms to classify the samples to various families.

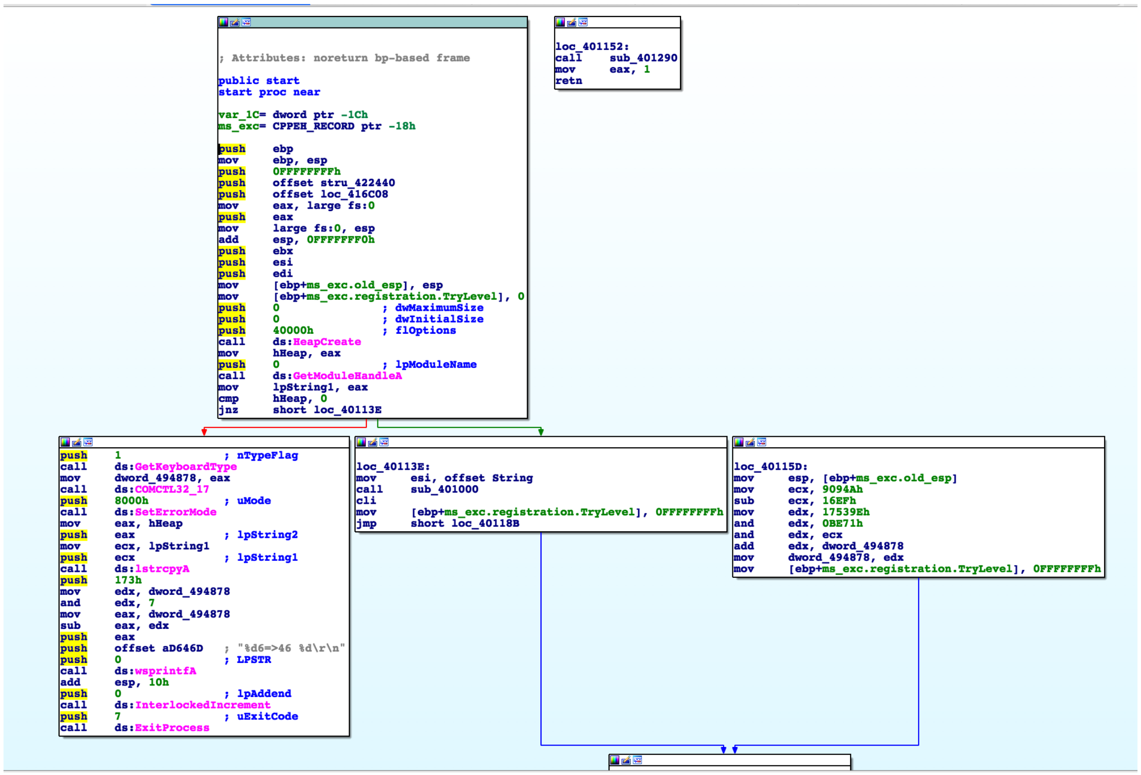
The idea behind this research is that it is possible to depict a malware executable as a binary string of zeros and ones and the resulting vector can be treated as a matrix and viewed as an image [43]. The malware binary is read as a vector of 8-bit unsigned integers and later organized into a 2D array which can be then visualized as a gray scale image [43]. The size of image can vary based on the file size over a range of file sizes. The malware images are distinguished based on the texture of the image, and the primary feature used to analyze textures is GIST descriptors [44] [45]. Then these previous authors apply k-Nearest Neighbors (kNN) classifier and performed various experiments for varying values of k. This research focuses on similar experiments for malware families.

2.8 Tools

This section talks about the tools used for extracting opcode sequences from malware files. The first subsection talks about IDA and the second subsection talks about objdump.

2.8.1 IDA

IDA stands for Interactive Disassembler which is a disassembler which converts machine executable code to assembly language source code. IDA does code analysis automatically. It also uses information about API calls and refers between sections of code [46]. The formats supported by IDA include a variety of executable formats across various platforms and processors. The output is represented in the form of a flow graph or text file which outlines the basic structure of the code. Figure 10 shows how a disassembled file looks like in IDA. This snapshot is for a sample of the Zbot family.



```
; Attributes: noreturn bp-based frame

public start
start proc near

var_1C= dword ptr -1Ch
ms_exc= CPPEH_RECORD ptr -18h

push    ebp
mov     ebp, esp
push    0FFFFFFFFh
push    offset stru_422440
push    offset loc_416C08
mov     eax, large fs:0
push    eax
mov     large fs:0, esp
add     esp, 0FFFFFFF0h
push    ebx
push    esi
push    edi
mov     [ebp+ms_exc.old_esp], esp
mov     [ebp+ms_exc.registration.TryLevel], 0
push    0               ; dwMaximumSize
push    0               ; dwInitialSize
push    40000h          ; flOptions
call    ds:HeapCreate
mov     hHeap, eax
push    0               ; lpModuleName
call    ds:GetModuleHandleA
mov     lpString1, eax
cmp     hHeap, 0
jnz     short loc_40113E
```

```
loc_401152:
call    sub_401290
mov     eax, 1
retn
```

```
push    1               ; nTypeFlag
call    ds:GetKeyboardType
mov     dword_494878, eax
call    ds:COMCTL32_17
push    8000h           ; uMode
call    ds:SetErrorMode
mov     eax, hHeap
push    eax             ; lpString2
mov     ecx, lpString1
push    ecx             ; lpString1
call    ds:lstrcpyA
push    173h
mov     edx, dword_494878
and     edx, 7
mov     eax, dword_494878
sub     eax, edx
push    eax
push    offset aD646D   ; "%d6=>46 %d\r\n"
push    0               ; LPSTR
call    ds:wsprintfA
add     esp, 10h
push    0               ; lpAddend
call    ds:InterlockedIncrement
push    7               ; uExitCode
call    ds:ExitProcess
```

```
loc_40113E:
mov     esi, offset String
call    sub_401000
cli
mov     [ebp+ms_exc.registration.TryLevel], 0FFFFFFFFh
jmp     short loc_40118B
```

```
loc_40115D:
mov     esp, [ebp+ms_exc.old_esp]
mov     ecx, 9094Ah
sub     ecx, 16EFh
mov     edx, 17539Eh
and     edx, 0BE71h
and     edx, ecx
add     edx, dword_494878
mov     dword_494878, edx
mov     [ebp+ms_exc.registration.TryLevel], 0FFFFFFFFh
```

Figure 10: IDA

2.8.2 Objdump

This is a command line program that displays information about object files on operating systems such as Linux. In our case, it is used to disassemble an executable file and view it assembly form. It is a part of the GNU Binutils [2]. The command used for disassembling a file is

```
objdump -Mintel -D <file_name>
```

The other possible operations of the objdump command are listed in Table 2 [2].

Argument	Description
a	archive-headers
b bfdname	target=bfdname
d	disassemble
D	disassemble-all
f	file-headers
h	section-headers
i	info
section	section
l	line-numbers
m	machine
r	reloc
R	dynamic-reloc
s	full-contents
t	syms
T	dynamic-syms

Table 2: Possible Operations of Objdump [2]

CHAPTER 3

Implementation

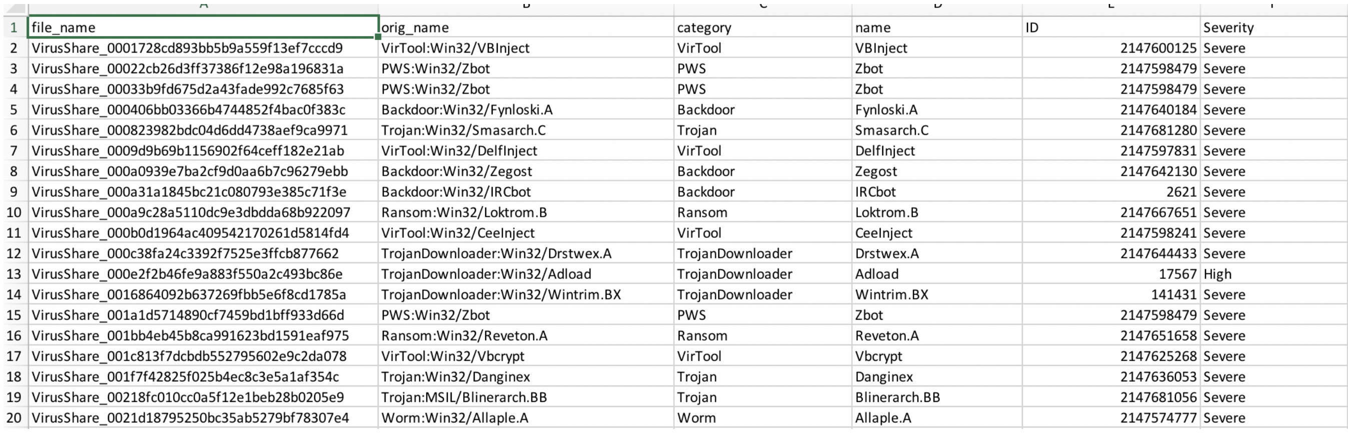
This chapter covers the details of the implementation of the project. The first section, Section 3.1 gives an overview of the dataset. Next, the Section 3.2 gives a detailed description about the extraction of features and generating an observation sequence to feed the model. The next section, Section 3.3 gives a description of training the HMMs. The last section, Section 3.4 gives a detailed explanation about how the HMM values are classified, along with the specifications and details about the methods.

3.1 Dataset

This project requires a huge number of malware samples since each HMM model is trained on an observation sequence of 50,000 symbols. The dataset [47] that I used for experimentation in this project is very large, containing nearly 1TB worth of data. It comprises of nearly 460,000 malware binaries belonging to various malware families. The total number of malware families present in this dataset is 2900. All the malware samples have either the `dll` (dynamic link library) or `exe` (executable) format. The dataset also contains csv (comma separated value) files which hold information about the malware files. For every malware sample, we know what family it belongs to, the category of the malware (whether it is virus or trojan, please refer Section 2) and how severe the attack of that malware could be. Some of the major malware families in the dataset include Zbot, Bancos, Bho, Alureon, Winwebsec, Vundo, Adload, Vobfus and so on. Figure 11 shows a snapshot of the csv file which holds all this data.

Table 3 depicts the count of the samples used for training the HMM. Below are the malware families which we are using for our experimentations

- Zbot belongs to a family of trojans. This malware focuses on stealing financial and confidential information from the target systems. This malware targets the



1	file_name	orig_name	category	name	ID	Severity
2	VirusShare_0001728cd893bb5b9a559f13ef7cccd9	VirTool:Win32/VBInject	VirTool	VBInject	2147600125	Severe
3	VirusShare_00022cb26d3ff37386f12e98a196831a	PWS:Win32/Zbot	PWS	Zbot	2147598479	Severe
4	VirusShare_00033b9fd675d2a43fade992c7685f63	PWS:Win32/Zbot	PWS	Zbot	2147598479	Severe
5	VirusShare_000406bb03366b4744852f4bac0f383c	Backdoor:Win32/Fynloski.A	Backdoor	Fynloski.A	2147640184	Severe
6	VirusShare_000823982bdc04d6dd4738aef9ca9971	Trojan:Win32/Smasarch.C	Trojan	Smasarch.C	2147681280	Severe
7	VirusShare_0009d9b69b1156902f64ceff182e21ab	VirTool:Win32/DelfInject	VirTool	DelfInject	2147597831	Severe
8	VirusShare_000a0939e7ba2cf9d0aa6b7c96279ebb	Backdoor:Win32/Zegost	Backdoor	Zegost	2147642130	Severe
9	VirusShare_000a31a1845bc21c080793e385c71f3e	Backdoor:Win32/IRCbot	Backdoor	IRCbot	2621	Severe
10	VirusShare_000a9c28a5110dc9e3dbdda68b922097	Ransom:Win32/Loktrom.B	Ransom	Loktrom.B	2147667651	Severe
11	VirusShare_000b0d1964ac409542170261d5814fd4	VirTool:Win32/CeeInject	VirTool	CeeInject	2147598241	Severe
12	VirusShare_000c38fa24c3392f7525e3ffcb877662	TrojanDownloader:Win32/Drstwex.A	TrojanDownloader	Drstwex.A	2147644433	Severe
13	VirusShare_000e2f2b46fe9a883f550a2c493bc86e	TrojanDownloader:Win32/Adload	TrojanDownloader	Adload	17567	High
14	VirusShare_0016864092b637269fbb5e6f8cd1785a	TrojanDownloader:Win32/Wintrim.BX	TrojanDownloader	Wintrim.BX	141431	Severe
15	VirusShare_001a1d5714890cf7459bd1bff933d66d	PWS:Win32/Zbot	PWS	Zbot	2147598479	Severe
16	VirusShare_001bb4eb45b8ca991623bd1591eaf975	Ransom:Win32/Reveton.A	Ransom	Reveton.A	2147651658	Severe
17	VirusShare_001c813f7dcbdb552795602e9c2da078	VirTool:Win32/Vbcrypt	VirTool	Vbcrypt	2147625268	Severe
18	VirusShare_001f7f42825f025b4ec8c3e5a1af354c	Trojan:Win32/Danginex	Trojan	Danginex	2147636053	Severe
19	VirusShare_00218fc010cc0a5f12e1beb28b0205e9	Trojan:MSIL/Blinerarch.BB	Trojan	Blinerarch.BB	2147681056	Severe
20	VirusShare_0021d18795250bc35ab5279bf78307e4	Worm:Win32/Allaple.A	Worm	Allaple.A	2147574777	Severe

Figure 11: Snapshot of the Metadata

Windows operating system. This enters by the websites which are hacked or usually via spam emails [48].

- The family Adload also belongs to the family of trojans. It is capable of attacking the target system by opening a backdoor so that it can download and install the potentially unwanted programs or adware. This malware gathers the information of the target system and send them to a remote location [49].

Malware Family	Count
Zbot	740
Adload	304

Table 3: Number of Samples Used for Training the HMM

3.2 Feature Engineering

This section describes the two main steps for the generation of features to give to the HMM model. We have used opcodes as the means to represent our malware samples. The extraction of opcodes serves as one of the key steps in the experiments for this project. The below Section 3.2.1 highlights the important steps and process followed for the extraction of opcodes. In the next Section 3.2.2, we discuss how we have used the opcodes as features for training the Hidden Markov Models, which then we further use for the classification of malware samples.

3.2.1 Opcode Extraction

The initial approach to extract the opcodes was to use the free version of the disassembler IDA. A detailed description about the same is covered in Section 2.8. But the problem with using the free version of IDA was that each malware sample had to be opened manually with IDA, one at a time. The opcodes which were generated would have to be then copied to a text file, after which the later steps could be carried out. But this would take a lot of time since it was manually impossible to do this for 1000 files. Purchasing the full version of IDA was very expensive. Hence, we came up with another solution for the same. We wrote a script which extracted the opcodes for us and generated a text file for the same. The below steps were followed for the extraction of the opcodes.

1. The dataset was divided into folders, each folder having the malware samples of a particular family.

2. A Virtual Machine was setup, as mentioned in Section 4.1, for running the script to extract the opcodes.

3. Transfer the scripts and the samples of a particular family for which the opcodes have to be extracted.

4. On the virtual machine, run the script which would output a shell script.

5. Give access to the script to be executed by any user and run the shell script. The shell script converts the malware samples into opcode sequences using the objdump command (please refer Section 2.8.2). The malware samples are converted into a file with extension opcecho_opcodes.

6. Then run another pythons script which converts the .opcecho_opcodes files to a text file. We now have each of the extracted opcodes in their respective text files.

7. Copy back the text files from the virtual machine to the host machine. Merge all the files into a single file to get a long list of opcodes for that family.

3.2.2 Opcode as Features

We now have the opcodes extracted from the malware exe or dll. Our next step is to convert the opcodes in a way that it can be used as features for the HMM. The approach we have followed to convert the opcodes to the observation sequence is as follows:

1. From all the opcode files, for all families, get the top 25 occurring opcodes.

2. Map the 25 most occurring opcodes to their respective number, as mentioned below, and group the rest of them as Others.

Table 4 shows a list of the top 25 opcodes along with the mappings used to convert them to an observation sequence. Figure 12 is a snapshot of a subset of the Zbot family. Figure 12a shows the opcodes which are extracted from the malware samples and Figure 12b shows the corresponding numeric representation which are used as features for the HMM.

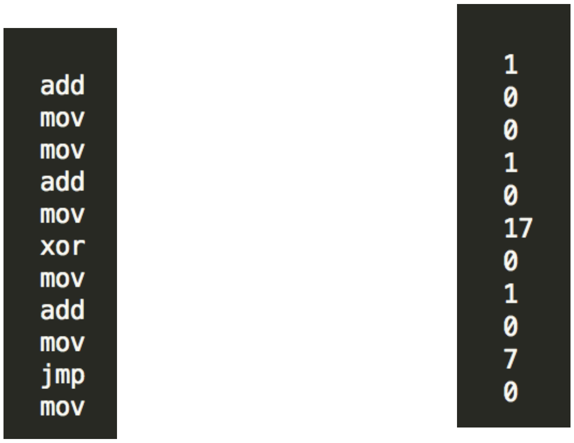
3.3 Training the HMM

Multiple experiments were run for this project. All the experiments follow the same steps, but the observation sequence for each model is changed. Each observation sequence is picked up from various malware families, such as Zbot and Adload. Every HMM is trained on a sequence of opcodes, extracted as described in Section 3.2.

Table 4: Opcode Mapping

MOV	0
ADD	1
SUB	2
PUSH	3
ADC	4
JNZ	5
INC	6
JMP	7
LEA	8
CMP	9
CALL	10
JZ	11
JB	12
OR	13
POP	14
ALIGN	15
JNB	16
XOR	17
JA	18
XCHG	19
AND	20
DEC	21
SHL	22
PUSHA	23
POPA	24
Others	25

We have trained several HMMs for each malware family from the dataset. Every HMM is trained for an observation sequence of length $T = 50,000$ opcode symbols. We have trained with $M = 26$ and the value of N varying from 2 to 26. The number of iterations has been set to 1000 in the cases when $N = 2, 3, 5, 10$. But for $N = 26$, the number of iterations is set to 200. The initialization of the A, B and π matrices is done by random values, maintaining the row stochastic nature of each of the matrices.



```
add
mov
mov
add
mov
xor
mov
add
mov
jmp
mov
```

(a) Extracted Opcodes

```
1
0
0
1
0
17
0
1
0
7
0
```

(b) Corresponding Numeric Representation

Figure 12: Snapshot of the Zbot Family

These set of experiments correspond to the problem 3 as defined in Section 2.2.3.1.

The HMM is trained by the EM algorithm [50] which consists of numerous iterations. Every iteration has one estimate and one maximize step. The entire process is repeated multiple times or when the log likelihood measure becomes constant, indicating that the model has converged. The maximize step aligns each observation vector with a state in the model and a log likelihood value is maximized. The estimate step estimates the parameters of a statistical model for the aligned vectors and the state transition probabilities, for every state. In the next iteration, the maximize step runs again with the updated models.

3.4 Classification of Malware

Once we have the results from training the HMM, we have to find some way to classify the HMM results into different malware families. This is done using the below two methods. Section 3.4.1 covers the classification of HMM results using Convolutional Neural Networks. Section 3.4.2 explains how the classification of the HMM results is done using Support Vector Machine.

3.4.1 Convolutional Neural Networks

In this method, we merge the A and B matrices from the HMM results and convert the merged matrix to an image using the PIL library in Python. In case for $N = 2$, we have the dimensions of the A matrix as 2×2 and the dimensions of the B matrix as 2×26. Hence the dimension of the merged matrix is 2×28, for which we create the image. Figure 13 shows a generated image of the merged matrix for the case $N = 2$. The malware sample belonged to the Zbot family. The A and B matrices from which Figure 13 was generated are

$$A = \begin{bmatrix} 0.5050 & 0.4950 \\ 0.5476 & 0.4524 \end{bmatrix} \quad \text{and} \quad B^T = \begin{bmatrix} 0.1070 & 0.1024 \\ 0.0569 & 0.0658 \\ 0.0230 & 0.0223 \\ 0.0682 & 0.0775 \\ 0.0354 & 0.0350 \\ 0.0000 & 0.0000 \\ 0.0352 & 0.0422 \\ 0.01517 & 0.0167 \\ 0.0041 & 0.0050 \\ 0.0230 & 0.0311 \\ 0.0050 & 0.0328 \\ 0.0000 & 0.0000 \\ 0.0061 & 0.0058 \\ 0.0397 & 0.0410 \\ 0.0435 & 0.0438 \\ 0.0000 & 0.0000 \\ 0.0000 & 0.0000 \\ 0.0309 & 0.0309 \\ 0.0040 & 0.0033 \\ 0.0226 & 0.0224 \\ 0.0293 & 0.0281 \\ 0.0298 & 0.0318 \\ 0.0033 & 0.0022 \\ 0.0078 & 0.0078 \\ 0.0042 & 0.0042 \\ 0.4058 & 0.3477 \end{bmatrix}$$

Figure 14 shows the HMM matrices visualized for the Adload family in the case where $N = 2$ and the dimension of the merged matrix 2×28. Figure 15 shows the

HMM matrices visualized for the Adload family in the case where $N = 26$ and the dimension of the merged matrix 26×52.

Figure 13: Zbot Malware

Figure 14: Adload Malware

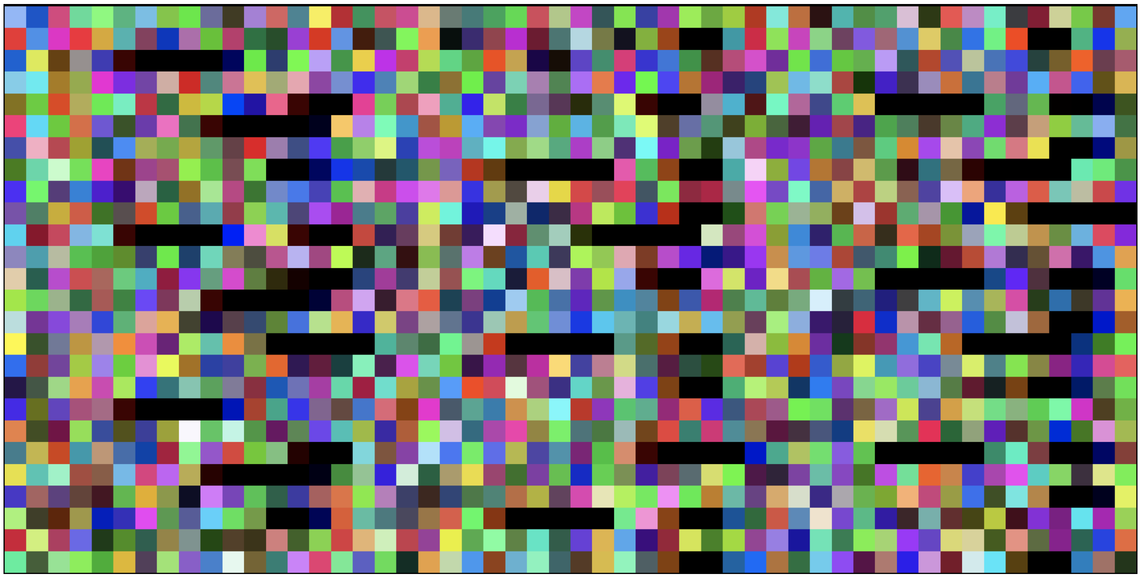


Figure 15: Adload Malware

Below listed are the specifications for the CNN used. The CNN used is from the keras module in python. We have used a sequential 2 dimensional CNN model which performs spatial convolution over images. Below mentioned is the architecture and specification details for the CNN used:

1. Input layer, which has the dimensions based on the images generated by the HMM. If the value of N used for training the HMM is 2, then we use an input

layer of size 2 × 28 whereas when we use the value of $N = 26$ for the HMM, we use an input layer of 26 × 52.

2. Convolutional layer (input_shape/2) filter maps of size 2×2 with the activation function as `relu` (Rectified Linear Unit)

3. Convolutional layer (input_shape/2) filter maps of size 2×2 with the activation function as `relu`)

4. Convolutional layer (input_shape/2) filter maps of size 2×2 with the activation function as `relu`)

5. Convolutional layer (input_shape/2) filter maps of size 2×2 with the activation function as `relu`)

6. Flatten layer which would remove all the dimensions of the matrix except for 1

7. Dense layer with dimensionality of the output space as 30 and activation function `relu`

8. Dense layer with dimensionality of the output space as 1 and activation function `sigmoid`

The number of layers is varied between 2 and 4 for experimentation purpose.

3.4.2 Support Vector Machine (SVM)

After running HMMs for various different families, we get the A, B and π matrix. We merge the A and B matrices from the HMM results and flatten the matrix to produce a single row of values. This is then used as a feature vector for the SVM algorithm. The SVM algorithm is used from the sklearn library in Python with a linear kernel. The SVM algorithm (please refer Section 2.5) will be used to classify the HMM models into different malware families.

In case for $N = 2$, we have the dimensions of the A matrix as 2×2 and the dimensions of the B matrix as 2×26. After merging we get a matrix of size 2×28, which is then flattened to get a matrix of size 1×56 for every HMM.

For example, if A matrix is $$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

and B matrix is $$\begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{126} \\ b_{21} & b_{22} & b_{23} & \dots & b_{226} \end{bmatrix}$$

Once we merge these two matrices and flatten it we get,

$$\begin{bmatrix} a_{11} & a_{12} & b_{11} & \dots & b_{126} & a_{21} & a_{22} & b_{21} & \dots & b_{226} \end{bmatrix}$$

3.5 Clustering of Malware

Once we have the results from training the HMM, we experiment with k-means clustering to cluster the different malware models into their respective families. We have the points for clustering from the A and B matrix obtained by training the several HMMs for each malware family. The feature vector is constructed as described in Section 3.4.2 after which we have experimented for various values of k.

CHAPTER 4

Experiments

In this chapter, we have specified the setup and results of the experiments of the project. The setup section covers the configurations and specifications of the host and guest machine. The second section covers the results and discussions about the results.

4.1 Setup

In this we have set up a virtual machine using Oracle VirtualBox. The virtual machine is used for generating the opcodes from the malware executables. The extraction of opcodes is done on the guest machine. Running the HMMs and classifying the malware families is done on the host machine. Below are the specifications for the guest and the host machine.

Host Machine

Operating System	Mac OS
Model	MacBook Pro
Processor	2GHz Intel Core i5
Software	Version 10.12.6
RAM	8GB

Table 5: Specifications of the Host Machine

Guest Machine

Operating System	Ubuntu 14.04.5 LTS
Software	Oracle VirtualBox version 5.2.8
System Type	64 bit OS

Table 6: Specifications of the Guest Machine

4.2 Results

In this section, we discuss the results obtained and the accuracy for various cases. We also have the Receiver Operating Characteristic (ROC) curve for each of the cases. We have done a 70-30 split to the trained HMM models and then performed the steps as discussed in Section 3.4. The families used in the experiments are Zbot and Adload.

The main purpose of our experiments is to compare the performance of CNN and SVM in distinguishing malware families. Also, the purpose of our experiments is to find the best value of N and the best value of the hyperparameters (for CNN) which give us the maximum value of accuracy in classifying the malware samples to their respective families. We start by training several HMMs for various malware families. We train 1000 HMMs for each malware family for $N = 2$, and 500 HMMs with $N = 3$ and $N = 5$ and, finally, 400 HMMs with $N = 10$ and $N = 26$. Once we have trained the HMMs, based on the values of the A and B matrix, we apply deep learning techniques such as CNN (refer Section 2.4) and machine learning techniques such as SVM (refer Section 2.5), in order to classify the trained HMM models into the various malware families. This is similar to applying machine learning on top of machine learning since we are applying machine learning on the trained HMM matrices, which are also obtained by applying a machine learning technique(HMM).

In order to use the trained HMM values for CNN, we merge the A and B matrix and convert it into an image. Then we apply CNN as an image classifier since we have converted the trained HMM values, which in turn means the malware samples from a family, to images. Next, to apply SVM, we merge the A and B matrices and flatten it, after which we apply SVM for classification of samples into various malware families. Also, ROC curves for the SVM classifier are generated. The region under the ROC curve gives an idea about how good the classifier is [51].

4.2.1 Classification of Malware Trained HMMs using CNN

We experimented with various values of N for the HMM and also by changing the hyperparameters of the CNN model. The specifications of HMM are as given in Table 7. These specifications are for the case of $N = 2, 3, 5, 10$. The specifications of HMM for the case of $N = 26$ are as given in Table 8.

M	26
T	50000
Iterations	1000

Table 7: HMM Specifications for $N = 2, 3, 5, 10$

M	26
T	50000
Iterations	200

Table 8: HMM Specifications for $N = 26$

Table 9 gives a list of the experiments, for the HMM specifications mentioned in Table 7, along with the specifications of the hyperparameters for CNN and the obtained accuracy for each case. Table 10 gives a list of the experiments, for the HMM specifications mentioned in Table 8, along with the specifications of the hyperparameters for CNN and the obtained accuracy for each case.

Figure 16 shows the confusion matrix for the case $N = 3$, where the number of epochs for training is set to 100, the steps per epoch is set to 5 and the number of layers is 4. Figure 17 shows the confusion matrix for the case $N = 5$, where the epochs run is 100, the steps ran per epoch is set to 10 and the number of layers is 4. Figure 18 shows the confusion matrix for the case $N = 10$, where the hyperparameters of the CNN include the number of epochs being set to 10, the steps per epoch set as 5, and the number of layers is set to 4.

Number of images used for training	Number of images used for testing	Number of states in HMM	Number of epochs	Number of layers	Steps per epoch	Accuracy
1400	600	2	10	2	5	0.52
1400	600	2	10	4	5	0.52
1400	600	2	100	2	5	0.67
1400	600	2	100	4	5	0.72
700	300	3	10	2	5	0.62
700	300	3	10	4	5	0.72
700	300	3	100	2	5	0.90
700	300	3	100	4	5	0.90
700	300	5	10	2	5	0.74
700	300	5	10	4	5	0.80
700	300	5	10	4	10	0.78
700	300	5	100	2	5	0.80
700	300	5	100	4	5	0.82
700	300	5	100	4	10	0.82
560	240	10	10	2	5	0.86
560	240	10	10	4	5	0.72
560	240	10	100	2	5	0.83
560	240	10	100	4	5	0.76

Table 9: Results for CNN for 2 families

Number of images used for training	Number of images used for testing	Number of states in HMM	Number of epochs	Number of layers	Steps per epoch	Accuracy
560	240	26	10	2	5	0.55
560	240	26	10	4	5	0.84
560	240	26	100	2	5	0.87
560	240	26	100	4	5	0.91
560	240	26	200	4	5	0.87

Table 10: Results for CNN for 2 families

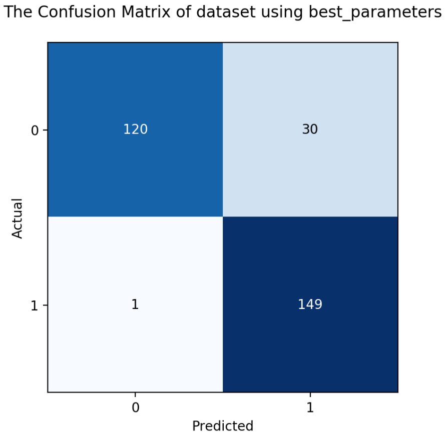


Figure 16: Confusion matrix for $N = 3$

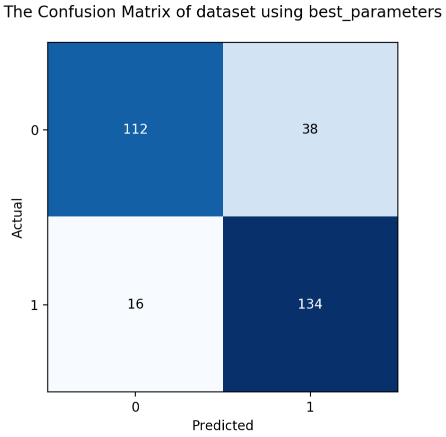


Figure 17: Confusion matrix for $N = 5$

Based on the experiments and the results shown in Tables 9 and 10, we can say that we get the highest accuracy of 0.91, in the case where we train the HMM for $N = 26$ and with the specifications as mentioned in Table 8. The CNN parameters include the number of epochs assigned as 100, the steps per epoch assigned as 5 and the number of layers is fixed at 4. Also, we can say that in the case where we train the HMM for $N = 3$ and with the specifications as mentioned in Table 7, we get almost the same result (the value of accuracy as 0.90) as in the previous case. The CNN parameters include the number of epochs being set to 100, the steps per epoch

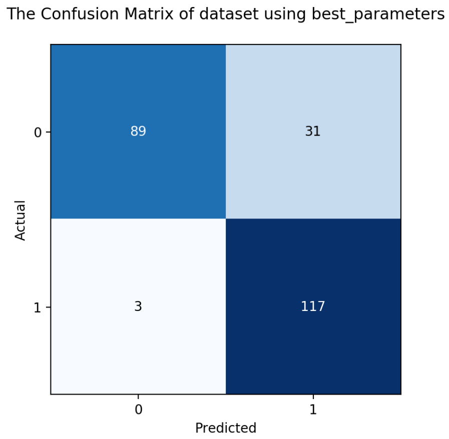


Figure 18: Confusion matrix for $N = 10$

being set at 5 and the number of layers is set at 4.

Computations for the case of $N = 3$ is lesser as compared to the case of $N = 26$. Also, the training for $N = 3$ will take lesser time as compared to the case of $N = 26$. Even though the number of states is significantly larger in the case of $N = 26$, we do not get any significant increase in the accuracy from the $N = 3$ case. Hence we can say that the HMM training of $N = 3$ along with the CNN parameters as number of epochs being set to 100, the steps ran per epoch set as 5 and the number of layers assigned to 4 works best for our case to distinguish between malware families.

In the case of $N = 2$, we see very low values of accuracy ranging from 0.52 to 0.72, even when the number of training samples is 1400 from both the classes. In the case of $N = 3$, the lowest accuracy we see is 0.62. In the case of $N = 5$, we see accuracy value ranging from 0.74 to 0.82, even when the number of training samples is 1400 from both the classes. In the case of $N = 10$, we see better accuracy results as compared to the case of $N = 2, 5$, ranging from 0.72 to 0.86, even when the number of training samples is reduced to 560 from both the classes. In the case of $N = 26$, the lowest accuracy we see is 0.55.

In general, we see that even though for the value of $N = 2$, we train with the maximum number of HMMs we get the lowest accuracy values. The explanation for such low accuracy results could be attributed to the fact that since the dimension of the A and B matrix is 2×2 and 2×26, a lesser essence of the malware family could be captured as compared to when the value of N is increased. Hence in the case for $N = 26$, even with lesser number of HMMs, we get a very good accuracy value.

We extended our research to a total of 10 families. In addition to Adload and Zbot, we included the families Bancos, Bho, Injector, Obfuscator, Renos, Vbinject, Vobfus and Winwebsec. The experimentations for 10 families were carried out with $N = 3$ since that value of N was giving us the best accuracy value when there were 2 families. We trained 500 HMMs per malware family for all 10 families. Table 11 gives a summary of the results obtained when all 10 families were considered for the experiments. As we can see from the Table 11, the accuracy does not change much even when the number of epochs is significantly changed from 1000 to 5000.

Number of epochs	Number of layers	Steps per epoch	Accuracy	Model Training Accuracy
1000	4	5	0.26	0.38
5000	4	5	0.21	0.40

Table 11: Results for CNN for 10 families

4.2.2 Classification of Malware Trained HMMs using SVM

We experimented with various values of N for the HMM and trained different SVM models for each value of N. The specifications of HMM are as given in Table 7. These specifications are for the case of $N = 2, 3, 5, 10$. The specifications of HMM for the case of $N = 26$ are as given in Table 8. After the matrices are merged and flattened, the feature vectors were normalized in order to eliminate any 0 values and the data was pre-processed to label the Adload family as 0 and the Zbot family as 1.

Then we trained the SVM models for different values of N.

Number of HMMs used for training	Number of HMMs used for testing	N	Accuracy
1400	600	2	0.97
700	300	3	0.99
700	300	5	0.96
560	240	10	0.94
560	240	26	0.98

Table 12: Results for SVM for 2 families

As we can see from Table 12, in the case of SVM, the accuracy is best in the case for $N = 3$ (0.99) whereas the least accuracy is observed in the case of $N = 10$. We see that the value of accuracy is highest for the cases of $N = 3$ and $N = 26$. But since the training of HMMs for $N = 26$ would require a huge amount of time, we would prefer the case of $N = 3$.

As in CNN, we extended our research to a total of 10 families. In addition to Adload and Zbot, we included the families Bancos, Bho, Injector, Obfuscator, Renos, Vbinject, Vobfus and Winwebsec. The experimentations for 10 families were carried out with $N = 3$ since that value of N was giving us the best accuracy value when there were 2 families. The accuracy we obtained from increasing the number of families to 10 was 0.61. Figure 19 shows the ROC curves for all the 10 families. We can say that our SVM classifier is able to identify class 0 (Adload) and class 7 (Vobfus) the best whereas it detects class 1 (Bancos) the worst.

We also see that as we kept adding the malware families, the accuracy dropped. Figure 20 shows the drop in accuracy level as the families were added. At each step, the average of the accuracy values is displayed for that particular number of families, chosen randomly.

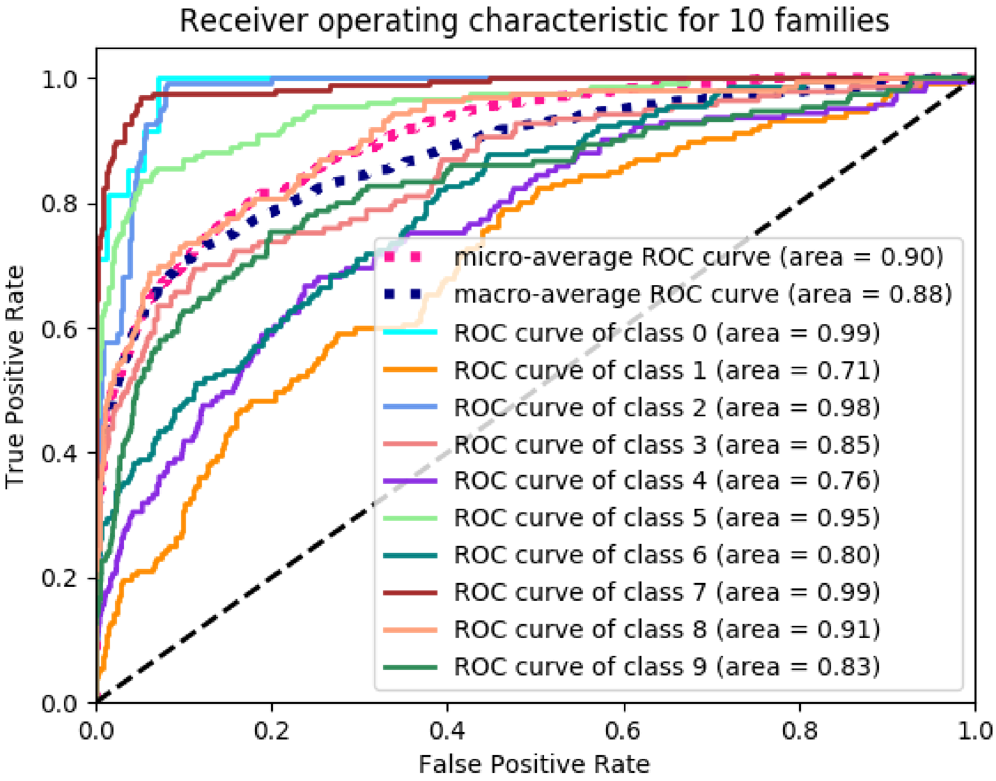


Figure 19: ROC for 10 families

4.2.3 Comparison of CNN and SVM

From the results observed in Tables 9, 10 and 12, we can say that SVM performs better than CNN in our case for the classification of malware into families. Even when the reserach was extended for 10 families, SVM proved to be a better choice as compared to CNN as the accuracy value we obtained from SVM was much higher than what we achieved in CNN. SVM has a maximum accuracy of 0.99 (when $N = 3$) and CNN has a maximum accuracy (optimal) of 0.90 (when $N = 3$), in the case of 2 families. Also, CNN gives an accuracy of 0.91 when $N = 26$, but that involves a lot of time and computation in the training phase. In comparison to CNN and SVM, we could say that SVM proves to be a better technique to classify the trained HMM

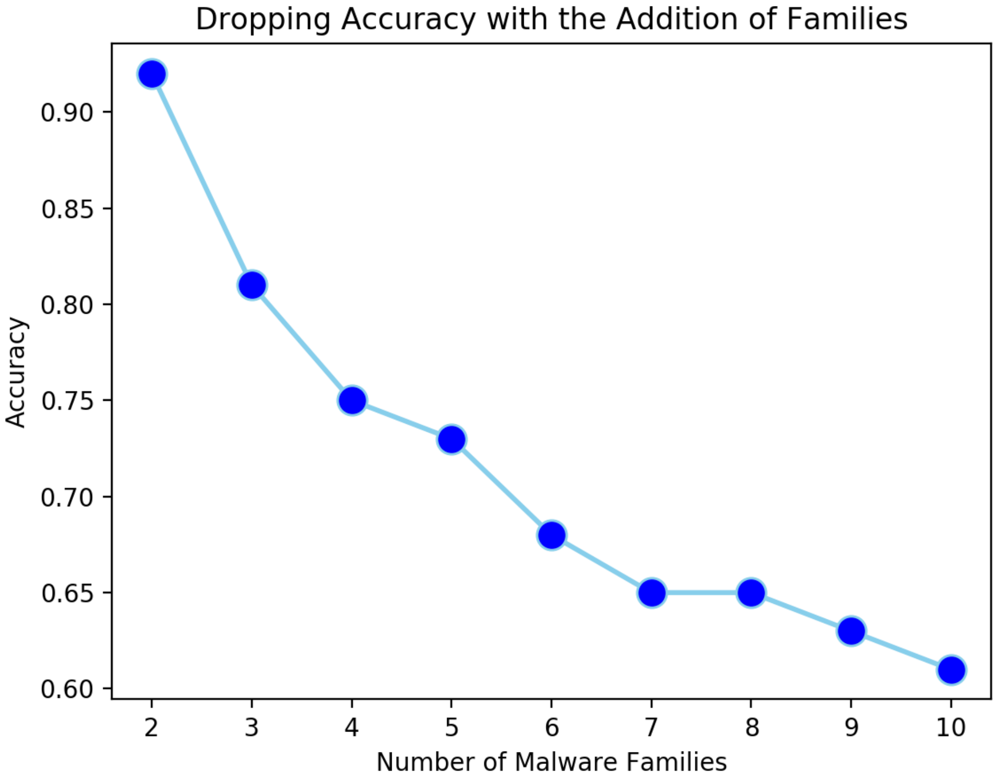


Figure 20: Graph for Dropping Accuracy Values

models as it gives a higher value of accuracy while consuming lesser computation power. Since CNN works on a multi layered structure, it takes a lot of time and computation resources to train the neural network as compared to a simpler, yet efficient machine learning method as SVM. Figure 21 shows a graph of the comparison of the accuracy values between CNN (highest accuracy value in each case) and SVM.

4.2.4 Clustering of Malware Trained HMMs

We carried out experimentations for k-means clustering for $N = 3$ since it was giving us the best accuracy in cases of CNN and SVM. When we applied k-means clustering for 2 families (Adload and Zbot), the accuracy was 0.91. But when we extended our experimentations to 10 families, the accuracy reduced drastically.

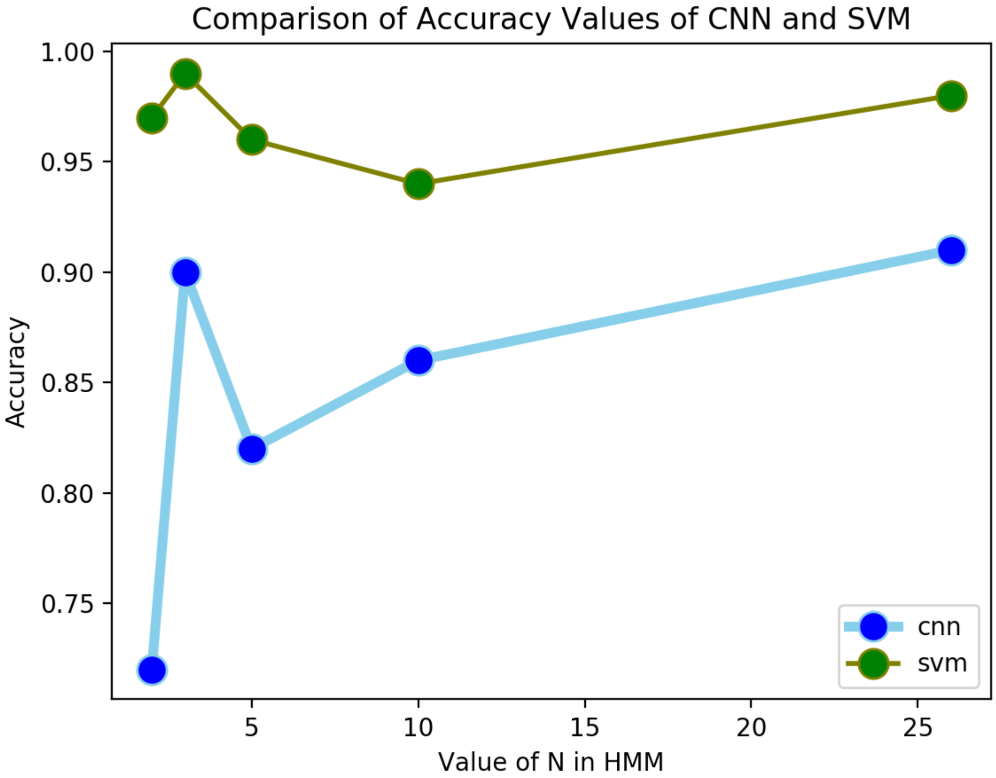


Figure 21: Accuracy Comparison for CNN and SVM

Although the accuracy is less, an interesting observation for 10 families came to light, i.e., the families of the same malware type were being grouped together. For example: HMM models of the families Adload, Bancos, Bho, came together as they are all types of Trojans. Another example was that HMM models of the families Injector, Obfuscator came together as they are all types of VirTool.

4.2.5 Importance of Classification

The classification of malware samples into their respective families is very important because if we are able to classify new malware, we get a better and more accurate way to remove those malware and also a faster way to detect those malware since the malware can be removed using the similar technique as the family to which it is classified to. Also, we would be able to identify the functionality and how the malware relates to other malware samples [19].

CHAPTER 5

Conclusion and Future Work

In this project, we carried out various experiments to determine ways in which we can classify different malware families. We worked with two different families: Zbot and Adload. The description of the different malware families is given in Section 3.1. The total number of samples which we used was 1000. We also saw the results of the comparison between the two methods (CNN and SVM). Please refer Section 4.2.3 for the same.

This project was twofold and took machine learning on another level. This is mainly because in this project, we did not follow an approach where we run HMMs and consider a threshold value to divide the various malware families. Instead we applied machine learning on the results which we obtained from running the HMM, i.e., it is like applying machine learning on top of machine learning.

In this project, we first ran several HMMs, ranging upto 1000 for each malware family. Then we had two approaches:

- We converted the trained HMM results into an image and ran CNN to classify images. This means we were classifying malware samples into their respective malware families.
- We flattened the matrix obtained from the trained HMM results and applied SVM on them with the intention of classifying malware samples into their particular families.

Various experiments were performed where we changed the value of N to 2,3,5,10 and 26. CNN (0.90) performed best in the case of $N = 3$ and similar results were observed in the case of SVM (0.99) also. From our experimentation, we observe that SVM performs better as compared to CNN. The test set consists of trained HMM samples from various families. Hence, we get to know the behavior of various malware families. This is helpful in cases of detecting zero-day malware and will speed up the process of unseen malware detection. It is important to classify zero-day malware into similar families since the detection method which would be used for the new malware sample would be similar to the one used for the existing family. The results showed that HMM in combination with SVM would serve as a great tool to identify zero-day malware and classify them into a malware family matching the known families.

In the case of our project, we use opcodes sequences as the input to our HMM model. The input to the CNN are the images which are generated from the matrices of the HMM and the input to the SVM is the flattened matrix from the training of HMM.

In this project, we have used opcodes as the observation sequence for the HMM. We can look into exploring features such as n-grams or call graph sequences or control flows for future experiments.

In this project, we used two malware families for our experimentation: Zbot and Adload. Along with these two, we can use various other malware families also for experimentation in the future.

An interesting approach which could be explored in the future would be using Profile Hidden Markov Models (PHMM) instead of HMM and then applying CNN or SVM to classify the malware samples.

LIST OF REFERENCES

[1] M. Stamp, "A revealing introduction to Hidden Markov Models," 2012. [Online]. Available: http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf

[2] "Objdump." [Online]. Available: https://linux.die.net/man/1/objdump

[3] "The ultimate list of cyber security statistics for 2019." [Online]. Available: https://purplesec.us/resources/cyber-security-statistics/

[4] D. Gibert Llauradó, "Convolutional neural networks for malware classification," Master's thesis, Universitat Politècnica de Catalunya, 2016.

[5] "Simple image classification using deep learning." [Online]. Available: https://medium.com/intro-to-artificial-intelligence/simple-image-classification-using-deep-learning-deep-learning-series-2-5e5b89e97926

[6] "Opencv tutorials (n.d.), Introduction to Support Vector Machines." [Online]. Available: http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

[7] V. N. Mandhala, V. Sujatha, and B. R. Devi, "Scene classification using Support Vector Machines," in *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*. IEEE, 2014, pp. 1807--1810.

[8] "K-means clustering." [Online]. Available: https://towardsdatascience.com/k-means-clustering-identifying-f-r-i-e-n-d-s-in-the-world-of-strangers-695537505d

[9] J. Aycock, *Computer Viruses and Malware.* Springer US, 2006.

[10] A. Vasudevan, "Maltrak: Tracking and eliminating unknown malware," in *2008 Annual Computer Security Applications Conference (ACSAC).* IEEE, 2008, pp. 311--321.

[11] "Wannacry." [Online]. Available: https://www.csoonline.com/article/3227906/what-is-wannacry-ransomware-how-does-it-infect-and-who-was-responsible.html

[12] J. Baltazar, J. Costoya, and R. Flores, "The real face of koobface: The largest web 2.0 botnet explained," *Trend Micro Research*, vol. 5, no. 9, p. 10, 2009.

[13] S. Pai, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "Clustering for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 2, pp. 95--107, 2017. [Online]. Available: https://doi.org/10.1007/s11416-016-0265-3

[14] U. Narra, F. Di Troia, V. A. Corrado, T. H. Austin, and M. Stamp, ‘‘Clustering versus SVM for malware detection,’’ *Journal of Computer Virology and Hacking Techniques*, vol. 12, no. 4, pp. 213--224, 2016.

[15] ‘‘Boot sector virus.’’ [Online]. Available: http://antivirus.about.com/cs/tutorials/a/bsvirus.htm

[16] R. B. Standler, ‘‘Examples of malicious computer programs,’’ *http://www. rbs2. com/cvirus. htm*, 2002.

[17] B. B. Rad, M. Masrom, and S. Ibrahim, ‘‘Evolution of computer virus concealment and anti-virus techniques: A short survey,’’ *IJCSI International Journal of Computer Science Issues*, vol. 8, 2011.

[18] P. Beaucamps, ‘‘Advanced polymorphic techniques,’’ *International Journal of Computer Science*, vol. 2, no. 3, pp. 194--205, 2007.

[19] C. Annachhatre, T. H. Austin, and M. Stamp, ‘‘Hidden Markov Models for malware classification,’’ *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 59--73, 2015.

[20] ‘‘The different examples of a trojan horse.’’ [Online]. Available: https://enterprise.comodo.com/example-of-a-trojan-horse.php

[21] ‘‘Symantec what is the difference between viruses, worms, and trojans?’’ 2009. [Online]. Available: http://www.symantec.com/business/support/index?page=content&id=TECH98539

[22] ‘‘Top 10 worst computer viruses.’’ [Online]. Available: https://www.telegraph.co.uk/technology/0/top-10-worst-computer-viruses2/

[23] PC Tools (n.d.), ‘‘What is spyware and what does it it do?’’ [Online]. Available: https://us.norton.com/internetsecurity-how-to-catch-spyware-before-it-snags-you.html

[24] M. Onatli, ‘‘Examples of spyware and what they are,’’ 2008. [Online]. Available: http://ezinearticles.com/?Examples-Of-Spyware-And-What-They-Are&id=1054106

[25] M. Stamp, *Information Security: Principles and Practice.* John Wiley & Sons, 2011.

[26] S. Priyadarshi, ‘‘Metamorphic detection via emulation,’’ Master’s thesis, San Jose State University, 2011, https://scholarworks.sjsu.edu/etd_projects/177/.

[27] N. Idika and A. P. Mathur, ‘‘A survey of malware detection techniques,’’ 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.4594&rep=rep1&type=pdf

[28] W.-J. Li, K. Wang, S. J. Stolfo, and B. Herzog, ‘‘Fileprints: Identifying file types by n-gram analysis,’’ in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop.* IEEE, 2005, pp. 64--71.

[29] L. Piyathilaka and S. Kodagoda, ‘‘Gaussian mixture based hmm for human daily activity recognition using 3D skeleton features,’’ in *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA).* IEEE, 2013, pp. 567--572.

[30] L. R. Rabiner, ‘‘A tutorial on Hidden Markov Models and selected applications in speech recognition,’’ *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257--286, 1989.

[31] W. Wong and M. Stamp, ‘‘Hunting for metamorphic engines,’’ *Journal in Computer Virology*, vol. 2, no. 3, pp. 211--229, 2006.

[32] S. Kazi and M. Stamp, ‘‘Hidden Markov Models for software piracy detection,’’ *Information Security Journal: A Global Perspective*, vol. 22, no. 3, pp. 140--149, 2013.

[33] Y. LeCun, Y. Bengio, and G. E. Hinton, ‘‘Deep learning,’’ *Nature*, vol. 521, no. 7553, pp. 436--444, 2015.

[34] L. Deng, D. Yu, *et al.*, ‘‘Deep learning: methods and applications,’’ *Foundations and Trends® in Signal Processing*, vol. 7, no. 3--4, pp. 197--387, 2014.

[35] N. Jmour, S. Zayen, and A. Abdelkrim, ‘‘Convolutional neural networks for image classification,’’ in *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET).* IEEE, 2018, pp. 397--402.

[36] T. Guo, J. Dong, H. Li, and Y. Gao, ‘‘Simple convolutional neural network on image classification,’’ in *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA).* IEEE, 2017, pp. 721--724.

[37] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, ‘‘Malware classification with deep convolutional neural networks,’’ in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS).* IEEE, 2018, pp. 1--5.

[38] ‘‘Support vector machines.’’ [Online]. Available: http://support-vector-machines.org/

[39] T. H. Austin, E. Filiol, S. Josse, and M. Stamp, ‘‘Exploring Hidden Markov Models for virus analysis: a semantic approach,’’ in *2013 46th Hawaii International Conference on System Sciences.* IEEE, 2013, pp. 5039--5048.

[40] U. Mishra, ‘‘An introduction to virus scanners,’’ *Available at SSRN 1916673*, 2010.

[41] I. Firdausi, A. Erwin, A. S. Nugroho, *et al.*, ‘‘Analysis of machine learning techniques used in behavior-based malware detection,’’ in *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies.* IEEE, 2010, pp. 201--203.

[42] S. Cesare and Y. Xiang, ‘‘Classification of malware using structured control flow,’’ in *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing-Volume 107.* Australian Computer Society, Inc., 2010, pp. 61--70.

[43] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, ‘‘Malware images: Visualization and automatic classification,’’ in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, ser. VizSec ’11. New York, NY, USA: ACM, 2011, pp. 4:1--4:7. [Online]. Available: http://doi.acm.org/10.1145/2016904.2016908

[44] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin, ‘‘Context-based vision system for place and object recognition,’’ *Artificial Intelligence Lab Publications*, 2003.

[45] A. Oliva and A. Torralba, ‘‘Modeling the shape of the scene: A holistic representation of the spatial envelope,’’ *International journal of computer vision*, vol. 42, no. 3, pp. 145--175, 2001.

[46] ‘‘Hex rays.’’ [Online]. Available: https://www.hex-rays.com/products/ida/

[47] ‘‘Dataset.’’ [Online]. Available: https://onedrive.live.com/?authkey=%21AGynjOJWX2vQrnw&id=7DE80EDD88C2C9DE%21115903&cid=7DE80EDD88C2C9DE

[48] ‘‘Trojan.zbot.’’ [Online]. Available: https://www.symantec.com/security-center/writeup/2010-011016-3514-99?om_rssid=sr-

[49] ‘‘Trojan.adload.’’ [Online]. Available: https://blog.malwarebytes.com/detections/trojan-adload/

[50] C. B. Do and S. Batzoglou, ‘‘What is the expectation maximization algorithm?’’ *Nature biotechnology*, vol. 26, no. 8, pp. 897--899, 2008.

[51] A. P. Bradley, ‘‘The use of the area under the roc curve in the evaluation of machine learning algorithms,’’ *Pattern recognition*, vol. 30, no. 7, pp. 1145--1159, 1997.