Spring 2022

Generative Adversarial Networks for Image-Based Malware Classification

Huy Nguyen
San Jose State University

Generative Adversarial Networks for Image-Based Malware Classification

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

Huy Nguyen

May 2022

The Designated Project Committee Approves the Project Titled

Generative Adversarial Networks for Image-Based Malware Classification

by

Huy Nguyen

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2022

Dr. Mark Stamp	Department of Computer Science
Dr. Fabio Di Troia	Department of Computer Science
Dr. Genya Ishigaki	Department of Computer Science

ABSTRACT

Generative Adversarial Networks for Image-Based Malware Classification

by Huy Nguyen

Malware detection and analysis are important topics in cybersecurity. For efficient malware removal, determination of malware threat levels, and damage estimation, malware family classification plays a critical role. With the rise in computing power and the advent of cloud computing, deep learning models for malware analysis has gained in popularity. In this paper, we extract features from malware executable files and represent them as images using various approaches. We then focus on Generative Adversarial Networks (GAN) for multiclass classification and compare our GAN results to other popular machine learning techniques, including Support Vector Machine (SVM), XGBoost, and Restricted Boltzmann Machines (RBM). We also evaluate the utility of the GANs generative models for adversarial attacks on image-based malware detection. We find that the AC-GAN discriminator is competitive with other machine learning techniques.

ACKNOWLEDGMENTS

I want to thank my advisor Dr. Mark Stamp for countless inspiring lectures about Computer Security and Machine Learning. I had an amazing opportunity to take CS 166, Information Security, by Dr. Stamp 6 years ago as an undergraduate and I knew my focus would be in the Security field for my master program. I wanted to know more about Machine Learning after graduating in 2017 so I applied for San Jose State University Master Program in 2020 and surprisingly Dr. Stamp taught CS 271, Machine Learning in my first semester. Even though, there were a lot of distractions and mistakes on my way to finish this project, Dr. Stamp has patiently guided me to complete and overcome them step by step.

I want to thank Dr. Fabio Di Troia and Dr. Genya Ishigaki for being in my committee. Dr. Fabio Di Troia also gave me project advices and the dataset for multiple experiments.

I also want to thank my family and friends for their encouragement, empathy as without them I could not finish the project.

Thank you.

TABLE OF CONTENTS

CHAPTER

CHAPTER 1

Introduction

The Covid-19 pandemic, which has run amok worldwide for two years, has drastically increased the trend of working from home. The remote work environment has also pushed another trend: increasing cyber attacks, including phishing, data breaches, and malware. According to CSO [1], in the second quarter of 2020, as compared to the same period a year earlier, cloud security incidents increased by 188%, ransomware attacks grew by over 40%, and email malware attacks were up by 600%.

Malware, short for "malicious software", consists of computer programs that are written to cause harm to computer and Internet users [2]. The most common types of malware are botnet, rootkit, spyware, Trojan, worm and spyware. Malware can be used to steal information, utilize hardware, cause disruption for financial or reputational gain, or other unauthorized activity. Malware defense is an ongoing battle with multiple layers: preventing malware from entering, alerting users that a system is compromised, removal of malware from compromised systems, and so on.

There has been considerable research into malware detection and classification. In recent years, malware classification based on machine learning has become a primary focus of such research. With more computing power from graphic processing units (GPU) and Google tensor processing units (TPU), which are specialized for machine learning techniques and simple feature extraction, costly deep learning image-based malware detection techniques have become viable options.

An image can be derived directly from the byte sequence of an executable file without executing (or emulating) or otherwise pre-processing the data to extract features. Powerful image-based techniques, including Convolutional Neural Networks (CNN) and Generative Adversarial Networks (GAN) have been used to classify malware samples with impressive results.

Malware detection is an arms race between detectors and malware writers, where each side tries to develop new and innovative ways to defeat the other side. According to the TV series Criminal Minds: "To catch a criminal, you have to think like one. The way to a criminal is through his mind." Following this logic, malware analysts must consider future attack strategies of malware developers. In one type of adversarial attack, a malware writer attempts to contaminate the training data, so that the resulting model is less effective. A possible approach to such an adversarial attack is to generate "deep fake" malware images to pollute the training dataset. From this perspective, we consider ways to make our models more robust at detecting fake malware images and thereby prevent possible future attacks.

GANs have been used to generate realistic fake images. Karras et. al. ,researchers at Nvidia, developed StyleGAN, a style-based architecture for GAN and the StyleGAN generator was 20% better than the traditional generator [3]. StyleGAN was also used to create the trending website "thispersondoesnotexist.com". Thus, we consider the utility of GANs for adversarial attacks on image-based malware analysis. MalGAN is a GAN technique that is designed specifically to deal with malware images [4, 5].

In addition to malware detection, malware classification is important, as it enables us to estimate the damage, determine the threat level, and to provide protection specific to the malware family. In this research, we employ auxiliary classifier GAN (AC-GAN) for multi-class classification of malware families and compare with other machine learning models, including Support Vector Machine (SVM), K-Nearest Neighbors (K-NN), multilayer perceptron (MLP), Random Forest (RF), Restricted Boltzmann Machines (RBM), XGBoost, and the deep residual network Resnet152 [6, 7, 8, 9, 10, 11]. We also develop SVM model to test the quality of fake images generated by AC-GAN generative model. The dataset that we use consists of more than 26,000 malware executables from 20 different families [12].

The remainder of this paper is organized as follows. In Chapter 2, we discuss related work and introduce the machine learning models used in our research. For Chapter 3, we analyze our dataset and discuss our evaluation criteria. Next, in Chapter 4, we introduce multiple ways to extract images from malware executables and our model implementations. In Chapter 5, we present our experimental results, with graphs, tables and analysis. Chapter 6 concludes the report and we discuss future work.

CHAPTER 2

Background

There are more and more malware families being developed everyday and they are developed by professional groups with tremendous resources. It was not too long ago that the SolarWinds zero day attack caused damage to several government agencies [13]. The attack was caused by a state-sponsored group and put the whole cyber security industry on high alert. The SolarWinds attack reminded everyone of the importance of malware defense and malware detection in particular. There are two types of malware detection: signature-based and anomaly-based. Signature-based malware detection keeps certain characteristics of previously-seen malware in a dictionary and prevents future attacks. There are three main disadvantages for this approach: the size of the dictionary is not scalable, new malware or zero day attacks cannot be detected, signatures can be changed using obfuscation to avoid detection. Anomaly-based detection sets the bar which softwares are normal and detect abnormal behaviors. Machine learning techniques are a subset of anomaly-based detection. The main disadvantage of anomaly-based detection is the training dataset can be polluted, making anomaly features become normal overtime. Our research's purpose is also based on this technique, generate fake malware images to trick sophisticated machine learning techniques.

The following section explains what others have done related to image-based malware analysis and machine learning techniques.

2.1 Related Work

As a first step into the topic of image-based analysis, Jain and Stamp [14] did a thorough analysis with different image sizes and used CNN and ELM to classify malware. The paper has a lot of details about project background, implementation and results. ELM was also experimented with different hyperparameters. Jain and

Stamp advised to look deeper into one type of malware, and try different techniques for image extraction such as zero padding and GIST descriptors of images for future work.

In another paper, Nagaraju and Stamp [15] worked on image-based malware analysis on a GAN architecture called Auxiliary-Classifier GAN(AC-GAN). They experimented with different image sizes from 32×32, 64×64 to as big as 512×512, and grayscale images were extracted and truncated from executable files to the desired sizes. Other than AC-GAN, CNN and Extreme Learning Machine (ELM) were used and CNN achieved impressive results in detecting fake images. For future work, they advised researchers to work on novel techniques like VG-199 or ResNet152.

Xiao et. al. [16] introduced a novel framework called MalCVS (Malware Classification using Colab image generation, VGG16 and Support Vector Machine (SVM)). The images were generated using Colab image generation, similar to grayscale images with thick colored lines to identify each section in the executable files. The images then were passed through VGG16 for feature extraction and fed to multi-class SVM for classification. The MalCVS framework achieved impressive results with 98.94% accuracy and F1-score at 97.91%. The MalCVS framework started to depart from grayscale images and enter the border of colored image representation of malware.

More data, more computing power, more layers seem like the trend for deep learning techniques recently. Let us say goodbye to the black and white images, and take a tour to the colorful world of colored images extracted from malware. Both Vasan et.al. [17] and Singh et. al. [18] generated a color map and used byte sequence from the executable files to represent colored images. Both papers also use similar techniques like CNN and Residual Neural Network (ResNet-50). Singh et. al. achieved impressive results with MalImg dataset: accuracy of 98.10% using ResNet-50, while Vasan et. al. scored 98.82% accuracy using Fine-tuned Convolutional Neural Network

Architecture. They concluded that red, blue and green (RGB) representation captured more pattern information, therefore achieving better results with the same dataset.

The idea of generating fake features of malware is not new. Hu et. al. and Kawai [4, 5] proposed MalGAN to bypass black-box machine learning based detection models. MalGAN took the output of the black-box models and used GAN to generate binary features based on the mixes of samples labeled as 1 and samples labeled as 0. MalGAN was able to decrease the detection rate to nearly 0. The process of training MalGAN was fast and efficient, made it hard for the black-box models to prevent future attacks.

In the area of multiclass classification, Fu et. al. [19] achieved impressive results with the accuracy score at 97.47% and F-measure at 96.85% in categorizing 15 different malware families. They focused heavily in extracting features from malware executables and combined global and local features. Color features were extracted using Portable Executable (PE) format parser and the colored images were built using RGB layers. The executable files were divided into sections and each section data such as entropy values, byte sequences and relative size were put together to represent each layer of RGB channels. The resulting images were colorful and we could see the shapes of data and code sections in the malware files. For future work, they suggested that deep learning models such as CNN would be developed for malware classification as CNN is excellent working with images.

Farhat and Rammouz [20] experimented with several pretrained Deep Convolutional Models and all of the pretrained models such as: VGG16, Resnet50, Resnet152 and mobilenet scored an accuracy score of 94% after just one training epoch for 9-class malware classification problem. The pretrained models' results are impressive considering we can customize input size and class labels to take advantage of powerful image-based models. We can use the pretrained models for most computer vision

problems and save a lot of training time.

We experimented with multiple machine learning techniques: SVM, K-NN, MLP, RBMs, two ensemble techniques: RF and XGBoost, and three deep learning models: DC-GAN, AC-GAN and Resnet152. In the next section, each technique's advantages and disadvantages are discussed and we will decide which technique is worth the tradeoff for image-based malware classification.

2.2 Machine Learning Models

K-Nearest Neighbors Classifier uses K nearest neighbors from the training set to classify new data. K-NN is simple, easy to understand and perform relatively well. However with high resolution images like $128 \times 128 \times 3$, the feature vector is 49152 byte long, making K-NN slower to compute distance. Support Vector Machine (SVM) or Support Vector Classifier (SVC) uses a hyperplane to separate and classify data. Figure 1 shows the 2D version of SVM hyperplane: a straight line separating two classes of data. SVM performs well, scored 93.20% on a 25-class classification problem [17]. However, similar to K-NN, when the feature space is large, SVM is quite slow to train and evaluate. RBMs can extract non-linear features from images and combine with a linear model like Logistic Regression can perform well: such as 94% accuracy for digit classification problem [21]. Both RBMs and Logistic Regression are simple and take a short time to train. We also have two ensembles: Random Forest and XGBoost. An ensemble is a group of models that combine together and function as a whole. The usual problem of ensemble is overfitting, XGBoost also requires large memory as the dataset needs to be preprocessing into proper form before training.

2.3 Deep Learning Models

There are a lot of GAN variations such as ProGAN, StyleGAN, and so on. We experiment with two basic GANs: DC-GAN for unsupervised generative model and

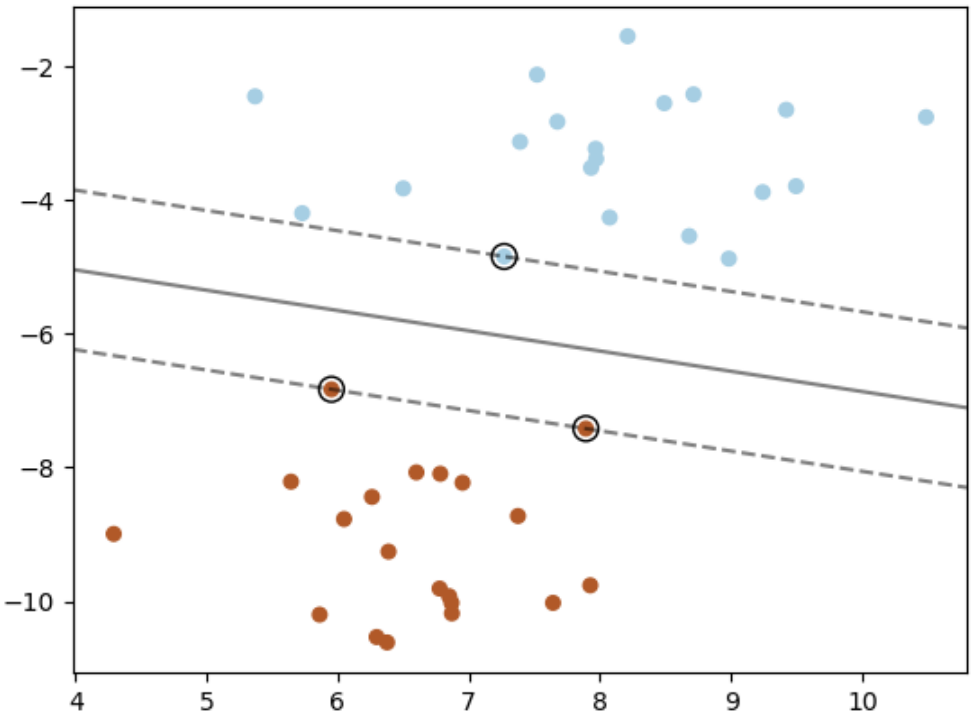


Figure 1: SVM Hyperplane [22]

AC-GAN for multi-class discriminator. GANs or Generative Adversarial Networks are a set of two models: the generator and the discriminator. The two models compete with each other and improve together when training. Another deep learning model we want to experiment with is Resnet152. Resnet152 is a pretrained deep residual network. Resnet152 can be customized to receive different input size, class labels and requires a few epochs to get good results.

CHAPTER 3

Methodology

The goal of this research is to examine both the discriminator and the generator models of GANs and compare them with various machine learning techniques. The discriminator can be used for a multiclass classification of malware families and binary classification of fake/real images. The generator of different GANs' architectures are compared against each other using evaluation models.

3.1 Dataset

Our dataset MalExe consists of 26,412 malware executable files from 20 different families. Each family has between 842 files to 3651 files. Figure 2 shows the exact number of files each family has. We have more samples from three families: Vundo, Winwebsec, and Zeroaccess while other families have similar amounts of samples. Table 1 shows the overview descriptions of the 20 malware families.

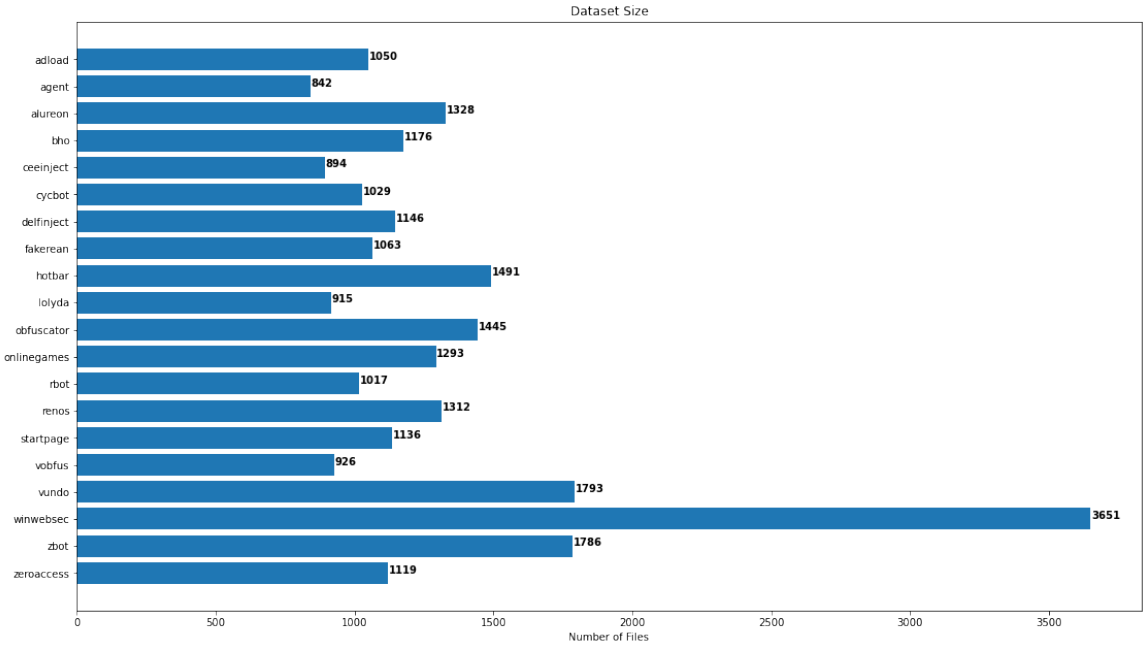


Figure 2: MalExe Dataset Overview

Let us take a look at the file sizes. Figure 3 shows the histogram of file sizes for

Table 1: Malware Families

Family	Type	Description
Adload	Adware	Shows ads, poses high threat [23]
Agent	General	Performs malicious actions [24]
Alureon	Trojan	Steals information [25]
Bho	Trojan	Steals information, redirects web sites [26]
Ceeinject	Virtool	Obfuscates itself to hide purposes [27]
Cycbot	Backdoor/Trojan	Provides backdoor access [28]
Delfinject	PWS	Steals passwords [29]
Fakerean	Rogue	Raises false alarms to make money [30]
Hotbar	Adware	Displays advertisements [31]
Lolyda	PWS	Monitors network activities [32]
Obfuscator	Virtool	Obfuscates itself to hide purposes [33]
Onlinegames	PWS/Trojan	Injects malicious files, steals information [34]
Rbot	Backdoor/Trojan	Provides backdoor access[35]
Renos	Trojan	Downloads unwanted softwares [36]
Startpage	Trojan	Changes internet browser homepage [37]
Vobfus	Worm	Downloads and spreads malwares [38]
Vundo	Trojan Downloader	Uses advanced defensive and stealth techniques [39]
Winwebsec	Rogue	Raises false alarms for money [40]
Zbot	Trojan	Steals information, gives access to hackers [41]
Zeroaccess	Trojan	Disables security features [42]

the whole dataset. We can see that the majority of files are smaller than 200KB while we have a fair amount of files between 200-500KB and a small number of big files with larger than 500KB size. Most image-based analysis choose an image size of 256×256 or 224×224, that would equal to 64KB or 49KB respectively. We can see that most of our files have enough data to extract and transform into the desired images.

Looking at each family in Figure 4, the average file size can help us see the difference. Based on the file size alone, we can easily distinguish Lolyda from other big-size families. Lolyda has an average file size of 35KB, while the average file size of adload is 602KB and startpage has the biggest average size at 1042KB. For machine learning techniques, we need fixed size input. With that much difference in size, there

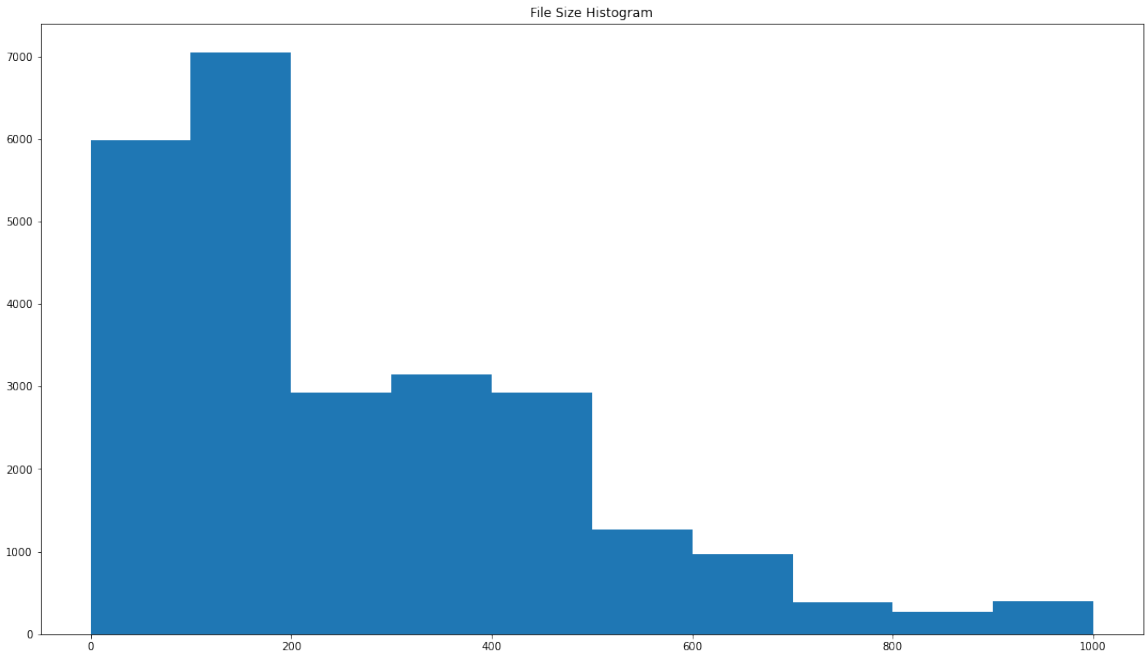


Figure 3: MalExe File Size Histogram

are multiple ways we can engineer to preprocess the data. One reasonable approach is to extract a fixed amount of bytes and filter out smaller files [14, 15]. However, we would not have enough files for the Lolyda family to experiment with. Another interesting way is to have variable sizes based on image size, then resize to the desired width and height.

In the next section, we discuss image extraction from executable files and experiment with different file sizes, grayscale and colored images.

3.2 Deep Convolutional GAN (DC-GAN)

As an unsupervised architecture, DC-GAN has an advantage over other architectures with unlabelled data. The resulting models from DC-GAN can be used to compare performance with other GAN architectures in binary classification of real/fake images. DC-GAN was first introduced by Radford A. and Metz L. [43] in 2015 as a way of unsupervised representation learning. Figure 5 visualizes convolutional layers

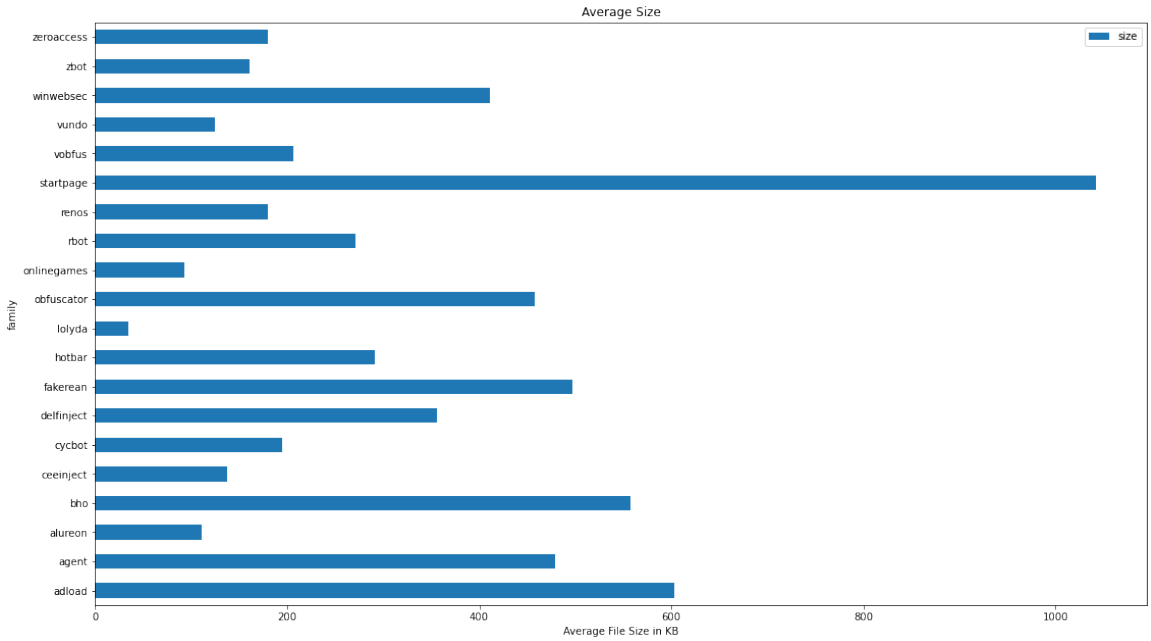


Figure 4: MalExe Average Size Per Family

of DC-GAN with no fully connected or pooling layers. A noise vector consists of 100 random numbers fed into the model and generate images based on training data. The basic architecture of DCGAN has four convolutional layers and the output is expected to be $64 \times 64 \times 3$. We can twist the settings in the convolutional layers to work with our grayscale $128 \times 128 \times 1$, $256 \times 256 \times 1$ or even RGB images $128 \times 128 \times 3$.

3.3 Auxiliary-Classifier GAN (AC-GAN)

There are multiple researches showing that AC-GAN performs well with multiclass data [15, 44, 45]. The intuition is clear as we have the class labels and we did not use them in DC-GAN as we just feed multiple images from different families into the models in an unsupervised representation. Using the extra data, the class labels, would also help us to generate the desired family when needed. The discriminator can also be used to solve the multiclass classification problems which is our main focus of this research. As we can see in Figure 6, input C representing class labels is fed into both the generator and the discriminator. There are two outputs for the discriminator:

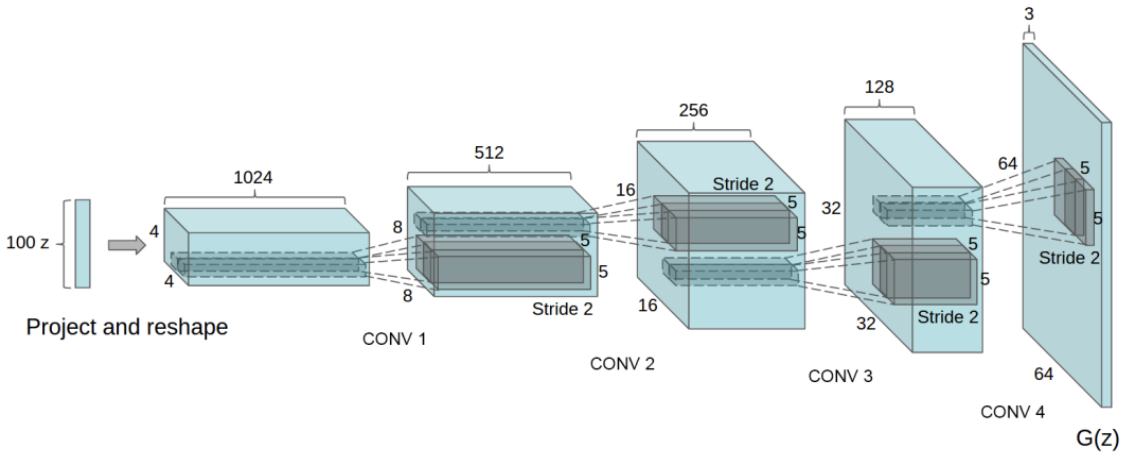


Figure 5: DC-GAN Generator [43]

validity of the image and the class label and the discriminator is trained based on the two outputs with two loss functions.

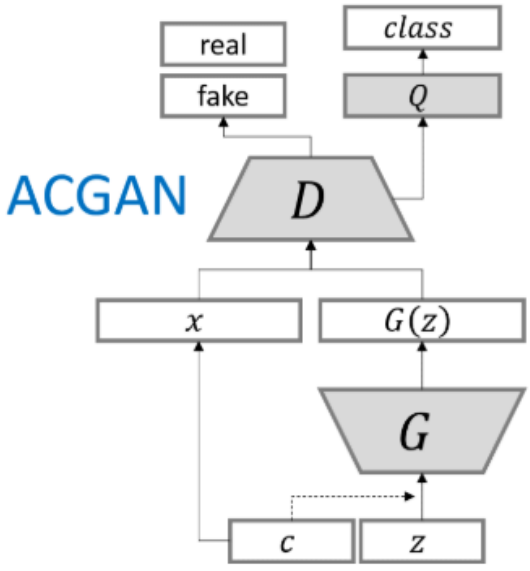


Figure 6: AC-GAN Architecture [46]

3.4 RBM

For RBM, we use BernoulliRBM, the implementation details can be found in [47]. The RBM acts as a layer to extract meaningful smaller images from malware images then the resulting images are fed into a Logistic Regression layer for multiclass classification. We also add a layer of AutoEncoder in front of the RBM layer to reduce noises in images. Figure 7 shows the RBM overview architecture.

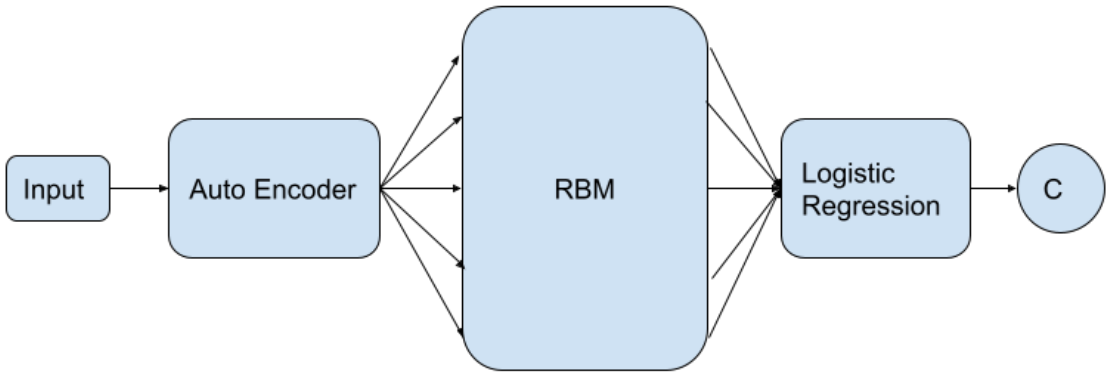


Figure 7: RBM Architecture

3.5 Resnet152

We use Tensorflow and Keras package: Resnet152v2, the pretrained weight is "ImageNet". To change the input size and class label, we add one Dense layer after the base Resnet output, freeze all the base model's weight and train the extra layer. After that, we unfreeze half of the layers of the pretrained models and train for several epochs.

3.6 Evaluation Plans

For the multiclass classification problem, to distinguish each malware family we use the AC-GAN discriminator model. We compare the performance of AC-GAN discriminator with other machine learning models like SVM, RBM and XGBoost. The

evaluation metrics are accuracy score, precision, recall and f1-score calculated based on True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN) as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$
$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

The Receiver Operating Characteristic (ROC) curve is a good way to of visualizing a classifier's performance; the area under the ROC curve(AUC) should be used as a single point of measurement to compare performance of various classifiers [49]. Figure 8 shows examples of the ROC curve and how to measure performance. After training DC-GAN and AC-GAN, we generate some fake malware images from the two architectures and use CNN, SVM to compare them with the performance of the generative models. The metrics that we use are accuracy score and AUC score.

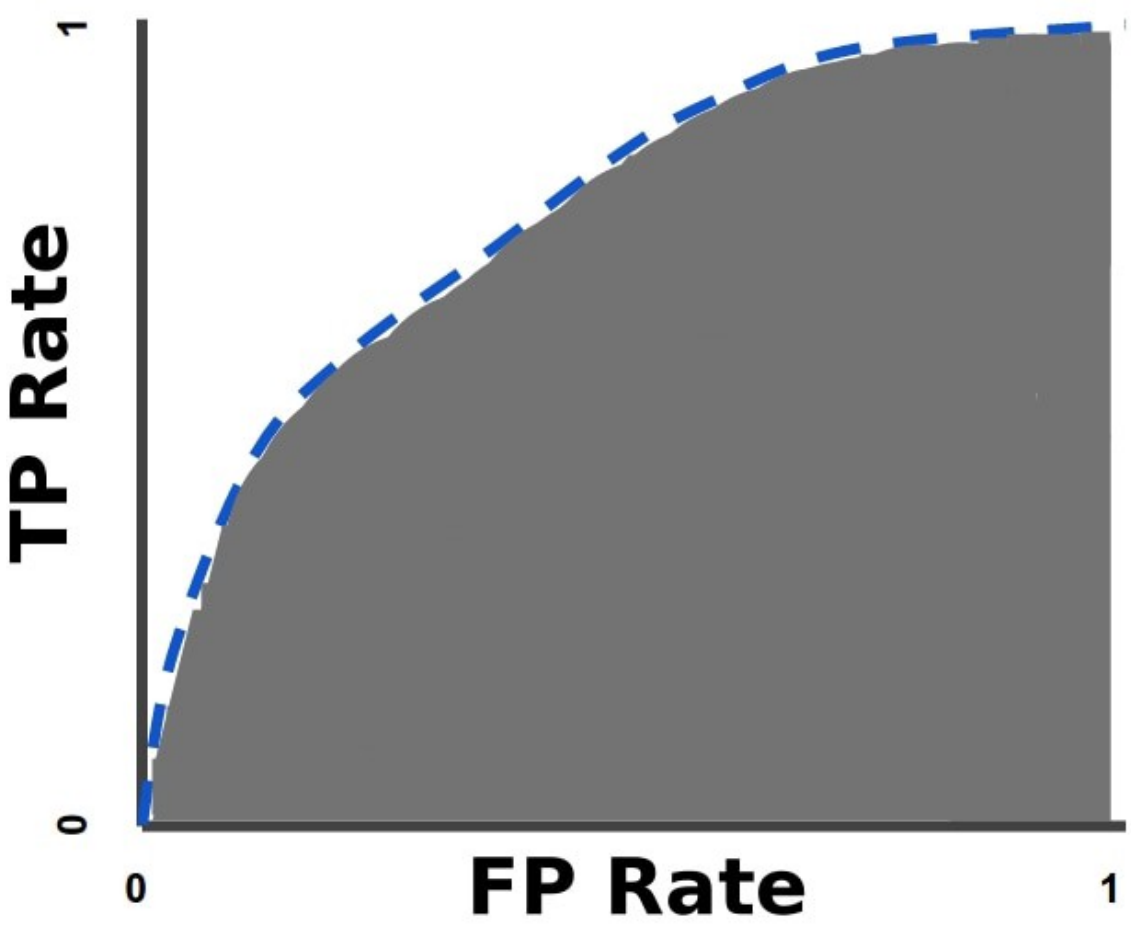


Figure 8: ROC Curve and Area Under ROC Curve [48]

CHAPTER 4

Implementation

The implementation can be divided into four sections: dataset overview, images and features extraction, data processing, and model architectures. We use Google Colab Pro+ to utilize Google's computing power to train multiple models using large datasets. Table 2 shows the runtime and memory settings for each model. XGBoost requires a lot of memory and we ran into issues using 35GB memory setup therefore we changed the runtime to GPU, increased memory limit to 51GB and changed our code to utilize GPU for faster training time.

Table 2: Runtime Environment Specifications

Models	Runtime	Memory
AC-GAN	TPU	35GB
DC-GAN	TPU	35GB
RBM	TPU	35GB
XGBoost	GPU	51GB
SVM	CPU	51GB
RF	CPU	51GB
KNN	CPU	51GB
MLP	CPU	51GB
Resnet152	TPU/CPU	35GB

4.1 Image Extraction

As the sizes of the executable files are different, we divide the files into bins and set corresponding image width for each bin. Each bin of files has fixed width and variable heights based on the size of the files [17]. Table 3 shows the details of bins and width setup.

After extracting images with different heights, we then resize them into 128×128 for experiments; we call this the Resizing method. Another method is to truncate the malware files into desired size and add some 0 paddings for smaller files. For

Table 3: Image Width Based on File Size

File Size	Image Width	File Size	Image Width
0 - 10KB	32	100KB - 200KB	384
10KB - 30KB	64	200KB - 500KB	512
30KB - 60KB	128	500KB - 1000KB	768
60KB - 100KB	256	>1000KB	1024

128×128 images, we only take the first 16,384 bytes of the executable files; this is the Truncating method.

4.1.1 Grayscale Images

Grayscale images are simple to extract from executable files. We can read the files and use 8-bit vectors to represent a pixel in the image [14, 15, 16]. The range of a grayscale pixel is from 0-255 which is the same range a byte in executable files can represent. Only the byte sequence of the files is used in this case and the process is fast as no calculation is needed.

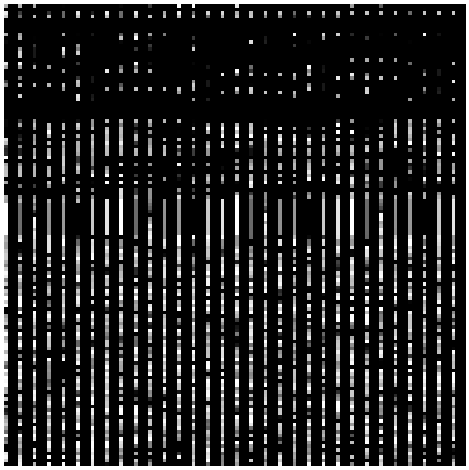


Figure 9: Adload - 256x256

Figure 10: Startpage - 128x128

4.1.2 Colored Images Using Color Map

In this section, we discuss how to extract colored images from executable files using color map [18, 17]. First, we need to generate a 2D color map, which should be a 16×16 2D array where each element corresponds to a RGB value. There are a total 256 colors in this palette so we extract from the byte sequence a byte or 8-bit vector. We then split the byte into two parts, the first half of the byte represents the y-coordinate and the second half is the x-coordinate. Then we can use the coordinates to get the RGB value from the color map. We use 'plasma' colormap, Figure 11 - a sequential color map, because the ordering in executable files is important, when the entropies of the files are big or there are big changes in the local data then we can see the difference in the colors [50]. This colormap also emphasizes the differences between the data representing the images.

After we extract the images using the color map, the images now have different sizes based on the file sizes. We then resize all images to the fixed 128×128 size like Figure 12. Each sample is now represented by an array of (128,128,3). The data is ready to feed into machine learning models for training. We refer to this method as the Colormap method.

4.1.3 Colored Images Using 3 Consecutive Bytes

This method is called 3-grams for future reference. We can extract more data from the executable files and represent each layer of RGB images. For 3 consecutive byte values, we use them as R, G and B layers. The resulting colored images were not as colorful as we expected. Therefore we keep Red, Grreen layers and replace Blue layer with the following formula: BlueLayer = 255 - ByteValue The 3-grams images have a blue background so we can distinguish them with grayscale images as shown in Figure 13.

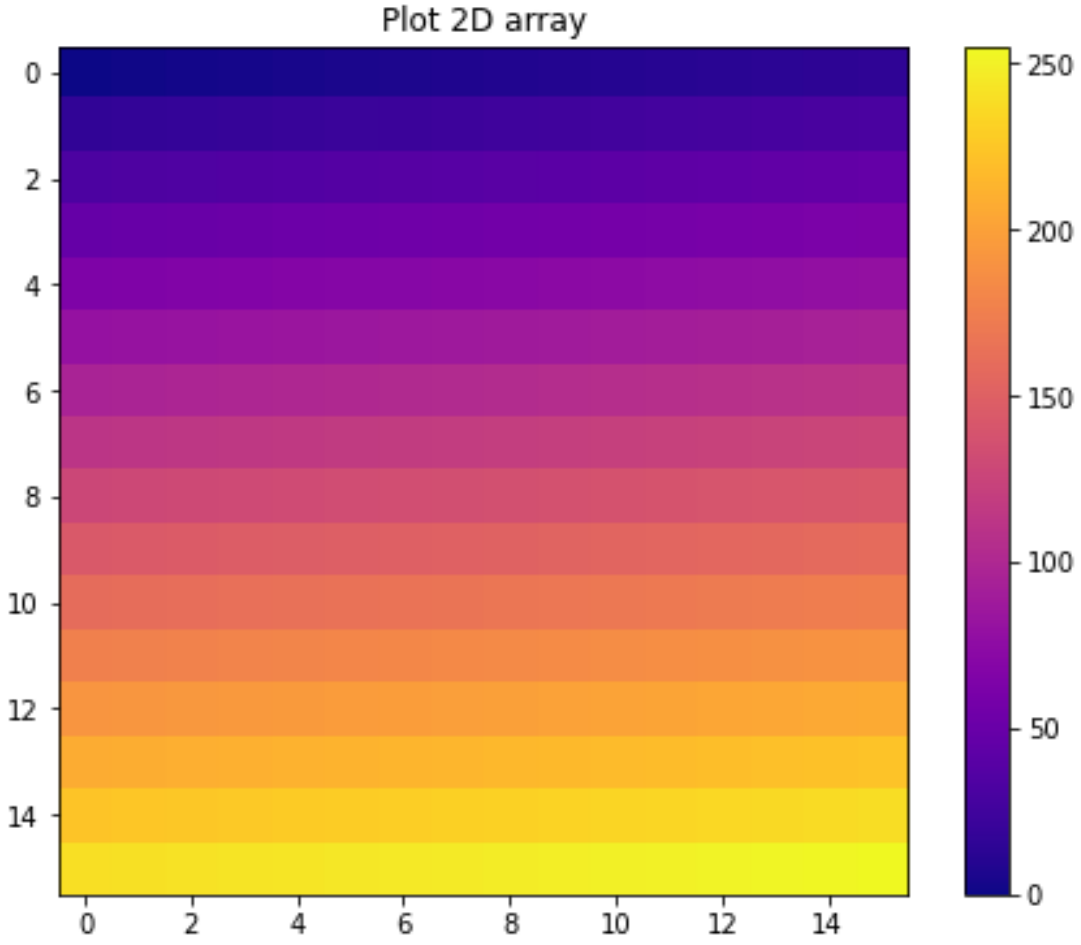
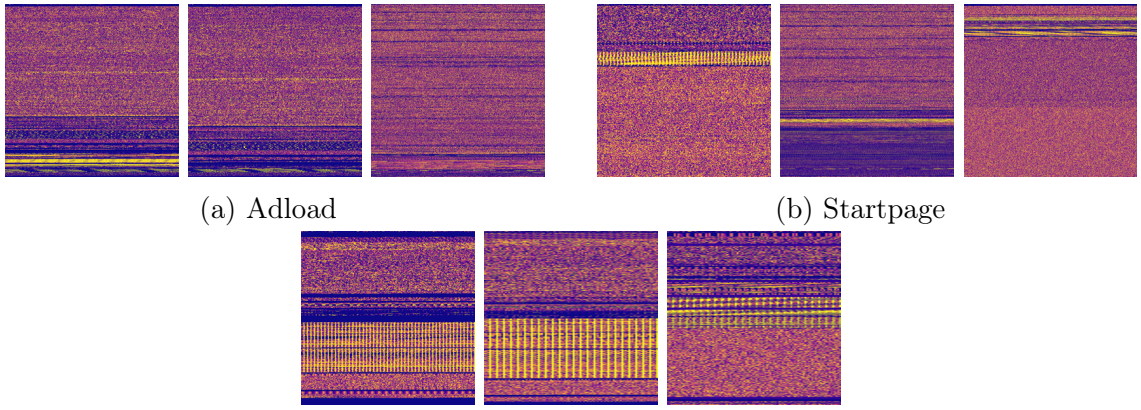


Figure 11: Plasma Colormap

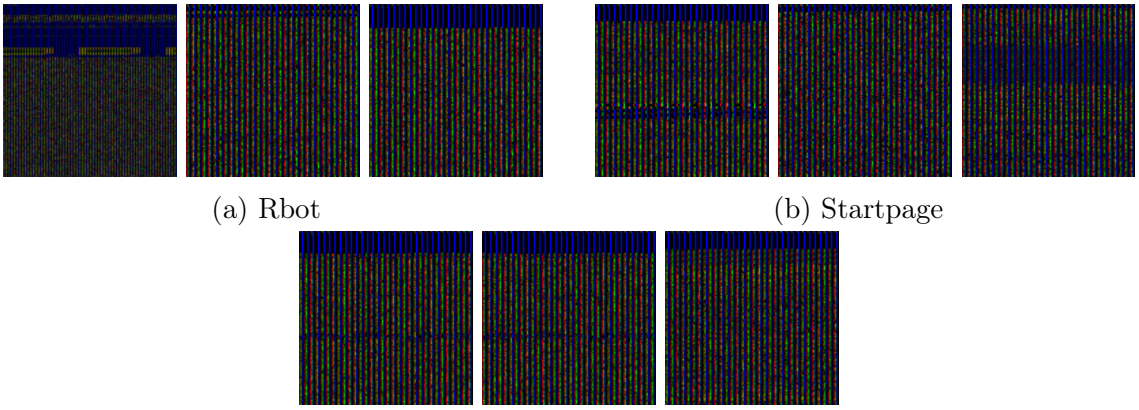


(a) Adload

(b) Startpage

(c) Ceeinject

Figure 12: Colormap Images of Different Families



(a) Rbot (b) Startpage

(c) Ceeinject

Figure 13: 3-grams Images of Different Families

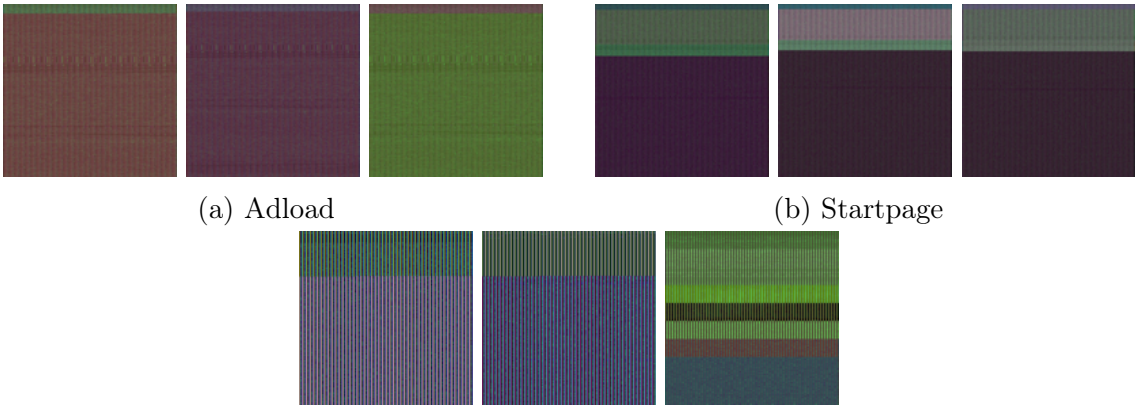
4.1.4 Colored Images Using PE Format

Fu et. al. [19] proposed to transform the malware files into PE files then used a PE reader to divide the files into meaningful sections. For each section, the entropy value was calculated once and used for the whole section. The B layer was represented using the section's relative size to total file size. The G layer was kept the same with byte values. We experimented with this method to extract colorful images from malware files. Entropy values and size ratios are calculated and scaled to 0-255 range. The formulas for red and blue layers are

$$\text{RedLayer} = \text{Entropy} \times \frac{255}{8}$$

$$\text{BlueLayer} = \frac{\text{SectionSize}}{\text{FileSize}} \times 255$$

Figure 14 shows the PE images of three families: Adload, Startpage, and Ceeinject. Ceeinject malwares obfuscate to hide their purpose, so the images from Ceeinject family are different. Therefore, Obfuscator, Ceeinject and Agent are the three types of families that are harder to classify as the images are not similar.



(a) Adload (b) Startpage

(c) Ceeinject

Figure 14: PE Images of Different Families

4.2 Data Processing

GANs require the data in 3-dimension array format, while other techniques like SVM, XGBoost require 1-dimension array input. Therefore the data needs to be flatten and fed to a scaling function such as Equation 1 for SVM where u is the mean and s is the standard deviation:

$$\bar{x} = \frac{x - u}{s} \tag{1}$$

Data is scaled to 0-1 range for faster and simpler computation before feeding into XGBoost or RBMs.

4.3 Models Tuning

Figure 15 and Figure 16 show the details of AC-GAN discriminator and generator implementation for colored images with size 128×128 [51]. For the generator, Manisha et. al. [52] showed that for colored images with big sizes such as 64×64 and above, increasing the noise dimension would significantly have a positive impact on the generative images, therefore both discriminator and generator models would gain benefits. We choose noise dimension vectors that are fed into the generator to be size (1000,1) instead of the regular (100,1) vectors for images size 28×28 [52]. After doing hyperparameter tunings for various models by using grid search, we choose the param-

eters as follows. Table 4 shows the parameters for Random Forest: 600 estimators, using *entropy* function, and having max depth of 6. For K-Nearest Neighbors, Table 5 lists 20 neighbors, using distance as weight as our final choice. Multilayer Perceptron has 4 hidden layers with sizes of 100,100,100,20, *relu* as activation function and 0.0001 penalty as shown in Table 6. SVM training and prediction takes a long time, therefore we can only experiment grid search with a few variables: *rbf* kernel and $C = 1$ are our final choice as in Table 7. Table 8 lists out all experimented parameters for XGBoost and the best parameters.

Table 4: Random Forest Parameters Search

Parameters	Description	All Values	Chosen Value
nestimators	Number of Estimators	100,200,400,600	600
criterion	Function to measure quality	gini, entropy	entropy
maxdepth	Max depth	3,4,5,6	6

Table 5: K-NN Parameters Search

Parameters	Description	All Values	Chosen Value
nneighbors	Number of Neighbors	5,10,20,40	20
weights	Function used in prediction	uniform, distance	distance

Table 6: MLP Parameters Search

Parameters	Description	All Values	Chosen Value
hiddenlayer	Size	(100,100,20), (100,100,100,20)	(100,100,100,20)
activation	Activation Function	Logistic, tanh, relu	relu
alpha	L2 Penalty	0.0001, 0.001, 0.01	0.0001

Table 7: SVM Parameters Search

Parameters	Description	All Values	Chosen Value
Kernel	Kernel Function	rbf, linear, poly	rbf
C	Regularization parameter	1,10,100	1

Table 8: XGB Parameters Search

Parameters	Description	All Values	Chosen Value
maxdepth	Max Depth	4,5,6,7	6
learningrate	Learning Rate	0.01, 0.02, 0.03	0.02
nestimators	Number of Estimators	200,400,600	600

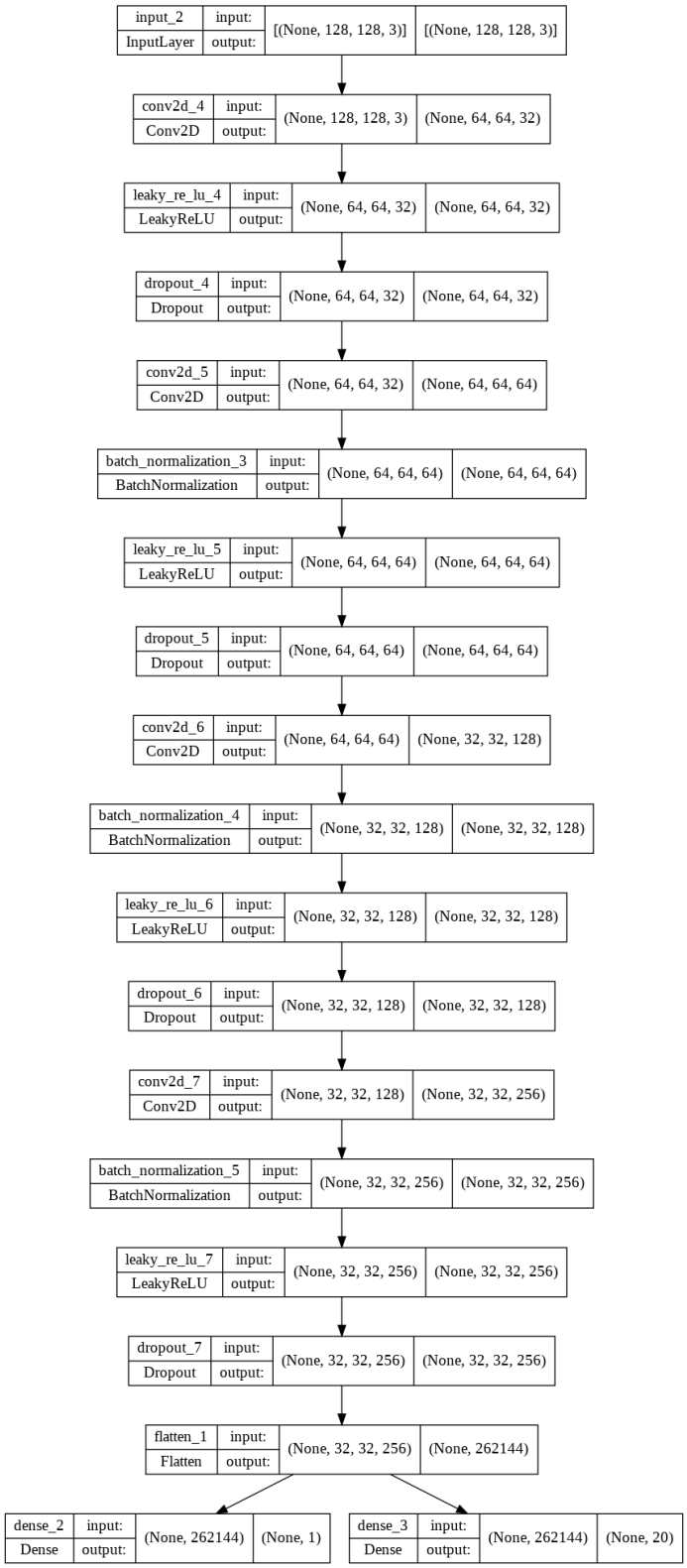


Figure 15: AC-GAN Discriminator Implementation

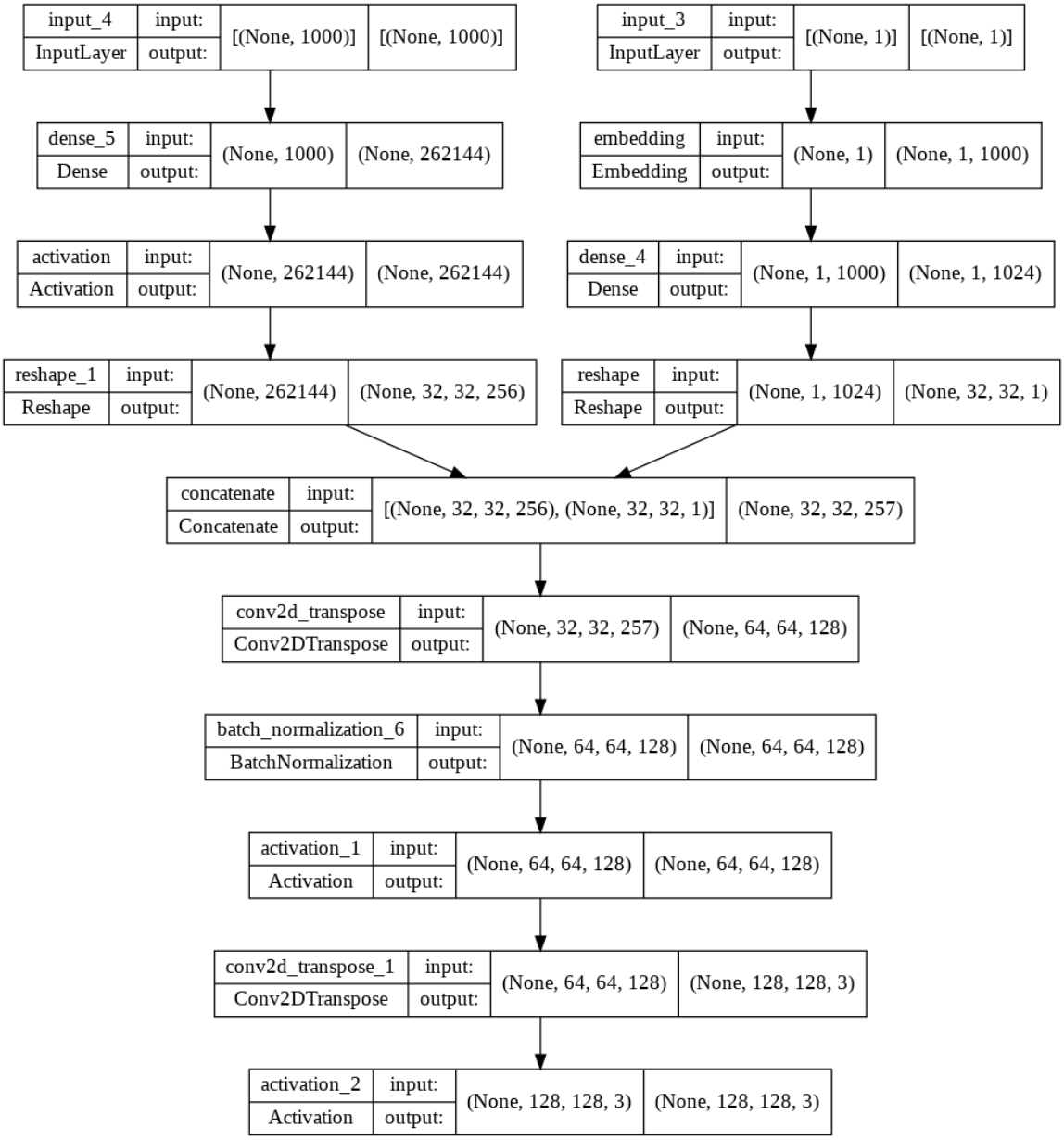


Figure 16: AC-GAN Generator Implementation

CHAPTER 5

Results and Analysis

5.1 Multiclass Classification

First we want to compare the performance of GANs vs RBMs on various image extraction methods. Then we pick one best method to compare GANs with other popular machine learning models.

5.1.1 Image Extraction Comparison

Figure 17 shows the comparison between different types of image extraction. The PE and 3-grams methods which contain more meaningful data did not perform as well as the ColorMap method overall. XGBoost, an ensemble technique, outperforms in all cases. In the next section, we use truncated Colormap method to extract images from executables, then compare the results with multiple machine learning models. To deal with the difference in file sizes, we only take the first 128×128 bytes of the executables, if the file size is smaller, we use 0 to for padding to the desired size.

5.1.2 Machine Learning Models Comparison

Figure 18 shows the comparison between training and test metrics for AC-GAN discriminator. After 30 epochs, there are not much improvement for the discriminator therefore we train AC-GAN discriminator for 30 epochs. AC-GAN discriminator performs well in image-based malware classification and the results are competitive against other models as shown in Table 9. Figure 19 compares the models' performance with training time. MLP and XGBoost perform exceptional well considering the training time is less than other deep learning models like AC-GAN. Resnet152 is our best model, as Resnet152 is a pretrained model, it also takes a short amount of time to customize for new class labels and input. AC-GAN takes approximately 500 seconds per epoch, or around 4 hours to finish training; SVM takes 3 hours for training and 2 hours for prediction. Other models are fast and take less than an hour to complete

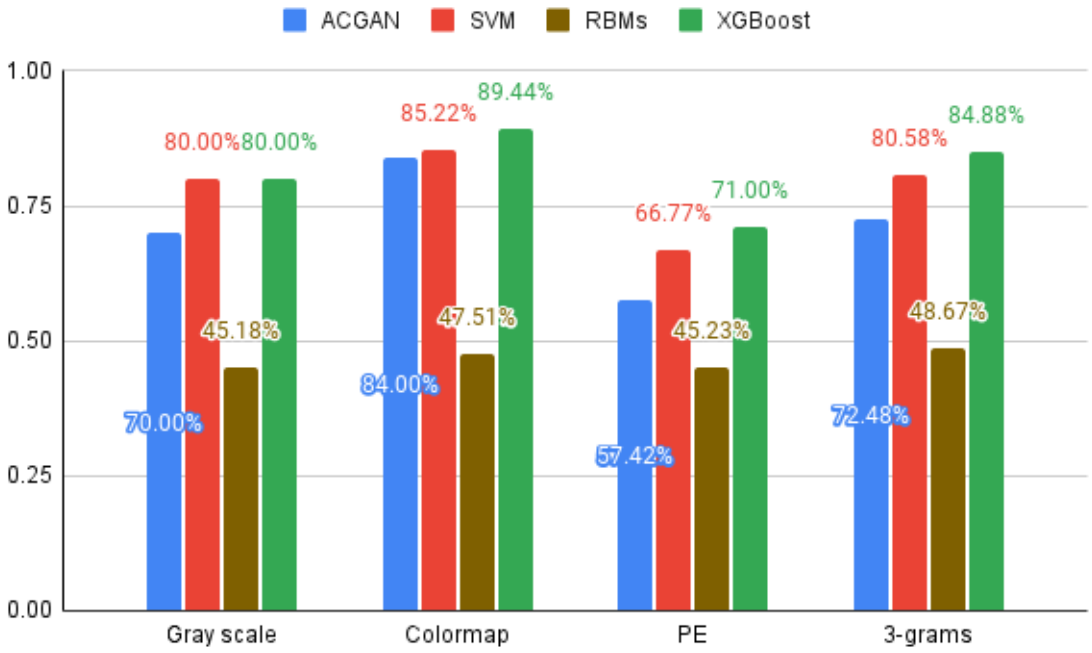


Figure 17: Accuracy score for malware families classification

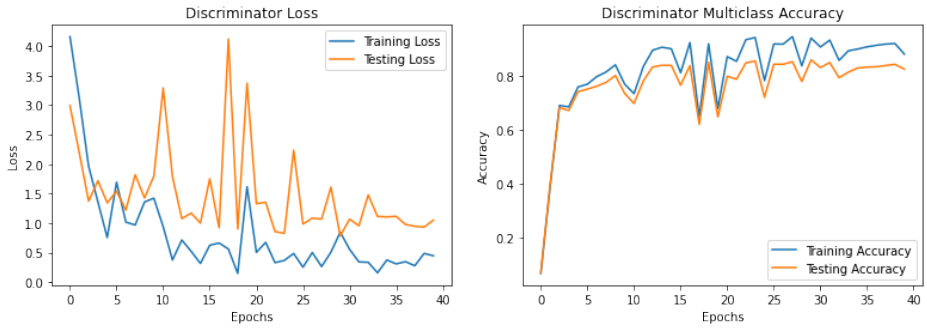


Figure 18: AC-GAN Discriminator Loss and Accuracy

training.

As Resnet152 is our best performer, we want to analyze its confusion matrix as in Figure 20. We can see that the three families that are causing classification problems for our models are Obfuscator, Rbot, and Agent. Obfuscator and Ceeinject obfuscate the code and hide their specific purposes therefore their images are different from each other in the same family, making the classification problem harder. Agent is a

Figure 19: Accuracy and Training Time Comparison

Table 9: Results on various models using truncating colormap

Models	Type	Accuracy	Precision	Recall	F1-Score
K-NN	Maching Learning	76.94%	87%	77%	79%
SVM	Machine Learning	85.22%	88%	86%	86%
MLP	Machine Learning	86.97%	86%	86%	86%
RF	Ensemble	72.56%	80%	69%	70%
XGBoost	Ensemble	89.44%	90%	89%	89%
AC-GAN	Deep Learning	84.00%	86%	86%	85%
Resnet152	Deep Residual Network	91.39%	91%	91%	91%

general type family that include malwares with multiple purposes, and it makes sense for machine learning models to have a problem classifying general types.

Figure 21 confirms our conclusion that some general or obfuscated families are hard to classify such as Agent, Ceeinject, Obfuscator, and Rbot. Resnet152 outperforms on most of the families. K-NN classifier even though does not perform well on most

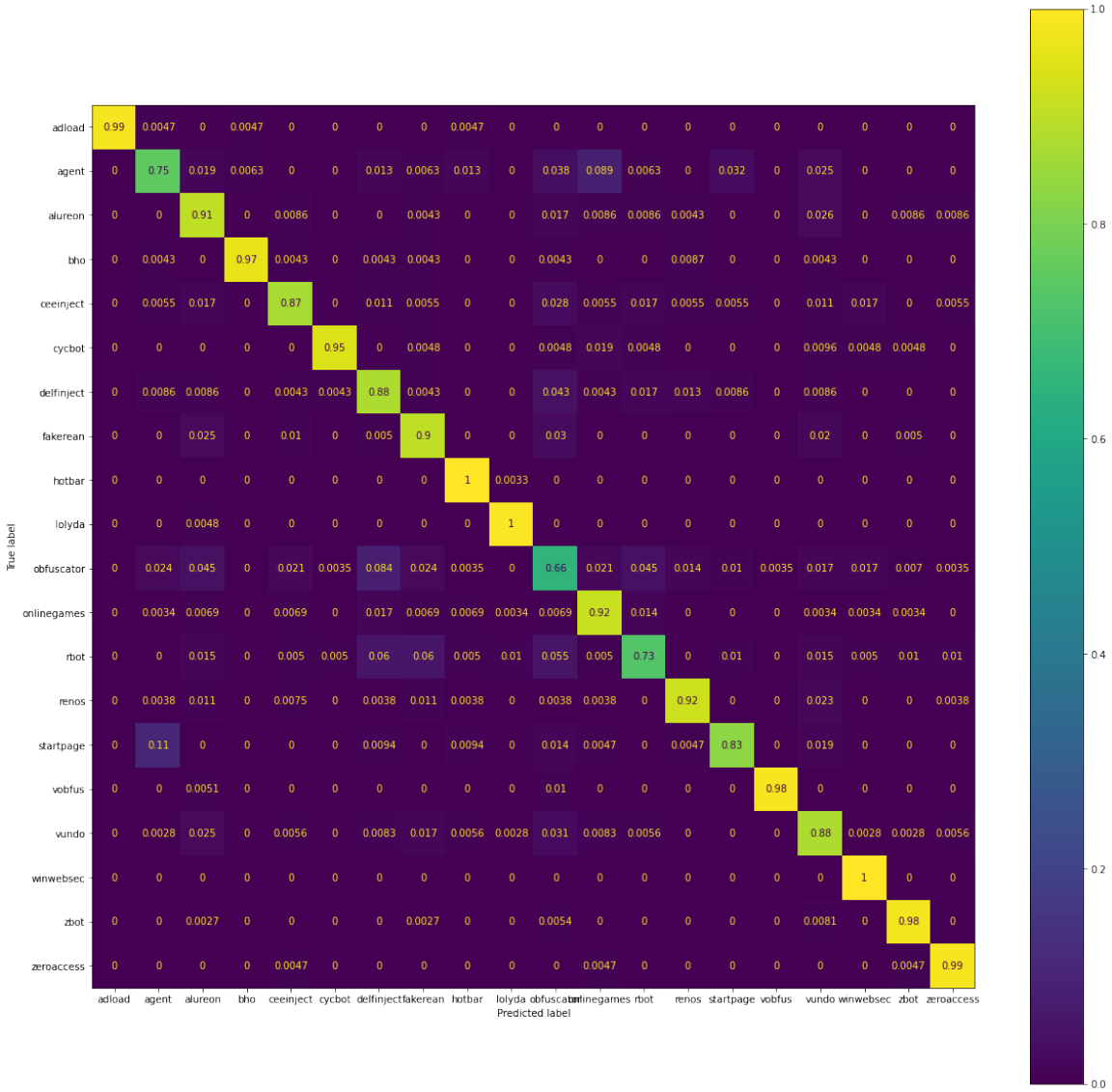


Figure 20: Confusion Matrix for Resnet152 using Colormap Method

families but scores high on a hard family: Agent.

5.1.3 The Ensemble Classifiers

As we want to improve accuracy score for "difficult" families such as Agent or Rbot, the idea is that we should take advantage of K-NN's strength to classify Agent and others for the "easy" families. For the first ensemble classifier, we just use majority voting. With our set of 7 classifiers: AC-GAN, K-NN, MLP, Resnet, RF,

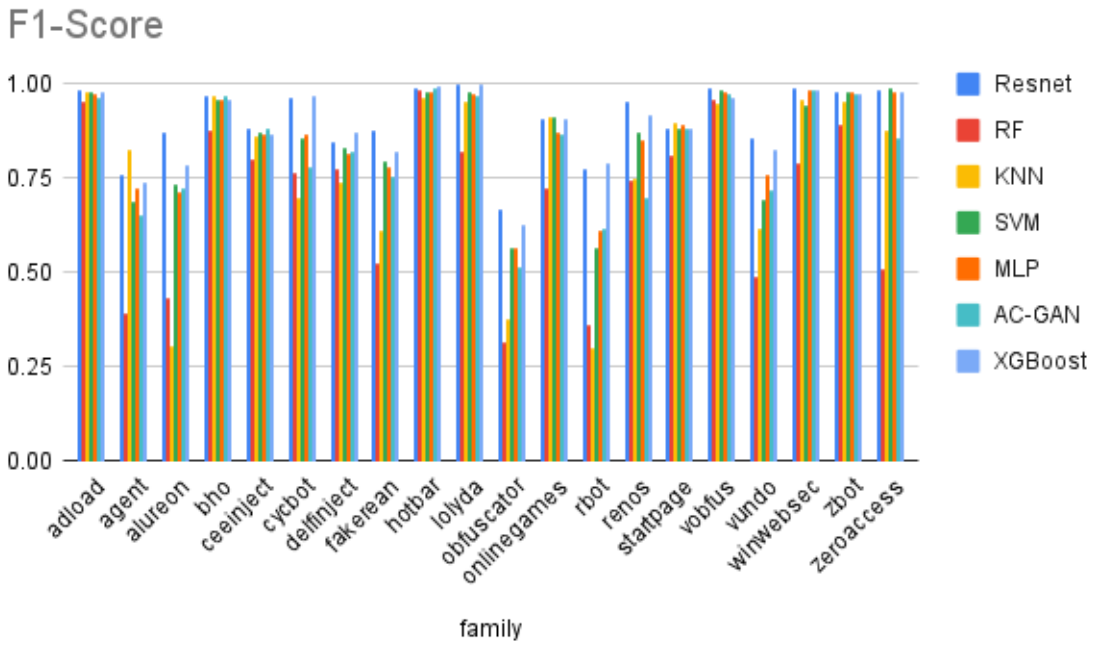


Figure 21: F1-score for each family

SVM and XGBoost, we choose the final prediction based on the prediction with the most occurrence. The ensemble improved a bit with 91.60% accuracy score. For the second experiment, we generate feature vectors using the 7 models we have. AC-GAN and Resnet generate 20×1 vectors while others generate 1×1 vectors. The feature vectors are concatenated and used to train a Random Forest model. The RF Final Ensemble scores 92.09% with improvements on "difficult" families and perform worse than Resnet with "easy" families such as Cycbot, Adload, Alureon because of the noises from other models as shown in Figure 22.

5.2 Generative Image Performance

As DC-GAN is an unsupervised architecture, we do not have accuracy scores for the discriminator. We use DC-GAN to compare with the generative power of AC-GAN generators. Figure 23 shows a comparison between real and generative images using DC-GAN. To analyze the performance of our generators, we experiment with binary

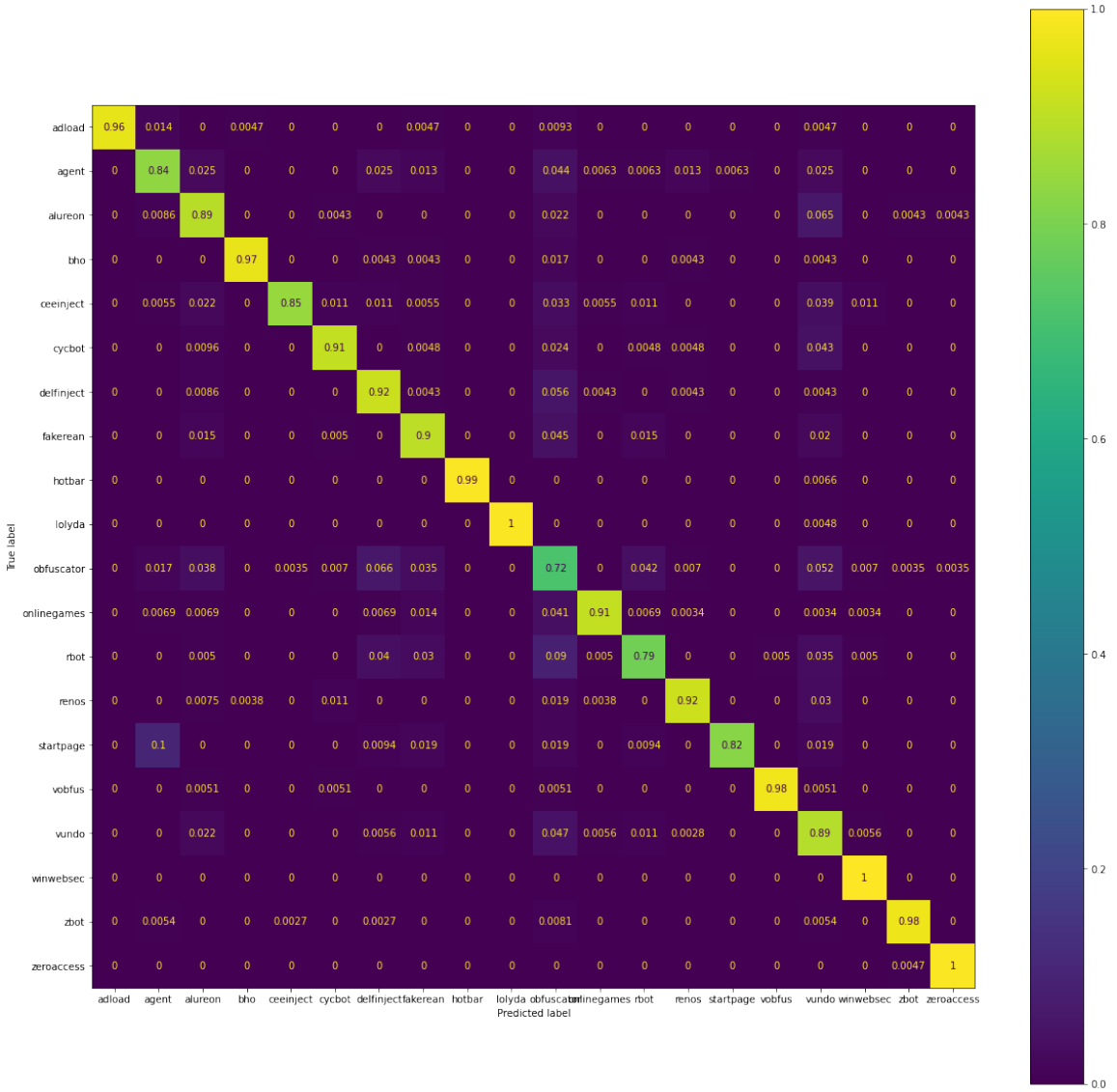
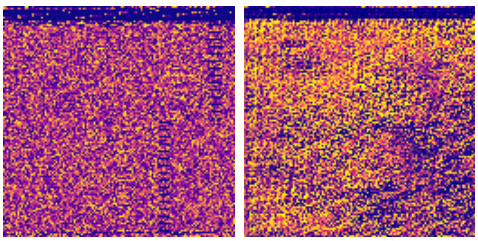
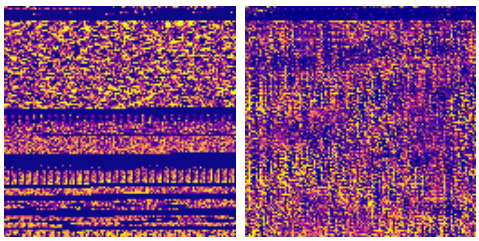


Figure 22: Confusion Matrix for RF Final Ensemble

classification: malware or benign. We use our dataset of 704 benign executables, extract images from them using the Colormap method then use SVM and CNN to classify them with our malware dataset. After that, we mix the benign samples with fake malware images from AC-GAN, DC-GAN and compare the detection performance. AC-GAN with an advantage of a multiclass GAN, can use a class label to generate images. Figure 24 and Figure 25 show real and generated images from Ceeinject and

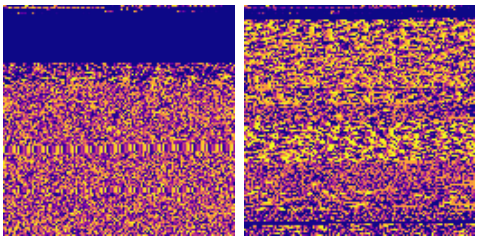


(a) Real vs Generated Images

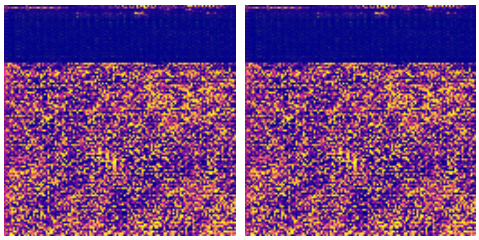


(b) Real vs Generated Images

Figure 23: Real Images and Fake Images Generated by DC-GAN



(a) Real Images



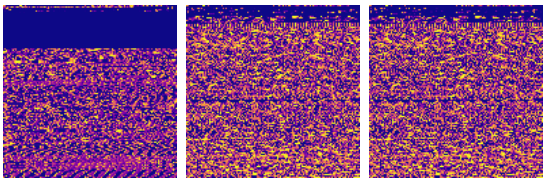
(b) Fake Images

Figure 24: Real and Fake Images of Ceeinject Family

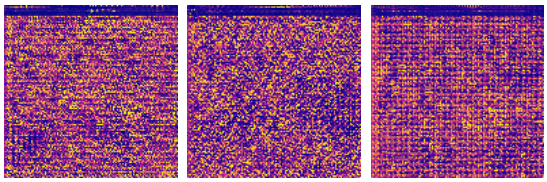
Adload family.

5.2.1 Binary Classification Results

Our dataset consists of 10,000 malware samples from 20 families and 704 benign samples. We use SVM for the detection experiment. SVM with 5-fold cross validation scores 99%-100% in accuracy and AUC scores. After mixing in 10,000 fake malware images, our SVM model still scores 99-100% in accuracy and AUC, with 5-fold cross validation. Our generative images are easy to detect and not good enough to trick popular machine learning models. We also tried to generate fake images from other



(a) Real Images



(b) Fake Images

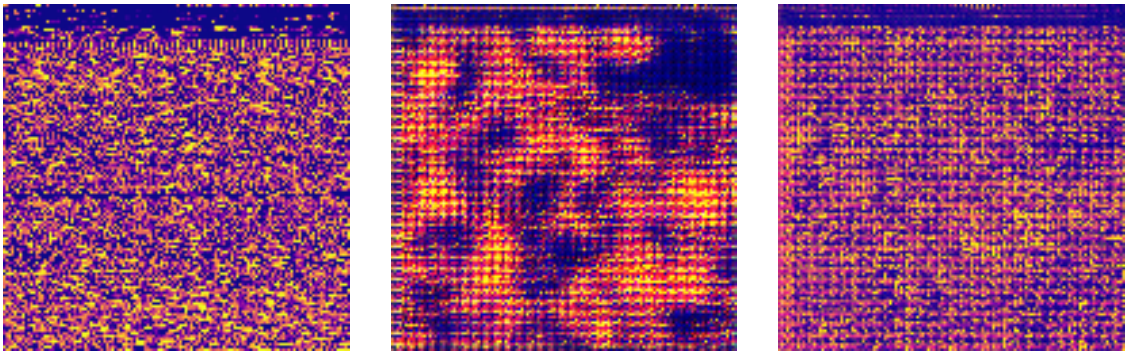
Figure 25: Real and Fake Colored Images of Adload Family

variations of GANs such as DC-GAN and Wasserstein GANs with gradient penalty (WGAN-GP). As WGAN-GP requires intensive computation power and training time, we can manage to train it for 100 epochs, and 300 epochs for DC-GAN. The results are still 99-100% for accuracy and AUC scores. The robustness of the colormap method may cause problems for the generated images because we only have 16×16 available colors in the colormap, so we experiment with another method: 3-grams using AC-GAN. The same result happens with 3-grams and AC-GAN: 99-100% accuracy and AUC scores.

5.3 Discussion

Intuitively, GANs with two set of models: the generator and the discriminator should compete with each other and improve together. However, with our experiments, AC-GAN discriminator performs best with 30 epochs of training while AC-GAN generator requires 200 epochs or more for better image generation. Odena et. al. [44] shows that AC-GAN generator can be trained for 50,000 mini batches or approximately 300 epochs. Figure 26 shows the comparison between a real image and generated images with different epochs. At 200 epochs, AC-GAN generator performs better however AC-GAN discriminator shows signs of overfitting and the accuracy score decreases to 65%.

As with malware classification, Resnet152 outperforms in accuracy score, f1-score and training time with the advantage of being a pretrained model. Image-based malware classification should be experimented more with pretrained models such as Resnet152 or VGG19. AC-GAN can be used as a classifier with competitive however considering AC-GAN long training time and not much of improvement over MLP, MLP and Resnet should be prioritized.



(a) Real Image (b) Generated Image After 30 Epochs (c) Generated Image After 200 Epochs

Figure 26: Real and Fake Colored Images of Adload Family

CHAPTER 6

Conclusion and Future Works

After trying three different ways of extracting images from malware executables :Grayscale, Colormap, 3-grams and PE, we conclude that with the same amount of data, Colormap method improves the overall results for all models and AC-GAN gains the most improvement for the malware families classification problem. XGBoost and RBM take less training time than SVM and GANs, however for the XGBoost memory requirement is high. We can change the runtime environments based on each model to reduce training time.

Based on the confusion matrices, our models are not doing well with malware that are general or obfuscated. For future work, experiments with a transformer or long short term memory (LSTM) are interesting to deal with obfuscated codes. The PE method extracts the most data from the executables and the images are colorful, however all of our models perform worse using PE data than Colormap images. Combining PE images with global, local features can help to achieve better results. Modern and advanced images based models such as CNN can be used to help with multiclass classification. There are many GAN's variations such as StyleGAN, MalGAN so future experiments are needed with different GAN architectures for malware classification as well as malware image generation. Designing new GAN architectures or changing AC-GAN loss function to speed up the training process of the AC-GAN generator are interesting to explore. We have tried multiple ways to extract images from malware executables, however there are still more novel efficient methods to extract images from both executable files and opcode sequences. Combining efficient feature engineering and model optimization would boost performance of GANs and other machine learning models.

LIST OF REFERENCES

[1] B. Carlson, ‘‘Top cybersecurity statistics, trends, and facts,’’ 2021. [Online]. Available: https://www.csoonline.com/article/3634869/top-cybersecurity-statistics-trends-and-facts.html

[2] M. F. A. Razak, N. B. Anuar, R. Salleh, and A. Firdaus, ‘‘The rise of ‘‘malware’’: Bibliometric analysis of malware study,’’ *Journal of Network and Computer Applications*, vol. 75, pp. 58--76, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804516301904

[3] T. Karras, S. Laine, and T. Aila, ‘‘A style-based generator architecture for generative adversarial networks,’’ *CoRR*, vol. abs/1812.04948, 2018. [Online]. Available: http://arxiv.org/abs/1812.04948

[4] W. Hu and Y. Tan, ‘‘Generating adversarial malware examples for black-box attacks based on GAN,’’ *CoRR*, vol. abs/1702.05983, 2017. [Online]. Available: http://arxiv.org/abs/1702.05983

[5] M. Kawai, K. Ota, and M. Dong, ‘‘Improved MalGAN: Avoiding malware detector by leaning cleanware features,’’ in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2019, pp. 040--045.

[6] M. Hearst, S. Dumais, E. Osuna, J. Platt, and B. Scholkopf, ‘‘Support vector machines,’’ *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18--28, 1998.

[7] J. Hegedus, Y. Miche, A. Ilin, and A. Lendasse, ‘‘Methodology for behavioral-based malware analysis and detection using random projections and k-nearest neighbors classifiers,’’ in *2011 Seventh International Conference on Computational Intelligence and Security*, 2011, pp. 1016--1023.

[8] F. C. C. Garcia and F. P. Muga, ‘‘Random forest for malware classification,’’ 2016. [Online]. Available: https://arxiv.org/abs/1609.07770

[9] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio, ‘‘Learning algorithms for the classification restricted boltzmann machine,’’ *The Journal of Machine Learning Research*, vol. 13, pp. 643--669, 03 2012.

[10] T. Chen and C. Guestrin, ‘‘Xgboost: A scalable tree boosting system,’’ *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[11] K. He, X. Zhang, S. Ren, and J. Sun, ‘‘Deep residual learning for image recognition,’’ 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[12] D. Dang, F. D. Troia, and M. Stamp, ‘‘Malware classification using long short-term memory models,’’ *CoRR*, vol. abs/2103.02746, 2021. [Online]. Available: https://arxiv.org/abs/2103.02746

[13] L. Lazarovitz, ‘‘Deconstructing the solarwinds breach,’’ *Computer Fraud & Security*, vol. 2021, no. 6, pp. 17--19, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361372321000658

[14] M. Jain, ‘‘Image-based malware classification with convolutional neural networks and extreme learning machines,’’ *SJSU ScholarWorks*, 2019. [Online]. Available: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1901&context=etd_projects

[15] R. Nagaraju and M. Stamp, ‘‘Auxiliary-classifier gan for malware analysis,’’ 2021. [Online]. Available: https://arxiv.org/abs/2107.01620

[16] M. Xiao, C. Guo, G. Shen, Y. Cui, and C. Jiang, ‘‘Image-based malware classification using section distribution information,’’ *Computers & Security*, vol. 110, p. 102420, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821002443

[17] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, ‘‘Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture,’’ *Computer Networks*, vol. 171, p. 107138, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128619304736

[18] A. Singh, A. Handa, N. Kumar, and S. K. Shukla, ‘‘Malware classification using image representation,’’ in *Cyber Security Cryptography and Machine Learning*, S. Dolev, D. Hendler, S. Lodha, and M. Yung, Eds. Cham: Springer International Publishing, 2019, pp. 75--92.

[19] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, ‘‘Malware visualization for fine-grained classification,’’ *IEEE Access*, vol. 6, pp. 14 510--14 523, 2018.

[20] H. Farhat and V. Rammouz, ‘‘Malware classification using transfer learning,’’ 2021. [Online]. Available: https://arxiv.org/abs/2107.13743

[21] ‘‘Restricted boltzmann machine features for digit classification.’’ [Online]. Available: https://scikit-learn.org/stable/auto_examples/neural_networks/plot_rbm_logistic_classification.html

[22] ‘‘Support vector machines.’’ [Online]. Available: https://scikit-learn.org/stable/modules/svm.html

[23] ‘‘Adware:win32/adload.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Adload&threatId=243639

[24] ‘‘Win32/agent.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FAgent

[25] ‘‘Win32/alureon.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Alureon

[26] ‘‘Trojan:win32/bho.bo.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/BHO.BO

[27] ‘‘Virtool:win32/ceeinject.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject

[28] ‘‘Win32/cycbot.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=Win32/Cycbot

[29] ‘‘Pws:win32/delfinject.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/DelfInject&threatId=-%202147241365

[30] ‘‘Win32/fakerean.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/FakeRean

[31] ‘‘Adware:win32/hotbar.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware%3AWin32%2FHotbar

[32] ‘‘Pws:win32/lolyda.bf.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS%3AWin32%2FLolyda.BF

[33] ‘‘Virtool:win32/obfuscator.c.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FObfuscator.C

[34] ‘‘Pws:win32/onlinegames.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS%3AWin32%2FOnLineGames

[35] ‘‘Backdoor:win32/rbot.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Rbot

[36] ‘‘Win32/renos.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FRenos

[37] ‘‘Trojan:win32/startpage.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Startpage&threatId=15435

[38] ‘‘Win32/vobfus.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FVobfus

[39] ‘‘Win32/vundo.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FVundo

[40] ‘‘Win32/winwebsec.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Winwebsec

[41] ‘‘Win32/zbot.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=win32%2Fzbot

[42] ‘‘Win32/sirefef.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Sirefef

[43] A. Radford, L. Metz, and S. Chintala, ‘‘Unsupervised representation learning with deep convolutional generative adversarial networks,’’ *arXiv preprint arXiv:1511.06434*, 2015.

[44] A. Odena, C. Olah, and J. Shlens, ‘‘Conditional image synthesis with auxiliary classifier GANs,’’ 2017.

[45] M. Kang, W. Shim, M. Cho, and J. Park, ‘‘Rebooting acgan: Auxiliary classifier GANs with stable training,’’ 2021.

[46] A. Waheed, M. Goyal, D. Gupta, A. Khanna, F. Al-Turjman, and P. Pinheiro, ‘‘Covidgan: Data augmentation using auxiliary classifier gan for improved covid-19 detection,’’ *IEEE Access*, vol. PP, pp. 1--1, 05 2020.

[47] ‘‘Bernoullirbm.’’ [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.BernoulliRBM.html

[48] M. Scognamiglio, ‘‘A beginner’s guide to roc curves and auc metrics.’’ [Online]. Available: https://medium.com/swlh/a-beginners-guide-to-roc-and-auc-curves-d279c1a5e0e6

[49] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320396001422

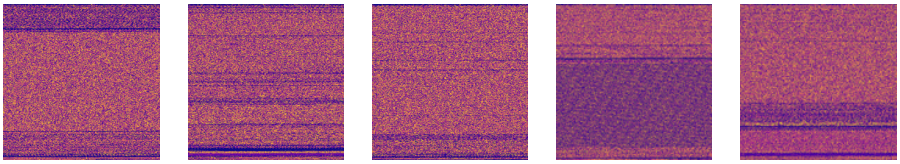
[50] ‘‘Choosing colormaps in matplotlib.’’ [Online]. Available: https://matplotlib.org/3.5.1/tutorials/colors/colormaps.html

[51] J. Brownlee, ‘‘How to develop an auxiliary classifier gan (ac-gan) from scratch with keras,’’ 2019. [Online]. Available: https://machinelearningmastery.com/how-to-develop-an-auxiliary-classifier-gan-ac-gan-from-scratch-with-keras/

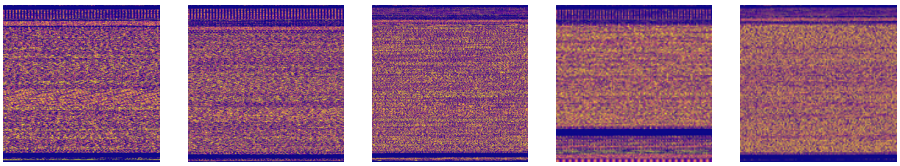
[52] P. Manisha, D. Das, and S. Gujar, ‘‘Effect of input noise dimension in GANs,’’ *CoRR*, vol. abs/2004.06882, 2020. [Online]. Available: https://arxiv.org/abs/2004.06882

APPENDIX A

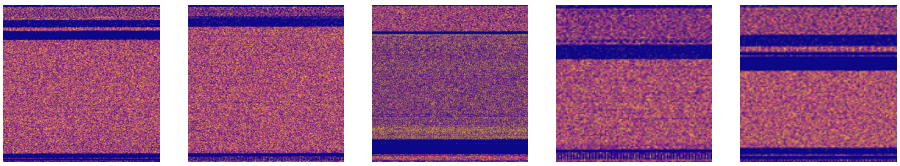
Malware Images



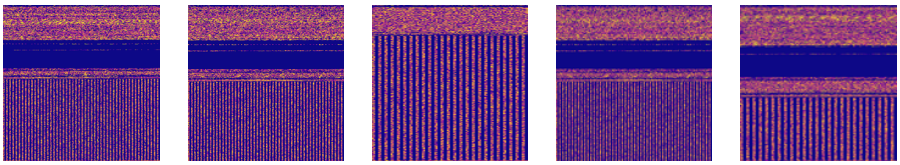
(a) Zeroaccess



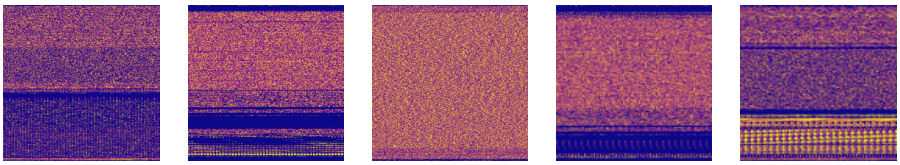
(b) Vobfus



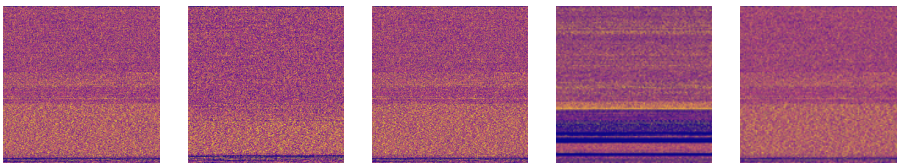
(c) Renos



(d) Onlinegames



(e) Vundo



(f) Hotbar

Figure A.27: Image Representation of Different Families

APPENDIX B

Confusion Matrices

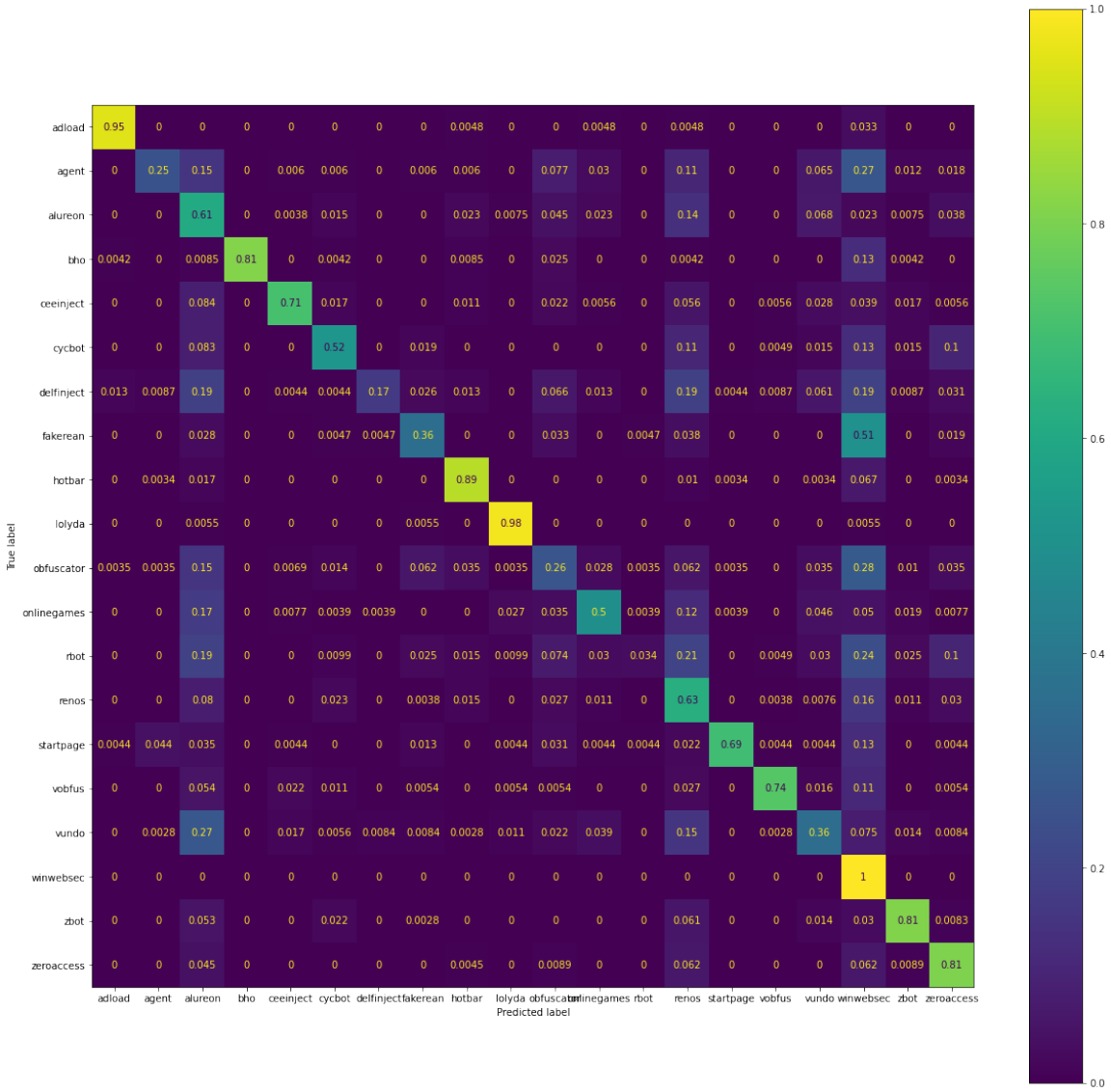


Figure B.28: Confusion Matrix for AC-GAN using RGB Layers Method

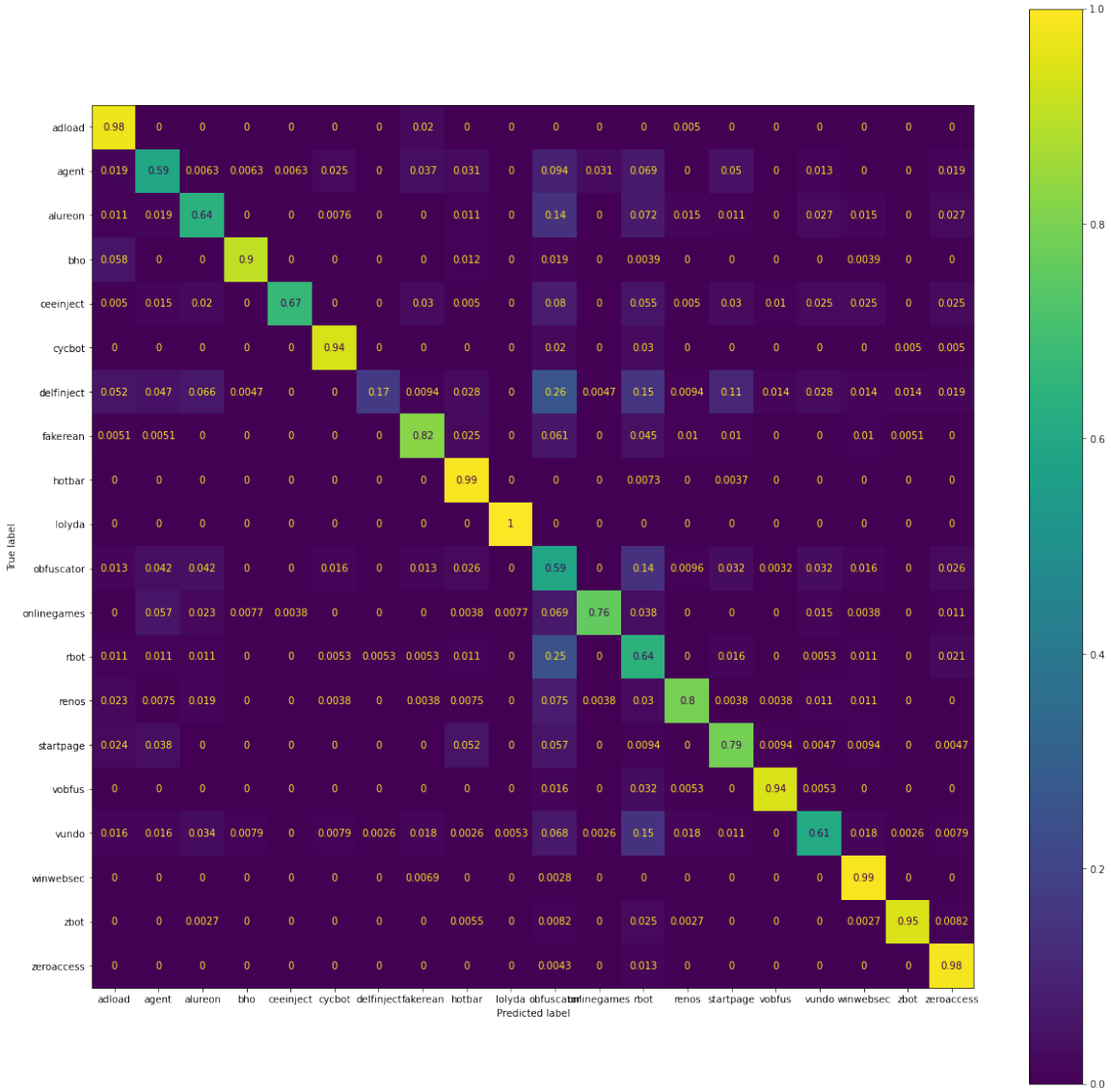


Figure B.29: Confusion Matrix for AC-GAN using Colormap Method

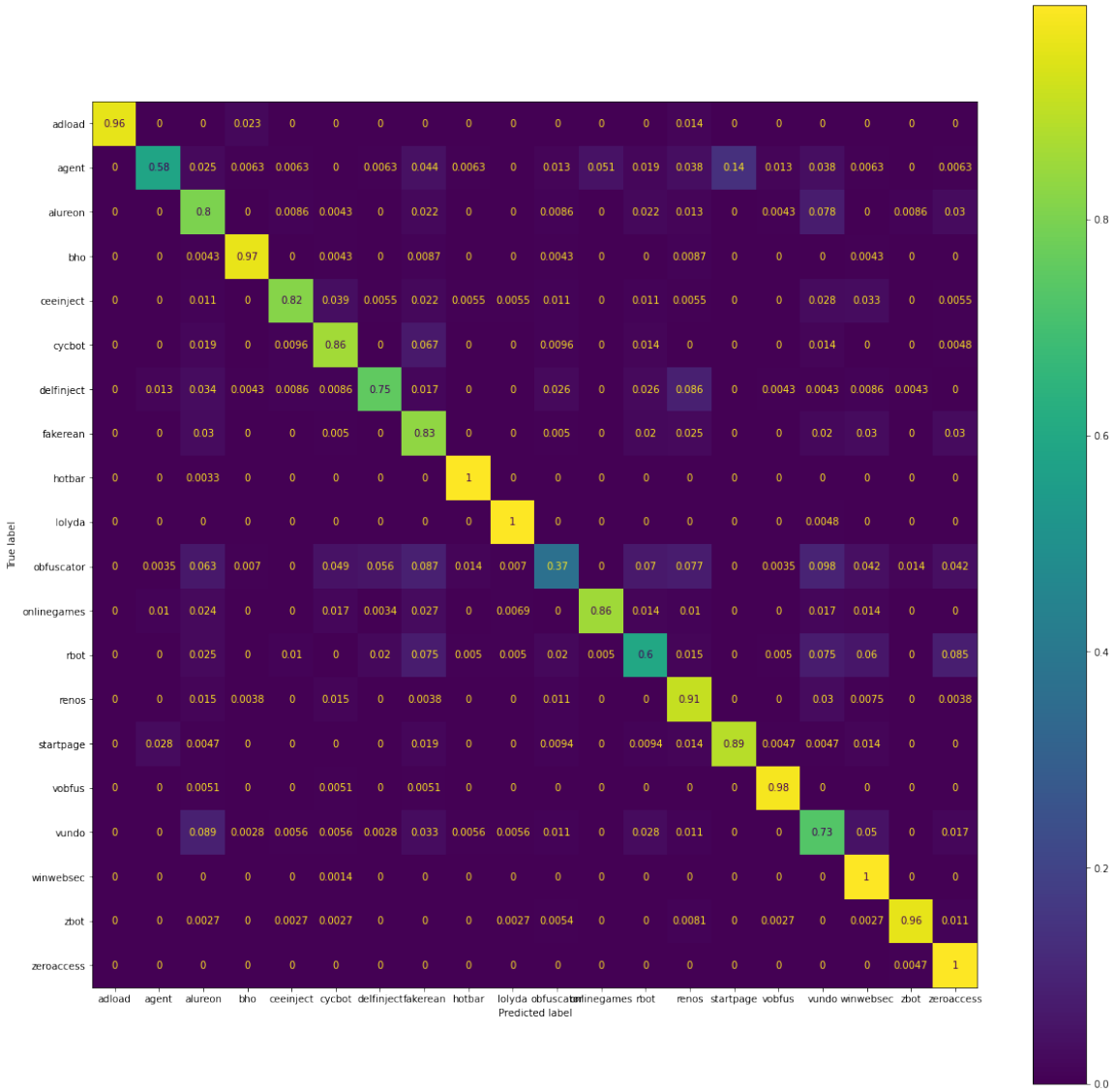


Figure B.30: Confusion Matrix for AC-GAN using Truncating Colormap Method

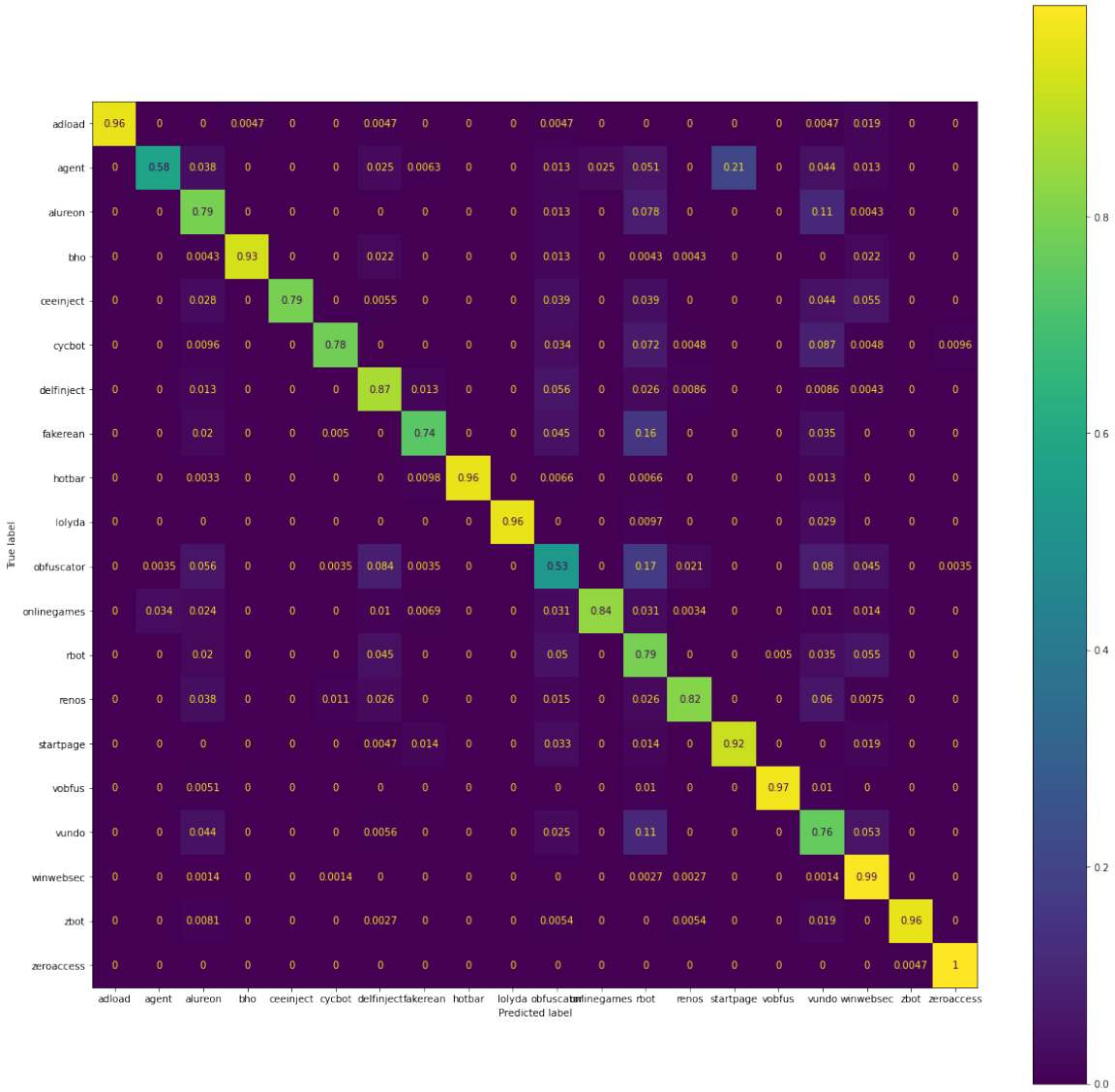


Figure B.31: Confusion Matrix for SVM using Truncating Colormap Method

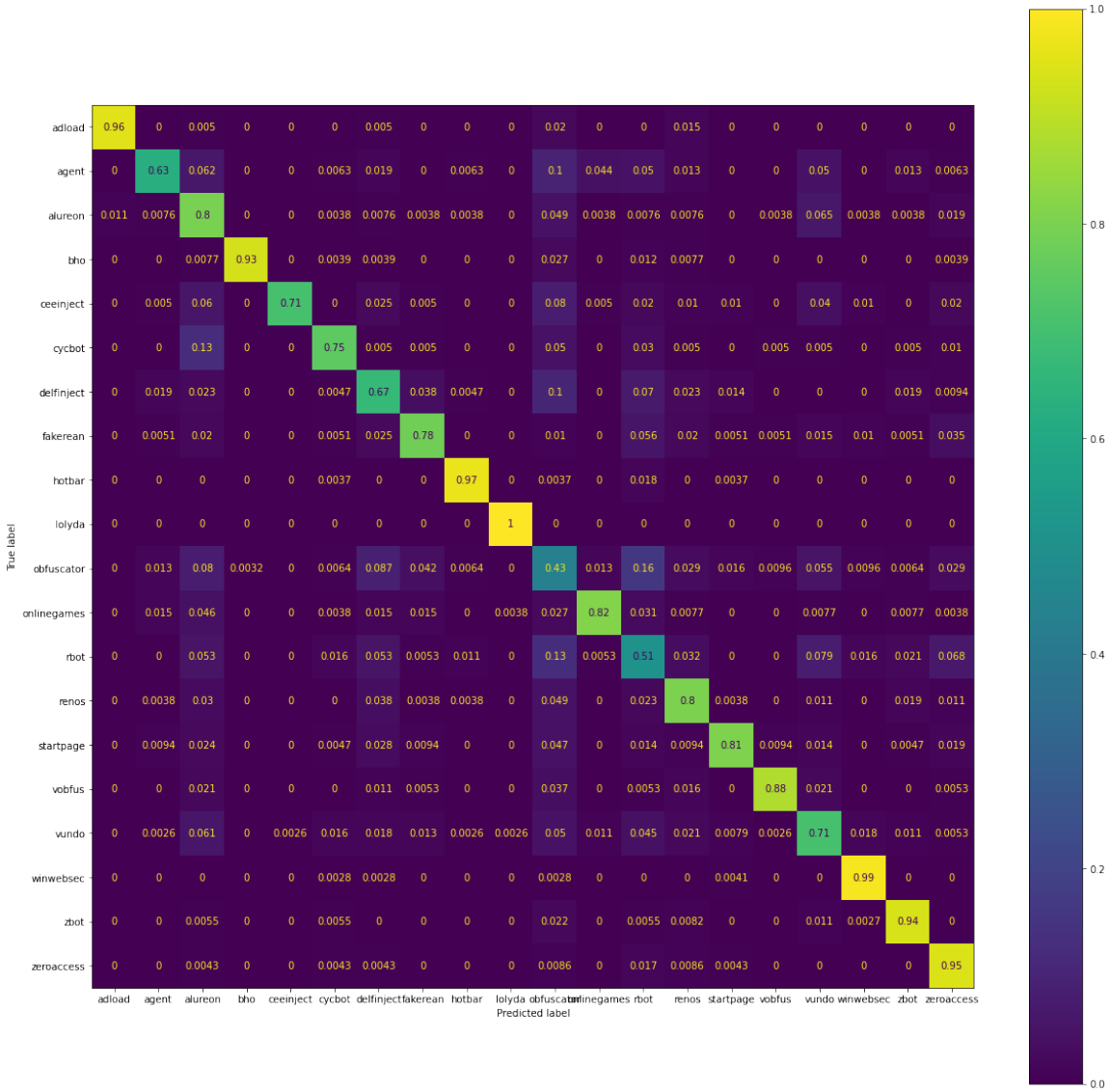


Figure B.32: Confusion Matrix for XGB using Colormap Method

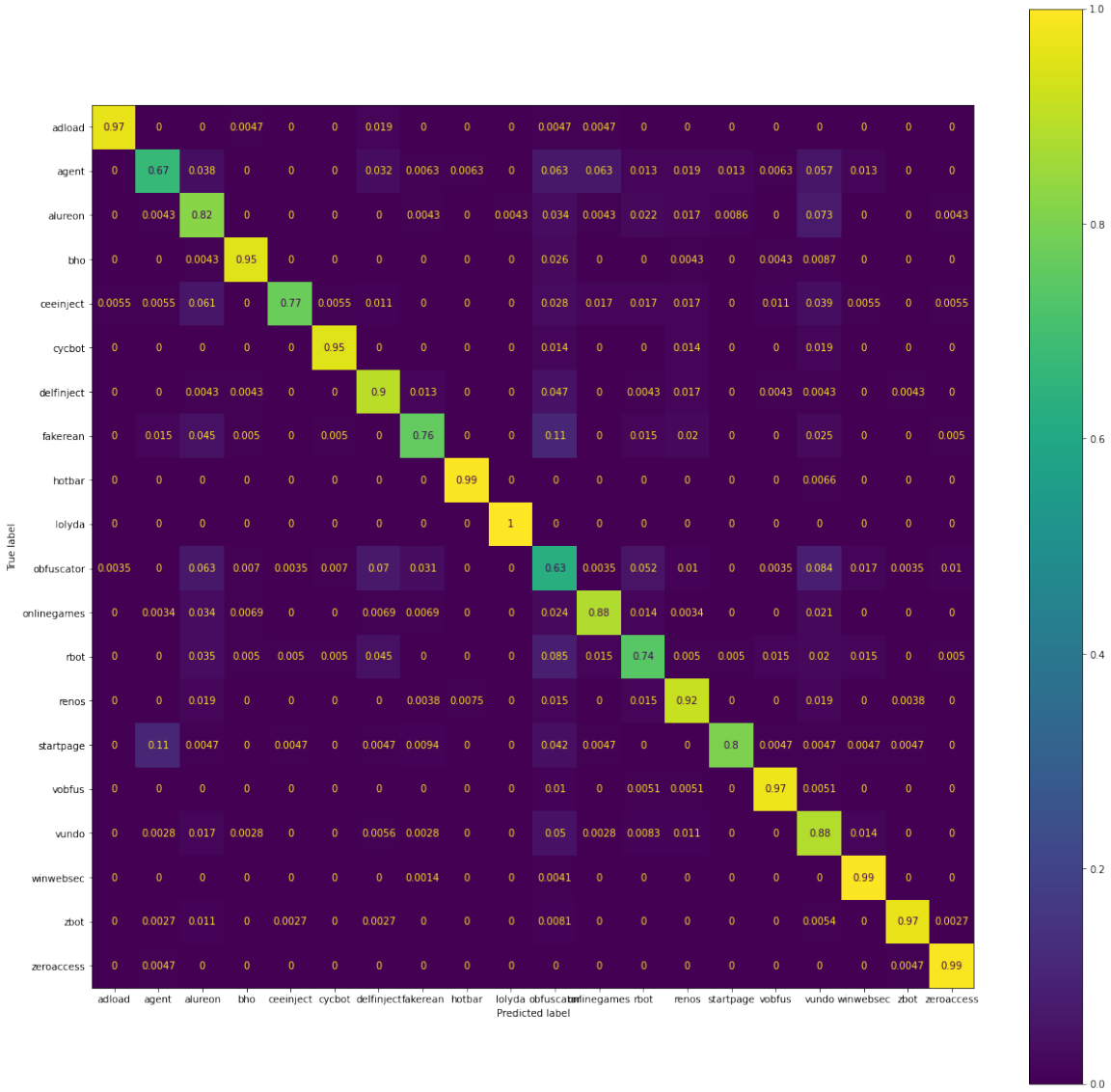


Figure B.33: Confusion Matrix for XGB using Truncating Colormap Method

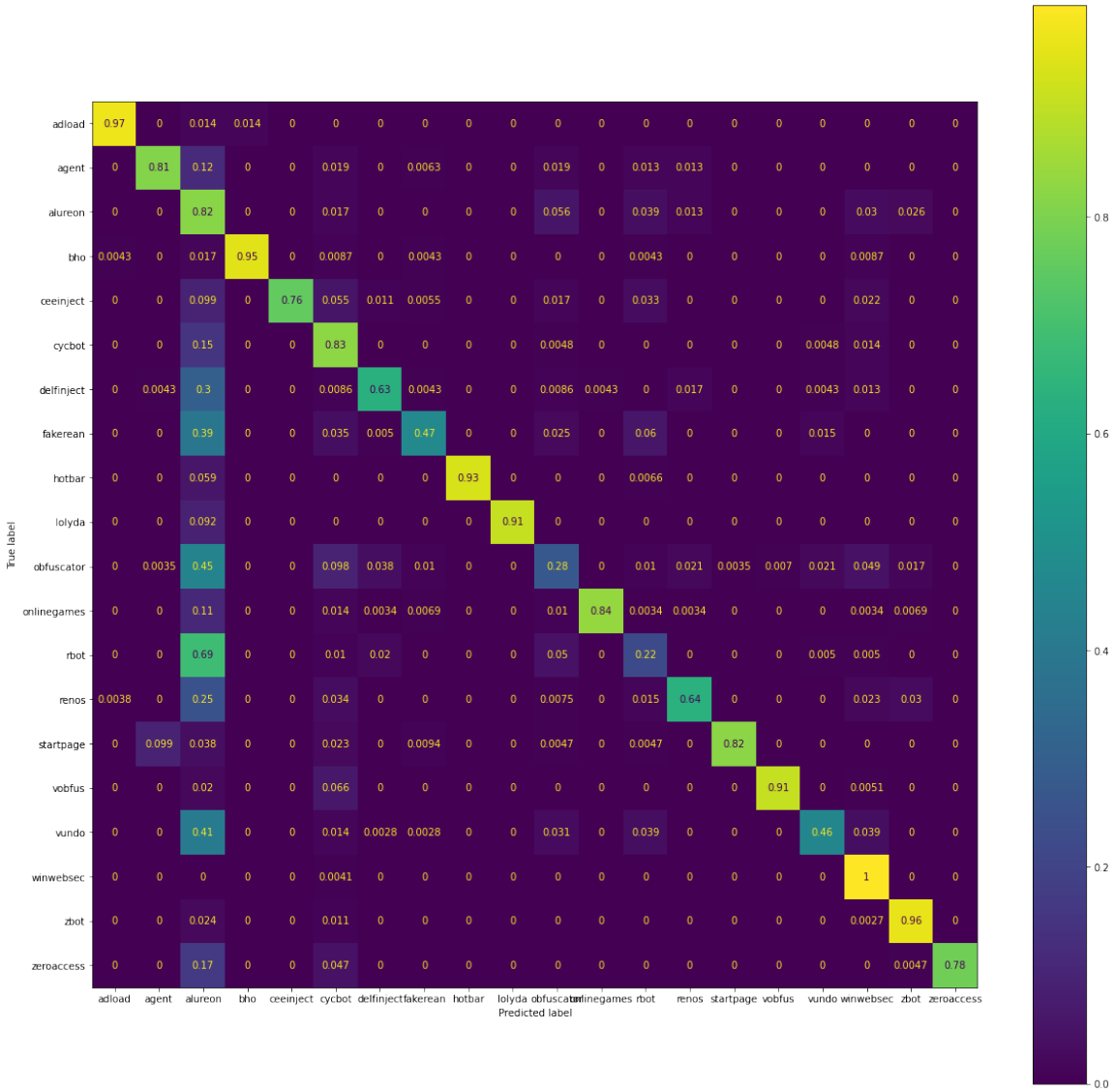


Figure B.34: Confusion Matrix for KNN using Truncating Colormap Method

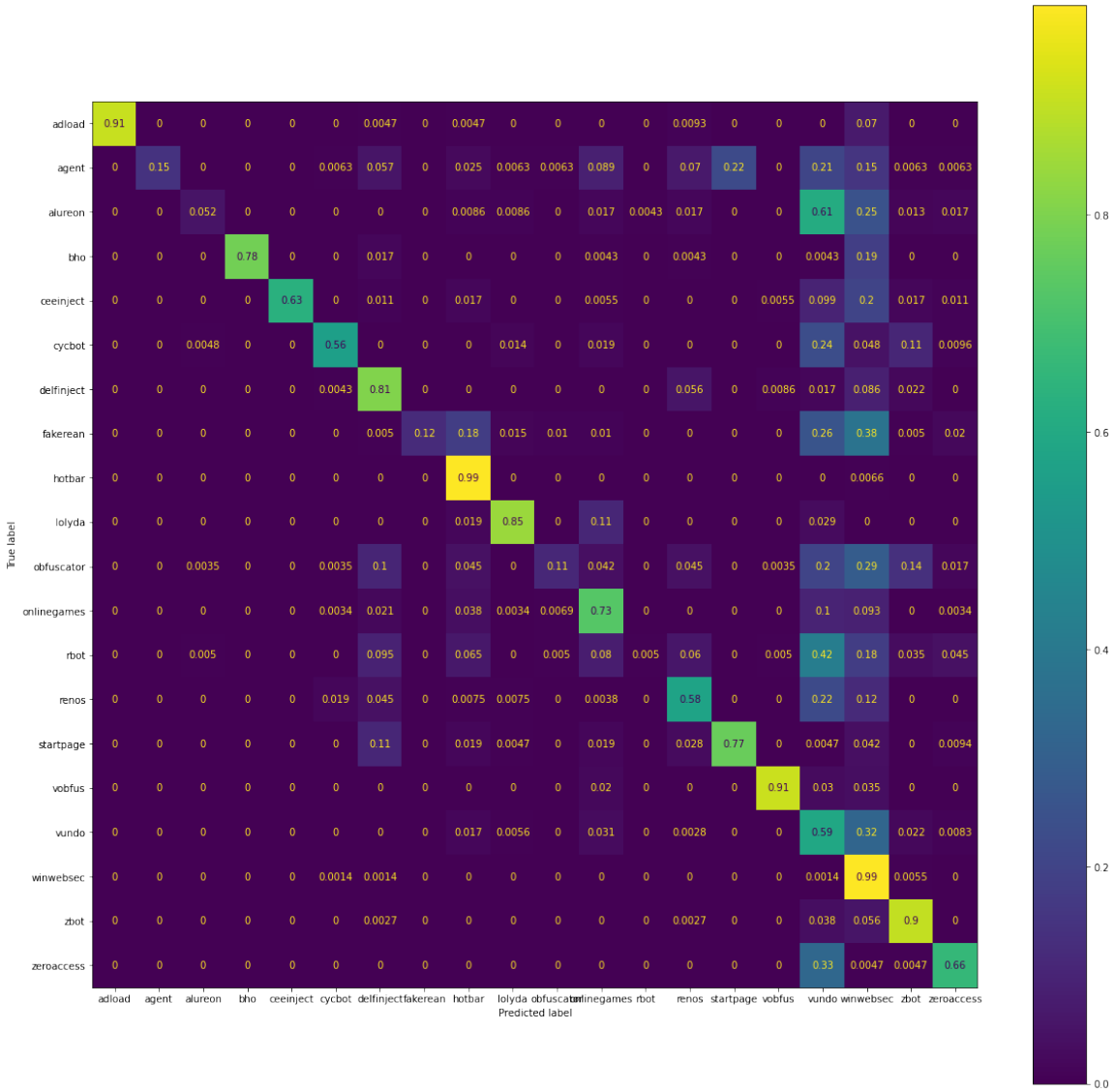


Figure B.35: Confusion Matrix for RF using Truncating Colormap Method

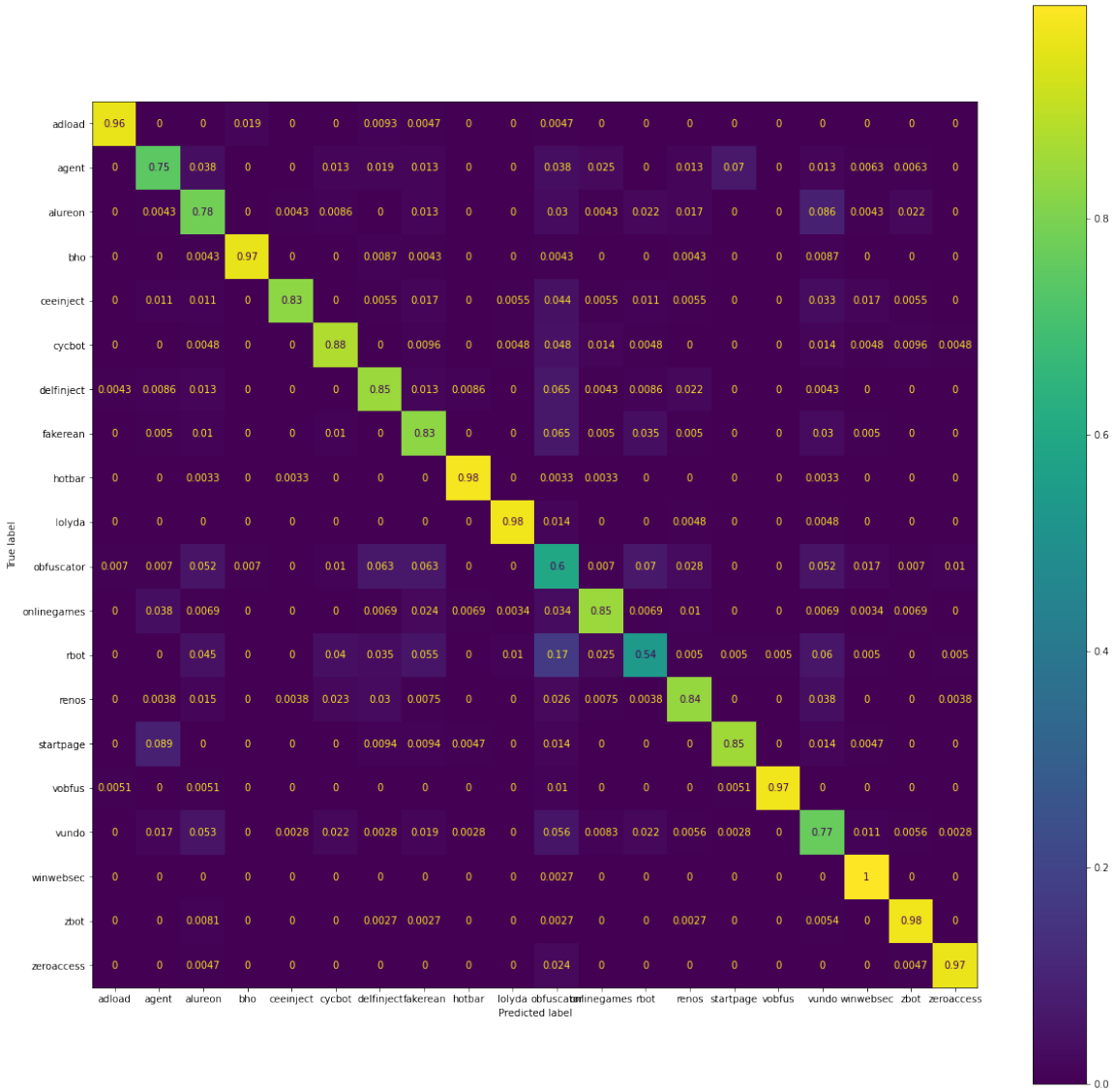


Figure B.36: Confusion Matrix for MLP using Truncating Colormap Method