San Jose State University
SJSU ScholarWorks

Master's Projects
Master's Theses and Graduate Research

Spring 2021

Fake Malware Classification with CNN via Image Conversion: A Game Theory Approach

Yash Sahasrabuddhe
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Artificial Intelligence and Robotics Commons, and the Information Security Commons

Recommended Citation
Sahasrabuddhe, Yash, "Fake Malware Classification with CNN via Image Conversion: A Game Theory Approach" (2021). *Master's Projects*. 999.
DOI: https://doi.org/10.31979/etd.q8vd-npfh
https://scholarworks.sjsu.edu/etd_projects/999

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Fake Malware Classification with CNN via Image Conversion: A Game Theory Approach

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

Yash Sahasrabuddhe

May 2021

© 2021

Yash Sahasrabuddhe

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

Fake Malware Classification with CNN via Image Conversion: A Game Theory Approach

by

Yash Sahasrabuddhe

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2021

Dr. Fabio Di Troia	Department of Computer Science
Dr. Nada Attar	Department of Computer Science
Dr. Katerina Potika	Department of Computer Science

ABSTRACT

Fake Malware Classification with CNN via Image Conversion: A Game Theory Approach

by Yash Sahasrabuddhe

Improvements in malware detection techniques have grown significantly over the past decade. These improvements have resulted in better security for systems from various forms of malware attacks. However, it is also the reason for continuous evolution of malware which makes it harder for current security mechanisms to detect them. Hence, there is a need to understand different malwares and study classification techniques using the ever-evolving field of machine learning. The goal of this research project is to identify similarities between malware families and to improve on classification of malwares within different malware families by implementing Convolutional Neural Networks (CNNs) on their executable files. Moreover, there are different algorithms through which we can resize images. Classifying these malware images will help us understand effectiveness of the techniques. As malwares evolve continuously, we will generate fake malware image samples using Auxiliary Classifier Generative Adversarial Network (AC-GANs) and jumble the original dataset to try and break the CNN classifier.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Fabio Di Troia for providing his wisdom and continuous guidance throughout the project. The insights provided by him made this project fun to work on and has improved this report tremendously.

I would also like to extend my thanks to my committee members, Dr. Nada Attar and Dr. Katerina Potika, for their valuable time and feedbacks.

Last, but not the least, I am eternally grateful of my family for supporting me and keeping their belief in me throughout this journey.

TABLE OF CONTENTS

CHAPTER

1 Introduction 1

2 History & Related Work 4

2.1 History 4

2.2 Related Work 4

3 Design and Implementation 6

3.1 Data 6

3.1.1 Malware Families 6

3.1.2 Image Conversion 10

3.2 Technical Concepts 11

3.2.1 Hidden Markov Model 11

3.2.2 Convolutional Neural Networks 13

3.2.3 Auxiliary Classifier Generative Adversarial Network 15

3.3 Approach for Experiments 17

4 Experiments & Results 19

4.1 HMM 19

4.2 CNN 20

4.2.1 CNN with Full Image Dataset 22

4.2.2 CNN with Hamming Images Dataset 22

4.2.3 CNN with Bicubic Images Dataset 22

4.2.4 CNN with Lanczos Images Dataset 23

4.2.5 CNN with Images from NxN bytes Dataset 24
4.3 AC-GAN: The Game Theory Approach 26
4.3.1 CNN on Consolidated Dataset 27
4.3.2 CNN on Poisoned Dataset 29
5 Conclusion & Future Work 42
5.1 Conclusion 42
5.2 Future Work 42
LIST OF REFERENCES 43
APPENDIX
Additional Results 47
A.1 Binary Classification Results 47
A.1.1 Obfuscator & Rbot 47
A.1.2 Obfuscator & Vundo 48
A.1.3 Alureon & Vundo 48
A.2 Loss Graphs 49

LIST OF TABLES

1 Malware Sample Breakdown 7

2 Resize Filter Metrics 11

3 HMM Notations 12

4 CNN Optimum Parameters 21

5 Classification Report for 2 Families 21

6 Classification Report Full Dataset 24

7 Classification Report: Resized with Hamming 26

8 Classification Report: Resized with Bicubic Filter 28

9 Classification Report: Resized with Lanczos Filter 30

10 Classification Report: Header Images Dataset 32

11 AC-GAN Optimum Parameters 32

12 Classification Report: Consolidated Dataset 36

13 Classification Report:Real Sample 37

14 Classification Report: Fake Samples 39

15 Classification Report: Poisoned Samples Classification 41

A.16 Classification Report: Obfuscator & Rbot 47

A.17 Classification Report: Obfuscator & Vundo 48

A.18 Classification Report: Alureon & Vundo 48

LIST OF FIGURES

1 High Level Illustration of HMM 12

2 Basic CNN Architecture 14

3 GAN Architecture 16

4 AC-GAN Architecture 16

5 ROC Curve of WinWebSec Classification 19

6 CNN Architecture 20

7 Confusion Matrix for 2 Family Classification 21

8 Confusion Matrix on Full Image Data 23

9 Confusion Matrix on Full Image Data Using Hamming Filter 25

10 Confusion Matrix on Full Image Data Using Bicubic Filter 27

11 Confusion Matrix on Full Image Data Using Lanczos Filter 29

12 Confusion Matrix on Header Images Dataset 31

13 Generator Architecture 31

14 Discriminator Architecture 33

15 Generator & Discriminator Loss 33

16 Confusion Matrix on Consolidated Dataset 34

17 Confusion Matrix: Real Samples Classification 35

18 Confusion Matrix: Fake Samples Classification 38

19 Confusion Matrix: Poisoned Samples Classification 40

A.20 Confusion Matrix: Obfuscator & Rbot 47

A.21 Confusion Matrix: Obfuscator & Vundo 48

A.22 Confusion Matrix: Alureon & Vundo 49

A.23 CNN: Winwebsec & Zbot Loss 49

A.24 CNN: Full Image Dataset Loss 50

A.25 CNN: Hamming Interpolation Loss 50

A.26 CNN: Bicubic Interpolation Loss 50

A.27 CNN: Lanczos Interpolation Loss 51

A.28 CNN: Header Image Data Loss 51

A.29 CNN: Consolidated Dataset Loss 51

A.30 CNN: Poisoned Dataset Loss 52

A.31 CNN: Obfuscator & Rbot Loss 52

A.32 CNN: Obfuscator & Vundo Loss 52

A.33 CNN: Alureon & Vundo Loss 53

CHAPTER 1

Introduction

Malwares are software programs designed to perform specific malicious tasks to harm computer systems as well as users. This can be done by gaining access to sensitive information or gaining unauthorized control to systems. Malwares can be of any form such as - viruses, ransomwares or trojan horses [1]. They are usually present in unsolicited softwares, or even through links which are made attractive in the hope that a user clicks on it on the web. Once they are inside a system, they replicate and infect the system itself.

To protect ourselves from such intrusions, malware detection and classification has become of utmost importance. There are several anti-malware softwares which help us to detect such attacks and prevent them from happening. There are various approaches behind the curtains of such software. Most of them are signature-based solutions where the files are scanned for a particular signature. However, evading these types of solution is easy for hackers if they use morphed codes or other obfuscation techniques [2]. One such approach is to detect and classify them using machine learning techniques. If we have a pre-trained model which can classify malwares, then classifying a new data point is a task which can be completed very efficiently.

As the detection and classification of malwares into their families is improving continuously, hackers are also able to come up with new techniques to hide the maliciousness of malwares more precisely. This has led to a game theory like problem whereas classification techniques grow stronger, so do the malwares and they become more difficult to recognize by the detection and classification techniques which were in place. Hence, understanding the nature of malwares themselves can help with classification techniques. Machine Learning, therefore, provides us with a unique solution where the software models can learn the characteristics of malwares themselves

to predict never seen before malwares which are based on similar set of properties [3].

Initial approaches for classification which used machine learning techniques were based on Hidden Markov Models (HMMs). For example, in [4] malware samples were classified as either benign or malign using HMM. One of the major steps in using HMM is to extract opcode sequences of the samples. HMM then tries to probabilistically determine the state sequences using which we can classify new samples.

Other techniques like Support Vector Machines (SVMs) were also used as they grew popular. SVM training is the process of finding a 'n-1'-dimensional plane within an n-dimensional dataset which could separate the dataset into two classes. The paper [5] talks about the details of finding this plane using gradient descent approach to find a solution for the quadratic optimization problem which is set up as part of this training process.

As we can see, the above techniques need feature extraction so that we can use them to build a model to classify. CNN, on the other hand, takes data as the input itself and tries to learn the characteristics of the data by re-computing weights between neurons. This implicit learning of data features makes it very suitable to use for classification and detection techniques where there not much information about the data itself. [6] also comments that the removal of human interference on feature extraction is one of the factors why CNN is a stronger classifier than the other techniques. Moreover, CNNs work better on images because of their high dimensionality.

In this research, we will try to leverage these advantages of CNNs in malware classification. We will first convert the malware executable files into images so that we can use CNN. We will then try to understand the relation between different families using the results of the experiments. Finally, we will try to generate fake malware samples using a Auxiliary Classifier Generative Adversarial Network (AC-GAN) and

poison our original dataset to observe the effect on our older data. These fake samples will then be used as a part of the training process of CNN to build a stronger classifier. The structure of the report is as follows: In Chapter 2, we will first look at the background of malware classification using machine learning. We will also provide a brief overview of previous works which were performed on similar lines. In Chapter 3, we will look at the details of our dataset, the concepts of the techniques which we will be using and discuss their implementations. In Chapter 4, we will look at the results of our experiments and its analysis. Lastly, in Chapter 5, we will conclude our research and discuss potential future work.

CHAPTER 2

History & Related Work

2.1 History

Malwares have grown a lot since the turn of 21st century. The paper [?] shows that the number of intrusions recorded the in the year 2005 grew exponentially from the past year. Early mechanisms to detect such malwares was a task which needed heavy computation. One such approach was to detect malware intrusion by assessing the sequence of system calls made by a program [7]. The paper [8] talks about classifying malwares using Hamming distances. Another approach is discussed in [9] where the common subplot of graph is used to classify the data points into their families.

Early malware classification approaches made use of pattern matching algorithms [10]. As the need of detecting malwares grew, research became broader and newer terchniques started to evolve, one of them being machine learning. As a result of the growth of stronger detection techniques, hackers developed more resilient malwares which made it difficult for the traditional machine learning classification techniques to classify them. Hackers came up with different techniques to try to hide malwares from the detection mechanisms. One such technique was to morph a malware into different forms. In [11], the author used pHMM technique was used to for classification by removing certain opcodes which were believed to be leading the morphed nature of malwares.

2.2 Related Work

After the successes seen in detection techniques which used machine learning, new ways to classify malwares grew with evolution of machine learning. Neural Networks have shown promising classification results with various malware datasets. The high dimensionality of images makes image classification an ideal problem for

CNNs. [12] talks about conversion of malware executables into images so that CNN can be leveraged to acheive higher accuracies in classification of malwares of various families.

In [13], the authors created grayscale images using a local mean method. They used them to classify the malwares using K-means techniques. [14] used Microsoft's dataset which is available through Kaggle. The authors implemented ensemble techniques with boosting. [15] and [16] also discuss about leveraging CNNs for multi-class classification. In the former, a character-level approach is used in CNN along with other techniques whereas in the latter, an analysis of classification is performed on two different datasets.

In [17], the author experimented on the Malimg dataset using CNN and ELM techniques and achieved high accuracy in the multi-class classfiication. These reviews and reading helped to understand different approaches taken to classify different malware datasets. Furthermore, in [18] GANs are implemented in an approach which is called as MalGAN where the output from GAN is fed to another feedforward network which makes the final classification if the sample is benign or not. Auxiliary Classifier GAN was a concept first provided in [19] where every generated sample includes a label of which the sample belongs to. These papers served as the inspiration for this research.

CHAPTER 3

Design and Implementation

In this chapter, we will look at the dataset used for the experiments in detail. Then we discuss the techniques used in the project, including image generation algorithms. Finally, we briefly describe the approach for experiments which were performed in this project. Python was used as the primary language for performing the experiments in this project due to its powerful library functions and ease of use.

3.1 Data

The dataset used in this project comprises of executable malware files which were consolidated by an alum. There are a total of 24545 files which belong to 18 families of malwares. The details of the number of samples from each of those 18 families can be seen in Table 1. Samples from each malware family are stored in its own directory with a main root directory providing access to all the families.

3.1.1 Malware Families

Let us look at the malware families individually and discuss them in brief-

alureon is a Trojan virus. These malwares monitor network traffic and steal sensitive information by scanning for passwords, credit card data [20].

bho is a Browser Helper Object which is a type of malware which presents itself through advertisements. When a user clicks on an advertisement, these programs install the BHO trojan which then changes browser settings, looks for sensitive information and tracks browsing behavior so that similar adwares can display more attractive ads leading to other malwares [21].

ceeinject is another Trojan virus where the intention of the malware is to hide the underlying malicious program by obfuscating it. The hidden malware can attack the

Table 1: Malware Sample Breakdown

Malware Name	No of Samples
alureon	1327
bho	1176
ceeinject	892
cycbot	1029
delfinject	1146
fakerean	1063
hotbar	1491
lolyda	915
obfuscator	1436
onlinegames	1293
rbot	1017
renos	1312
startpage	1135
vobfus	964
vundo	1793
winwebsec	3651
zbot	1785
zeroaccess	1119

system in any number of ways. The core is to hide it itself. This is done in such fashion that it attempts to undetected to the anti-virus software [22].

cycbot is a malware whose characteristics are that it gives a backdoor access to the infected system. This allows the hackers to install or uninstall programs on the system, retrieve information from them, update existing programs or even visit web links. The backdoor access means it goes undetected by the detection systems which are in place. They can also be used to launch Denial of Service (DoS) attacks [23].

delfinject is an access point malware. These are generally used to gain access to systems through backdoor mechanisms. Once it is successful, a hacker can install unwanted software on the system. The initial access occurs when a user visits

unsolicited web links or other malicious websites [24].

fakerean is installed on a system through careless installation of softwares. FakeRean malware does not infect a system but rather impersonates anti-virus softwares with fake logos and interfaces. The program then scans your computer and displays some files as infected files and asks the user to pay the owner of the software in order to clean the system. Such malicious programs are categorized as FakeRean malwares [25].

hotbar is an adware where the malware gets installed on the system and installs its own malicious components by changing the appearance of other applications. The malware is then capable of tracking the information flowing through those applications which leads to encountering more ads throughout the time you are browsing [26].

lolyda is a trojan malware which is specifically found in gaming. They usually extract the gaming account information. They may even be responsible for random files being installed on the system [27].

obfuscator is a malware which as the name suggests, tries to hide underlying malicious program which can be anything. Such malwares are programmed in such a way that these help other malwares go undetected from anti-virus softwares [28].

onlinegames is a family of harmful malwares which usually gains access by installation of softwares especially games through false resources. Once they are in the system, they can record keyboard behavior, modify, and install additional malicious programs and steal information [29].

rbot are backdoor trojans which usually affect through network. It launches its attack based on weak passwords and they can be the cause of DoS attacks while retrieving user information as well [30].

renos is similar to FakeRean where a spyware tells the user that the system is infected and requests for money to clean the system even though there might not be any other malicious content present in the system [31].

startpage are very unpredictable and usually change browser settings through unauthorized use. They may also be responsible for other malicious installation of softwares on the system. [32]

vobfus are worms which are responsible for installing other malwares on a system. They can be transmitted through flash drives or can even be downloaded through other media [33].

vundo is another adware where unrelated ads are displayed to the users while browsing. These tend to be access points to BHO malwares [34].

winwebsec is like ransomware where the winwebsec malware displays that the system is infected through a malicious software and locks other applications or the system itself and asks the user to pay to clean the system even though there aren't any other infected files. It may also affect other applications and be the cause of DoS attacks from within the system where it blocks usage of websites [35].

zbot is a trojan virus which attacks systems with the intent to steal sensitive information about users. They can also be responsible for providing a remote access of your system to the hacker(s) [36].

zeroaccess is another rootkit malware which remains hidden and is responsible for pay-per-click fraud. It attacks by retrieving information from browsers while user surfs the internet. This is done through advertising their own network and retrieving information from other middle men and web traffic [37].

3.1.2 Image Conversion

Since the focus of this research project is to leverage the advantages neural networks, we convert the executable samples into grayscale images to increase their dimensionality. To do this, we converted each byte of the bytecode sequence of the executable files into their ASCII value and added this stream of data to a 2D array. These values represent a grayscale image, and a corresponding image is saved in a directory. All this can be done using the *image.save* API from PIL.

After this process was completed, we needed the size of the images of all malwares to be consistent. We chose 64x64 as an ideal choice since the model will not become heavy due to a big size but will have enough information so that the model can learn some features of each family. This choice was bolstered after reading [38] paper in which the author achieved good results with these dimensions for a lightweight classifier.

Hence, to convert the images to 64x64 dimensions, we used the below 2 approaches-

1. Using PIL Image Library Filters:

   This library provides the APIs to resize an input image to a desired dimension using different filters like-

   HAMMING - Produces sharper pixels based on Hamming interpolation. Has better performance but the image quality is not the best.

   BICUBIC - Performs a cubic interpolation between all contributing pixels to determine the resulting pixel's value. Lower performance than Hamming

filter but the quality is better.

LANCZOS - Determines the resulting pixel's value based on a high quality Lanczos filter which is a truncated sinc filter. It produces high quality images but lags on its performance.

Table 2: Resize Filter Metrics

Filter	Quality	Performance
HAMMING	Good	Best
BICUBIC	Better	Better
LANCZOS	Best	Best

2. Reading NxN amount of bytes

   As it was seen from the results of the paper [39], the use of headers of malware executables also results in great classification results. Hence, using this as a benchmark, we will read only a certain number of bytes from the malware file and construct an image from this data itself. Since we are building 64x64 images we will read the first 4096 bytes from the malware executable file and create a grayscale image.

3.2 Technical Concepts

In this section, we will look briefly investigate the concepts of the techniques which have been used in this project. We will also describe the architecture and the optimum parameters that were found for setting up the experiments.

3.2.1 Hidden Markov Model

Hidden Markov Model (HMM) is based on the Markov process in which the Markov chain is basically a set of hidden states. The goal of the algorithm is to compute likelihood of transitions between the hidden states. This can be done by

observing the states from the observation sequence which are probabilistically linked to the hidden states. The Figure 1 depicts a high-level architecture of HMMs.

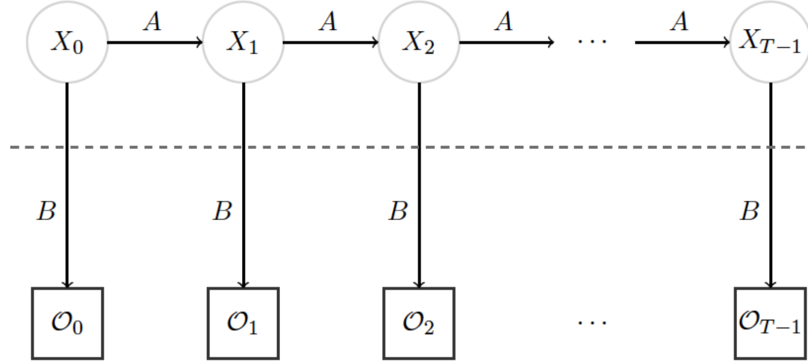

Figure 1: High Level Illustration of HMM

Table 3 shows a subset of notations that are generally used in HMM. [40].

Table 3: HMM Notations

Sybmbol	Meaning
A	Hidden State Transition Probabilities
B	Observation Probability Matrix
π	Initial State Distribution
O	Observation Sequence
N	Number of Hidden States
M	Unique Observation Symbols

The probability distribution of the hidden states is represented in A matrix whose dimensions are based on the number of hidden states. The observation sequence is denoted by the symbol 'O'. The probability of each data point i.e., an observation, to occur against a particular hidden state is calcualted in the 'B'matrix. Therefore, the dimension of this matrix is a 2-D matrix which is of size N x M.

There are three classic problems which can be solved using HMM - first, to score

an observation sequence based on a given trained model, second, to uncover the hidden states and finding their optimal state transition probabilities, and third, to train a model λ which fits the sequence of observations.

The training of the model from problem 3 is done based on a hill climb algorithm known as the Baum-Welch re-estimation algorithm. The A, B and π matrices are randomly initialized, but are always row stochastic. They are then re-estimated each step using an Expectation Maximization algorithm (EM) until the training converges to a local maximum.

3.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were first introduced in [41] after back-propagation techniques for feed-forward neural networks were described from the book [42] by C.M. Bishop.

The main architecture bases of CNN are shared weights, degree shift and receptive fields [41]. There are layers of neurons which are fully connected with each other. They may be of different dimensions where the receiving layer receives a small neighborhood of inputs which were present in the previous layer. The neurons have a set of inputs and outputs. A weight is associated with each of those synapses (like a neuron in brain). A loss or error rate is calculated with each iteration of training. This becomes a error minimization problem, where we adjust the weights associated with each input/output between the layers so as to minimize the loss and increase the accuracy. The re-computation of weights is implemented using a backpropagation approach [41].

The advantages of using CNNs is that it removes the need to extract features from data and classify using only that set of features. Moreover, extracting features of high dimensional data like images can be a computationally expensive task. CNN on the other hand, does not require a set of features specifically. The layers of neural

networks can extrapolate the information needed to classify data implicitly. Each neuron has some information which is necessary for the overall result and training of a CNN model.

For a model to train, there are various types of layers that we can add to the architecture of a CNN model. We will look into the concepts of some of these layers in brief. Figure 2 is a high-level depiction of CNN architecture.

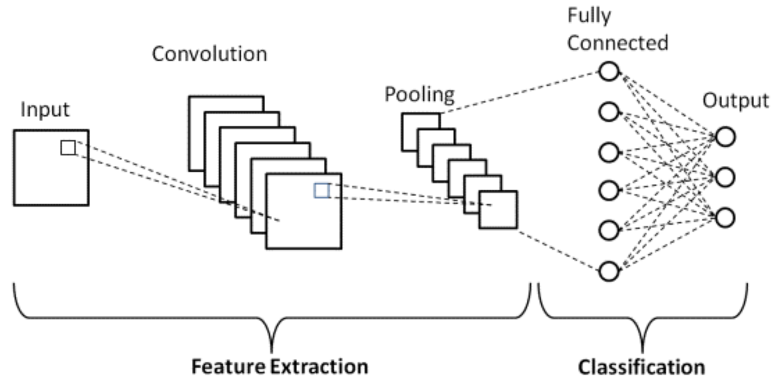

Figure 2: Basic CNN Architecture

Let us now look at the different types of layers we can use in a CNN model.

1. Convolution Layer - This layer consists of some number of neurons which have a tensor input. Tensor input means, the dimensions of the input are determined by the height, width of the images, number of images and the channels of the input images. The function of this layer is to convolve the input based on a size known as kernel size and feed forward the results to the next layer.

2. Max-Pooling Layer - Pooling layers reduce the dimensionality of the input layer and achieving invariance. Clusters of neurons from input layer for an output of single neuron using a kernel map of some size. The max pooling layer considers the maximum value from those clusters as the output as a single neuron [43].

3. Dropout Layer - A dropout layer is used to regularize the parameters of the convolutional layers [44]. It randomly sets a percent of inputs to 0. The other unchanged inputs are scaled so that the overall sum of parameters remains unchanged.

4. Flatten Layer - This layer is used to convert a multidimensional input into a single dimensional output of the tensor. This is normally used just before a dense layer as a single column input to it [45].

5. Dense Layer - The dense layer is a deep layer, i.e., fully connected network where each neuron present in the network receives an input from the previous layer. This layer performs matrix multiplications and recomputes the weights of neurons using backpropagation. An activation function, usually relu, is used to help the model learn complex patterns by helping to decide if a neuron should be activated i.e., used as an output or not.

3.2.3 Auxiliary Classifier Generative Adversarial Network

The Auxiliary Classifier Generative Adversarial Networks or AC-GANs are a development over GANs. The were first introduced in [19] by Augusts Odena, Christoper Olah and Jonathon Shlens.

GANs are a setup of a generator model and discriminator model which are neural networks. They are trained in opposition of each other, like a setup of game theory, where the generator is training to generate better quality of fake data from a noise input. The discriminator model is a classifier which tries to detect if the generated fake sample data is real or not. A loss is calculated to determine the gap between generated fake sample and real data samples. This process goes back and forth until the discriminator model is not able to distinguish between real and fake data. The loss is calculated based on the following equation-

$$L = E[logP(S = real)|X_{real}] + E[logP(S = fake)|X_{fake}] \text{ [19]} \quad (1)$$

The discriminator tries to maximize the first term of the above equation whereas the generator tries to minimize the second term of the above equation. Figure 3 shows a basic architecture of GANs.

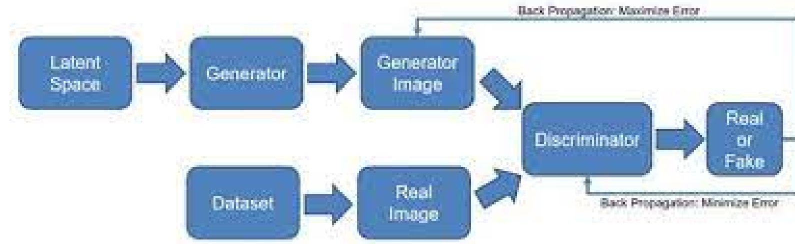

Figure 3: GAN Architecture

AC-GANs is a variant of the above architecture in which the discriminator produces two outputs. One is whether the generated image is real or fake and to which class it belongs to. The new architecture is shown in Figure 4 below.

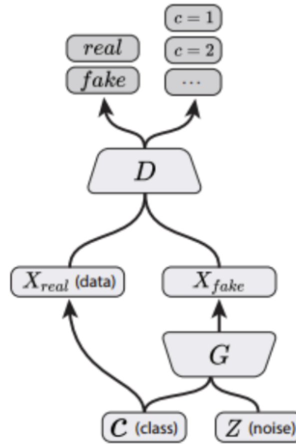

Figure 4: AC-GAN Architecture

3.3 Approach for Experiments

This section describes the roadmap for experiments which are to be conducted. As we are using CNN as our main classifier, we need some kind of a benchmark result to compare the results from CNN. Hence, the first step is to train a basic HMM model based on their opcodes. We will train *winwebsec* family as it has the greatest number of samples available and then test the model against samples from *winwebsec* as well as other families. As we have a benchmark, we will then build CNN model with *winwebsec* and *zbot* samples and compute accuracies to validate the result by comaparing them with HMM results. To do this, we will first convert the malware executable files into images so that we can provide those as inputs to CNN. A basic image conversion is performed, and data is reseized using a built-in API. After we compare the results and tune the model, we can use the same architecture later experiments as well.

We will now move on to a multi-class classification to classify the 18 families of malwares on a CNN model and observe its results. We will follow it up with some further binary classification if necessary, for certain families. Next, we will perform the same experiment with different datasets generated using the approaches mentioned in Section 3.1.2.

Now, we will work with our game theory like approach where we will generate fake samples of malware by setting up an AC-GAN model. For this, we will use the original converted images dataset, which contains images of variable size, as source for training as they contain much more information than resized images. After training the AC-GAN model and generating fake images for all the malware families, we perform a 36 class (18 real & 18 fake) classification on a subset of data to analyze the confusion of CNN classifier based on the same architecture as of previous experiments. This appraoch will lead us to propose our final experiment in which we will create a

poisoned dataset which will consist of both real and fake samples of each family.

Finally, as this poisoned dataset is created, we will train a CNN model on it which would complete our game theory in the sense that the inclusion of fake samples would help us counter the reduction in classification accuracies which we see from the 36-class classification. We will score 3 testing samples based on this trained model - on real data only, on fake data only and on poisoned test samples which contain both real and fake data and observe their results.

CHAPTER 4

Experiments & Results

In this section, we will see the set up for each of the experiments i.e., model architectures and analyze the results of each of them by observing the confusion matrices and classification reports generated by the experiments.

4.1 HMM

For the basic classification, we trained an HMM model on *winwebsec* family by considering the most used opcodes from the samples. The opcodes define the dimensions of the 'B'matrix. The 'A'matrix is the length of the number of hidden states we keep for the problem. We performed the experiment with 10000 random restarts for training. The HMM model uses Baum-Welch re-estimation algorithm and once the training completes, we scored the samples from other families as well as from *winwebsec*. Figure 5 graph shows the results of experiment-

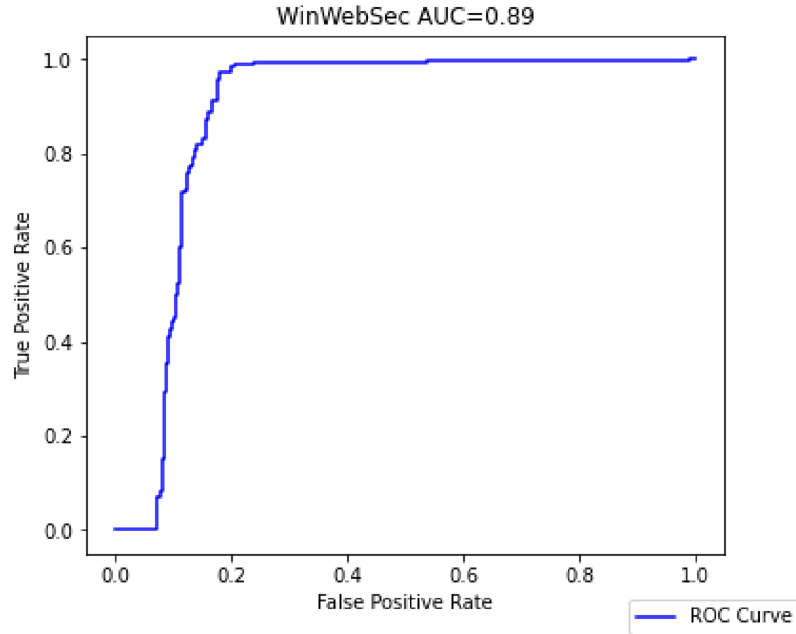

Figure 5: ROC Curve of WinWebSec Classification

As we can see, we score winwebsec samples and samples from different families

using this model. We get 89% accuracy for this binary classification.

4.2 CNN

We first implement a CNN model based on two families. The architecture which yielded the best results is shown below in Figure 6. We will use this architecture for further experiments as well. Table 4 shows the optimum parameters for training a CNN model.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 30)        300
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 30)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 15)        4065
_________________________________________________________________
dropout (Dropout)            (None, 29, 29, 15)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 15)        0
_________________________________________________________________
flatten (Flatten)            (None, 2940)              0
_________________________________________________________________
dense (Dense)                (None, 128)               376448
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                6450
_________________________________________________________________
flatten_2 (Flatten)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 18)                918
=================================================================
Total params: 388,181
Trainable params: 388,181
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: CNN Architecture

We use all the samples belonging to the *winwebsec* and *zbot* samples. The data is then split into training and testing using the train_test_split library with 70% of the data for training and the remaining 30% for testing. Figure 7 shows the confusion matrix of the classification. Table 5 shows the classification report.

Table 4: CNN Optimum Parameters

Parameter	Value
KernelSize	3
PoolSize	2
Activation Function	relu
Epochs	50
Optimizer	Adam
Loss Function	Categorical Crossentropy

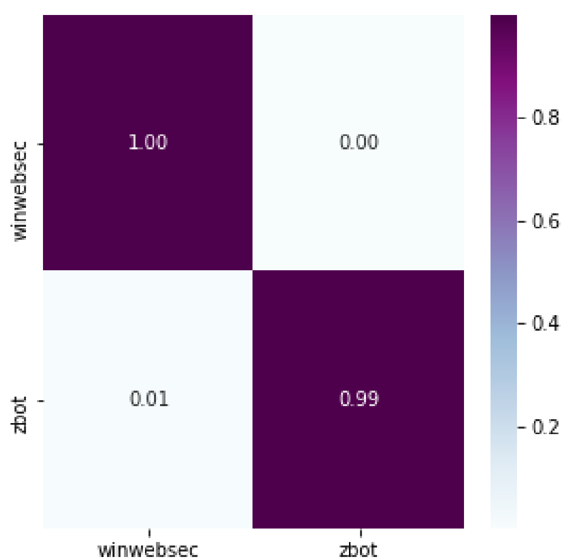

Figure 7: Confusion Matrix for 2 Family Classification

Table 5: Classification Report for 2 Families

Class	Precision	Recall	F1-Score	Support
0	1.00	1.00	1.00	1124
1	0.99	0.99	0.99	507
accuracy			0.99	1631
macro avg	0.99	0.99	0.99	1631
weighted avg	0.99	0.99	0.99	1631

As we have acheived a better accuracy to that of the HMM results, it can be said that CNN can be a better classifier. We will now implement a multi-class classification for all the datasets.

4.2.1 CNN with Full Image Dataset

We use the dataset created by reading all the bytes of the malwares and creating images of various sizes. This dataset is then loaded to the CNN model using PIL library's ImageDataGenerator API. This resizes the images into the target size of 64x64 for our CNN model. The model is trained for 50 epochs using *relu* activation for the dense layers as used in the previous experiment. Figure 8 shows the confusion matrix.

Table 6 shows the classification report.

As we can see, we achieved an 80% accuracy for an 18-class classification which seems great. But can we do better? Let's look at the results in the next sections.

4.2.2 CNN with Hamming Images Dataset

As seen earlier, the resizing of the images from the dataset occurs internally through ImageDataGenerator. As a different approach, we resized the images to 64x64 size using the 'Hamming'filter on the whole dataset. Figure 9 shows the confusion matrix achieved for classifying the images based on this filter.

The Table 7 shows the classification report.

As we can see, the maximum accuracy we achieved using Hamming resize for the images is 86%. We can see the improvement from the previous result which justifies the use of resize filters.

4.2.3 CNN with Bicubic Images Dataset

We resize the images to 64x64 size using the 'Bicubic'filter on the whole image dataset. Figure 10 shows the confusion matrix achieved for classifying the images

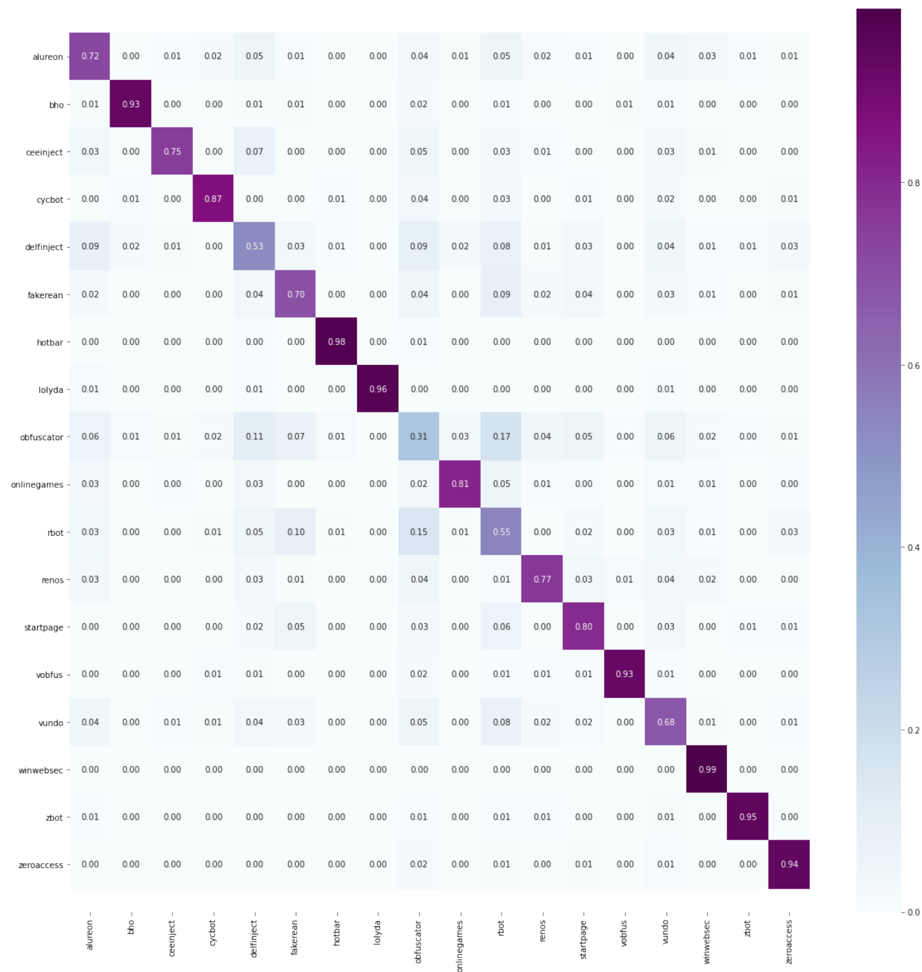

Figure 8: Confusion Matrix on Full Image Data

based on this filter.

The Table 8 below depicts the classification report.

As we can see, the maximum accuracy we achieved using the Bicubic filter for resizing the images is 85%. We can say that the Hamming filter provides better results than Bicubic filters.

4.2.4 CNN with Lanczos Images Dataset

We resize the images to 64x64 size using the 'Lanczos'filter on the whole image dataset. Figure 11 shows the confusion matrix achieved for classifying the images

Table 6: Classification Report Full Dataset

Class	Precision	Recall	F1-Score	Support
0	0.72	0.67	0.69	418
1	0.93	0.94	0.93	343
2	0.75	0.91	0.83	220
3	0.87	0.89	0.88	308
4	0.53	0.52	0.52	360
5	0.70	0.65	0.67	343
6	0.98	0.96	0.97	436
7	0.96	0.99	0.98	274
8	0.31	0.38	0.34	364
9	0.81	0.91	0.86	368
10	0.55	0.39	0.45	440
11	0.77	0.83	0.80	330
12	0.80	0.77	0.78	387
13	0.93	0.94	0.93	288
14	0.68	0.73	0.70	493
15	0.99	0.96	0.98	1150
16	0.95	0.97	0.96	510
17	0.94	0.88	0.91	336
accuracy			0.80	7368
macro avg	0.79	0.79	0.79	7368
weighted avg	0.80	0.80	0.80	7368

based on this filter.

The Table 9 below depicts the classification report.

As we can see, the maximum accuracy we achieved using the Lanczos filter for resizing the images is 86%.

4.2.5 CNN with Images from NxN bytes Dataset

We create the new dataset by reading the first 4096 bytes of the malware executables and using these bytes to create images of size 64x64. We classify these 'header images now using the CNN model. Figure 12 shows the confusion matrix of the observed results.

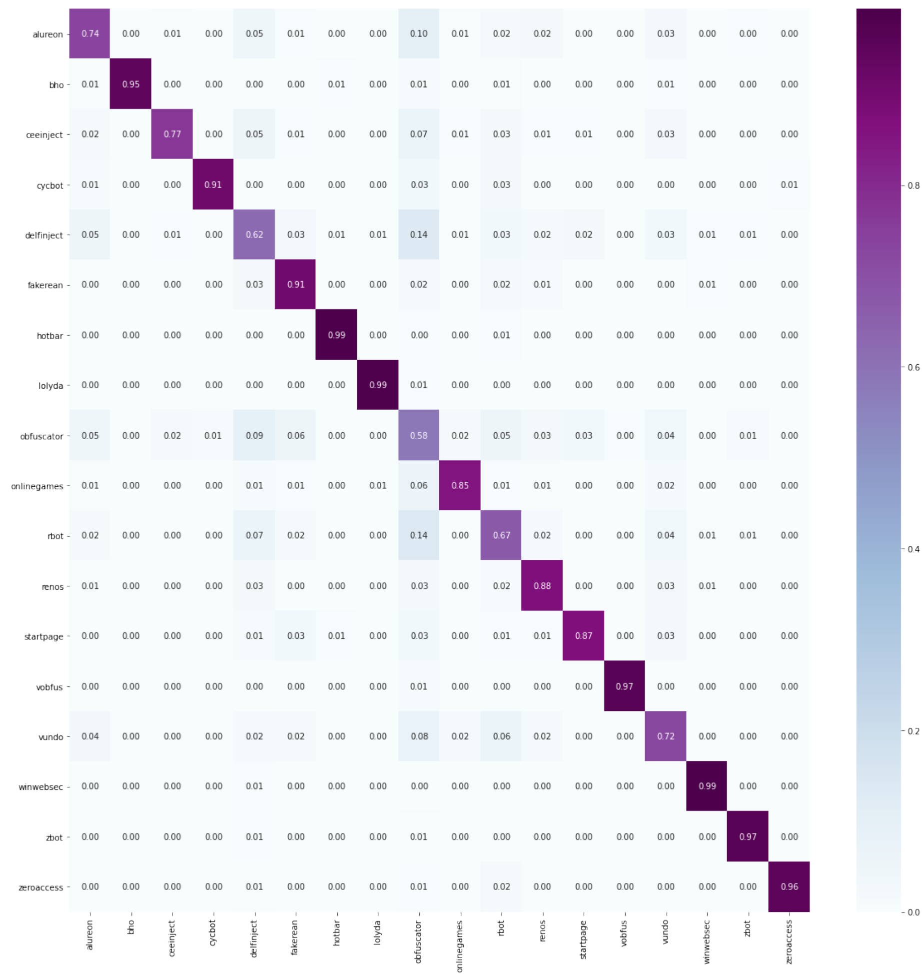

Figure 9: Confusion Matrix on Full Image Data Using Hamming Filter

The Table 10 shows its classification report.

We achieved a 92% accuracy for this dataset. This means that the header information from the malware executables was enough information to classify them into each of their classes. This means that in other resizing filters, the information was getting blurred due to which their accuracy is lower than what we see here. It might also be interpreted that the whole malware file's data might make the classification more complex.

Table 7: Classification Report: Resized with Hamming

Class	Precision	Recall	F1-Score	Support
0	0.74	0.77	0.75	381
1	0.95	1.00	0.97	314
2	0.77	0.91	0.83	220
3	0.91	0.98	0.94	278
4	0.62	0.58	0.60	352
5	0.91	0.81	0.86	375
6	0.99	0.98	0.98	480
7	0.99	0.97	0.98	275
8	0.58	0.47	0.52	512
9	0.85	0.92	0.88	347
10	0.67	0.63	0.65	324
11	0.88	0.85	0.86	402
12	0.87	0.92	0.90	336
13	0.97	0.98	0.97	293
14	0.72	0.81	0.77	500
15	0.99	0.99	0.99	1142
16	0.97	0.97	0.97	526
17	0.96	0.97	0.96	311
accuracy		0.86	7368	
macro avg	0.85	0.86	0.86	7368
weighted avg	0.86	0.86	0.86	7368

4.3 AC-GAN: The Game Theory Approach

We set up an AC-GAN using PyTorch, leveraging GPUs for faster execution. Figure 13 & Figure 14 illustrate the architectures of generator and discriminator models respectively. As mentioned in Section 3.3, we use the full image dataset as the input for the AC-GAN. Moreover, this sets up the game theory like approach when we try to induce a negative effect on the previous results by introducing the fake samples to the classifier. Once we see this negative effect on the accuracies, we will then try to improve on it by considering these generated fake samples in the training process as a part of the real dataset, thus creating a "poisoned dataset". This will

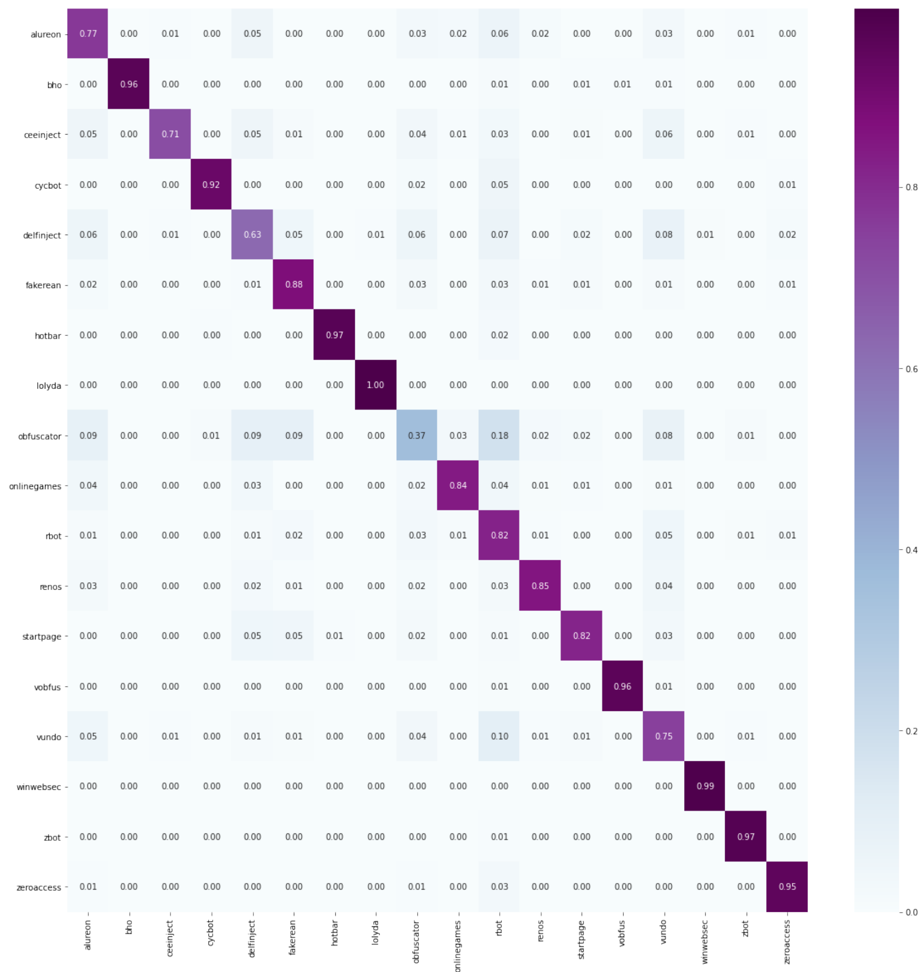

Figure 10: Confusion Matrix on Full Image Data Using Bicubic Filter

lead to an improvement on the classification from previous results.

Table 11 describes the optimum parameters which were used to train the model.

The Figure 15 shows the variation in generator and discriminator loss over the epochs while training AC-GAN.

4.3.1 CNN on Consolidated Dataset

We generate 500 samples from a generator model which produces a reasonable discriminator accuracy (around 50%). In the experiment, generators from the range of batch numbers 7000 to 13000 were producing results which were seen as a fit for

Table 8: Classification Report: Resized with Bicubic Filter

Class	Precision	Recall	F1-Score	Support
0	0.77	0.68	0.72	425
1	0.96	0.99	0.98	353
2	0.71	0.93	0.81	216
3	0.92	0.97	0.95	297
4	0.63	0.63	0.63	359
5	0.88	0.76	0.81	395
6	0.97	0.99	0.98	447
7	1.00	0.98	0.99	263
8	0.37	0.57	0.45	278
9	0.84	0.91	0.88	353
10	0.82	0.48	0.61	541
11	0.85	0.92	0.89	366
12	0.82	0.89	0.85	335
13	0.96	0.99	0.97	265
14	0.75	0.73	0.74	561
15	0.99	0.99	0.99	1060
16	0.97	0.96	0.97	523
17	0.95	0.95	0.95	331
accuracy			0.85	7368
macro avg	0.84	0.85	0.84	7368
weighted avg	0.86	0.85	0.85	7368

generation. We generate the samples using the generator model of batch 11600.

After the generation, we will build a consolidated dataset containing 36 classes of data, 18 fake and 18 real, and see observe how the results are different from the results in Section 4.2.1. The Figure 16 below shows the confusion matrix for the 36 classes and the Table 12 describes its classification report.

As we can see, the CNN model is not able to classify the samples as precisely as it was able to before. This is due to the inclusion of these fake samples which has made it harder for the CNN to classify the samples into each of their respective families.

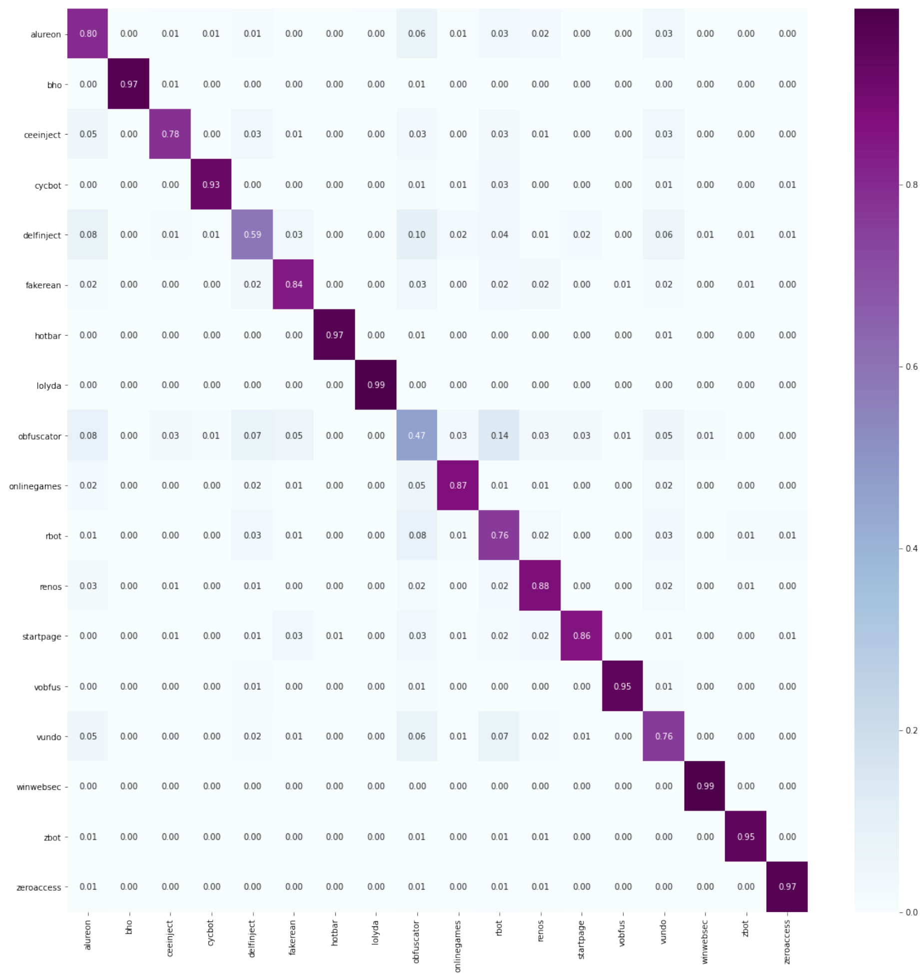

Figure 11: Confusion Matrix on Full Image Data Using Lanczos Filter

4.3.2 CNN on Poisoned Dataset

As discussed from Section 3.3, we have trained a CNN model on the poisoned dataset based on the same optimum architecture. The training data was based on a 50:50 split of real and fake samples. We chose 400 real samples from each family and 400 fake samples of each family for training the model. We then scored 100 samples of real and fake and 200 samples of poisoned data using the trained model. We observe these results for the three classifications-

1. **Scoring Real Samples** - Figure 17 shows the confusion matrix for classifying

Table 9: Classification Report: Resized with Lanczos Filter

Class	Precision	Recall	F1-Score	Support
0	0.80	0.70	0.75	474
1	0.97	0.97	0.97	349
2	0.78	0.87	0.82	234
3	0.93	0.95	0.94	301
4	0.59	0.69	0.64	287
5	0.84	0.82	0.83	341
6	0.97	0.99	0.98	421
7	0.99	0.99	0.99	286
8	0.47	0.50	0.48	399
9	0.87	0.90	0.88	381
10	0.76	0.56	0.65	393
11	0.88	0.84	0.86	398
12	0.86	0.91	0.88	330
13	0.95	0.96	0.96	275
14	0.76	0.79	0.77	501
15	0.99	0.99	0.99	1139
16	0.95	0.97	0.96	496
17	0.97	0.96	0.96	363
accuracy			0.86	7368
macro avg	0.85	0.85	0.85	7368
weighted avg	0.86	0.86	0.86	7368

100 real samples against the trained model. Its classification report is constructed in Table 13.

The real samples are classified with an accuracy of 62%. This accuracy is great when we compare these results with the results from Section 4.3.1. We can see at some individual results which have performed a lot better. For example, *alureon* classification has significantly improved from 33% to 80%.

2. **Scoring Fake Samples** - Figure 18 shows the confusion matrix for classifying 100 fake samples against the trained model. Its classification report is constructed in Table 14.

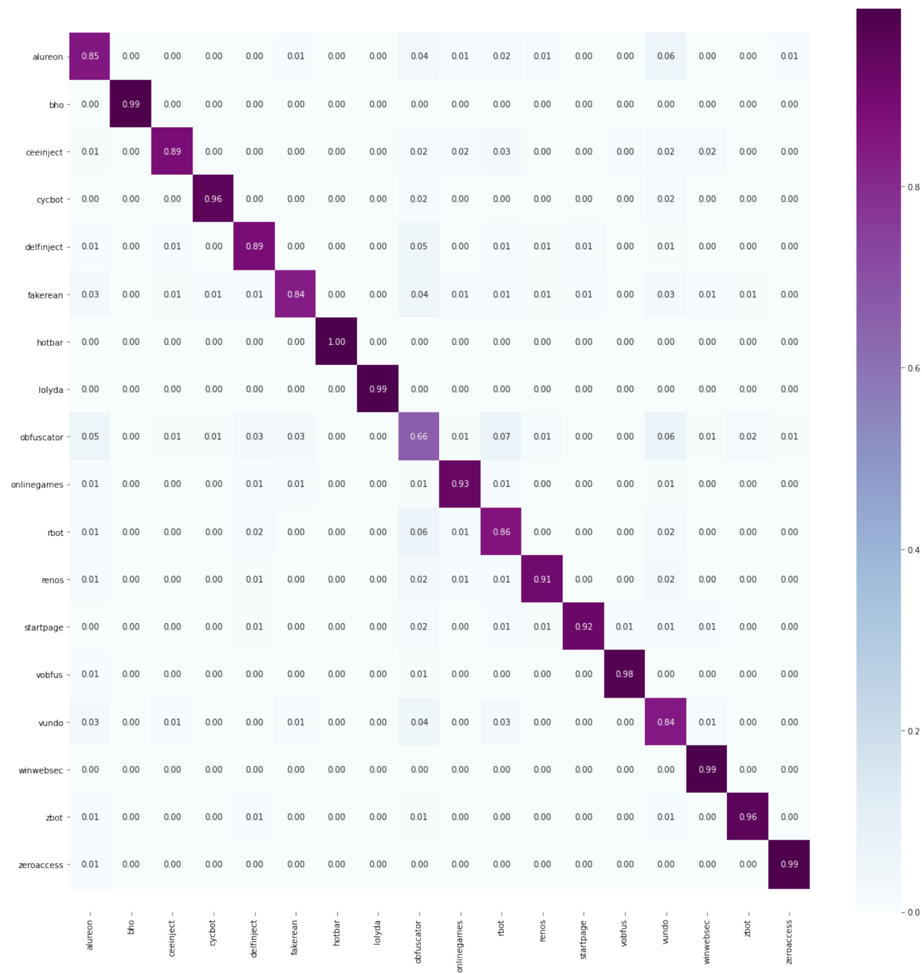

Figure 12: Confusion Matrix on Header Images Dataset

```
Generator(
  (label_emb): Embedding(18, 100)
  (l1): Sequential(
    (0): Linear(in_features=100, out_features=32768, bias=True)
  )
  (conv_blocks): Sequential(
    (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): Upsample(scale_factor=2.0, mode=nearest)
    (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Upsample(scale_factor=2.0, mode=nearest)
    (6): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): Tanh()
  )
)
```

Figure 13: Generator Architecture

This is an interesting result as the fake samples are also getting better classified into their own family. Although, this is because of inclusion similar fake samples

Table 10: Classification Report: Header Images Dataset

Class	Precision	Recall	F1-Score	Support
0	0.85	0.82	0.84	410
1	0.99	1.00	0.99	350
2	0.89	0.91	0.90	248
3	0.96	0.96	0.96	326
4	0.89	0.89	0.89	335
5	0.84	0.89	0.86	326
6	1.00	1.00	1.00	436
7	0.99	0.99	0.99	268
8	0.66	0.70	0.68	423
9	0.93	0.95	0.94	384
10	0.86	0.77	0.81	343
11	0.91	0.94	0.93	352
12	0.92	0.97	0.95	322
13	0.98	0.97	0.97	290
14	0.84	0.82	0.83	576
15	0.99	0.98	0.99	1072
16	0.96	0.97	0.96	557
17	0.99	0.97	0.98	346
accuracy			0.92	7364
macro avg	0.91	0.92	0.91	7364
weighted avg	0.92	0.92	0.92	7364

Table 11: AC-GAN Optimum Parameters

Parameter	Value
Batch Size	64
Learning Rate	0.0002
Epochs	100
Beta1	0.5
Beta2	0.999
Optimizer	Adam
Loss Function	BinaryCrossEntropy
Activation Function	LeakyReLU

```
Discriminator(
  (conv_blocks): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout2d(p=0.25, inplace=False)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Dropout2d(p=0.25, inplace=False)
    (6): BatchNorm2d(32, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Dropout2d(p=0.25, inplace=False)
    (10): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (11): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (12): LeakyReLU(negative_slope=0.2, inplace=True)
    (13): Dropout2d(p=0.25, inplace=False)
    (14): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
  )
  (adv_layer): Sequential(
    (0): Linear(in_features=2048, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (aux_layer): Sequential(
    (0): Linear(in_features=2048, out_features=18, bias=True)
    (1): Softmax(dim=None)
  )
)
```

Figure 14: Discriminator Architecture

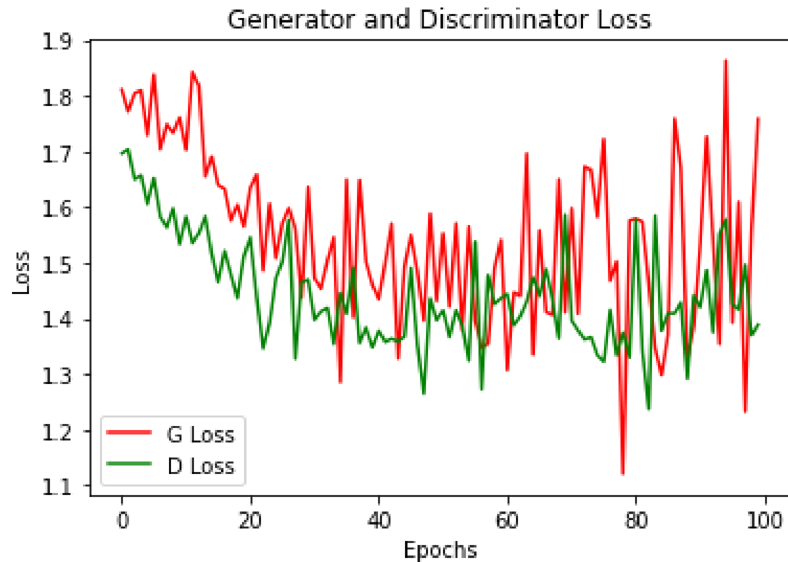

Figure 15: Generator & Discriminator Loss

in training. The 77% accuracy is a strong classification when compared with previous results.

3. **Scoring Poisoned Samples** - Figure 19 shows the confusion matrix for classifying 200 poisoned samples consisting of 100 fake and 100 real samples of each family against the trained model. Its classification report is constructed in

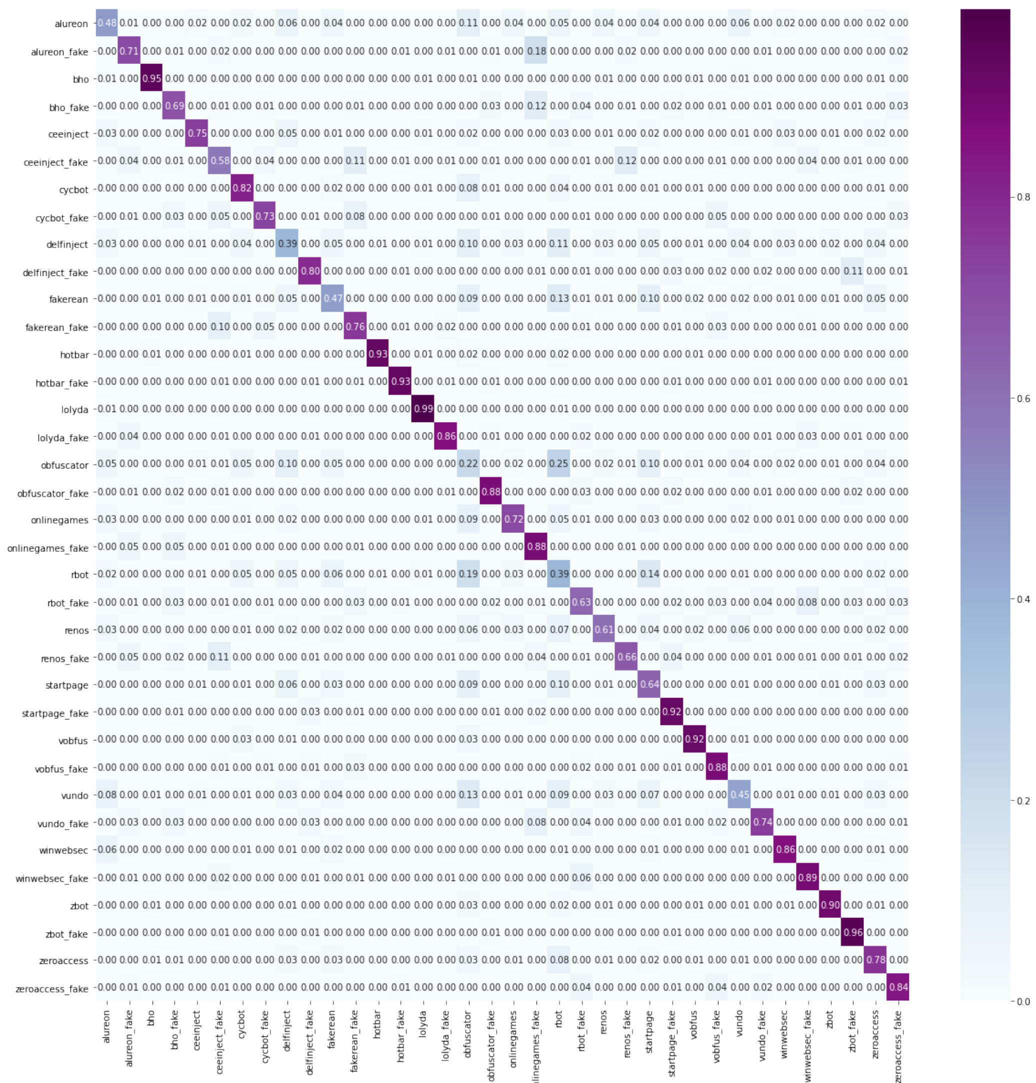

Figure 16: Confusion Matrix on Consolidated Dataset

Table 15.

The poisoned test samples show a 77% accuracy which is great considering the disparities between the real and fake data existing in the same data.

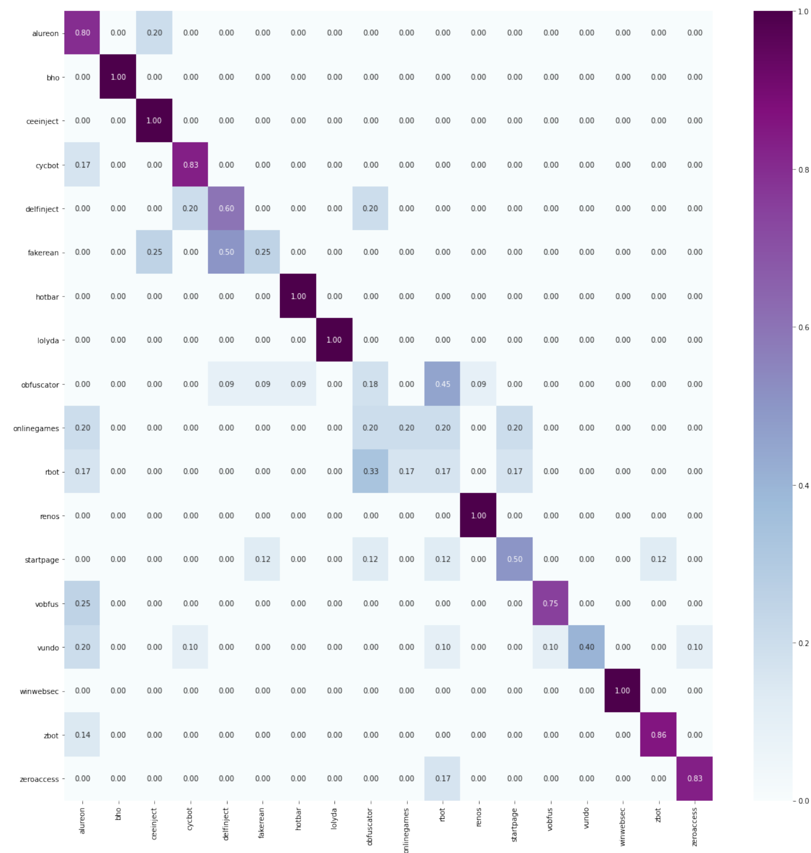

Figure 17: Confusion Matrix: Real Samples Classification

Table 12: Classification Report: Consolidated Dataset

Class	Precision	Recall	F1-Score	Support
0	0.48	0.54	0.51	114
1	0.71	0.73	0.72	143
2	0.95	0.96	0.95	153
3	0.69	0.76	0.73	144
4	0.75	0.90	0.81	115
5	0.58	0.58	0.58	142
6	0.82	0.78	0.80	174
7	0.73	0.85	0.79	128
8	0.39	0.44	0.41	131
9	0.80	0.85	0.82	124
10	0.47	0.59	0.52	133
11	0.76	0.74	0.75	165
12	0.93	0.99	0.96	150
13	0.93	0.95	0.94	149
14	0.99	0.93	0.96	153
15	0.86	0.93	0.89	131
16	0.22	0.18	0.20	201
17	0.88	0.87	0.87	152
18	0.72	0.82	0.77	126
19	0.88	0.67	0.76	212
20	0.39	0.27	0.32	224
21	0.63	0.68	0.65	145
22	0.61	0.73	0.66	102
23	0.66	0.79	0.72	117
24	0.64	0.50	0.56	199
25	0.92	0.82	0.87	148
26	0.92	0.91	0.92	160
27	0.88	0.82	0.85	169
28	0.45	0.61	0.52	115
29	0.74	0.84	0.79	128
30	0.86	0.88	0.87	151
31	0.89	0.85	0.87	179
32	0.90	0.94	0.92	147
33	0.96	0.83	0.89	170
34	0.78	0.72	0.75	168
35	0.84	0.82	0.83	138
accuracy			0.74	5400
macro avg	0.74	0.75	0.74	5400
weighted avg	0.74	0.74	0.74	5400

Table 13: Classification Report:Real Sample

Class	Precision	Recall	F1-Score	Support
0	0.80	0.36	0.50	11
1	1.00	1.00	1.00	3
2	1.00	0.67	0.80	6
3	0.83	0.71	0.77	7
4	0.60	0.50	0.55	6
5	0.25	0.33	0.29	3
6	1.00	0.75	0.86	4
7	1.00	1.00	1.00	7
8	0.18	0.29	0.22	7
9	0.20	0.50	0.29	2
10	0.17	0.10	0.12	10
11	1.00	0.75	0.86	4
12	0.50	0.67	0.57	6
13	0.75	0.75	0.75	4
14	0.40	1.00	0.57	4
15	1.00	1.00	1.00	2
16	0.86	0.86	0.86	7
17	0.83	0.83	0.83	6
accuracy			0.62	99
macro avg	0.69	0.67	0.66	99
weighted avg	0.68	0.62	0.62	99

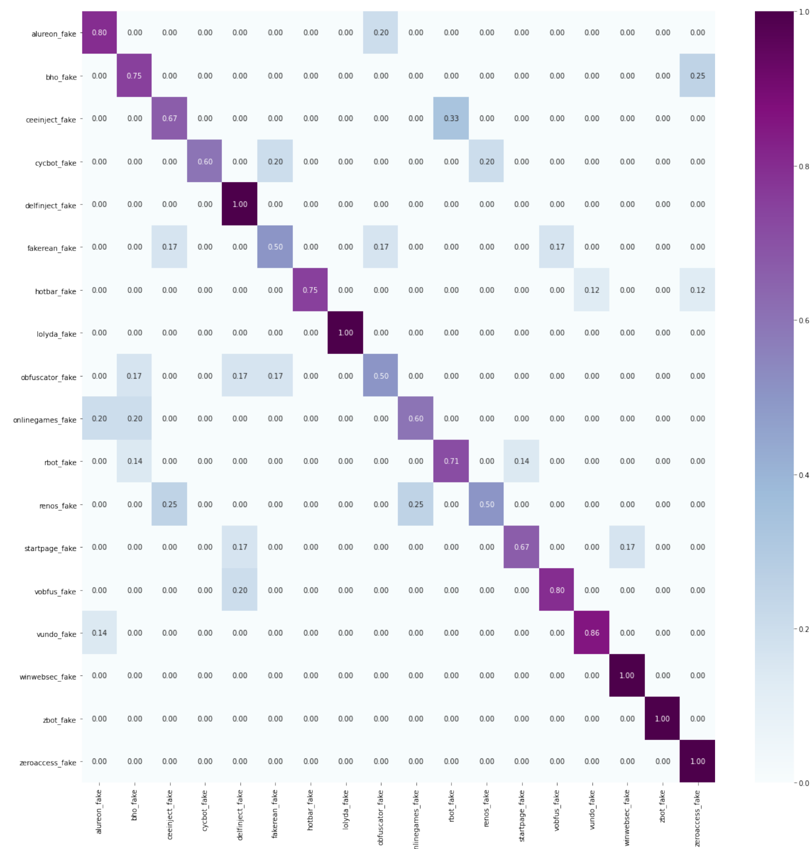

Figure 18: Confusion Matrix: Fake Samples Classification

Table 14: Classification Report: Fake Samples

Class	Precision	Recall	F1-Score	Support
0	0.80	0.67	0.73	6
1	0.75	0.50	0.60	6
2	0.67	0.50	0.57	4
3	0.60	1.00	0.75	3
4	1.00	0.62	0.77	8
5	0.50	0.60	0.55	5
6	0.75	1.00	0.86	6
7	1.00	1.00	1.00	8
8	0.50	0.60	0.55	5
9	0.60	0.75	0.67	4
10	0.71	0.83	0.77	6
11	0.50	0.67	0.57	3
12	0.67	0.80	0.73	5
13	0.80	0.80	0.80	5
14	0.86	0.86	0.86	7
15	1.00	0.86	0.92	7
16	1.00	1.00	1.00	6
17	1.00	0.60	0.75	5
accuracy			0.77	99
macro avg	0.76	0.76	0.75	99
weighted avg	0.80	0.77	0.77	99

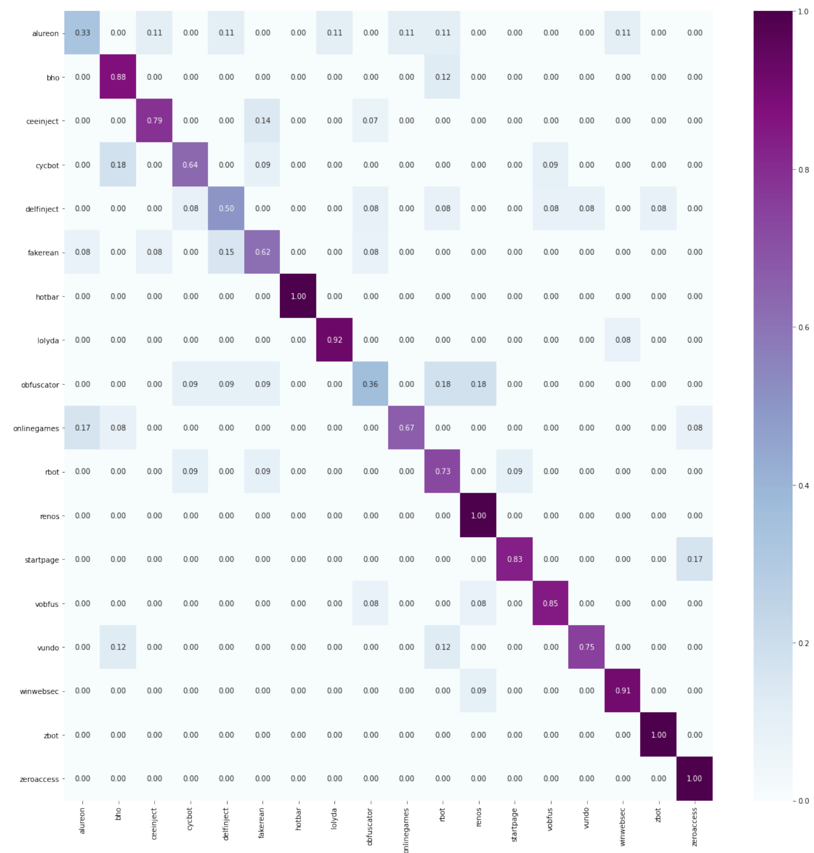

Figure 19: Confusion Matrix: Poisoned Samples Classification

Table 15: Classification Report: Poisoned Samples Classification

Class	Precision	Recall	F1-Score	Support
0	0.33	0.50	0.40	6
1	0.88	0.64	0.74	11
2	0.79	0.85	0.81	13
3	0.64	0.70	0.67	10
4	0.50	0.60	0.55	10
5	0.62	0.62	0.62	13
6	1.00	1.00	1.00	12
7	0.92	0.92	0.92	13
8	0.36	0.50	0.42	8
9	0.67	0.89	0.76	9
10	0.73	0.57	0.64	14
11	1.00	0.79	0.88	19
12	0.83	0.83	0.83	6
13	0.85	0.85	0.85	13
14	0.75	0.86	0.80	7
15	0.91	0.83	0.87	12
16	1.00	0.91	0.95	11
17	1.00	0.83	0.91	12
accuracy			0.77	199
macro avg	0.76	0.76	0.76	199
weighted avg	0.80	0.77	0.78	199

CHAPTER 5

Conclusion & Future Work

5.1 Conclusion

In this research, we were able to convert malware executable files into images and analyzed resizing techniques and their impact on classification of data. We also created an image from the headers of the malware files by reading the start bytes and found that they give optimum results. To try to break this classifier, we generated fake samples using AC-GAN and found that the CNN model's classification accuracy deteriorated with the inclusion of fake samples. By retraining the model with a poisoned dataset, we obtain better classification results than before.

5.2 Future Work

Fake malwares can be generated using other techniques like HMM as well. It could be worth exploring to create images from the fake executable byte codes. Other techniques like data obfuscation can also be performed to generate similar samples of data from a particular family. It can be interesting to see what can happen if we introduce fake samples generated from multiple techniques in the same dataset.

The classification techniques used in this research project were limited to CNN to maintain the consistency in results we obtain so that it is easier to analyze the effects of introduction of fake samples and other image generation techniques. Other techniques like Multi-Layer Perceptron (MLPs) and ResNets have also shown great classification results on other datasets.

GANs are themselves a broad area of research. Other GAN models like DC-GAN (Deep Convolutional GANs), Progressive GANs and Cycle GANs can be used to generate the fake images. The results of AC-GAN were based on BinaryCrossEntropy Loss. There are other loss functions like SSIM (Structural Similarity Index Measurement) which can also be explored in training AC-GAN models.

LIST OF REFERENCES

[1] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, ‘‘Learning and classification of malware behavior,’’ in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, 2008, pp. 108--125.

[2] D. Lin and M. Stamp, ‘‘Hunting for undetectable metamorphic viruses,’’ *Journal in computer virology*, vol. 7, no. 3, pp. 201--214, 2011.

[3] L. Liu, B.-s. Wang, B. Yu, and Q.-x. Zhong, ‘‘Automatic malware classification and new malware detection using machine learning,’’ *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 9, pp. 1336--1347, 2017.

[4] L. Liu, B.-s. Wang, B. Yu, and Q.-x. Zhong, ‘‘Automatic malware classification and new malware detection using machine learning,’’ *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 9, pp. 1336--1347, 2017.

[5] T. Joachims, ‘‘Making large-scale svm learning practical,’’ Technical report, Tech. Rep., 1998.

[6] B. Zhang, C. Quan, and F. Ren, ‘‘Study on cnn in the recognition of emotion in audio and images,’’ in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).* IEEE, 2016, pp. 1--5.

[7] K. Borders, X. Zhao, and A. Prakash, ‘‘Siren: Catching evasive malware,’’ in *2006 IEEE Symposium on Security and Privacy (S&P’06).* IEEE, 2006, pp. 6--pp.

[8] M. Gheorghescu, ‘‘An automated virus classification system,’’ in *Virus bulletin conference*, vol. 2005. Citeseer, 2005, pp. 294--300.

[9] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, ‘‘Fast malware classification by automated behavioral graph matching,’’ in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, 2010, pp. 1--4.

[10] Y. Ye, D. Wang, T. Li, and D. Ye, ‘‘Imds: Intelligent malware detection system,’’ in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 1043--1047.

[11] N. Runwal, R. M. Low, and M. Stamp, ‘‘Opcode graph similarity and metamorphic detection,’’ *Journal in computer virology*, vol. 8, no. 1, pp. 37--52, 2012.

[12] 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: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2016904.2016908

[13] L. Liu and B. Wang, "Malware classification using gray-scale images and ensemble learning," in *2016 3rd International Conference on Systems and Informatics (ICSAI)*. IEEE, 2016, pp. 1018--1022.

[14] Y. Zhang, Q. Huang, X. Ma, Z. Yang, and J. Jiang, "Using multi-features and ensemble learning method for imbalanced malware classification," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 965--973.

[15] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 2482--2486.

[16] 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.

[17] M. Jain, W. Andreopoulos, and M. Stamp, "Convolutional neural networks and extreme learning machines for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 3, pp. 229--244, 2020.

[18] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," 2017.

[19] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," 2017.

[20] D. MacRae, "viruses to be on the alert for in 2014," *Computer Business Review*, 5.

[21] M. S. Intelligence. "Adware:win32/bho.g." 2009. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/BHO.DW&threatId=-2147325123

[22] M. S. Intelligence. "Win32/ceeinject." 2007. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/CeeInject&threatId=-2147369055

[23] M. S. Intelligence. “Win32/cycbot.” 2011. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Cycbot&threatId=

[24] M. S. Intelligence. “Win32/delfinject.” 2008. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDropper:Win32/DelfInject&threatId=-2147367014

[25] M. S. Intelligence. “Win32/fakerean.” 2011. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/FakeRean&threatId=

[26] M. S. Intelligence. “Win32/hotbar.” 2006. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Hotbar&threatId=6204

[27] M. S. Intelligence. “Win32/lolyda.” 2008. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Lolyda&threatId=

[28] M. S. Intelligence. “Win32/obfuscator.” 2011. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Obfuscator&threatId=

[29] M. S. Intelligence. “Win32/onlinegames.” 2015. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/OnLineGames&threatId=

[30] M. S. Intelligence. “Win32/rbot.” 2005. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Rbot&threatId=

[31] M. S. Intelligence. “Win32/renos.” 2006. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Renos&threatId=16054

[32] M. S. Intelligence. “Win32/startpage.” 2011. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Startpage&threatId=15435

[33] M. S. Intelligence. “Win32/vobfus.” 2010. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Vobfus&threatId=-2147335447

[34] M. S. Intelligence. “Win32/vundo.” 2009. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Vundo&threatId=100135

[35] M. S. Intelligence. "Win32/winwebsec." 2010. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Winwebsec&threatId=

[36] M. S. Intelligence. "Win32/zbot." 2014. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=MSIL/Zbot&threatId=

[37] S. Hittel and R. Zhou, "Trojan. zeroaccess infection analysis," *Symantec Corporat., Mountain View, CA, USA, Tech. Rep*, 2012.

[38] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of iot malware based on image recognition," in *2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)*, vol. 2. IEEE, 2018, pp. 664--669.

[39] A. Walenstein, D. J. Hefner, and J. Wichers, "Header information in malware families and impact on automated classifiers," in *2010 5th International Conference on Malicious and Unwanted Software.* IEEE, 2010, pp. 15--22.

[40] M. Stamp, *Introduction to machine learning with applications in information security.* CRC Press, 2017.

[41] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[42] C. M. Bishop *et al.*, *Neural networks for pattern recognition.* Oxford university press, 1995.

[43] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International conference on rough sets and knowledge technology.* Springer, 2014, pp. 364--375.

[44] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," in *Asian conference on computer vision.* Springer, 2016, pp. 189--204.

[45] K. A. Reference. "Droput layers." [Online]. Available: https://keras.io/api/layers/regularization_layers/dropout/

APPENDIX

Additional Results

A.1 Binary Classification Results

As malware belonging to obfuscator, vundo and alureon families had a low accuracy among the multi-class classification results, it implies that the similarity between these malware is more. Hence, to bolster their classification, a binary classication between them is performed. The subsequent sub-sections illustrate the findings of these experiments.

A.1.1 Obfuscator & Rbot

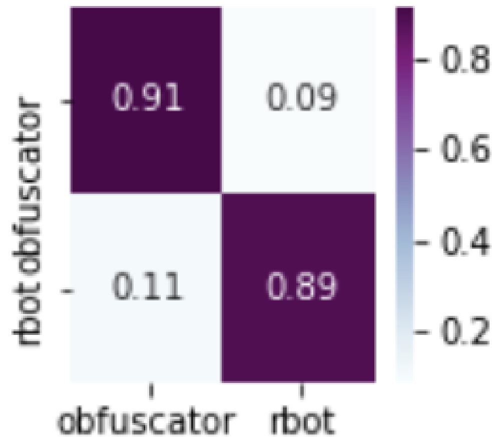

Figure A.20: Confusion Matrix: Obfuscator & Rbot

Table A.16: Classification Report: Obfuscator & Rbot

Class	Precision	Recall	F1-Score	Support
0	0.91	0.92	0.91	431
1	0.89	0.87	0.88	305
accuracy			0.90	736
macro avg	0.90	0.89	0.90	736
weighted avg	0.90	0.90	0.90	736

A.1.2 Obfuscator & Vundo

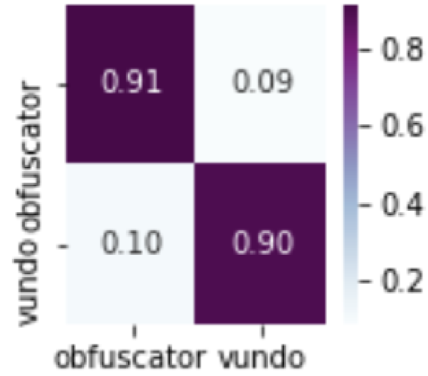

Figure A.21: Confusion Matrix: Obfuscator & Vundo

Table A.17: Classification Report: Obfuscator & Vundo

Class	Precision	Recall	F1-Score	Support
0	0.91	0.88	0.90	451
1	0.90	0.93	0.91	518
accuracy			0.91	969
macro avg	0.91	0.90	0.90	969
weighted avg	0.91	0.91	0.90	969

A.1.3 Alureon & Vundo

Table A.18: Classification Report: Alureon & Vundo

Class	Precision	Recall	F1-Score	Support
0	0.95	0.94	0.94	389
1	0.96	0.97	0.96	547
accuracy			0.95	936
macro avg	0.95	0.95	0.95	936
weighted avg	0.95	0.95	0.95	936

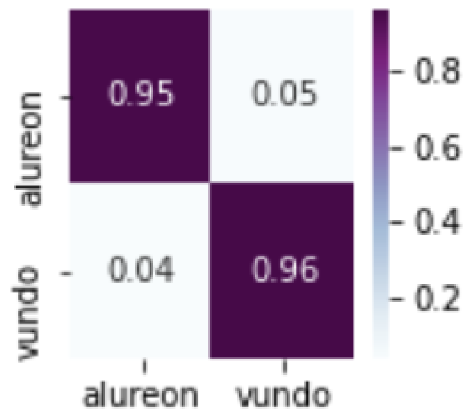

Figure A.22: Confusion Matrix: Alureon & Vundo

A.2 Loss Graphs

This section consists of the loss graphs of each experiment.

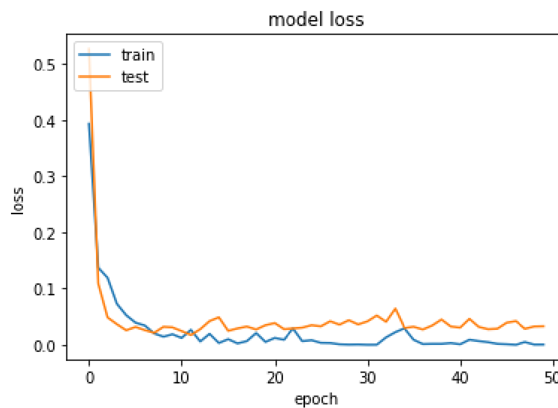

Figure A.23: CNN: Winwebsec & Zbot Loss

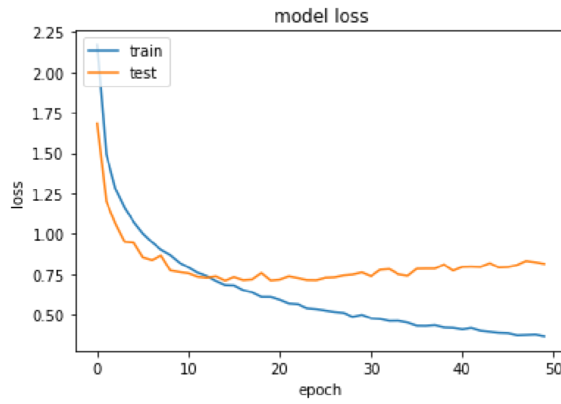

Figure A.24: CNN: Full Image Dataset Loss

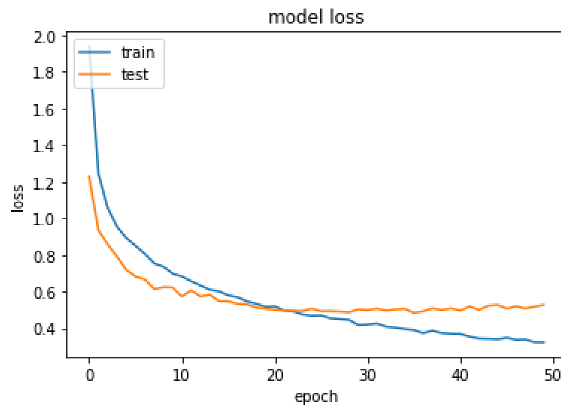

Figure A.25: CNN: Hamming Interpolation Loss

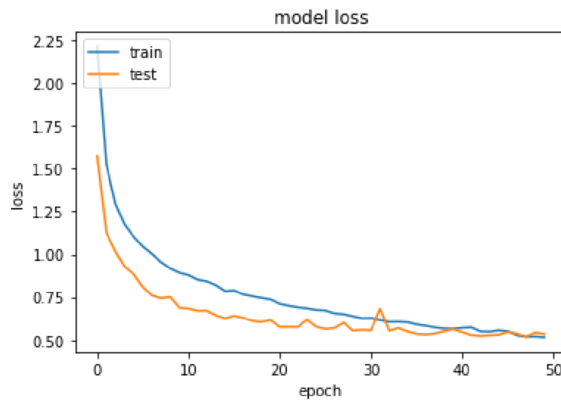

Figure A.26: CNN: Bicubic Interpolation Loss

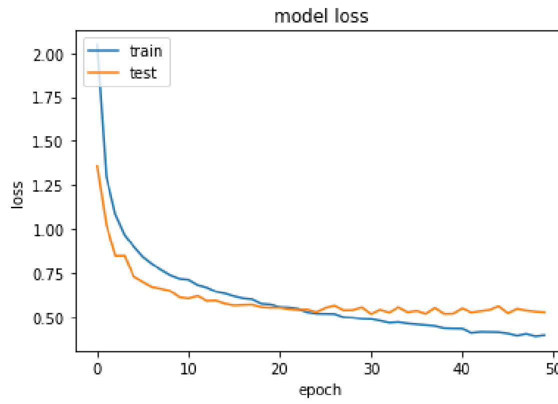

Figure A.27: CNN: Lanczos Interpolation Loss

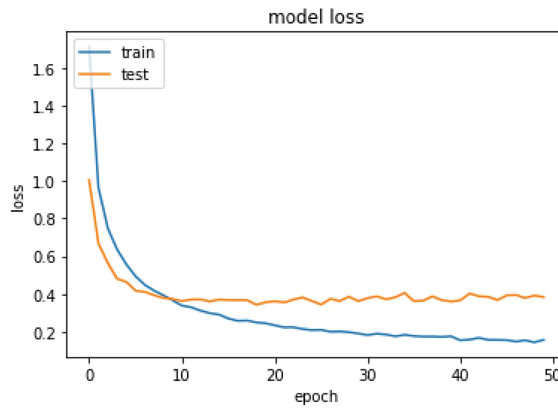

Figure A.28: CNN: Header Image Data Loss

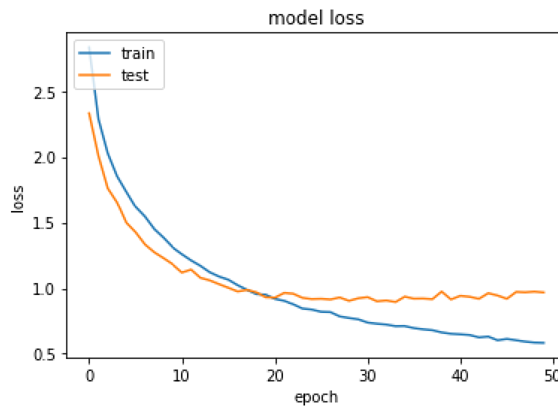

Figure A.29: CNN: Consolidated Dataset Loss

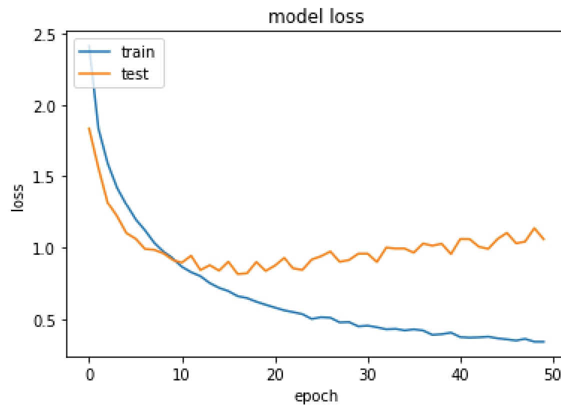

Figure A.30: CNN: Poisoned Dataset Loss

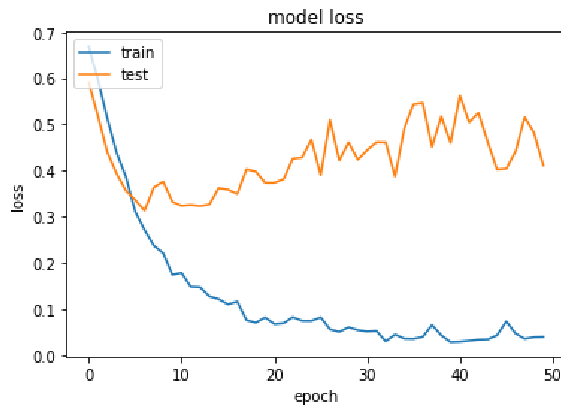

Figure A.31: CNN: Obfuscator & Rbot Loss

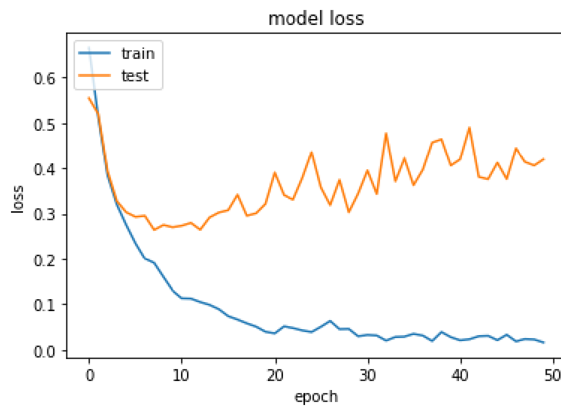

Figure A.32: CNN: Obfuscator & Vundo Loss

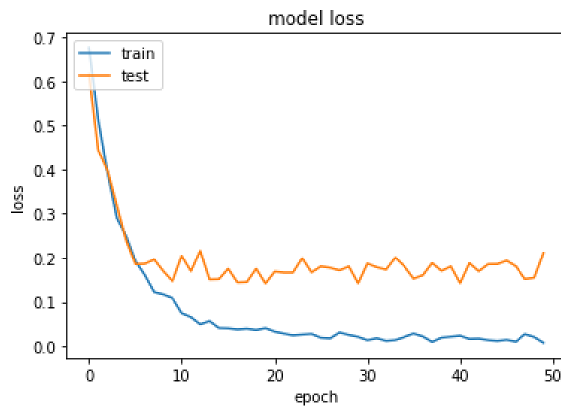

Figure A.33: CNN: Alureon & Vundo Loss