Spring 2021

Data Augmentation with Malware as Images

Aditi Walia
San Jose State University

Data Augmentation with Malware as Images

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

Aditi Walia

May 2021

The Designated Project Committee Approves the Project Titled

Data Augmentation with Malware as Images

by

Aditi Walia

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2021

Dr. Mark Stamp	Department of Computer Science
Dr. Nada Attar	Department of Computer Science
Dr. Fabio Di Troia	Department of Computer Science

ABSTRACT

Data Augmentation with Malware as Images

by Aditi Walia

Machine learning and deep learning techniques for malware detection and classification play an important role in the mitigation of cybersecurity threats. However, such techniques are often limited by a lack of data. Previous research has shown promising classification results by treating malware executables as images. In this research, we consider data augmentation using noise addition, geometric transformations, and Auxiliary Classifier Generative Adversarial Networks (AC-GAN) for data augmentation of malware images. We train convolution neural networks (CNN) to verify that our generated images accurately model the original malware samples.

ACKNOWLEDGMENTS

I am immensely thankful to Dr. Mark Stamp for his constant guidance, and support throughout this research. His knowledge has played a huge role in the shaping of this project. The Friday zoom meetings and Monday updates have played a huge role in making sure the work was on track, and that all the queries were addressed.

I am thankful to Dr. Nada Attar and Dr. Fabio Di Troia for being a part of the committee for this research. I'm grateful for their time, feedback, and support. I would also like to thank all my professors and faculty members at SJSU for sharing their knowledge and for helping me in this journey.

Finally, I want to thank my family and friends for all their love and support. Without them I wouldn't have made it this far.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malware is malicious software, that is, software that is designed to cause damage to systems. There are different types of malware, including viruses, trojans, and worms [3]. The famous WannaCry [4] malware targeted Microsoft Windows machines and encrypted important data. To obtain the decryption key, a ransom had to be paid in the form of Bitcoin. Attacks like this make it clear that malware classification and identification is imperative for cybersecurity.

Malware classification helps us understand how a particular malware affects systems and also what steps can be taken to protect against it. Classification techniques are often limited by a lack of data needed to train and test models, which can result in weaknesses, such as overfitting [5]. There are many data augmentation techniques that have proven successful in generating data that closely matches the original dataset.

Previous research in the malware domain has focused on a wide variety of features, including opcode sequences, byte n-grams, system calls, and so on. Recently, image-based malware analysis has been shown to be highly effective. Therefore, in this paper, we consider data augmentation for malware images. We experiment with Geometric Transformations using Skimage and also work with deep learning technique for data augmentation called Auxiliary Classifier Generative Adversarial Network (AC-GAN).

We experiment with multiple malware image datasets containing significant numbers of images over a large number of families. To determine the effectiveness of our augmentation techniques, we apply Convolutional Neural Networks (CNN) to distinguish between our (fake) generated samples and the original (real) samples in the same family. We experiment with a variety of hyperparameters and find that CNNs cannot accurately distinguish the real and fake samples, which provides strong evidence that our data augmentation is generating highly realistic samples.

The remainder of this paper is organized as follows. In Chapter 2, we present various background topics, including previous research, techniques and dataset used in this research. Chapter 4 contains our experiments and results. The final Chapter 5 provides our conclusions and includes a discussion of possible directions for future work.

CHAPTER 2

Background

In this section, we discuss previous work done in the area of malware classification, especially in terms of images. We also go over previous works in the field of data augmentation and highlight how having more data to train our models is immensely useful. Next, we give a brief summary of the dataset and also go over different techniques used in this research including - Data Augmentation, AC-GANs, and classification techniques like CNNs.

2.1 Related Work

There are different types of malware families like Winwebsec, Rbot, etc. Often malware from a family have distinct characteristics specific to that family. Ekta Gandotra et al. [5] in their paper highlight that knowing which family a malware is from plays a vital role in mitigating potential attacks if countermeasures for the family that the malware belongs to are already known. Here, we discuss some previous work related to malware classification.

Ekta Gandotra et al. [5] in their paper highlight that there are three main ways of classifying malware which includes static analysis, dynamic analysis, and using machine learning and deep learning.

In static analysis, malware is analyzed without running it. Some common techniques used under static analysis are based on extracting features like opcode frequency distribution, string signatures, and byte sequences. Islam et al. [6] in their paper work with malware classification using string and function feature selection. Shultz et al. [7] in their research work with byte sequences as features. The problem with this type of analysis is that malware executables' code can be easily obfuscated.

In dynamic analysis, malware is analyzed while it is running in a controlled sandbox environment by making use of different monitoring tools like Procmon and

Wireshark. Some previous research work with call graphs where the vertices represent functions, and edges correspond to the calls made to the functions. Another approach was proposed by Tobiyama et al. [8] in their paper which involves running malwares in a controlled environment and analyzing the generated log files. Malware developers have however been very successful in making malwares behave differently in controlled sandbox environments thereby making such techniques ineffective.

When it comes to machine learning and deep learning, techniques like Support Vector Machines, Random Forest, CNN have been used for malware classification. With the rise in the number of malware attacks, researchers find it easier to use machine learning and deep learning techniques as opposed to manual analysis of malware.

Santos et al. [9] used TF-IDF (Term Frequency Inverse Document Frequency) to create feature vectors from binary files but this involved disassembly for opcode extraction. Kolter et al. [10] in their paper used Naive Bayes, SVMs with boosting which they found was the best in malware classification. They made use of n-grams of byte codes to create feature vectors. This technique works well but multiple combinations of n-grams are difficult to evaluate. Coming to deep learning methods, Saxe et al. [11, 12] in their paper provided a technique for static malware analysis using feedforward deep neural networks. The problem with these approaches is that feature extraction is tedious.

Conti et al. [13] in their paper first proposed a taxanomy for binary fragments and their visualization. For malwares the images are representative of four major sections. The code is contained in the .text section and .data consists of initialized and uninitialized code. The .rsrc consists of resources used by the executable.

Nataraj et al. [14] classified malware by converting executable binary files into gray-scale bitmap images. They then applied an abstract representation technique

to compute texture features to classify malware.We build on the idea of converting malware binary files to grayscale images and then work on data augmentation techniques, and classification on our dataset using CNN which is widely used for image based classification.

For data augmentation with images, there are a lot of popular approaches which fall under three major categories as stated in [12]. These include basic image manipulations like geometric transformations and noise addition, deep learning approaches like GANs, and meta learning approaches like AutoAugment.

Odena et al. first came up with a variant of GAN called AC-GAN by making use of conditional image generation. AC-GAN works well with high resolution images for training. Like standard GANs, AC-GANs have a discriminator and a generator model. The discriminator however in case of AC-GANs predicts whether the image is real or fake as well as which class it belongs to in contrast to GAN discriminator which just predicts real or fake. We make use of AC-GANs to generate fake malware images to augment our dataset.

For data augmentation, Catak et al. [15] in their work built on the idea that malware developers try and evade malware from being flagged by classification techniques by incorporating noise in the binary files. They worked with 3 channel images and introduced noise in their dataset to make it resilient to introduction of noise in the executables. The classification results after introducing Poisson, Laplace and Gaussian noise looked very promising.

In this paper, we work with noise related augmentation using Poisson and Laplace noise, Geometric Transformations using SkImage and Deep Learning techniques like AC-GAN. We evaluate the datasets generated using these techniques by setting CNN classification on our original un-augmented dataset as baseline.

2.2 Techniques

In this section we go over the techniques and terminologies used in this research. We first go over the idea behind data augmentation. Then we briefly discuss data augmentation techniques followed by CNN, and AC-GAN.

2.2.1 Data Augmentation

Deep learning models perform better with more data however it is not always feasible to have enough data to train. Due to limited training data, models are often prone to overfitting. Data Augmentation is a technique that can be used to create more data to increase the training dataset and hence deal with overfitting.

Multiple types of data can be augmented like text, audio, images. etc. For this paper, we first experiment with Poisson and Laplace noise using Python library imagug and then work Geometric transformations - using Skimage

Poisson noise [16], also called as photon noise or shot noise is caused due to the quantitative nature of light which cannot be divided into fractional parts. the signal magnitude is directly related to poisson noise magnitude. Laplacian noise [17] is created by adding noise from a laplace distribution. Like Poisson noise, it is also statistical in nature. Imgaug Python library consists of multiple APIs for augmenting images including arithmetic augmentors AdditiveLaplaceNoise and AdditivePoissonNoise.

Skimage, also known as scikit-image is library which comes with sub-packages like transform, color which can be used for creating more images from existing dataset. We also use deep learning technique AC-GAN for our augmentation experiments.

2.2.2 Convolutional Neural Network

CNN is a popular deep learning algorithm when it comes to images. It is used to identify local structure by assigning importance to aspects in the input image. Traditional neural networks have fully connected layers but with huge images, the

complexity of the models increase. In CNN, we have different layers which are not fully connected so it performs more efficiently due reduced number of weights and parameters.

The input image can be of different types like RGB, Grayscale, etc. ConvNet reduces images in size which makes it easier to process them while preserving the important aspects of the image.

A CNN layer is primarily composed of Convolution Layer and a Pooling Layer. The architecture of a vanilla CNN is as describe in Figure 1 .

Convolution Layer also called as Kernel or Filter performs convolution function using a matrix by going over the input image matrix depending on the stride length. The convolution function is used to extract features such as edges and colors.

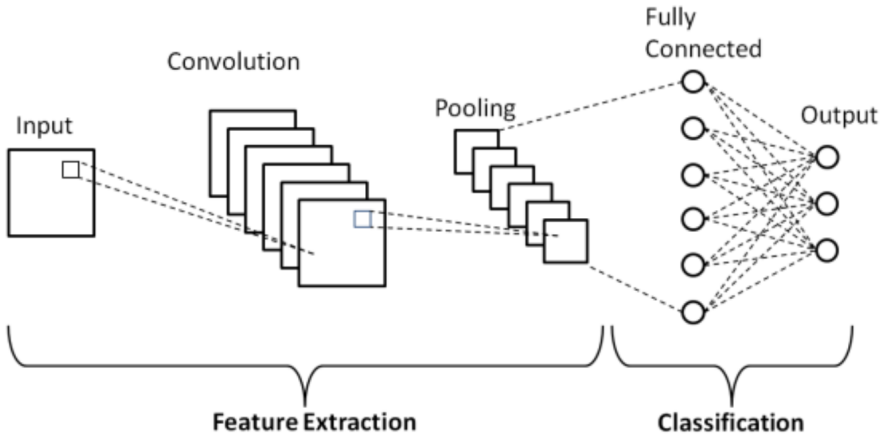

Figure 1: CNN architecture [1]

The Pooling Layer is similar to Convolution Layer but it doesn't perform learning and solely focuses on reducing the size of data. There are two types of pooling layers- Max Pooling and Average Pooling. Max pooling substitutes the pixels covered by the filter with max among the pixel values. It is helps in reducing the noise in images. The Average pooling substitutes the pixels covered by the filter with average of the

pixel values and is used to smooth the images. The output from the convolution layers is then flattened and fed to the neural network for classification purpose.

2.2.3 Auxiliary Classifier Generative Adversarial Network

The fundamental behind the working of GANs can be best described by the analogy of generating more complex random numbers by making some simpler random numbers go through functional transformations. To find this function, a direct or indirect training method can be used. In direct training, the difference between true and generated values is backpropagated. In indirect training, the true and generated values are fed to a downstream task which optimizes itself and the generator task in a game like way to achieve generation close to the true values.

GANs make use of indirect training. The downstream task is called a discriminator task. The discriminator tries to distinguish the real and fake images and the generator tries to fool the discriminator by generating images as close as possible to the real samples. The discriminator minimizes the classification error and the generator maximizes it it.

AC-GAN was developed by researchers from Google. AC-GANs are widely used to train and generate multi-class family images without any manual intervention. They are know to provide high quality images as compared to other models.

The underlying architecture is similar to GAN and consists of a generator and a discriminator. It was build on the idea of a Conditional GAN (C-GAN) wherein the class label is also provided as an input to the generator which makes the image generation conditional on the input label. The difference between C-GAN and AC-GAN is that the discriminator is not provided the class label. The discriminators distinguishes whether an image is real or fake, and also to predict what class it belongs to. The difference in architectures is depicted in Figure 2

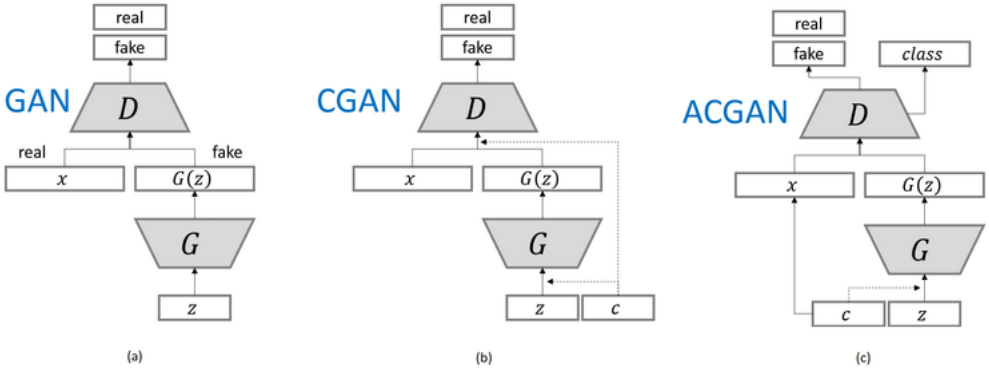

Figure 2: GAN, C-GAN, AC-GAN architecture comparison [2]

CHAPTER 3

Implementation

In this chapter, we give an overview of the dataset used for this research. We also discuss the software and hardware specifications used to run the experiments explained in Chapter 4

3.1 Dataset

The dataset was acquired from Dang and Stamp [18]. It consists of 26412 samples from 20 families as listed in Table 1. Samples for Winwebsec, Zbot, and Zeroaccess families are from the Malicia dataset. The other families' samples are obtained from dataset [19]. We briefly go over each of the malware families in the following paragraphs.

Adload disables proxy settings on windows by downloading an exe file which is stored remotely [20].

Agent downloads unwanted software from remote server [21].

Alureon makes confidential data like usernames, passwords, and credit card information available to hackers and is also capable of corrupting driver files [22].

BHO enables hackers to perform malicious actions on the infected system [23].

CeeInject is obfuscated and can perform any actions. The obfuscation makes it hard to detect [24].

Cycbot provides hackers control of a system by connecting it to a server and executing commands received through it. It also lets attackers perform backdoor functions [25].

DelfInject steals sensitive information like usernames and passwords [26] .

Table 1: Malware families in dataset

Family	Type	Samples
Adload	Trojan Downloader	1050
Agent	Trojan	842
Alureon	Trojan	1328
BHO	Trojan	1176
CeeInject	VirTool	894
Cycbot	Backdoor	1029
DelfInject	VirTool	1146
FakeRean	Rogue	1063
Hotbar	Adware	1491
Lolyda.BF	Password Stealer	915
Obfuscator	VirTool	1445
OnLineGames	Password Stealer	1293
Rbot	Backdoor	1017
Renos	Trojan Downloader	1312
Startpage	Trojan	1136
Vobfus	Worm	926
Vundo	Trojan Downloader	1793
Winwebsec	Rogue	3651
Zbot	Password Stealer	1786
Zeroaccess	Trojan Horse	1119
Total		26412

FakeRean fake scans the system and raises false vulnerabilities. It then prompts to pay to get the system cleaned [27].

Hotbar displays ads on browsers. It can be installed along side different softwares and through unauthentic websites [28].

Lolyda.BF monitors user's network activity to steal user names and passwords and sends them to remote servers [29].

Obfuscator can perform any actions and the obfuscation makes it hard to detect [30].

OnLineGames tries to get login information of online games by injecting dynamic link library files into processes and capturing keystroke events [31].

Rbot like Cycbot provides hackers backdoor access to system [32].

Renos downloads spyware detecting software like SpyDawn and raises false warnings of the system being infected by spyware and prompts system users to pay money to fix the issues [33].

Startpage usually change browser's homepage but can also perform other malicious actions [34].

Vobfus downloads malware on the system and is usually spread through USB drives [35].

Vundo uses pop-up ads and can download malware to systems [36].

Winwebsec like Fakerean raises false vulnerability alerts and asks for money to clean up the system [37].

Zbot steals personal information and can provide hackers access to systems. It is spread through spam emails [38].

Zeroaccess downloads malware on host machines [39] .

3.1.1 Sample Images

Figures 3 4 shows grayscale images belonging to Alureon and Fakerean. These images highlight similarities between images of the same family and differences between images of different families.

3.2 Programming details

The experiments for this research were run on Google Colab Pro and on a MacBook Pro 2019 system using Jupyter Notebook. The software details are specified in Table 2 .

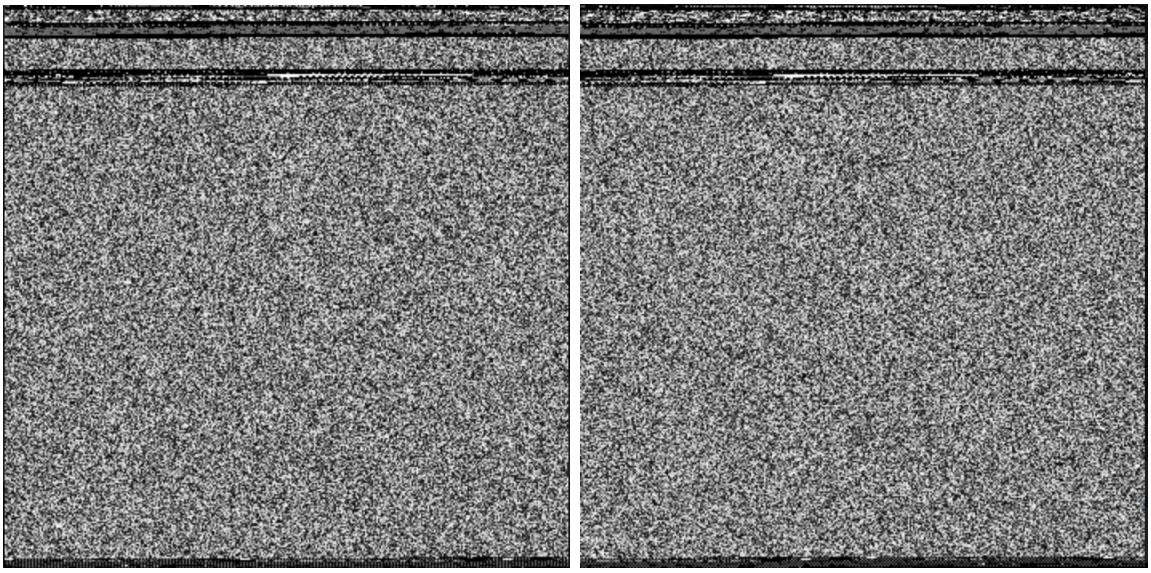

Figure 3: Sample images alureon

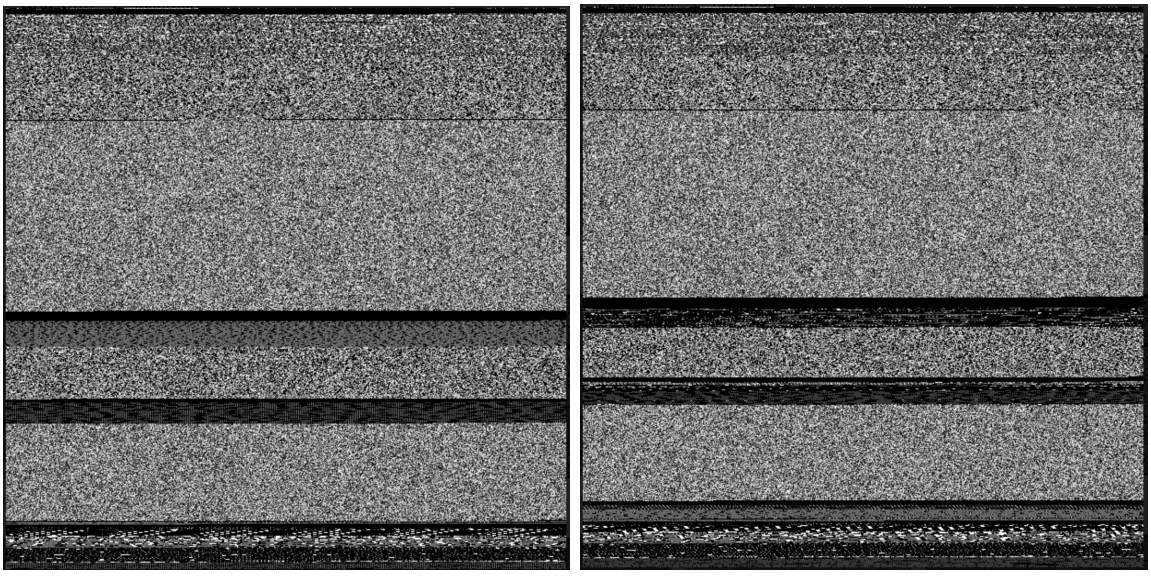

Figure 4: Sample images fakerean

3.3 CNN Hyper parameters

For running our CNN models, we experiment with different hyper parameters. The best results are obtained with relu activation function, adam optimizer with learning rate 0.001 and 50 epochs. Table 3 lists all the tested values and and also which selection of parameters resulted in best accuracy for malware classification.

Table 2: Software details

Software	Version
OS	macOS Mojave
Python	Python 3.7.4
Jupyter Notebook	6.0.1
Anaconda Navigator	1.9.12
Google Colab	Pro
Tensorflow	2.4.1
PIL	7.1.2

Table 3: CNN model parameters

Parameter	Value
Classes	18
Batch Size	32 64 **128**
Test Size	30%
Kernel Size	3×3, 5×5
Pool size	2×2
Activation function	sigmoid, tanh, **relu**
Optimizer	**adam**
Epochs	**50**, 100

CHAPTER 4

Results

In this section we go over the results of our experiments. We first present results for experiments conducted to create images from the executable malware files. These experiments involve experimenting with image size by resizing the original image and also by creating specific sized images using specific number of bytes from executables in the dataset.

Next we experiment with multi-class classification using CNN on the original dataset, and also on a dataset comprising of 700 files from each of the families for training and 100 files each for testing to create baseline for our next set of experiments for data augmentation. We chose 700 as number of images for training purpose per class because Agent has 842 files in our exe dataset. We reserved 100 number of images per class for testing.

For the experiments related to data augmentation, we first try some basic geometric transformations using Skimage. We then experiment with Python imgaug library and introduce Laplacian and Poisson noise in our dataset. We create datasets with 700 fake malware images for each of the families. We experiment with including different percentages of the fake images in our training dataset and then validate by testing against the real dataset comprising of 100 images from each of the families.

Next we experiment with AC-GAN. We follow a similar approach as the previous augmentations. Using our saved generator model, and we create 700 fake malware images dataset. We use this to experiment with inclusion of different percentages of fake data in our dataset to see how our CNN classification accuracy is affected.

4.1 Exe to grayscale image experiments

Our dataset consists of executables from different malware families. The binaries are stored under folders representing each of the families. For converting the binaries

to images, we first walk through the directory and then for each of the files, we read the file byte by byte and store it in a byte array. Depending on the size of the byte array, the height and width of the image is determined. For all the experiments, we work with square images so the height and width are the same for all images in the dataset.

We use Image module from PIL in Python to create image from byte array. We create the grayscale image dataset using these images. Next, we run CNN on our dataset. The train accuracy is 89% and the test accuracy is 78% . Figure 5 shows the model accuracy and Figure 6 shows the model loss for CNN run on the original grayscale dataset. Figure 7 shows a plot of the confusion matrix. Obfuscator, Agent and Rbot are the most misclassified families in this experiment. 35%, 45% and 46% samples are classified correctly for these families.

For a baseline for our augmentation techniques, we then create a real images dataset for training using 700 images from each of the families. We use 100 images each to create a real images dataset for testing purpose. The train accuracy is 87% and the test accuracy is 76%. Figure A.12 shows the model accuracy and Figure A.13 shows the model loss for CNN run on dataset comprising of 700 samples per family for training. Figure A.14 shows a plot of the confusion matrix. The most misclassifed families are again Obfuscator, Rbot, and Agent with only 24%, 44% and 55% samples getting classified correctly.The drop in the accuracy can be attributed to using less number of samples compared to the previous experiment. Next we experiment with different image re-sizing techniques.

4.1.1 Resizing using python-resize-image

We use Python library python-resize-image to resize images. Internally the library uses Lanczos resampling [40] to resize the image. We create 64×64 sized image

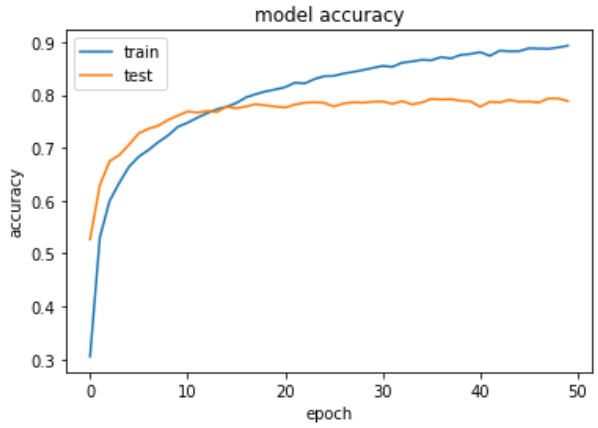

Figure 5: CNN model accuracy original grayscale dataset

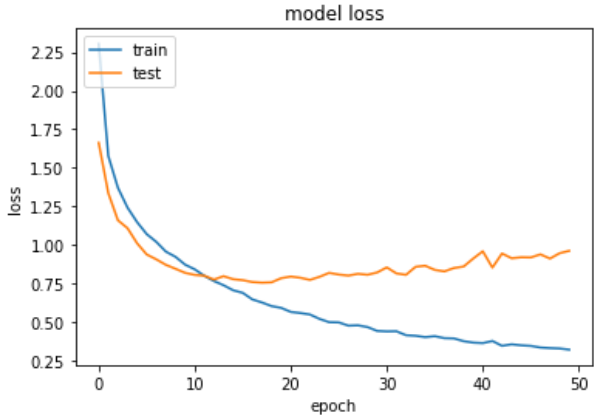

Figure 6: CNN model loss original grayscale dataset

datasets. With 64×64 dataset the test accuracy is 83% and the train accuracy is 82%. Figure A.21 shows the model accuracy and Figure A.22 shows the model loss. Figure A.23 shows a plot of the confusion matrix. Compared to the original grayscale dataset, more percentage of samples are classified correctly.

4.1.2 Fixed size images using fixed number of bytes

Next, we experiment with creating 32×32 and 64×64 sized image datasets using the initial 1024 and 4096 bytes from the executable files. The test accuracy with 32×32 is 91% and the train accuracy is 90%. Figure A.15 shows the model accuracy and Figure A.16 shows the model loss. Figure A.17 shows a plot of the confusion

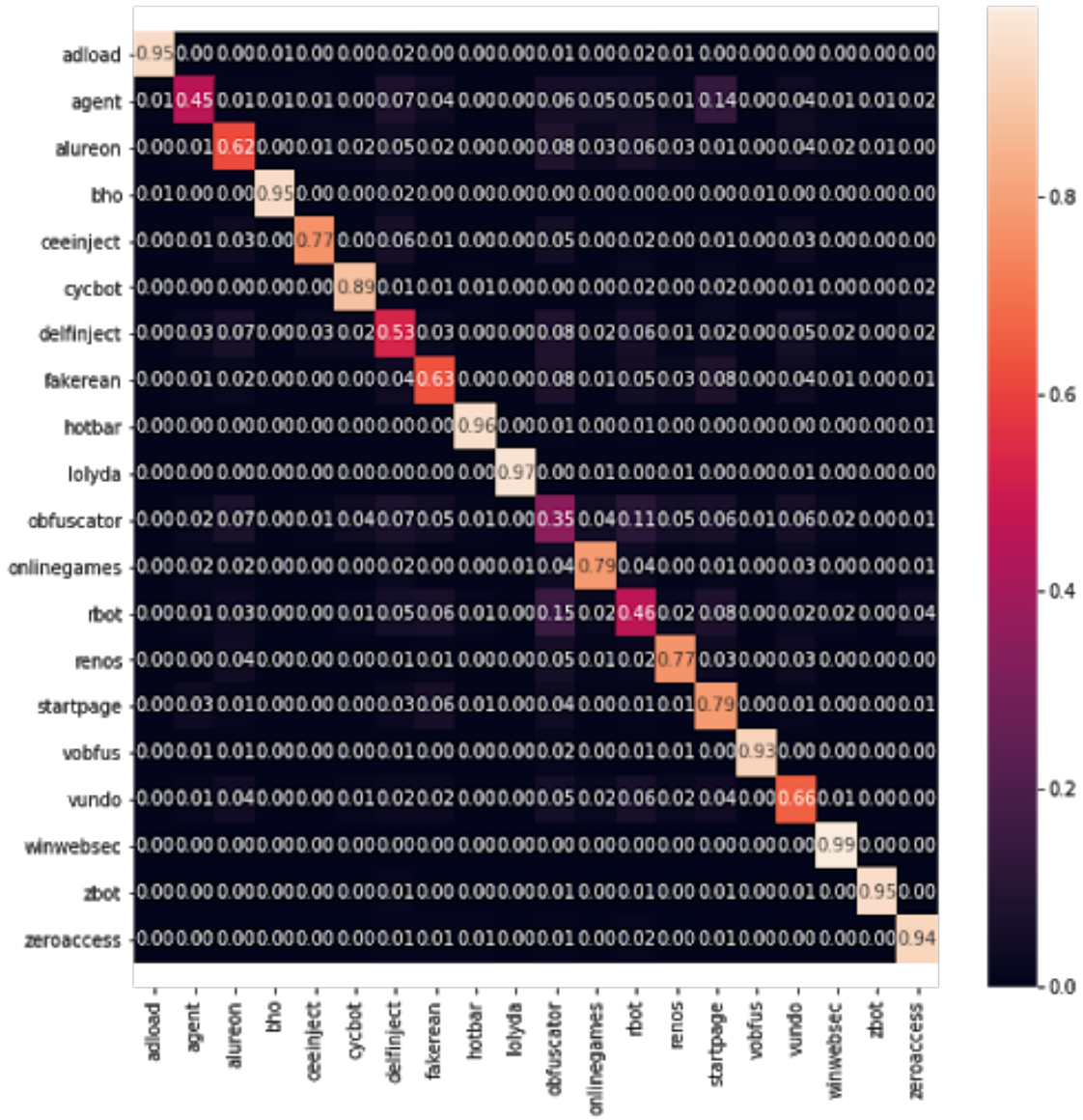

Figure 7: CNN confusion matrix original grayscale dataset

matrix. The jump in accuracy could be indicative of some pattern in the initial 1024 bytes which is enough to classify the malware.

With 64×64 dataset test accuracy is 90% and the train accuracy is 95%. Figure A.18 shows the model accuracy and Figure A.19 shows the model loss. Figure A.20 shows a plot of the confusion matrix.

Even though the accuracy improved a lot in both these cases, we lost the visual patterns which were very evident in the original dataset. We tried to generate AC-GAN images using these datasets but the generated images looked like noise. Since our next set of augmentation experiments rely on spatial orientation of features, we chose to skip using this technique to resize. We finally ended up using Pytorch transforms to resize our original dataset to 64×64 internally when using AC-GANs.

4.2 Augmentation

For all the augmentation experiments, we set a baseline of creating 700 fakes per family which results in 14,000 fake samples divided over 20 families. We use the dataset created using 700 real images per family which results in 14,000 real samples over 20 families. For testing purpose, we use 100 images from each of the families which are not part of the real 14,000 image dataset. This provides us 2000 images for testing divided over 20 families. We train by including permutations of different percentages of samples from real, and fake datasets. This is explained in more detail in the next subsections.

4.2.1 Augmentation using Skimage

For the first set of experiments, we create fake images by using Skimage library. We perform a random operation on the input image out of three specified operations- Random rotation, random , horizontal flip. We create 700 fake images each for all the families.

First we create a combined dataset with families like adload, adload fake, winwebsec real, winwebsec fake, etc. We run CNN on this dataset to see how distinguishable our generated images are. The confusion matrix is as listed in Figure 8. We note that some samples of Hotbar and Startpage are misclassified as the fake Hotbar and fake Obfuscator class.

For the next experiment, for training our CNN. we use different permutations from real and fake datasets, and observe the results. For example if we use 10% images from the fake dataset, we pick 10% from each of the malware families for testing along with 90% from each of the malware families in the real dataset. Tables 4, 5, 6 describe all the the combinations tested for and the percentage of fake samples in the training dataset along with the precision, recall and F1-scores. From the results, we can conclude that the fakes are close to real images. We see best improvement in F1-score when the dataset has less number of samples. The confusion matrices for 7000 real and 7000 fake samples, and 0 real and 14,000 fake samples is provided in Figure A.25 and Figure A.24 .

Table 4: Experiment results Skimage Augmentation keeping samples count 14,000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	14,000	0.76	0.76	0.76
1400	12,600	0.77	0.76	0.76
2800	11,200	0.76	0.75	0.75
4200	9800	0.76	0.75	0.75
5600	8400	0.74	0.73	0.73
7000	7000	0.75	0.73	0.74
8400	5600	0.74	0.71	0.72
9800	4200	0.72	0.71	0.71
11,200	2800	0.71	0.70	0.70
12,600	1400	0.73	0.71	0.72
14,000	0	0.70	0.68	0.69

4.2.2 Augmentation using Poisson Noise

Next, we add Poisson noise using imgaug library on the input image. We use lambda parameter value for poisson distribution as 5.0. The recommended values are between 0.0 to 10.0.

First we create a combined dataset with families like adload, adload fake, winwebsec real, winwebsec fake, etc. We run CNN on this dataset to see how distinguishable

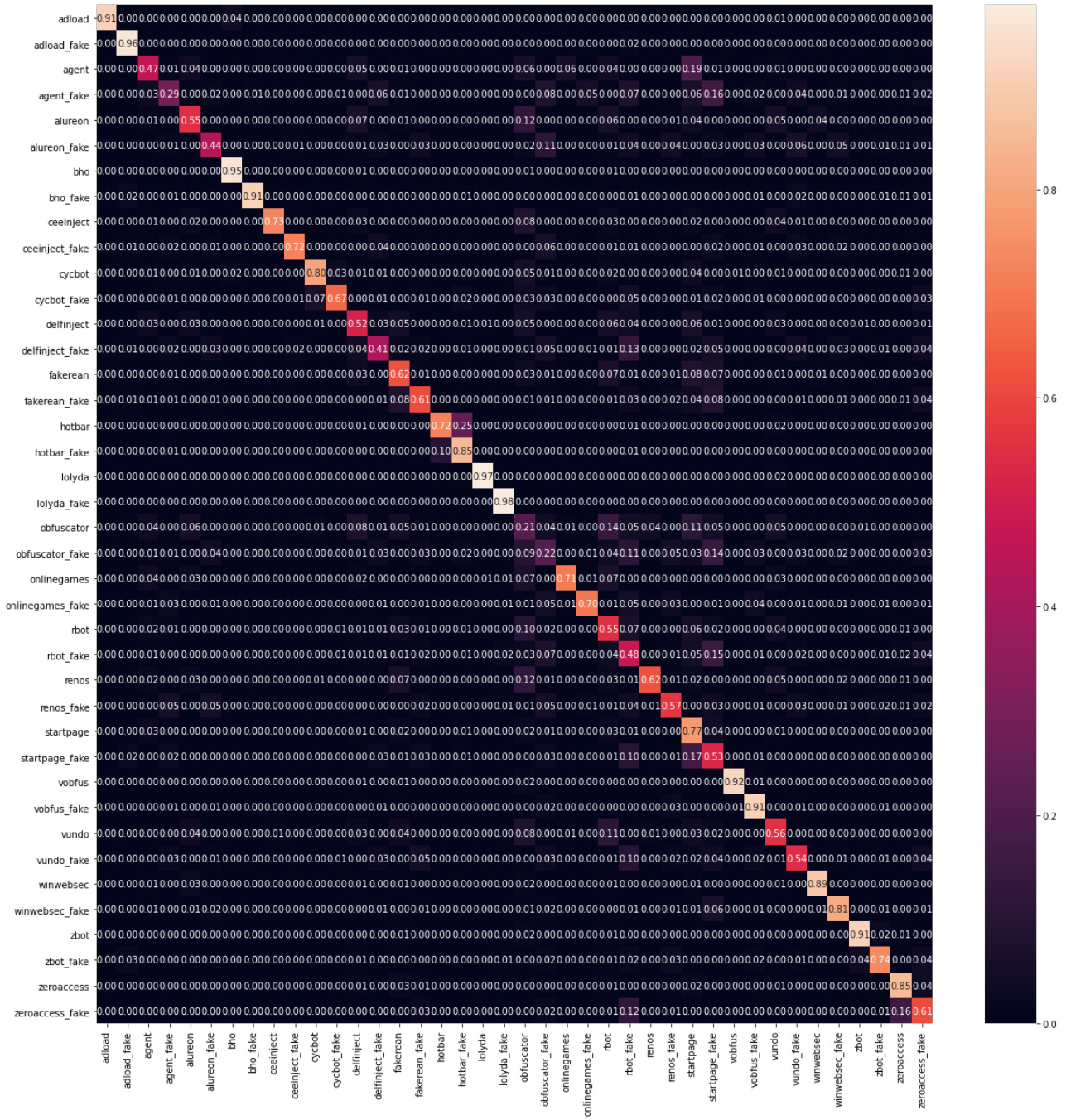

Figure 8: CNN confusion matrix 40 classes Skimage augmentation

our generated images are. The confusion matrix is as listed in Figure 9. We get the most indistinguishable images using Poisson noise as is evident from the confusion matrix.

Table 5: Experiment results Skimage Augmentation keeping real samples count 14,000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	14,000	0.78	0.77	0.77
1400	14,000	0.77	0.76	0.76
2800	14,000	0.76	0.76	0.76
4200	14,000	0.78	0.77	0.77
5600	14,000	0.78	0.76	0.77
7000	14,000	0.77	0.75	0.76
8400	14,000	0.77	0.76	0.76
9800	14,000	0.77	0.76	0.76
11,200	14,000	0.78	0.76	0.77
12,600	14,000	0.79	0.77	0.78
14,000	14,000	0.79	0.77	0.78

Table 6: Experiment results Skimage Augmentation keeping real samples count 2000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	2000	0.59	0.58	0.58
200	2000	0.57	0.58	0.57
400	2000	0.63	0.61	0.62
600	2000	0.60	0.58	0.59
800	2000	0.62	0.61	0.61
1000	2000	0.63	0.60	0.61
1200	2000	0.65	0.62	0.63
1400	2000	0.61	0.60	0.60
1600	2000	0.64	0.62	0.63
1800	2000	0.62	0.60	0.61
2000	2000	0.66	0.64	0.65

We create 700 fake images each for all the families. For training our CNN, we use different permutations and observe the results. Tables 7, 8, 9 describe all the the combinations tested for the percentage of fake samples in the dataset. This experiment also creates fakes which are very close to the real samples.The confusion matrices for 7000 real and 7000 fake samples, and 0 real and 14,000 fake samples is provided in Figure A.27 and Figure A.26 .

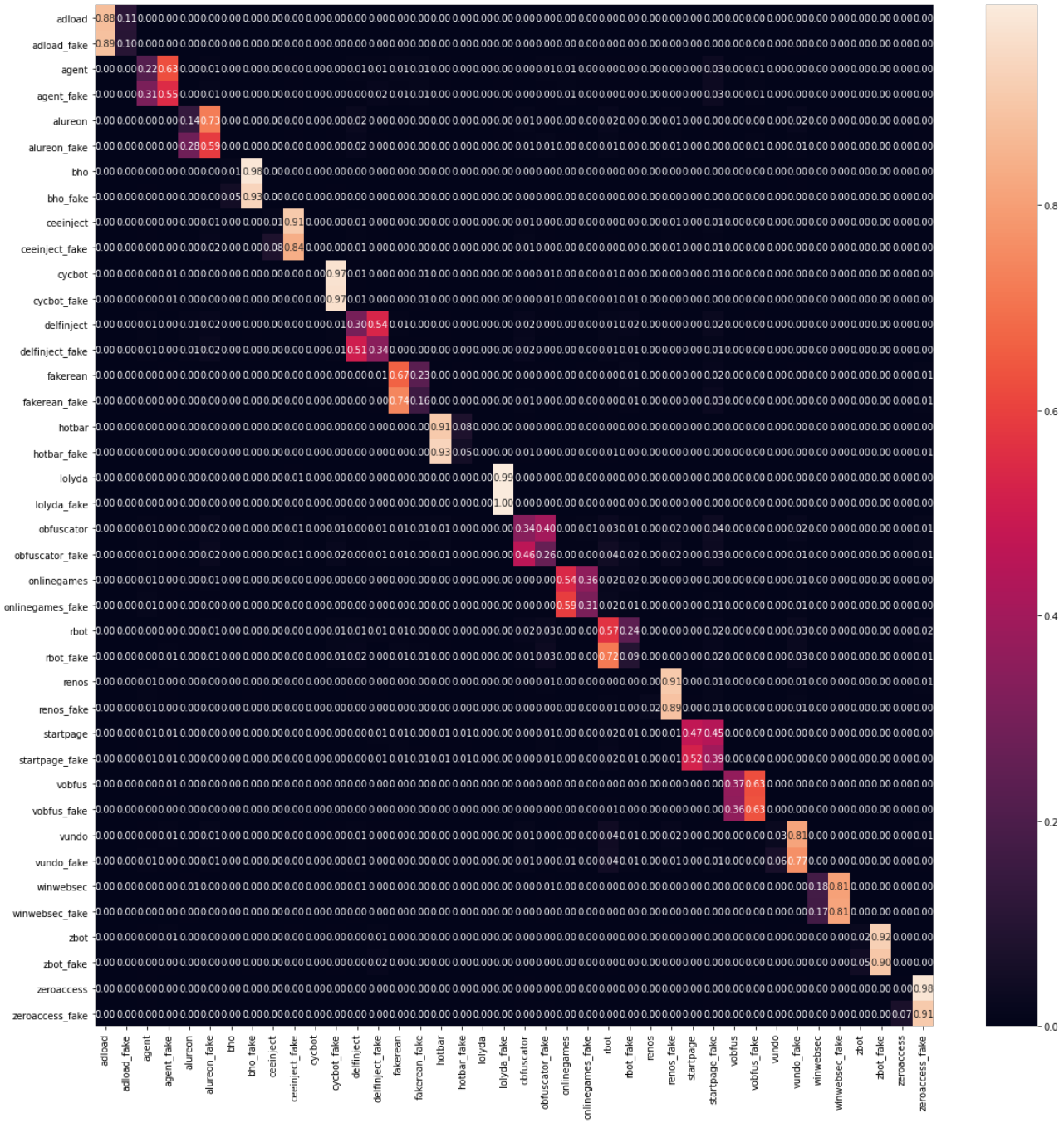

Figure 9: CNN confusion matrix 40 classes Poisson Noise augmentation

Table 7: Experiment results Poisson Noise keeping samples count 14,000

Fake samples count	Real samples count	Precision	Recall	F1-score
0	14,000	0.76	0.75	0.75
1400	12,600	0.77	0.76	0.76
2800	11,200	0.76	0.76	0.76
4200	9800	0.77	0.75	0.76
5600	8400	0.74	0.74	0.74
7000	7000	0.75	0.74	0.74
8400	5600	0.72	0.72	0.72
9800	4200	0.75	0.75	0.75
11,200	2800	0.77	0.75	0.76
11,200	1400	0.76	0.76	0.76
14,000	0	0.79	0.77	0.78

Table 8: Experiment results Poisson Augmentation keeping real samples 14,000 count

Fake samples count	Real samples count	Precision	Recall	F1-score
0	14,000	0.78	0.76	0.77
1400	14,000	0.78	0.76	0.77
2800	14,000	0.77	0.77	0.77
4200	14,000	0.75	0.75	0.75
5600	14,000	0.76	0.74	0.75
7000	14,000	0.77	0.76	0.75
8400	14,000	0.76	0.75	0.75
9800	14,000	0.77	0.76	0.76
11,200	14,000	0.76	0.75	0.76
12,600	14,000	0.77	0.76	0.76
14,000	14,000	0.78	0.77	0.77

4.2.3 Augmentation using Laplacian Noise

Similarly, we apply Laplacian noise using imgaug library on the input image. We use scale=0.1*255 for the noise. We create 700 fake images each for all the families.

First we create a combined dataset with families like adload, adload fake, winwebsec real, winwebsec fake, etc. We run CNN on this dataset to see how distinguishable our generated images are. The confusion matrix is as listed in Figure 10 . We note that the images are distinguishable and compared to Poisson noise results, this technique

Table 9: Experiment results Poisson Augmentation keeping real samples count 2000

Fake samples count	Real samples count	Precision	Recall	F1-score
0	2000	0.62	0.60	0.61
200	2000	0.66	0.62	0.64
400	2000	0.63	0.61	0.62
600	2000	0.65	0.61	0.63
800	2000	0.61	0.61	0.61
1000	2000	0.64	0.62	0.63
1200	2000	0.63	0.62	0.62
1400	2000	0.62	0.60	0.61
1600	2000	0.64	0.64	0.64
1800	2000	0.64	0.63	0.63
2000	2000	0.64	0.63	0.63

didn't perform well.

For training our CNN, we use different permutations and observe the results. Tables 10, 11, 12 describe all the the combinations tested for the percentage of fake samples in the dataset. This experiment also creates fakes which are very close to the real samples. We again see an increase in F1-score when the inital dataset has lesser number of training samples and is then augmented.The confusion matrices for 7000 real and 7000 fake samples, and 0 real and 14,000 fake samples is provided in Figure A.29 and Figure A.28 .

4.3 ACGAN

Next, we train AC-GAN generator and discriminator, and then generate 700 fake samples for each of the family. First we create a combined dataset with families like adload, adload fake, winwebsec real, winwebsec fake, etc. We run CNN on this dataset to see how distinguishable our generated images are. The confusion matrix is as listed in Figure 11.

For the next experiment, for training our CNN, we use different permutations and observe the results. Tables 13, 14, 15 describe all the the combinations tested for

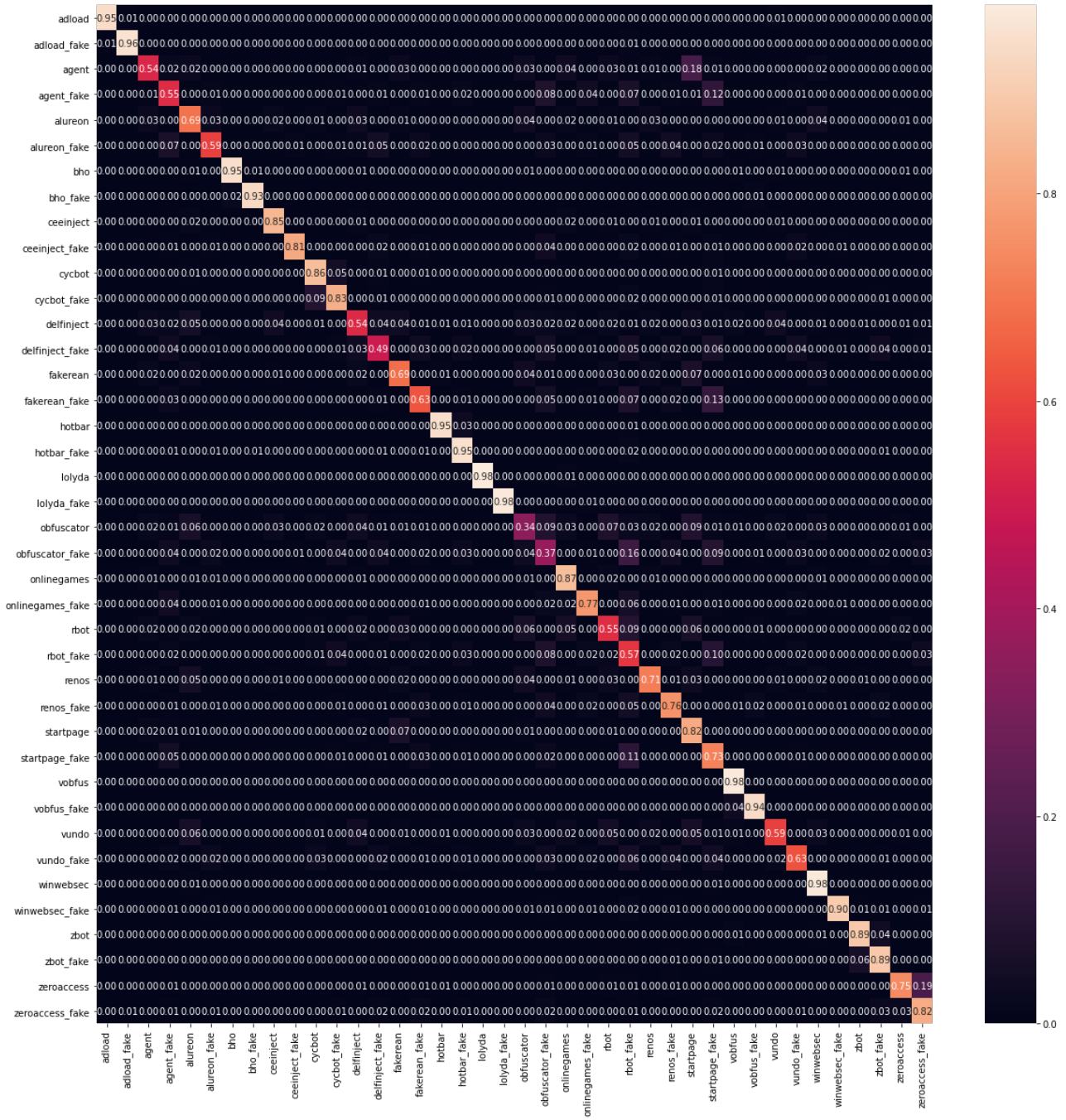

Figure 10: CNN confusion matrix 40 classes Laplace augmentation

the percentage of fake samples in the dataset. The introduction of these fake samples actually skews the classifier. The more the number of fakes are added, the lower the classification accuracy is. The confusion matrices for 7000 real and 7000 fake samples,

Table 10: Experiment results Laplacian Noise keeping samples count 14,000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	14,000	0.77	0.76	0.76
1400	12,600	0.76	0.76	0.76
2800	11,200	0.76	0.75	0.75
4200	9800	0.75	0.74	0.74
5600	8400	0.75	0.74	0.74
7000	7000	0.75	0.74	0.74
8400	5600	0.73	0.73	0.73
9800	4200	0.76	0.75	0.65
11,200	2800	0.74	0.73	0.73
12,600	1400	0.77	0.75	0.76
14,000	0	0.77	0.76	0.76

Table 11: Experiment results Laplace Augmentation keeping real samples count 14,000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	14,000	0.76	0.75	0.75
1400	14,000	0.76	0.76	0.76
2800	14,000	0.78	0.76	0.77
4200	14,000	0.75	0.75	0.75
5600	14,000	0.77	0.76	0.76
7000	14,000	0.75	0.75	0.75
8400	14,000	0.77	0.75	0.76
9800	14,000	0.77	0.76	0.76
11,200	14,000	0.76	0.75	0.75
12,600	14,000	0.77	0.77	0.77
14,000	14,000	0.76	0.75	0.75

and 0 real and 14,000 fake samples is provided in Figure A.31 and Figure A.30 .

Table 12: Experiment results Laplace Augmentation keeping real samples count 2000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	2000	0.59	0.58	0.58
200	2000	0.63	0.61	0.62
400	2000	0.62	0.60	0.61
600	2000	0.62	0.60	0.61
800	2000	0.64	0.61	0.61
1000	2000	0.63	0.63	0.63
1200	2000	0.63	0.61	0.62
1400	2000	0.63	0.62	0.62
1600	2000	0.62	0.61	0.61
1800	2000	0.64	0.64	0.64
2000	2000	0.65	0.63	0.64

Table 13: Experiment results AC-GAN keeping samples count 14,000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	14,000	0.77	0.76	0.76
1400	12,600	0.76	0.75	0.75
2800	11,200	0.71	0.70	0.70
4200	9800	0.75	0.73	0.74
5600	8400	0.70	0.70	0.70
7000	7000	0.70	0.68	0.69
8400	5600	0.66	0.64	0.65
9800	4200	0.63	0.60	0.61
11,200	2800	0.58	0.57	0.57
12,600	1400	0.52	0.47	0.49
14,000	0	0.24	0.07	0.11

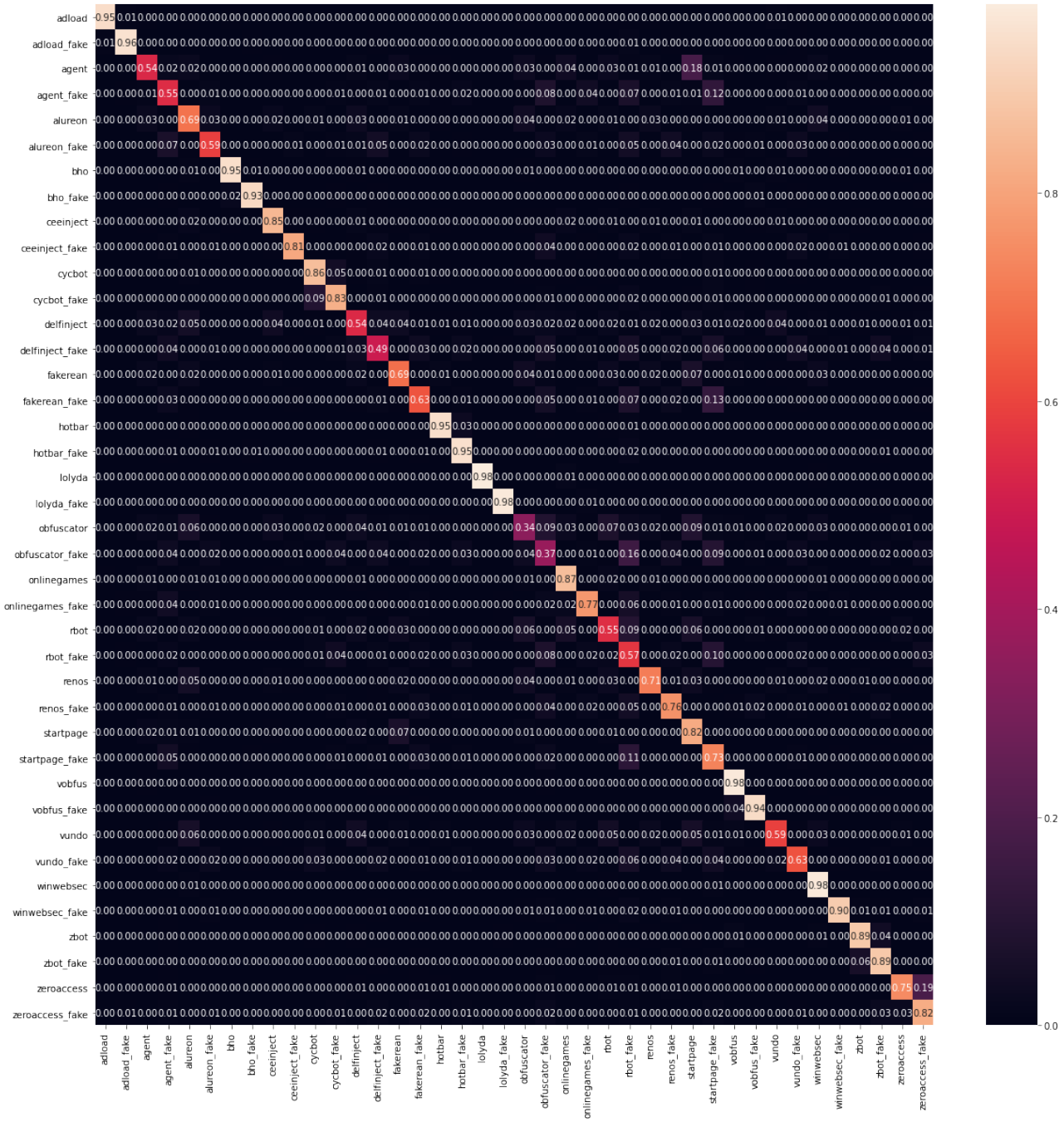

Figure 11: CNN confusion matrix 40 classes AC-GAN augmentation

Table 14: Experiment results AC-GAN keeping real samples count 14,000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	14,000	0.77	0.76	0.76
1400	14,000	0.76	0.75	0.75
2800	14,000	0.76	0.76	0.76
4200	14,000	0.75	0.74	0.74
5600	14,000	0.75	0.74	0.74
7000	14,000	0.77	0.75	0.76
8400	14,000	0.73	0.73	0.73
9800	14,000	0.73	0.72	0.72
11,200	14,000	0.75	0.73	0.74
12,600	14,000	0.74	0.73	0.73
14,000	14,000	0.73	0.72	0.72

Table 15: Experiment results AC-GAN keeping real samples count 2000

Fake samples count	**Real samples count**	**Precision**	**Recall**	**F1-score**
0	2000	0.60	0.58	0.59
200	2000	0.58	0.57	0.57
400	2000	0.59	0.58	0.58
600	2000	0.56	0.55	0.55
800	2000	0.61	0.59	0.60
1000	2000	0.61	0.57	0.59
1200	2000	0.58	0.56	0.57
1400	2000	0.58	0.56	0.57
1600	2000	0.58	0.56	0.57
1800	2000	0.57	0.55	0.56
2000	2000	0.57	0.55	0.56

CHAPTER 5

Conclusion and Future Work

Malware classification is imperative for mitigation when it comes to malware related attacks. The classification models are however often limited by the lack of data. Previous research had worked on malware as images for classification and also on data augmentation. The best classification accuracy through our vanilla CNN model was 91% with 64×64 images however we lost spatial features by picking up just the initial 4096 bytes. With the non-augmented dataset using original grayscale images, the best accuracy was 78%

We explored techniques like Noise addition, Random flip and also deep learning technique AC-GAN. We were able to generate images closes to the real ones using Poisson Noise. WIth AC-GAN we were able to create fakes which skew up CNN classification when added to the dataset. This can be used by malware hackers to poison datasets. We propose to make the classifier strong by including such fakes in our dataset. With the other techniques, we achieved precision and recall of around 70% irrespective the number of fake in the training dataset because of the similarity in the fakes.

In terms of future work, we can experiment with conversion of executable files to three channel images. We can also work on stabilizing our AC-GAN model to generate images close to fakes and also experiment with 128×128 sized images. We can also experiment with augmentation techniques like AutoAugment and explore other noise additions to enhance our classifiers.

LIST OF REFERENCES

[1] V. H. Phung, E. J. Rhee, *et al.*, ''A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets,'' *Applied Sciences*, vol. 9, no. 21, p. 4500, 2019.

[2] A. Mino and G. Spanakis, ''Logan: Generating logos with a generative adversarial neural network conditioned on color,'' in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 965--970.

[3] 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.

[4] ''Wannacry ransomware attack,'' Apr 2021. [Online]. Available: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack

[5] E. Gandotra, D. Bansal, and S. Sofat, ''Malware analysis and classification: A survey,'' *Journal of Information Security*, vol. 05, pp. 56--64, 01 2014.

[6] R. Islam, R. Tian, L. Batten, and S. Versteeg, ''Classification of malware based on string and function feature selection,'' in *2010 Second Cybercrime and Trustworthy Computing Workshop*. IEEE, 2010, pp. 9--17.

[7] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, ''Data mining methods for detection of new malicious executables,'' in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 2000, pp. 38--49.

[8] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, ''Malware detection with deep neural network using process behavior,'' in *2016 IEEE 40th annual computer software and applications conference (COMPSAC)*, vol. 2. IEEE, 2016, pp. 577--582.

[9] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, ''Opcode sequences as representation of executables for data-mining-based unknown malware detection,'' *Information Sciences*, vol. 231, pp. 64--82, 2013.

[10] J. Z. Kolter and M. A. Maloof, ''Learning to detect malicious executables in the wild,'' in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 470--478.

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

[12] C. Shorten and T. M. Khoshgoftaar, ''A survey on image data augmentation for deep learning,'' *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.

[13] G. Conti, S. Bratus, A. Shubina, A. Lichtenberg, R. Ragsdale, R. Perez-Alemany, B. Sangster, and M. Supan, ''A visual study of primitive binary fragment types,'' *White Paper, Black Hat USA*, 2010.

[14] 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*, 2011, pp. 1--7.

[15] F. O. Catak, J. Ahmed, K. Sahinbas, and Z. H. Khand, ''Data augmentation based malware detection using convolutional neural networks,'' *PeerJ Computer Science*, vol. 7, p. e346, 2021.

[16] ''Poisson.'' [Online]. Available: https://en.wikipedia.org/wiki/Shot_noise

[17] ''Laplacian.'' [Online]. Available: https://en.wikipedia.org/wiki/Additive_noise_mechanisms

[18] D. Dang, F. Di Troia, and M. Stamp, ''Malware classification using long short-term memory models,'' *arXiv preprint arXiv:2103.02746*, 2021.

[19] P. Prajapati and M. Stamp, ''An empirical analysis of image-based learning techniques for malware classification,'' in *Malware Analysis Using Artificial Intelligence and Deep Learning*. Springer, 2020, pp. 411--435.

[20] ''Trojandownloader:win32/adload.'' [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader%3AWin32%2FAdload

[21] ''Trojandownloader:win32/agent.'' [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Agent&ThreatID=14992

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

[23] ''Trojan:win32/bho.'' [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/BHO&threatId=-2147364778

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

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

[26] ‘‘Pws:win32/delfinject.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/DelfInject&threatId=-2147241365

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

[28] ‘‘Adware:win32/hotbar.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Hotbar&threatId=6204

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

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

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

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

[33] ‘‘Trojandownloader:win32/renos,.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Renos&threatId=16054

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

[35] ‘‘Win32/vobfus.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Vobfus&threatId=

[36] ‘‘Win32/vundo.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Vundo&threatId=

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

[38] ‘‘Pws:win32/zbot.’’ [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/Zbot&threatId=-2147368817

[39] ‘‘Trojan.zeroaccess.’’ [Online]. Available: https://www.symantec.com/security-center/writeup/2011-071314-0410-99

[40] ‘‘Lanczos.’’ [Online]. Available: https://en.wikipedia.org/wiki/Lanczos_resampling#:~:text=Lanczos%20resampling%20is%20typically%20used,or%20rotate%20a%20digital%20image.

APPENDIX

Appendix

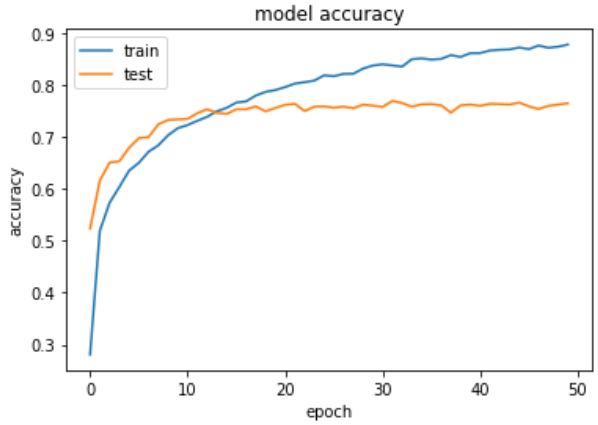

Figure A.12: CNN model accuracy 700 samples per family for training

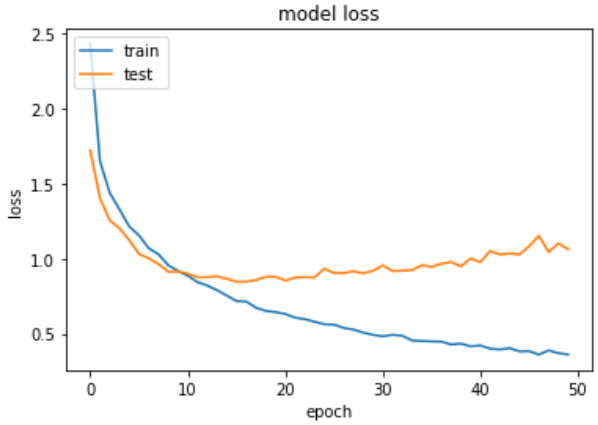

Figure A.13: CNN model loss 700 samples per family for training

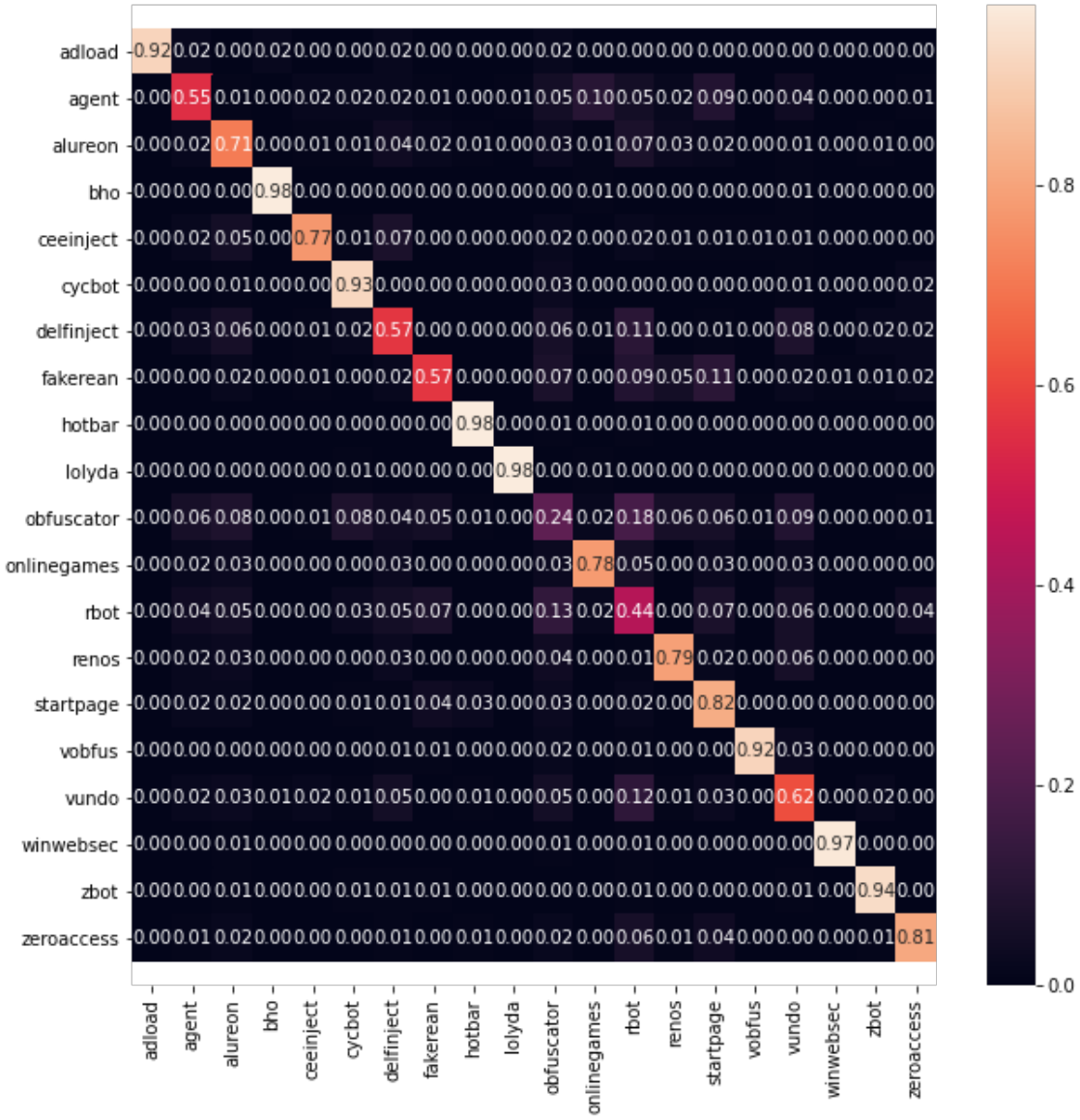

Figure A.14: CNN model confusion matrix 700 samples per family for training

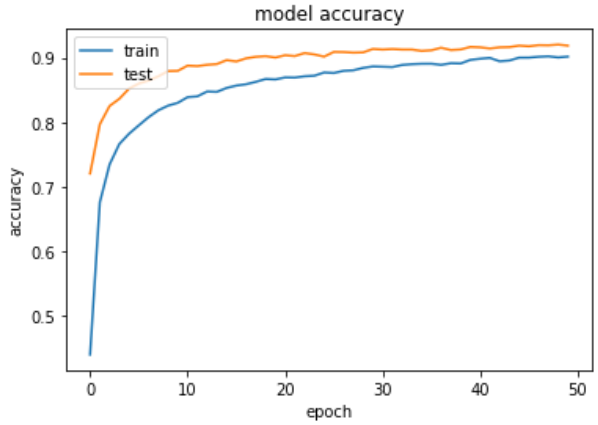

Figure A.15: CNN model accuracy dataset 32×32 created using 1024 bytes

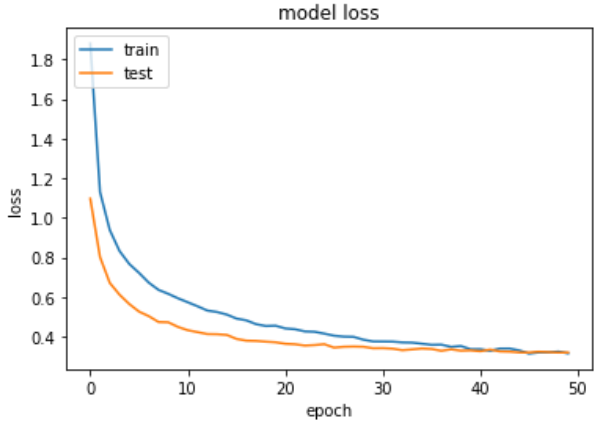

Figure A.16: CNN model loss Dataset 32×32 created using 1024 bytes

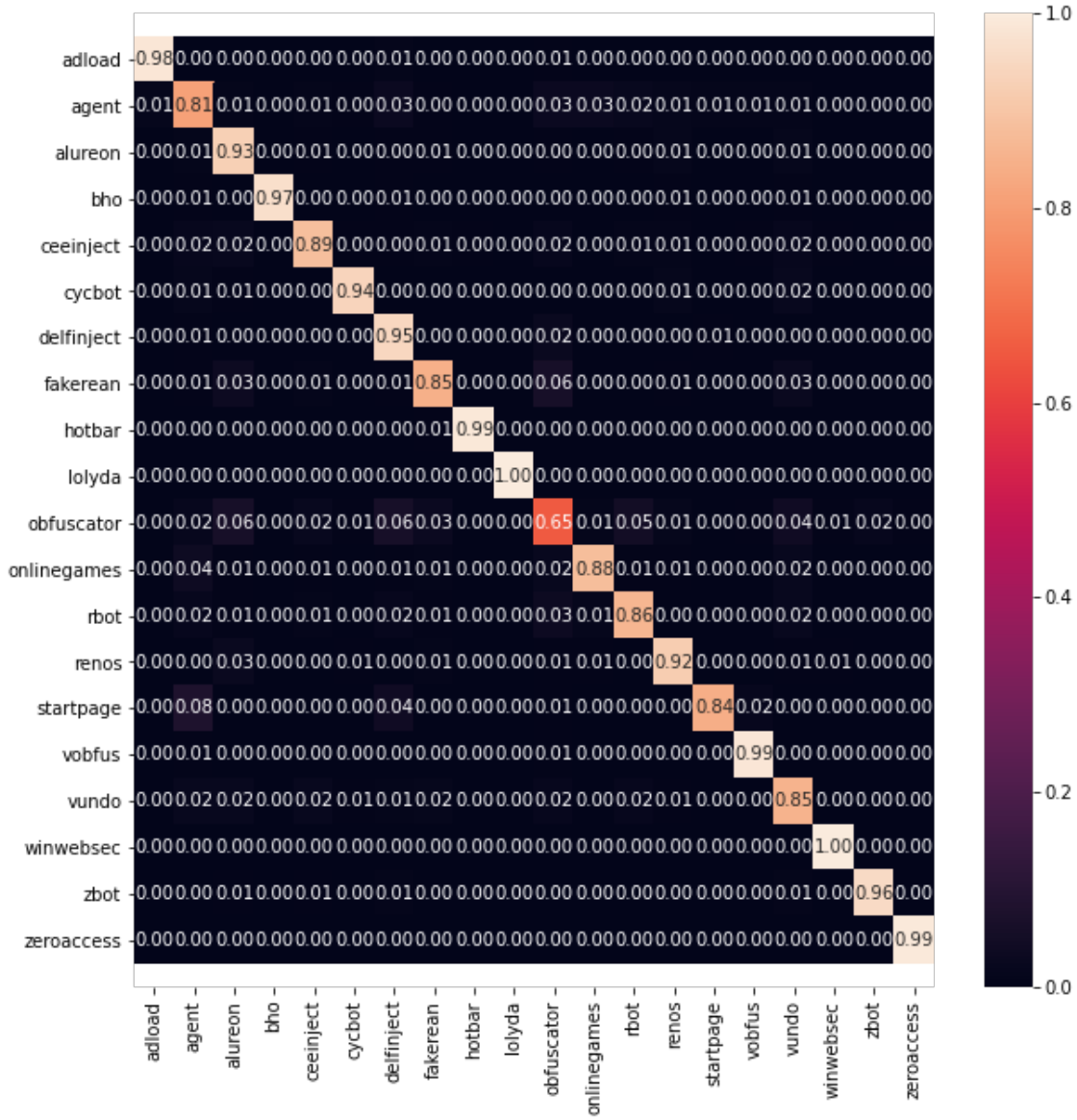

Figure A.17: CNN confusion matrix Dataset 32×32 created using 1024 bytes

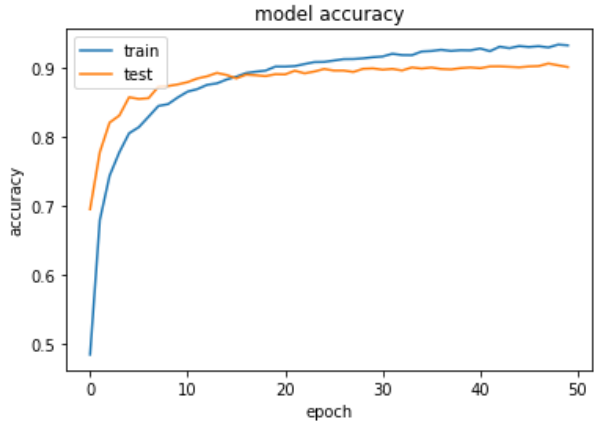

Figure A.18: CNN model accuracy dataset 64×64 created using 4096 bytes

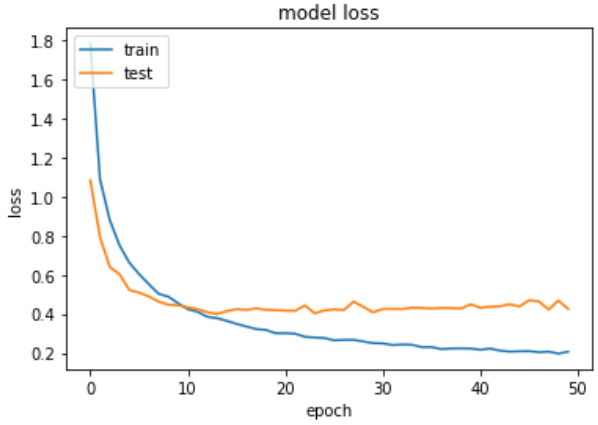

Figure A.19: CNN model loss dataset 64×64 created using 4096 bytes

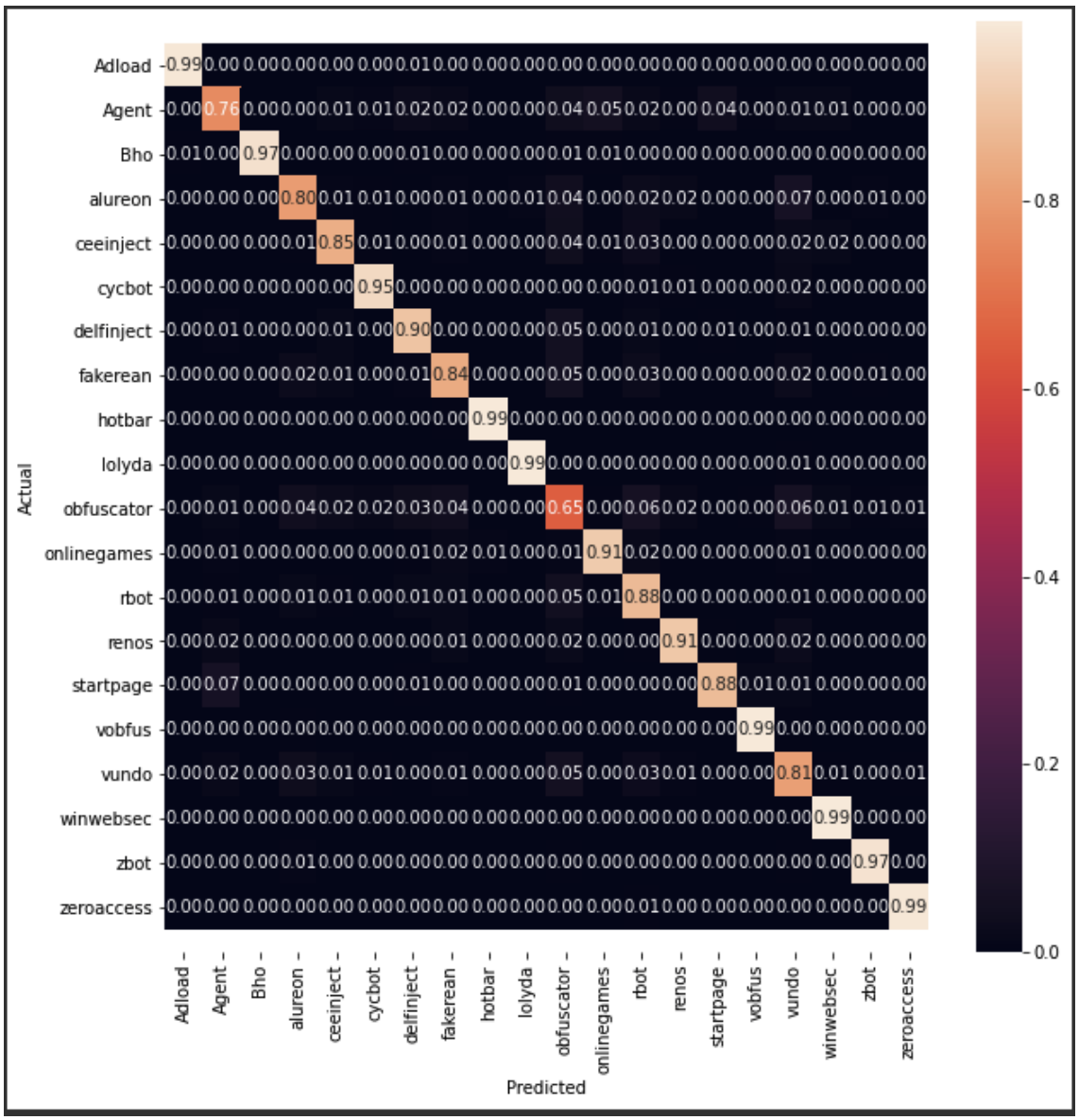

Figure A.20: CNN confusion matrix dataset 64×64 created using 4096 bytes

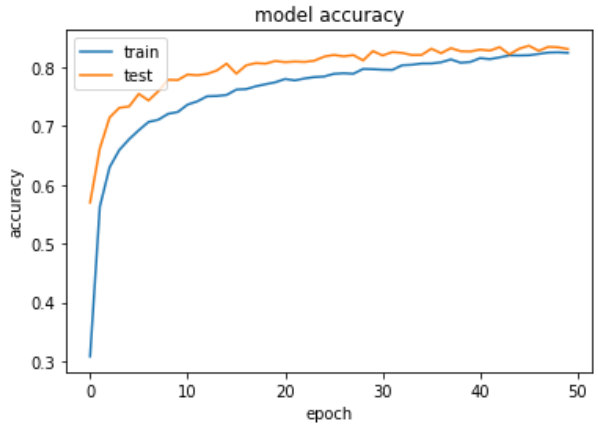

Figure A.21: CNN model accuracy dataset 64×64 created using python-resize-image

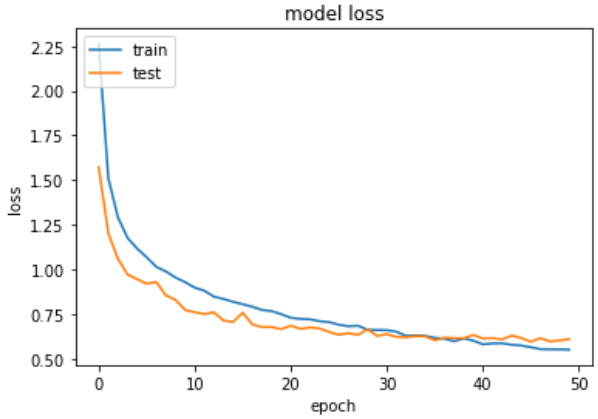

Figure A.22: CNN model loss dataset 64×64 created using python-resize-image

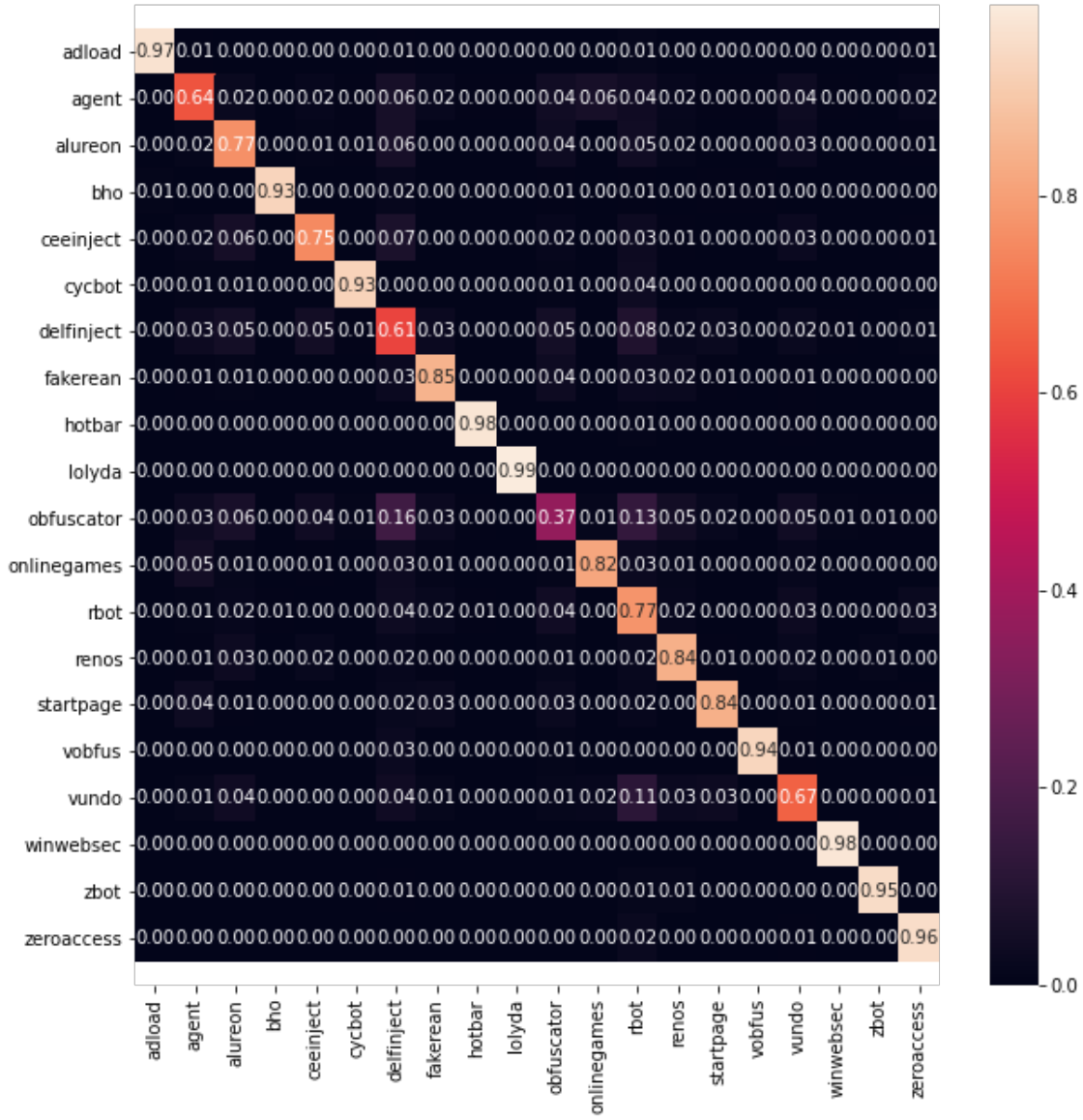

Figure A.23: CNN confusion matrix dataset 64×64 created using python-resize-image

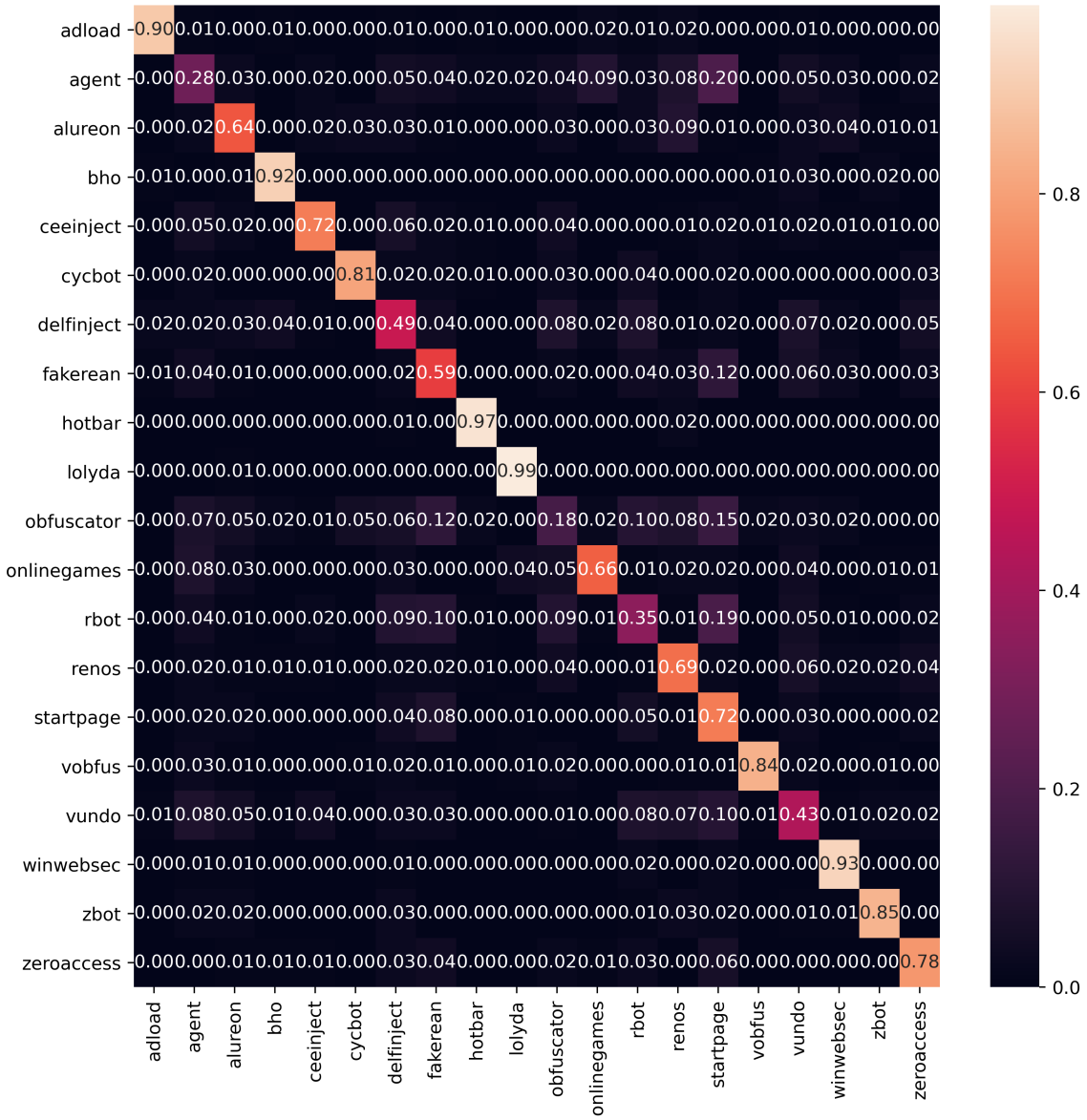

Figure A.24: CNN model confusion matrix Skimage augmented dataset with 0 real images and 14,000 fake images

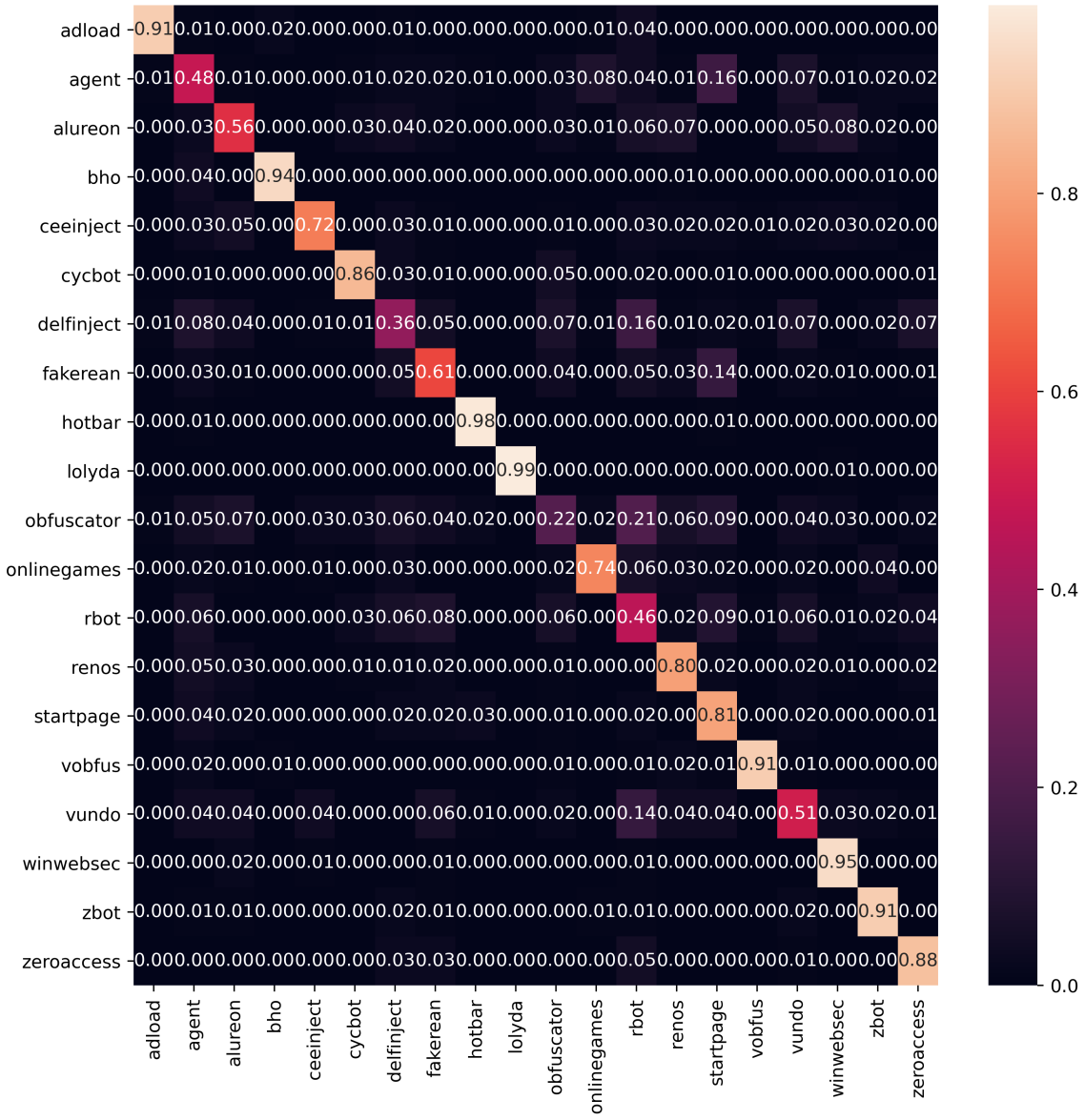

Figure A.25: CNN model confusion matrix Skimage augmented dataset with 7000 real images and 7000 fake images

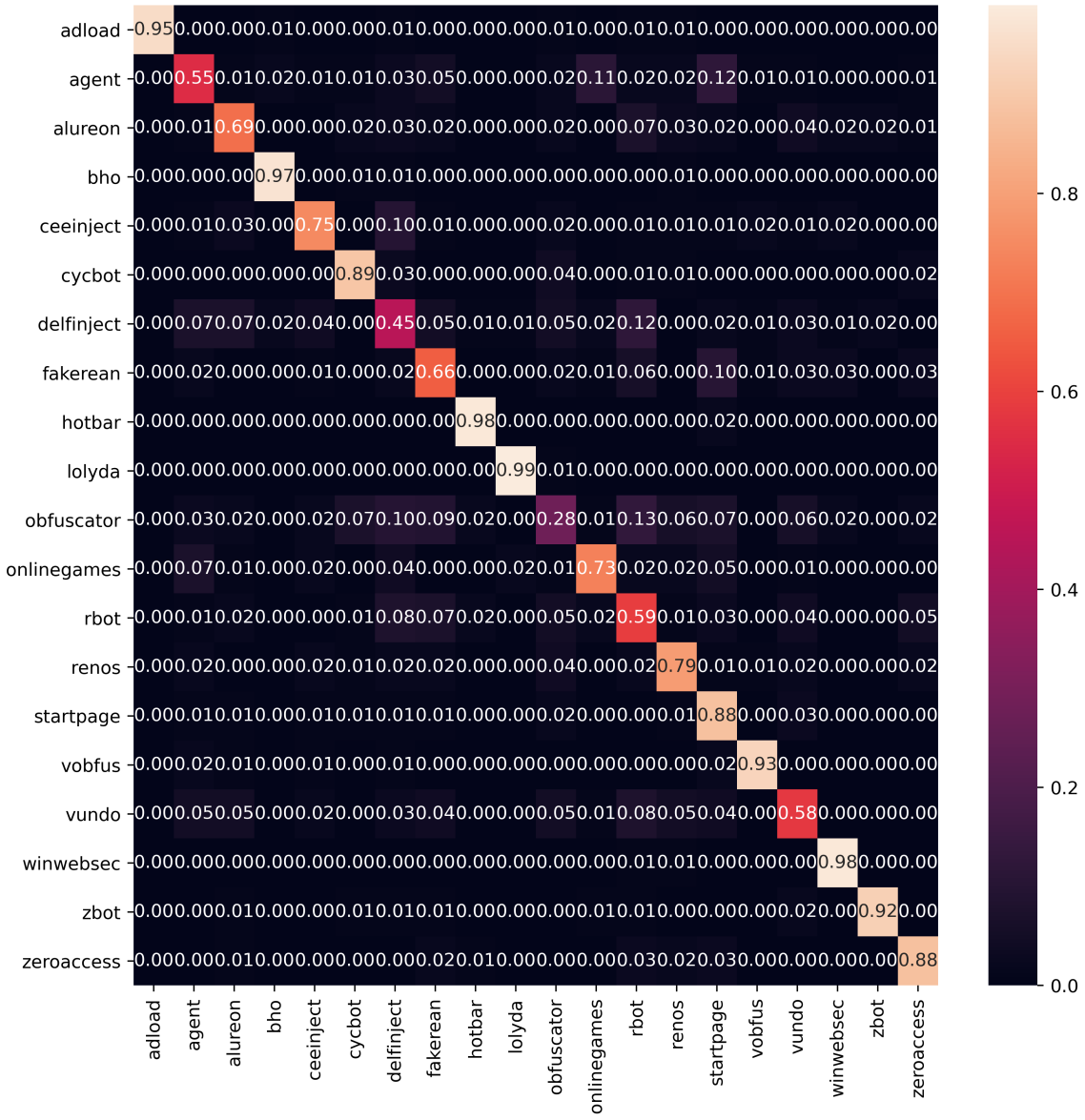

Figure A.26: CNN model confusion matrix Poisson augmented dataset with 0 real images and 14,000 fake images

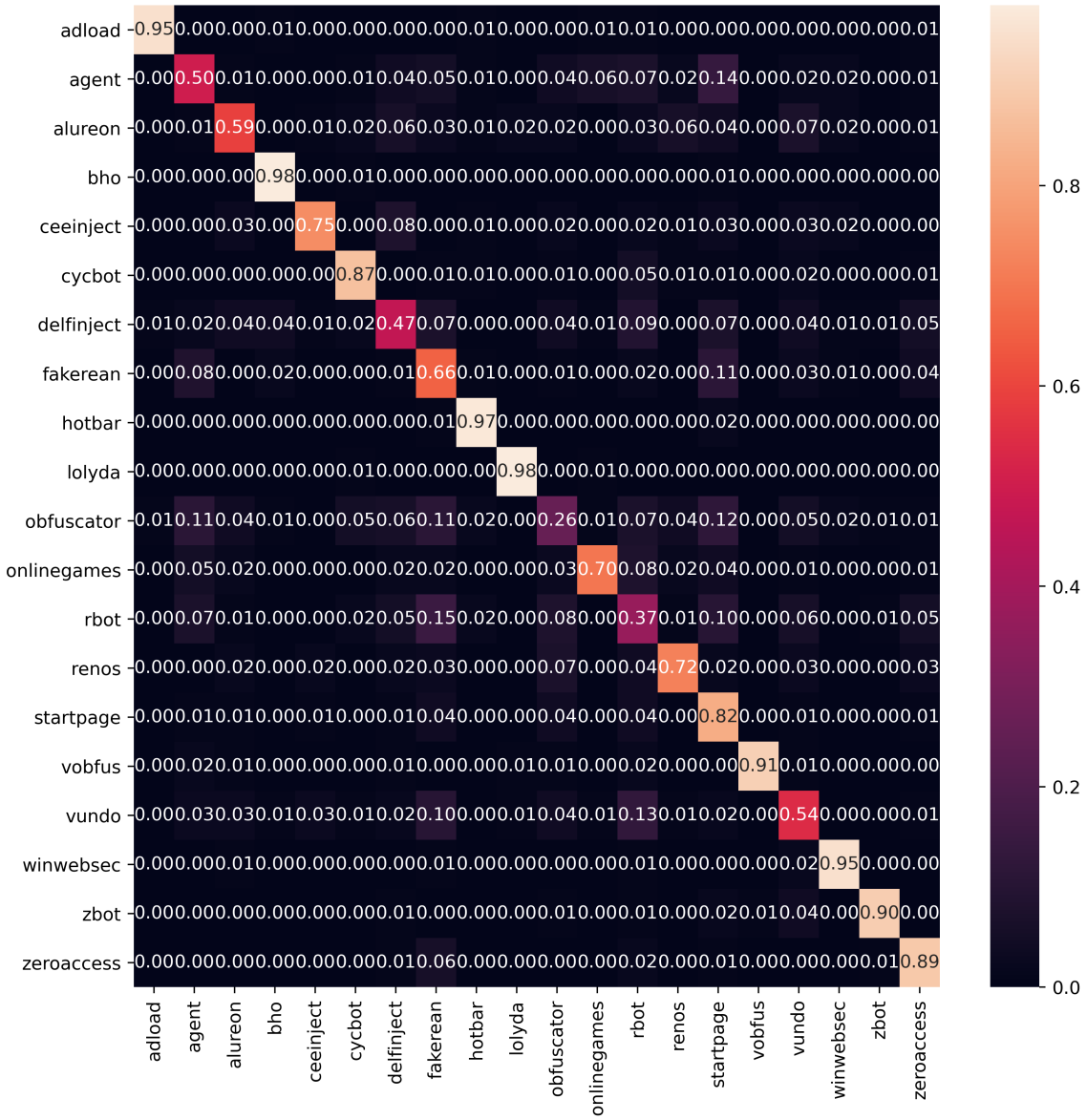

Figure A.27: CNN model confusion matrix Poisson augmented dataset with 7000 real images and 7000 fake images

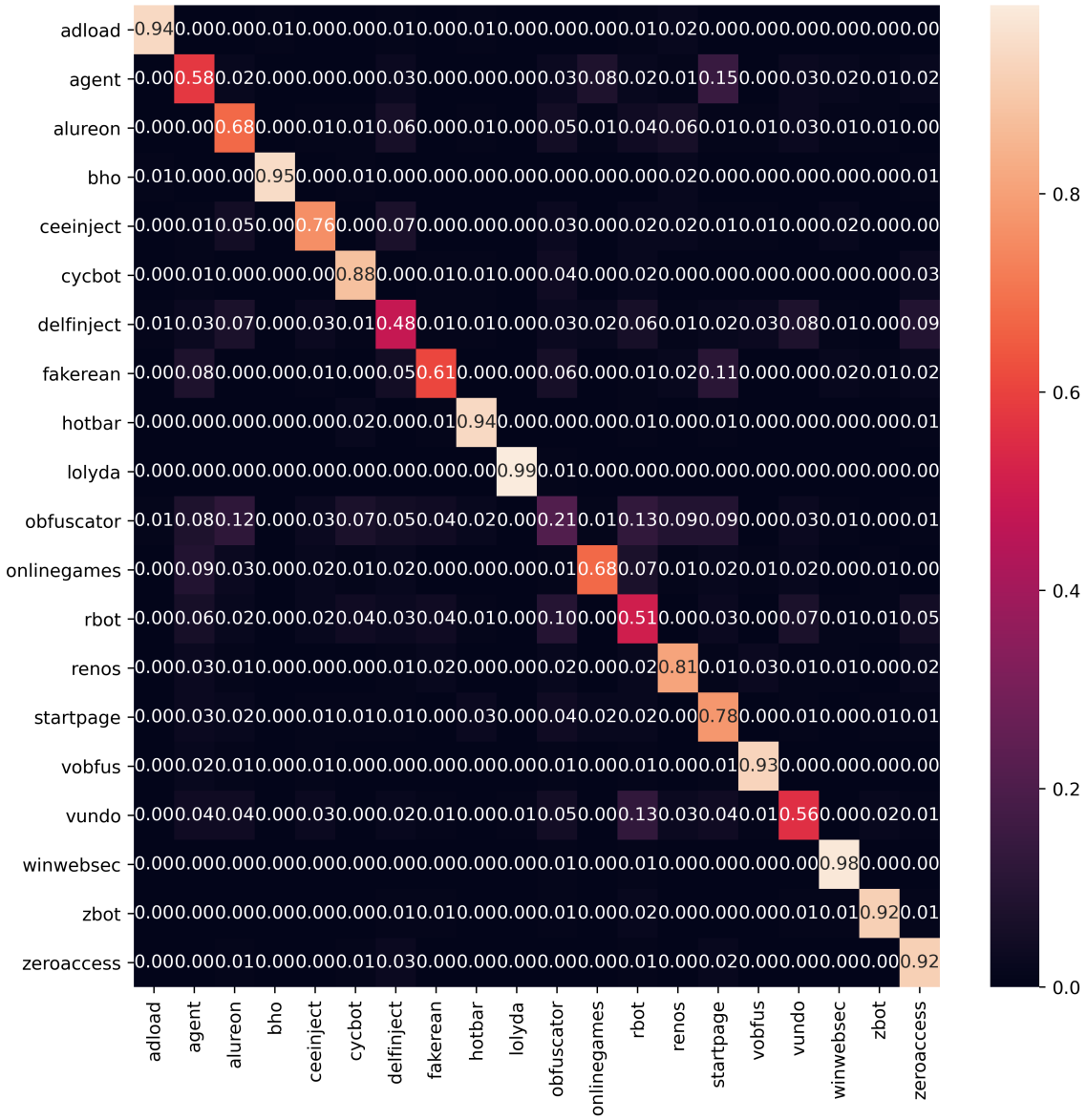

Figure A.28: CNN model confusion matrix Laplace augmented dataset with 0 real images and 14,000 fake images

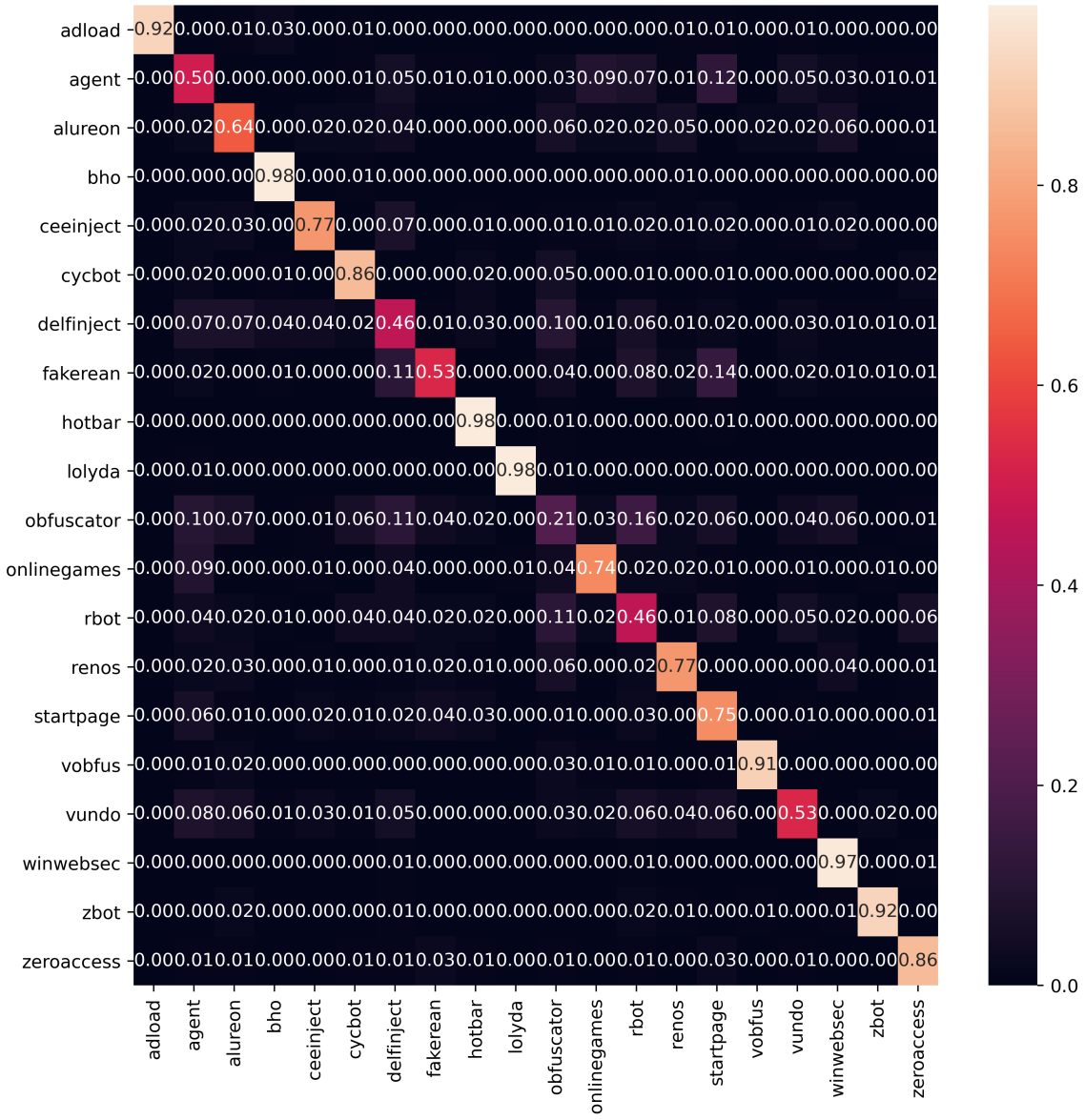

Figure A.29: CNN model confusion matrix Laplace augmented dataset with 7000 real images and 7000 fake images

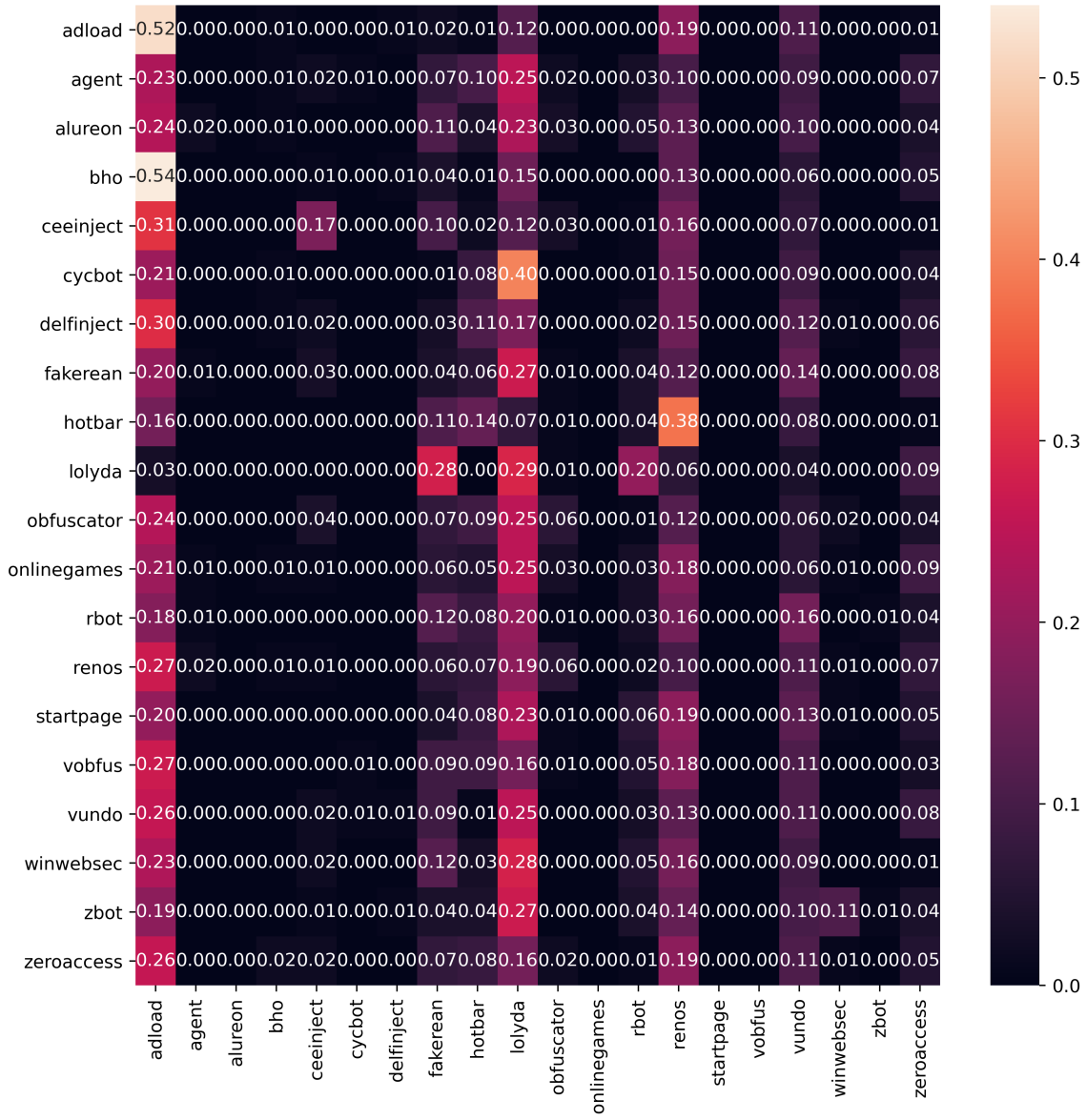

Figure A.30: CNN model confusion matrix AC-GAN augmented dataset with 0 real images and 14,000 fake images

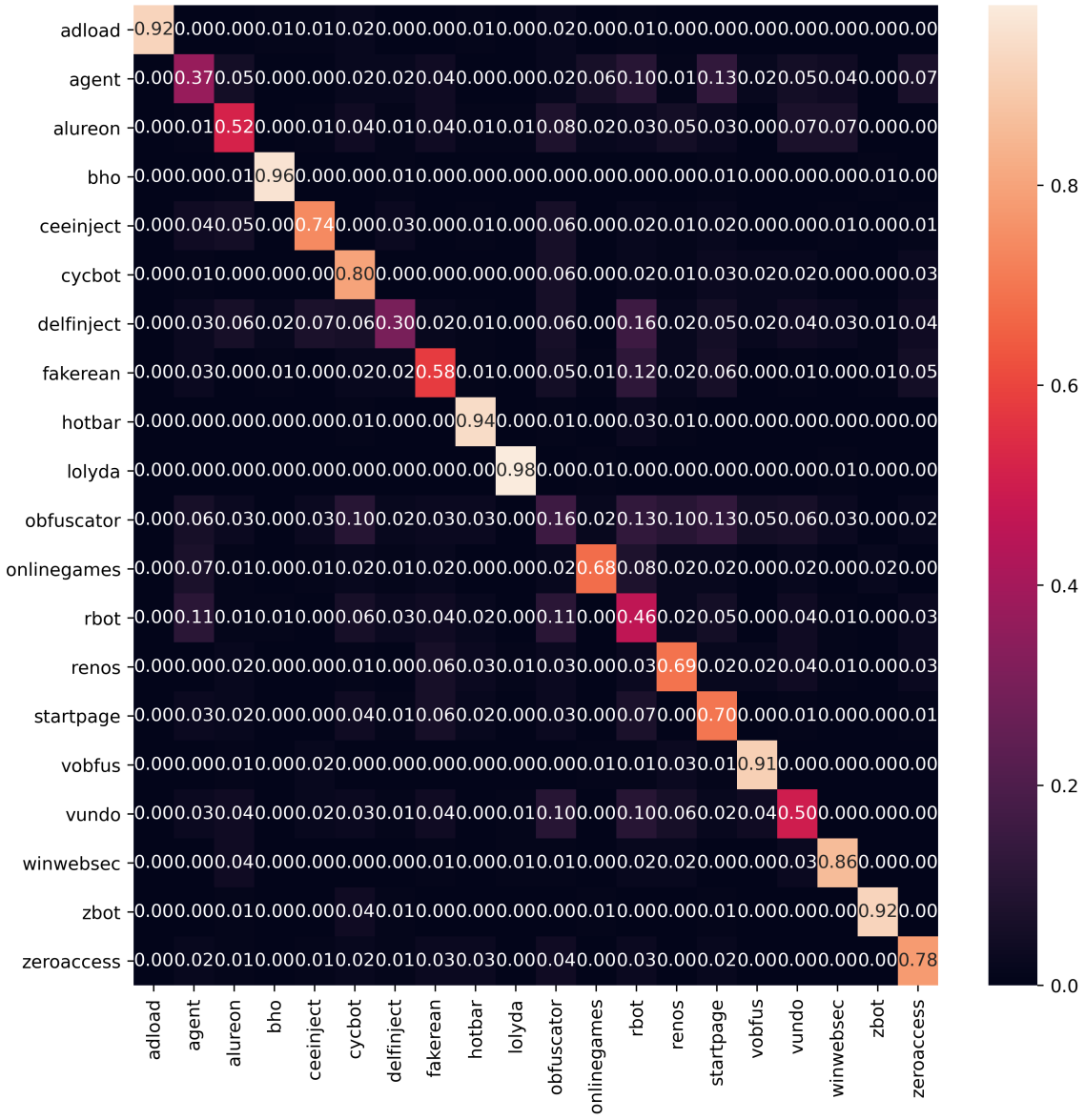

Figure A.31: CNN model confusion matrix AC-GAN augmented dataset with 7000 real images and 7000 fake images