University of Southampton

Faculty of Engineering and Physical Sciences

Electronics and Computer Science

Future Malware Prediction Through Generative Modelling

by

Alexander Newton

September 2020

Supervisor: Dr Leonardo Aniello
Second Examiner: Dr Enrico Gerding

A dissertation submitted in partial fulfilment of the degree
of MSc Artificial Intelligence

University of Southampton

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
ELECTRONICS AND COMPUTER SCIENCE

Master of Science

by Alexander Newton

Malware prediction and classification is a growing research field that aims to develop approaches to help identify malware running on computer systems. Current malware models are trained on previously identified malware samples. However, malware developers can take advantage of this knowledge by adapting malware to circumvent current malware identifiers. This has led to a new research field known as future malware prediction which aims to predict new malware variants and would enable malware analysts to pre-emptively identify future malware samples. Using general adversarial networks (GAN), it is possible to learn a distribution of malware samples that allow new pseudo malware to be generated which can be added to the original dataset. When malware samples are added, there is shown to be improvement in future malware prediction when a convolutional neural network is trained to identify whether an executable file is malware or benign. A GAN was also trained with a small amount of available malware using differentiable augmentation and the generated samples were added to the original dataset to fix class imbalances; this consequently increased the performance of the CNN classifier.

Statement of Originality

- I have read and understood the ECS Academic Integrityinformation and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead tothe imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption andcite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report,you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.
PyTorch, PyTorch-GAN, data-efficient-gans, data-efficient-gans diagram , GAN architecture diagram

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work(this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader),or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August2018, Alex Weddell aioffi-cer@ecs.soton.ac.uk

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Malware is a type of malicious software that is designed to cause damage to a computer or computer system. Some of these actions include causing disruption, stealing data, deleting user files or actively monitoring a user's computer usage. The term malware is used broadly to describe many different types of malicious software with a number of common variants including Ransomware, Adware, Viruses and Trojans. Malware has become an increasing problem in recent years as malicious actors use malware for potential monetary gain. CryptoLocker, for example, was a ransomware that raised 41,928 in bitcoins, the equivalent of $27 million dollars at the time when it was active in 2013 (Blue (2013)). Malware is also actively being used by state actors to cause political influence and often involves the use of zero-day exploits. These exploits are undisclosed and subsequently lead to malware that cannot be protected against as operating systems do not have the required patches for the vulnerabilities. A malware with zero-days continues to infect computers until the exploit is discovered and patched.

Traditional antivirus engines make use of signature detection in order to discover potential threats on a computer system. However, malware authors can easily circumvent signature detection through the use of code obfuscation and code polymorphism. Signature detection also suffers from scalability issues due to the vast amount of potential malware signatures that may need to be checked. Malware can even evade dynamic analysis by detecting when it is being executed for analysis.

Machine learning (ML) and deep learning (DL) have become a new focal point as a means of addressing the current issues of malware detection. By extracting common features of malware samples, machine learning models can be trained to predict new malware. Machine learning can give probabilities of its' belief that a sample is malware and can also perform multi-class classification to categorise malware. ML and DL models have the advantage of being able to use a combination of static and dynamic analysis features which allows models to be more robust in analysing malware which employs evasion tactics.

A new area of research focuses on the prediction of future malware. Malware authors have the upper hand when developing new malware; a new malware variant can be tested against commonly available anti-virus engines to check if they are detectable as discovered exploits are added to open databases such as the common vulnerabilities and exposures (CVE) dictionary. Therefore, malware analysts have to constantly play catchup by quickly discovering new techniques used by malware developers and add these to current malware detection systems. If malware can be predicted before it is used "in the wild", then anti-malware products have the ability to identify not previously seen malware. An example of this is the use of the zero-day EternalBlue exploit that was used in the WannaCry ransomware on the 12^{th} of May 2017 and then reused a month later by the NotPetya ransomware on the 27^{th} of June 2017. Both caused a large amount of disruption, however, if a ML model could have predicted a malware pattern using the EternalBlue exploit, then NotPetya could have been potentially identified earlier.

Generative modelling is a recent research area that aims to learn a distribution of data which can then be used to generate new samples from the learnt distribution. By training a generative model over a large set of malware samples, a distribution can be learnt which contains the salient features that contribute to malware. Pseudo malware are generated and added to the training set of malware classifiers which in turn could increase the likelihood that they will be able to detect new malware in the future.

To summarise the paper's contributions:

- A method for using generative models to learn a malware distribution and generate pseudo malware samples is outlined.
- The ability of the added samples is evaluated to see if they increase the likelihood of detecting future malware.
- An approach using differentiable augmentation using a small set of training data is demonstrated as well as showing the increase in classifier performance gained from this technique when training with a small number of malware samples.

The structure of this paper is as follows. Firstly, Chapter 2 will outline the background techniques used throughout the paper. Chapter 3 will then discuss the related work and how previous research has approached future malware prediction.

Chapter 4 outlines the methodology used to conduct the experiments and Chapter 5 analyses results obtained from these. Chapter 6 discusses the results from the experiments as a whole and Chapter 7 concludes the main findings from the paper. Finally, Chapter 8 lays out potential future ideas that could take the results and findings from this research further.

Chapter 2

Background

2.1 Malware Detection

This chapter explains the background knowledge for future malware prediction by outlining the general research area of malware detection. This is followed by an examination of the specialised generative models that will be used throughout the rest of the report.

Malware detection is a long standing research area which aims to detect malware running on a computer system. Malware detection systems use signature detection to identify known threats. Recent research has seen a shift away from rule based methods to machine learning due to a number of recent developments (Gibert et al. (2020)); namely, the increase in the availability of labelled malware data, the increase in the amount of computational power and rapid development of the machine learning research field.

There has been a wide range of research exploring many different types of classical ML models. Ucci et al. (2019) summarises the ML models and their corresponding features. The most popular models include support vector machines (SVM), decision trees, naïve Bayes, k-nearest neighbour (k-NN) and multilayer perceptrons. Classical ML models depend on the creation of input features based on domain expertise and require that feature engineering gives a valid representation of the data. Many anti-virus vendors choose their own combination of features which come from a range of static and dynamic analysis techniques. Malware analysis could involve decompiling the binary to its assembly code or running the malware in a sandboxed environment. However, the process of extracting these features is time and computationally consuming as malware binaries need to be analysed.

State of the art malware detection makes use of deep learning models to overcome these problems with its ability to learn features needed for classification without manual feature engineering. A wide variety of deep learning architectures have been explored including Convolutional Neural Networks (CNNs), Long Short Term Memory networks (LSTM) and Autoencoders (AEs) as summarised by Gibert et al. (2020).

2.2 Generative Models

Generative models are a class of models that take a training set with samples drawn from the distribution p_{data} and then learns an estimate of that distribution. Generative models aim to find a distribution p_{model} by finding the parameters θ that give the maximum likelihood of the training data although not every generative model makes use of maximum likelihood. Finding the maximum likelihood estimation is analogous to minimising the Kullback–Leibler divergence between the data generating distribution p_{data} and the model p_{model}. A generative model is successful if the distribution p_{data} lies within the family of learnt distributions $p_{model}(x; \theta)$ (Openai).

2.3 General Adversarial Networks

General adversarial networks (GAN) are an example of models where $p_{model}(x)$ is not explicitly defined and we only learn to sample from the distribution. This is achieved by sampling from a simple distribution (usually Gaussian) and using a neural network to learn a transformation to the training distribution. As shown in Figure 2.1, a GAN comprises of two key elements; a generative network G that generates datapoints and a discriminative network D that aims to distinguish between real datapoints and fake datapoints from the generator. At each iteration of the training, the generative network updates its weights in order to produce new samples that increase the classification error of the discriminator whereas the discriminative network updates its weights in order to decrease this error. The overall effect is that the generator is able to produce new datapoints which are indistinguishable from real data. A GAN converges when the distribution of the data generated by the generator is the same as the distribution of the real data and this occurs when the generator and discriminator have reached a Nash Equilibrium. The generator has then learnt the appropriate mapping to the training distribution and hence samples can be generated. The generator outputs pseudo samples by passing a noise variable z which transforms the vector to an image in the learnt training distribution.

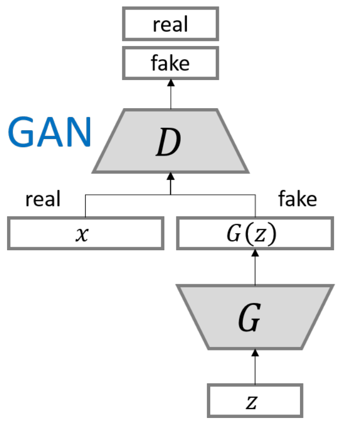


Figure 2.1: A diagram outlining the architecture of a GAN model. During training, random latent vectors z are passed to the generator G to produce fake images. The real and fake images are passed to the discriminator D a batch at a time where D has to predict if an image is real or fake. The cross entropy error is backpropagated through the D and G in order to make D better at discerning real and fakes and for G to improve its distribution of images to generate more realistic images. Source: `https://github.com/hwalsuklee/tensorflow-generative-model-collections`

The formulation for the GAN loss function starts with the cross entropy loss shown in equation 2.1 which quantifies the difference between two probabilities distributions. This measures how far away the predictions are from the true model. In the GAN model the discriminator outputs a probability of whether an image is real or fake hence $D(x) = \hat{y}$. This equation is the same as the expectation (weighted average) over the distribution of real labels y. When we train a GAN we pass half real images and half fake images to the discriminator. Therefore we know we have $\frac{N}{2}$ real images (with a label of 1) which is the first summation in equation 2.2 and another $\frac{N}{2}$ for fake images (with a label of 0) as the second summation. We can then convert the sums into expectations of data drawn from the real images p_{data} and fake images drawn from the generator p_g to form the final discriminator loss.

$$L(\hat{y}, y) =[y.log\hat{y} + (1 - y.log(1 - log\hat{y}))] \tag{2.1}$$

$$= - \left[\frac{1}{N} \sum_{i=1}^{N/2} y_i \; log(D(x_i)) + \frac{1}{N} \sum_{i=N/2}^{N} (1 - y_i) \; log((1 - D(x_i))) \right] \tag{2.2}$$

$$L_D = - \left[\; \mathbb{E}_{x \sim p_{data}}[log \; D(x)] + \; \mathbb{E}_{z \sim p_z}[log \; (1 - D(G(z)))] \right] \tag{2.3}$$

The discriminator loss function can be thought of as a zero-sum game. The discriminator's goal is to minimise it's loss or, as shown in equation 2.4, to maximise it's negative loss. The generator on the other hand wants to minimise $-J^{(D)}$ and therefore $J^{(G)} = -J^{(D)}$. Hence, we can set $V(D, G) = -J^{(D)}$ with G aiming to minimise and D aiming to maximise. This forms the final value function shown in equation 2.5.

$$\max_D [-J^{(D)}] = \max_D \left[\; \mathbb{E}_{x \sim p_{data}}[log \; D(x)] + \; \mathbb{E}_{z \sim p_z}[log \; (1 - D(G(z)))] \right] \tag{2.4}$$

$$\min_G \max_D V(D, G) = \min_G \max_D \left[\; \mathbb{E}_{x \sim p_{data}}[log \; D(x)] + \; \mathbb{E}_{z \sim p_z}[log \; (1 - D(G(z)))] \right] \tag{2.5}$$

2.4 Variational Autoencoders

Variational autoencoders (VAE) are another type of generative model that are able to learn a data distribution and generate samples. VAEs are a probabilistic version of a normal autoencoder network that works by defining an intractable density distribution of the data and using variational inference to optimise the variational lower bound. Generally VAE are useful when representation learning is a priority, however, the generated images are noisier than images generated by a GAN. A GAN is preferred when higher quality images are needed (Rosca (2018)). As this work aims to generate fake malware samples, only GANs will be considered for image generation.

Chapter 3

Related Work

This chapter investigates previous and similar research in the area of future malware prediction. The results from each paper are explained with critical analysis.

Research for future malware prediction is relatively new with a small number of recent papers. It is also somewhat fragmented; several papers have been published with similar ideas which are categorised with different nomenclature. For example, some papers view the idea of malware prediction as data augmentation (Burks et al. (2019)) whilst other papers see it as the application of training malware samples on generative models (Moti et al. (2019)). Malware prediction is also very closely related to the research area of Adversarial Example generation which focuses on generating malware samples that bypass malware detectors (Hu & Tan (2017),Kawai et al. (2019), Castro et al. (2019)).

Evasion attacks aim to create malware samples that can bypass malware detector systems (MDS) by adding slight modifications that cause a detector to incorrectly classify malware samples. Anderson et al. (2018) used a reinforcement learning agent that learns to modify the Portable Executable (PE) section of a windows executable to bypass PE malware detection. Hu & Tan (2017) and Kawai et al. (2019) use a generative adversarial network (GAN) to learn which API calls to modify a sample to bypass a MDS. Castro et al. (2019) explored the use of Genetic Programming to evolve malware. Perturbations were injected into the malware PE header and checked if they were still functional. The resulting samples are checked to see if they evade the MDS. They were then assigned a fitness depending on whether the sample is corrupted, evaded or got detected and the fittest samples are crossed over for the next generation. Creating adversarial examples does generate new malware, however, the new samples only have small changes to the existing malware to bypass MDS and consequently the adversarial malware will be very similar to the original malware. In contrast, future malware prediction aims to create completely new malware samples without the explicit aim of bypassing an MDS.

One of the initial papers on future malware prediction by Juzonis et al. (2013) explored the use of genetic algorithms to evolve new malware. An initial population was created with each individual having a set of parameters that is found in common Android malware. Over many evolution stages the chromosomes were randomly crossed over and mutated and only the fittest individuals were kept. The final chromosome with the highest fitness represented features that a future malware could have. The paper manually defined a fitness function with set features and their belief of their fitness contribution. This constrains the prediction of new malware only to have the manually defined features which may fall out of usage over time. There is also a potential bias within the fitness function which will cause malware only to evolve if it contains features determined to be dangerous by the author of the fitness function.

Howard et al. (2017) made use of a ML approach to predicting future malware variants by producing a predictive malware defense system (PMD) that is split into five stages. The first stage uses dimensionality reduction of static and dynamic analysis features using singular value decomposition (SVD) to select the most important features of the dataset. These chosen features are then used to train their Malware Analysis and Attribution using Genetic Information (MAAGI) system which uses a clustering algorithm to group similar malware into families. A neural network is then trained to make predictions of future malware by taking the features of the most recent family members in a cluster and then using the signature of the next family member as the output label. Once trained, the neural network will predict new malware signatures which are then added to the overall malware dataset. The new dataset with the generated malware signatures is then clustered with a k-nearest neighbour algorithm using the labels of the known malware to classify a cluster as malware or benign. Any new malware is added to the nearest cluster and given the corresponding label. The neural network was trained with 923 families and thus added 923 predicted signatures to the dataset. After training with the additional samples, the K-NN algorithm detected 11 more malware samples compared to using the dataset without the additional generated samples. The approach to generating the new malware samples is interesting, however, it is unknown how much contribution the newly generated malware samples add to the final classification model. Furthermore, the MAAGI system is closed source and developed by a private company which makes reproducibility very difficult.

The concept of GANs has been explored in zero day malware detection where a generator network could be trained to produce new realistic malware samples. Moti et al. (2019) trained a GAN to generate new malware samples that were then added to the original dataset. This method is similar to Howard's method with the main difference being that the GAN generates the data instead of using clustering and neural networks to predict new samples. The results found that adding the new samples increased the accuracy of ML classification models by a few percent. Lu & Li (2019) also made use of a GAN to learn the distribution of data over the MalImg dataset. They similarly showed that training a Resnet18 image classifier with added samples generated by a GAN also increased the classification performance. Work by Burks et al. (2019) explored the idea of data augmentation for malware through the use of conditional VAE. The VAE was trained with the MalImg dataset (Nataraj et al. (2011)) and then 2,000 samples were generated and added to the original dataset. The dataset with the additional samples was trained on a CNN classifier and found to increase classification performance by 2%. They concluded that VAE images do not have as great an effect on increasing accuracy as images generated from GANs. However, the key problems with these works are that they do not evaluate whether the GAN or VAE, if trained on old malware samples, could help classify future malware samples. Therefore the trained models real life performance is not truly represented.

Kim et al. (2018) took an alternative approach in the use of the discriminator as a malware detector. Once the discriminator is trained against the generator, the convolution layers of the discriminator have sufficiently learnt the characteristic of real malware samples and this ability is transferred to a malware detector. The paper assumes that zero-day attacks can be modelled by adding noise to existing malware samples. Noise was added to each sample and the similarity to the original samples was measured using structural similarity of images (SSIM). They found that as similarity to original images decreased (as a result of adding more noise) conventional ML models accuracy significantly decreased while the GAN discriminator accuracy only saw a small drop. This shows that the GAN's discriminator model is more robust to new samples and hence could detect malware further into the future.

3.1 MALWARE IMAGE CLASSIFICATION WITH TEMPORAL AND SPATIAL BIAS

Many previous works have demonstrated that a CNN can achieve very high classification scores on malware images. Nataraj et al. (2011) and Gibert et al. (2019) both used CNNs to achieve accuracy scores of 0.948 and 0.9178 on the MalImg dataset respectively. Kawai et al. (2019) obtained accuracy scores of of 0.745 and 0.8836 on ResNet (He et al. (2016)) and GoogleNet (Szegedy et al. (2015)) architectures when trained on the Microsoft Malware Classification dataset.

While at first these scores seem impressive, they fail to take into account the key issue of concept drift. Drift is a common problem where we assume that data has been sampled from a stationary distribution and thus our model cannot adapt to new data over time. This assumption may be too crude as the characteristics of data evolves, especially in malware development where malware authors are constantly evolving malware to avoid new forms of detection. A trained model may start incorrectly classifying malware after a certain period of time as the features it uses for detection may have also changed. Concept drift as defined by Pendlebury et al. (2019) "is a problem that occurs in machine learning when a model becomes obsolete as the distribution of incoming data at test-time differs from that of training data". Pendlebury et al. (2019) showed that the high accuracy of most ML and DL malware classifiers are not representative of real life performance. They introduce the idea that previous malware classification research has temporal bias where the splits of training and testing sets do not take into account the dates that the malware appeared. For example, if a malware detector is trained on old malware and then tested on future malware, the classifiers perform significantly worse than if they are trained with malware covering the full time frame.

They also introduce spatial experiment bias which is concerned with the percentage of malware in the testing and training. If a testing set has more malware than goodware, then the F_1-Score is artificially increased.

With both temporal and spatial bias taken into account, classifiers which were reporting accuracy scores of 0.97 had a new accuracy of 0.58. When training a classifier, these biases need to be taken into account to accurately show if the added samples have an effect on improving classifier performance. To measure the ability of added samples to predict future malware, the datasets used to train and test the classifiers must be temporally consistent. The spatial consistency also needs to be extended to training on generative models to ensure that the pseudo samples produced are only based on the distribution of previously seen malware.

Chapter 4

Method

This section covers the methodology used to train GAN models and explains how new samples can then be generated and added to a dataset. It also covers how the datasets will incorporate temporal and spatial consistency as well as describing the chosen GAN models used for the experiments.

4.1 Malware Binary Image Conversion

Generative models are traditionally trained on images with the primary research focus of improving image quality and stability of the training process. Training generative models on non-image data can be difficult as it requires modifying traditional generative models loss functions and training loops while ensuring that common problems such as mode collapse, non-convergence and diminishing gradients are avoided. Therefore, malware binaries are converted in a 2D grey scale image. Visualising a malware binary as an image involves converting each byte into an integer value between [0, 255] and arranging these values into a 2D array. These new integer values represent the greyscale colour value of each pixel where 0 and 255 represent black and white respectively. The width and height of the greyscale image is determined by the size of the malware binary according to the equations shown in equations 4.1 and 4.2 (Kim et al. (2018)) where k is the malware's file size. The appropriate image widths for an malware's size are shown in Table 4.1.

File Size	Image Width
<10 kB	32
10 kB – 30 kB	64
30 kB – 60 kB	128
60 kB – 100 kB	256
100 kB – 200 kB	384
200 kB – 500 kB	512
500 kB – 1000 kB	768
>1000 kB	1024

Figure 4.1: A table which gives the appropriate image size depending on image width (Nataraj et al. (2011))

$$W = 2^{\frac{\log \sqrt{16k}}{\log 2}+1} \tag{4.1}$$

$$H = \frac{16k}{C} \tag{4.2}$$

As well as being able to use the images in generative models, the conversion of malware to images does not require computationally expensive static or dynamic analysis that is traditionally used in malware detection systems. A visual representation of the image also shows different sections of a binary executable. If old malware is slightly modified the overall visual structure will remain similar and hence easily detectable while retaining the global structure of the malware binary. Malware authors also make use of zero-padding for block alignment and to reduce entropy of a malware binary. This can be easily detected in images by the large areas of black pixels Gibert et al. (2020).

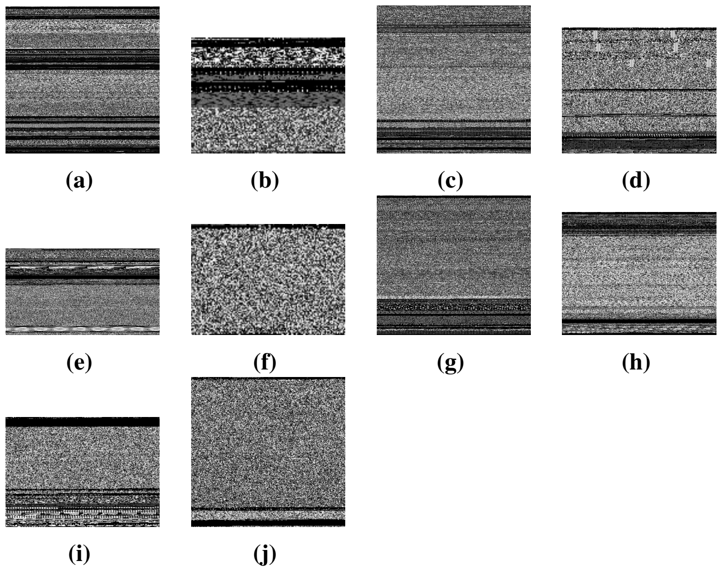


Figure 4.2: a-b: Backdoor, c-d: Rootkit, e-f: Trojan, g-h: Virus, i-j: Worm

Figure 4.2 shows an example grid of malware images from five types of malware families: backdoor, rootkit, trojan, virus and worm. In the images, large sections of black pixels indicate zero padding and in image e, icons can be seen in the bottom of the image which are as part of .rsrc section which contains the resources for the executable program.

4.2 Dataset

The most common datasets for image malware classification are the MalImg (Nataraj et al. (2011)) and the Microsoft Challenge dataset (Ronen et al. (2018)) and both are used as a benchmark for classification performance. However, both datasets do not have labels referring to the date that the malware was first discovered and do not have the portable executable (PE) header which allow a specific malware to be identifiable. Therefore, the date the malware was first seen in the wild can not be obtained and thus the datasets cannot be made temporally consistent. Therefore, the less commonly known MC-dataset-multiclass dataset (de O. Andrade (2018)) was chosen which consists of 16,450 malware samples taken from the VirusShare (`www.virusshare.com`) repository. Unlike the previous datasets, the PE header has not been removed from the malware samples which makes it possible to extract time and date information.

Initially the compilation time was parsed from the PE header from each malware sample to identify the date of the malware. However, a very high majority had their dates set to the start of unix time (1970-01-01) or set to "None" in order to avoid analysis. To overcome this, the MD5 hash of each malware sample was analysed with the virus total (`www.virustotal.com`) API in order to retrieve the date that the sample was first submitted to the virus total website.

Benign samples were taken from the MC-dataset-multiclass dataset and from the Yong et al. (2020) and Tuan et al. (2018) datasets. The first submitted to virus total date was also retrieved for the benign samples and was removed from the dataset if it was not on the virus total database. Overall there are 7,842 benign samples in the dataset.

4.3 GAN Models

Three types of GAN variations were used to compare the effectiveness of their generated samples. These include the Deep Convolutional GAN (DCGAN) (Radford et al. (2015)), Boundary Equilibrium GAN (BEGAN) (Berthelot et al. (2017)) and the Wasserstein Divergence for GAN (WGAN-DIV) (Wu et al. (2018)). The DCGAN is the baseline GAN for the experiments. It was the first network that applied a deconvolutional neural network to upsample a latent vector to an image sample. It is considered the vanilla architecture that many GAN architectures are based on. BEGAN was chosen as it adapts the network architecture of a traditional GAN by using an autoencoder for the discriminator. This changes the loss function to being a function of reconstruction quality achieved by the discriminator on real and generated images. BEGAN was shown to produce high quality samples with an improved Fréchet Inception Distance (FID) score compared to DCGAN. WGAN-DIV is a variation of a Wasserstein GAN and was chosen as an example of a GAN which uses an alternate loss function. Wasserstein GANs use a Wasserstein distance as a loss compared to traditional GANs which use Jensen–Shannon Divergence between the data and real sample distributions. WGAN-DIV uses a relaxed version of Wasserstein distance which has shown to produce higher quality samples over a WGAN.

All the GAN models were taken from the PyTorch GAN library (`https://github.com/eriklindernoren/PyTorch-GAN`) and were modified to load the malware images for training. Some notable GANs that were not used inlcude BigGAN Brock et al. (2018) and StyleGan2 Karras et al. (2020b) as these models architectures are too complex for conventional training on a single GPU. Using the continuation of training from pretrained models was also not possible as the malware image dataset greatly differed from the original datasets used to the train them.

4.4 Experiment methodology

To evaluate whether generated pseudo images have an effect on future malware prediction, firstly a convolutional neural network (CNN) is trained on the original dataset with no added pseudo malware images and these results act as the baseline score used for comparison. The GAN models are then trained with the dataset to learn the distribution of images. When a random latent vector z is sampled from some prior distribution (normally Gaussian) and is passed to the generator G, $G(z)$ returns an image from the learnt distribution p_{model} (Openai). We can then proceed to generated pseudo images which are added to the original dataset to increase the amount of images that a classifier can be trained on. Subsequently, the new datasets with the added GAN samples are trained and the results can be compared with the baseline score to see if the added samples improve the classification metrics.

Chapter 5

Experiments

This chapter analysis the results from adding samples to a dataset from GANs trained with and without temporal consistency. It then goes on to demonstrate how GAN training can be used in a realistic scenario where the number malware samples are limited. These experiments will be run in order to test the following hypothesis: training GAN models with malware image samples and then adding generated pseudo samples to the original dataset will increase a classifiers ability to predict future malware samples.

5.1 Experimental Details

The malware samples in the dataset were converted into their greyscale representation and then used to train the GAN models. The models were trained on a system with 16GB of RAM and an RTX 2070 graphics card. Due to the 8GB memory limit of the graphics card, the images were all scaled to a size of 64x64 pixels. GAN models expect images to be in the RGB format and therefore the single channel of the greyscale images were duplicated to 3 channels. Apart from resizing the image, no other preprocesing was applied to the greyscale images. In most cases, normalising the images in the range between $[0 - 1]$ caused problems with GAN training and led to mode collapse and therefore was omitted in the training of the final models. Each of the GAN models were trained for a maximum of 10,000 epochs or until the GAN training had reached a stable convergence. The parameters used for each GAN model are shown in Table A.1 in the appendix.

A pretrained Resnet18 (He et al. (2016)) CNN was trained on the binary class dataset to predict if a sample was malware or benign. The Resnet classifier was initially trained without any added samples to give a baseline score. Once the GAN model has been trained, pseudo malware images are generated by passing random latent vectors to the GAN's generator. These samples are subsequently added to the original dataset and the Resnet classifier is retrained. A pretrained Resnet18 model was taken from the Pytorch Hub (`https://pytorch.org/hub/`) and trained using Pytorch Lightning (Falcon (2019)) with early stopping enabled after three epochs of no improvement in loss.

For comparison, the GANs were trained on datasets with and without temporal constraints. The dataset without temporal constraint is randomly split into training and testing sets. When pseudo malware images are added for training, pseudo images were not used in the testing set. This is to ensure that only real malware images are used for testing the accuracy of the classifier. The dataset with temporal constraint was ordered by date that the sample was first seen. To enforce temporal constraints on a dataset, Pendlebury et al state that three constraints need to be enforced.

- C1) Temporal training consistency:

$$\operatorname{time}(s_i) < \text{ time } (s_j), \forall s_i \in \mathrm{Tr}, \forall s_j \in Ts \tag{5.1}$$

  Here s_i is a malware sample in the training set T_r or testing set T_s. Equation 5.1 enforces the constraint that no future malware sample is used to train the classifier.

- C2) Temporal malware/benign windows consistency:

$$t_i^{\min} \leq \text{time}\left(s_k\right) \leq t_i^{\max}, \quad \forall s_k \text{ in time slot } \left[t_i, t_i + \Delta\right) \tag{5.2}$$

  Equation 5.2 states that all malware samples used for training or testing must be in a defined time slot Δ where t_i^{min} and t_i^{max} are the dates of the earliest and latest malware samples in that slot.

- C3) Realistic malware-to-benign ratio in testing ensures that the average percentage of malware in the testing data is a close as possible to the percentage of malware in the wild $\delta \simeq \hat{\sigma}$. Here δ is the average percentage of malware in testing and $\hat{\sigma}$ is the estimated percentage of malware in the wild.

For the dataset used in these experiments, t_i^{min} and t_i^{max} for the whole dataset are 2006-06-12 and 2018-04-08 respectively (YY-DD-MM format). A specific date was chosen that would split the malware and benign samples. Anything less than the chosen date would be added to the training set and any date after would be in the testing set. Figures 5.1 and 5.2 show the distribution of dates for the malware and benign samples respectively. By trial and error a date of 2015-12-01 was chosen. This date resulted in 2,957 and 3,724 samples in the benign and malware test sets and 4,539 and 12,726 in the benign and malware training sets respectively. This split date allowed enough samples in the training set to train GANs while also having a fairly equal malware-benign split in the test set. Using this date split results in $\delta = 0.55$. The average percentage of malware in the wild $\hat{\sigma}$ is not known for the dataset so having a balanced ratio of benign and malware samples in the test set will ensure that there is no bias when calculating binary classification scores.

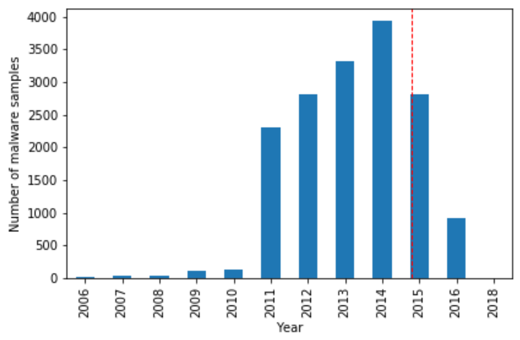


Figure 5.1: A chart showing the distribution of malware samples. For the malware samples the earliest sample was seen in 2006 and the latest in 2018. The majority of the samples occurred between 2011 and 2016.

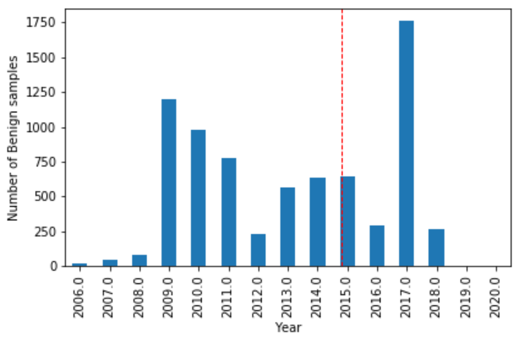


Figure 5.2: A chart showing the distribution of benign samples. The benign samples have a wider date range with the earliest and latest sample dates being 2006 and 2020 respectively. The benign date samples are also are more spread out than the malware samples with samples mainly occurring between 2009 and 2017.

5.2 GAN Training without Temporal Consistency

Firstly, the GANs are trained with the dataset that violates the **C1** and **C2** constraints. Therefore, the GANs are trained over the full dataset irrespective of the dates of malware samples. Subsequently malware images were generated and added to the new dataset. The Resnet was separately retrained with an extra added 500, 200 and 10,000 samples from each of the GAN models to the datasets with and without the temporal constraints. Table 5.1 shows the results from each of these experiments. The baseline model without temporal constraint achieved an accuracy score of 0.92 compared to the lower score of 0.81 with temporal constraint. The 10% score difference between the two is already a good indication that the accuracy of malware detection models are overstated and that they give a biased view of the real life performance.

Adding samples to the the dataset without temporal constraint gives an average accuracy increase of 2.7%, however, in the best cases accuracy increased by 4%. These results fall inline with work by

Lu & Li (2019) who found that adding samples from a GAN increased the accuracy score by 6% from 84% to 90%. The results from each individual GAN shows that there is no one particular set of samples that improve the accuracy scores over the others. While WGAN-DIV produced higher accuracy scores on average compared to BEGAN and DCGAN, the differences are too marginal to indicate that it produced higher quality samples.

When samples are added to the dataset with temporal consistency, the same increase in accuracy is not observed. The average accuracy score remains the same as the baseline at 0.81 with a highest increase of 2%. For DCGAN and WGAN-DIV with 500 samples, the accuracy score dropped by a percentage worsening the classification performance of the model.

	No Temporal Constraint			Temporal Constraint		
Baseline	0.92			0.81		
	#Samples			#Samples		
	500	2000	10000	500	2000	10000
DCGAN	0.96	0.94	0.94	0.79	0.8	0.82
BEGAN	0.94	0.95	0.95	0.82	0.83	0.82
WGAN-DIV	0.95	0.94	0.96	0.79	0.81	0.82

Table 5.1: Accuracy results for Resnet18 model trained with 500, 2,000 and 10,000 extra added samples from a GAN trained without temporal consistency. The samples were added to two different datasets, one with and one without temporal constraint. The baseline models were trained with no added samples.

5.3 GAN Training with Temporal Consistency

A GAN was trained with temporal consistency by only training it on malware samples before 2015-12-01. Again, 500, 2,000 and 10,000 malware samples were generated from the trained generator and added to the datasets with and without temporal consistency. Training the BEGAN over the temporal consistent dataset was difficult and kept resulting in mode collapse. It was therefore replaced with WGAN-GP with latent optimisation for general adversarial networks (LOGAN) Wu et al. (2019). LOGAN optimises GAN training by optimising the latent vector z using natural gradient descent. Therefore, instead of using a randomly sampled vector $z \sim p(z)$, an optimised latent vector is used $\Delta z = \alpha \frac{\partial f(z)}{\partial z} \quad z' = z + \Delta z$. The LOGAN optimisation has been shown to improve the image quality when applied to GAN training and therefore is a good replacement for BEGAN.

Table 5.2 shows the accuracy scores for each of the added samples. Generally the results are very similar for the GAN trained without temporal consistency. The samples added to the dataset without temporal consistency increased the accuracy score by 2% on average from 0.92 to 0.94 with a highest increase of 3%. The samples added to the dataset with temporal consistency gave an average accuracy increase of 1% with highest increase of 2%.

The results compared to the GAN trained without temporal consistency are very similar. Both indicate that samples added to the temporal consistency dataset only result in a small increase in accuracy. There are no significant differences between the two results showing that training the GAN with temporal consistency does not have a worse effect compared to the GAN without temporal consistency. This is probably due to the fact that the number of malware samples before 2015-12-01 is 12,726 samples. Hence, the temporal training set contains 77% of the malware samples and the distribution learnt will be very similar to the GAN trained on the whole dataset. The precision, recall and F_1 scores all followed the relative changes in the accuracy score with no classification metric being significantly different from the others. The prominent reason for this being that the temporally consistent test set is fairly balanced and hence no one class is over represented.

Whilst the accuracy scores do indicate whether the classifier model has achieved a better performance, it does not fully reflect a model's capabilities to predict future malware. Instead, Pendlebury proposed that the area under time (AUT) metric is used which evaluates the performance f of a classifier against time decay over N time units in a realistic setting. Equation 5.3 shows how the AUT metric is calculated. The equation is formulated from the area under curve (AUC) metric used to calculate the performance of a binary classifier for a receiver operator characteristic (ROC) curve.

	No Temporal Constraint			Temporal Constraint		
Baseline	0.92			0.81		
	#Samples			#Samples		
	500	2000	10000	500	2000	10000
DCGAN	0.95	0.93	0.95	0.82	0.81	0.83
LOGAN	0.95	0.94	0.94	0.82	0.82	0.81
WGAN-DIV	0.95	0.94	0.95	0.82	0.83	0.83

Table 5.2: Accuracy results for Resnet18 model trained with 500, 2,000 and 10,000 extra added samples from a GAN trained with temporal consistency.

At each time unit x_k where k is the test slot number $k \in [1, N]$, the metric f is calculated for the interval. The AUT is the area under the curve produced from the point estimates which is calculated using the trapezoidal rule. The perfect classifier which predicts all future malware correctly will have the highest score at each time interval and thus will produce a straight line with the whole area underneath the curve. However, in most cases the curve will drop over time as the model becomes less effective at predicting future malware.

$$\mathrm{AUT}(f, N) = \frac{1}{N-1} \sum_{k=1}^{N-1} \frac{[f(x_{k+1}) + f(x_k)] \cdot (1/N)}{2} \tag{5.3}$$

Table 5.3 shows the AUT scores for the samples added to the temporally consistent dataset. The AUT was calculated with f as the F_1 score with a time decay N as 12 months and with time slots set at each month. Thus the metric $\mathrm{AUT}(F_1, 12m)$ was calculated. The baseline AUT score is 0.601 with 10,000 samples from WGAN-DIV resulting in the largest AUT increase of 6.8%. Adding 10,000 samples gave the highest average increase in AUT of 5.6% compared to adding 500 and 2,000 samples. The increase in AUT scores shows that the added malware samples improve the detection of future malware more significantly than the accuracy metric indicates. The highest accuracy scores obtained for the temporally consistent dataset in Table 5.2 also have high AUT scores indicating that the two metrics do coincide with each other. The greater granularity of the AUT score also shows that the WGAN-DIV samples improve the ability to detect future malware over DCGAN and LOGAN which in turn suggests that the WGAN-DIV are of higher quality and give a better representation of malware image samples.

	Temporal Constraint		
Baseline	0.601		
#Samples	500	2000	10000
DCGAN	0.59	0.602	0.659
WGAN-DIV-LOGAN	0.656	0.609	0.643
WGAN-DIV	0.675	0.649	0.669

Table 5.3: AUT scores for Resnet18 model with added samples from temporally consistent GANs.

Figures 5.3 and 5.4 show the performance metrics for the malware class over time for the baseline classifier with no added samples and the WGAN-DIV model with 10,000 samples which achieved the highest increase in AUT score. The x-axis shows the testing period where the classification metrics are calculated for the malware and benign samples for that month and the corresponding score is on the y-axis. On observing the two figures, the baseline plot initially performs well and then the scores drop at months 4 to 6 before recovering back up to scores between 0.7 and 1.0. The WGAN-DIV plot shows that, on average, the performance metrics are higher than the baseline with a far less significant drop between months 4 and 6. A key observation is that the recall does not have any kind of comparable drop in score compared to the precision and F_1 scores and remains fairly consistent across each month. However, while this indicates that the model now identifies a higher number of malware samples, the precision still has significant drops at months 1 and 5. This shows that in these months a higher proportion of benign samples are being incorrectly classified as malware. Overall the F_1 scores are consistently higher than the baseline performance. The main differences between the two graphs are the areas underneath the lines which indicate the AUT scores. As the WGAN-DIV has the higher average performance, the area underneath the plot is greater and hence reflects the higher AUT score.

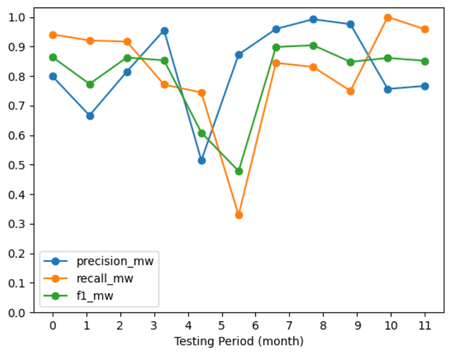


Figure 5.3: Baseline performance metric plot with the malware class precision, recall and F_1 score for each month in the testing set.

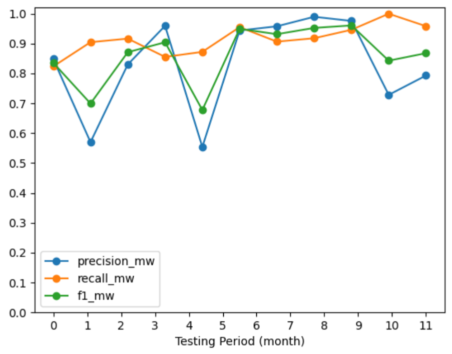


Figure 5.4: Model performance metric plot trained on dataset with 2,000 WGAN-DIV samples showing the malware class precision, recall and F_1 score for each month in the testing set.

5.4 GAN Training with Differentiable Augmentation

In Pendlebury's et Al's paper, they state that the realistic setting for training and testing a model is with both temporal and spatial constraints enforced. To prevent spatial bias, only a realistic amount of malware should be used for training and testing. If there is more malware than benign samples then this can positively inflate the f_1 score and this is a problem that generally occurs in imbalanced datasets. Pendlebury showed that training with only 10% of the available malware can lead to significant reductions of accuracy in the model.

It may be the case that the availability of certain types of malware samples to train with might be low as a new malware variant may only been seen briefly in the wild and subsequently, a malware classifier will perform poorly at detecting it. Having the ability to generate more malware samples to correct class imbalances could be an effective method of increasing classifier performance.

However, it has not been previously possible to train GANs with small amounts of data with common datasets containing a significant amount of images. Training with small amounts of images causes the discriminator to overfit to the training examples and subsequently the feedback to the generator becomes ineffective and divergence will occur (Karras et al. (2020a)). Data augmentation for GAN training is a technique that has been explored to increase the amount of available samples that the generator and discriminator can learn from. Zhao et al. summarises several methods that have been attempted to augment the images during GAN training, however, they cause instability between the generator and discriminator and cause the model to collapse. In Figure 5.5, previous work followed steps i and ii by augmenting the real and fake images when the discriminator D is updated. By performing step iii which propagates the gradients of the augmented samples to the generator G as well as the discriminator, the GAN learns suitable distribution of the training data. By ensuring the methods of image augmentation $T(x)$ are differentiable, the gradients can be backpropagated through G. This results in the ability to train a GAN with as little as 300 images.

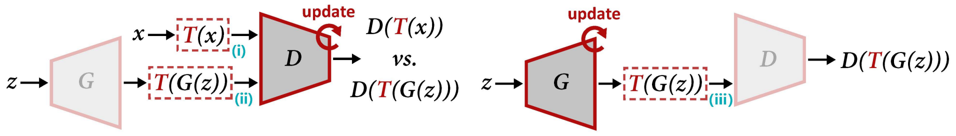
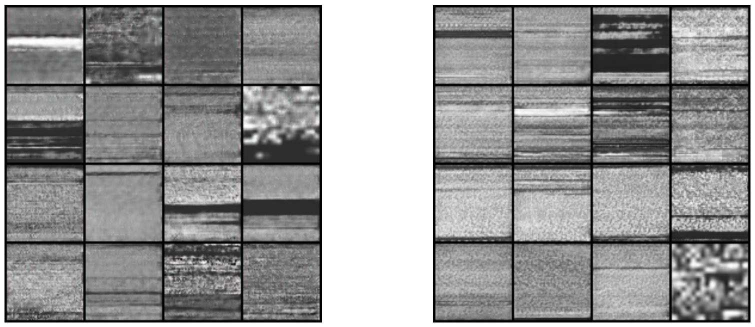


Figure 5.5: Overview of DiffAug. DiffAug applies augmentation to the real and fake images. The transformation T has its gradients backpropagated back through the generator G (Zhao et al.).

The use of Differentiable Augmentation (DiffAugment) allows training a GAN with a small number of malware images and hence can improve classifier performance with extra added samples. To test the basis of this hypothesis, a DCGAN model was modified to apply augmentation to the real and fake images when updating G and D. The augmentations functions are preimplmented and available from the authors GitHub repo (`https://github.com/mit-han-lab/data-efficient-gans`).

The malware samples from the dataset with temporal consistency were randomly downsampled to 1,645 samples which represents 10% of all malware samples in the dataset. These images were trained on the DCGAN with DiffAug with all three types of augmentation applied to each image batch (color, translation and cutout) and the model was ran until it had reached convergence. The use of DiffAugment enables the training process to produce high quality samples even with small amounts of available malware images. Figure 5.6 shows a comparison of the samples malware images from a DCGAN trained over the whole dataset and DCGAN+DiffAug over the downsampled dataset. The two sets of samples look very similar although, on closer inspection, the DCGAN+DiffAug samples in Figure 5.6b are sharper whilst the DCGAN samples in Figure 5.6a are slightly blurred. This is only a visual inspection and is prone to human bias. A comparable image quality statistic such as Fréchet Inception Distance could be used to confirm if this is the case.



(a) Malware samples from DCGAN **(b)** Malware samples from DCGAN+DiffAug

Figure 5.6: Sets of samples generated from DCGAN with/without DiffAug

The Resnet18 classifier was then trained on the same 10% of the malware used to train the GAN along with all the benign samples (4,441 samples) from the temporal dataset. The same temporally consistent test set of malware and benign samples after 2015-12-01 was used to give equal comparison to the previous results.

As shown in Table 5.4, the baseline accuracy score for the downsampled dataset without any added samples is 0.755. The lower accuracy score is caused by the smaller number of malware samples and also 2.7 times more benign than malware samples in the training set. Adding 500 extra samples increases the accuracy by 1.5% and adding 2,000 samples by 3%. Adding 10,000 samples causes the accuracy to increase by 0.5% which is most likely caused by the large number of extra malware samples imbalancing the dataset. The effect of adding the extra samples to balance the dataset gives a considerable increase to the accuracy score of the model.

Table 5.5 shows the precision, recall and f1-scores for each of the trained Resnets with an added 500, 2,000 and 10,000 generated samples. After 500 extra pseudo malware samples are added, the classifier becomes better at identifying benign samples as the benign class recall increased from

	Temporal Constraint		
Baseline	0.755		
#Samples	500	2000	10000
DCGAN + DiffAug	0.77	0.785	0.750

Table 5.4: Accuracy scores from adding samples DCGAN+DiffAug to the temporally consistent dataset with 10% of the original malware samples.

0.75 to 0.82 while the precision remains at 0.71. The malware class precision increase from 0.79 to 0.84 shows that the classifier is more accurate at labelling benign samples correctly at the cost of identifying less malware samples as the malware class recall drops from 0.76 to 0.74. When 2,000 pseudo malware samples are added to the malware class, the recall increases to 0.91 with a drop in recall for the benign class from 0.82 to 0.63 showing that malware samples are being over predicted. Finally, 10,000 added samples causes significant drops in most metrics showing the adverse effect on a high class imbalance. This demonstrates that whilst added samples can improve classifier performance, adding too many samples can degrade it. The number of samples between 500 and 2,000 extra samples can be fine tuned in order to give the optimal increase of performance for both classes.

#Samples		precision	recall	f1-score	support	AUT
0	Benign	0.71	0.75	0.73	2916	0.55
	Malware	0.79	0.76	0.78	3724	
500	Benign	0.71	0.82	0.76	2916	0.54
	Malware	0.84	0.74	0.78	3724	
2000	Benign	0.84	0.63	0.72	2916	0.62
	Malware	0.76	0.91	0.83	3724	
10000	Benign	0.62	0.85	0.72	2916	0.57
	Malware	0.84	0.6	0.7	3724	

Table 5.5: Classification metrics from adding samples DCGAN+DiffAug to the temporally consistent dataset with 10% of the original malware samples.

Table 5.5 also shows the AUT scores for each number of added samples. The baseline AUT with no added samples has an AUT score of 0.55 and by adding 2,000 samples gives the best score of 0.62. The AUT score seems closely linked to the overall accuracy of the model as the models with the highest accuracy scores also reflect the best AUT scores. As mentioned in the previous results when training the GANs with/without temporal consistency, the balanced test set means that the accuracy reflects the performance of a model.

Figures 5.7 and 5.8 show the malware performance metrics of the baseline and 2,000 added samples models as point estimates over time. As time elapses the precision, recall and F_1 scores follow each other which is again down to the balanced sized test set. The model with 2,000 samples scores are higher on average than the baseline model with scores only getting as low as 0.5 compared to the baseline model which has a lowest score of 0.25. Overall this is shown in a larger area underneath the curve for the 2,000 samples model and reflects the higher AUT score.

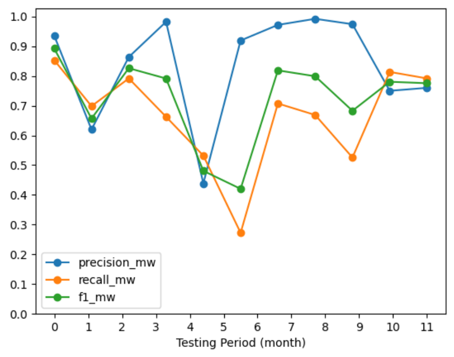


Figure 5.7: Baseline performance metric plot on the smaller temporally consistent dataset with the malware class precision, recall and F_1 score for each month in the testing set.

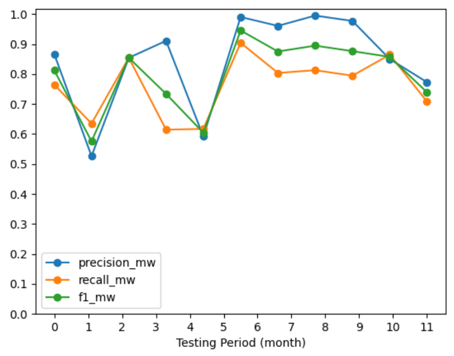


Figure 5.8: Model performance metric plot trained on the smaller temporally consistent dataset with 2000 WGAN-DIV samples showing the malware class precision, recall and F_1 score for each month in the testing set.

Chapter 6

Discussion

This chapter discusses the results from each of the experiments as well as reflecting on the project planning and time management.

From the results it can be seen that malware samples generated by a GAN can contribute to increasing the classification performance of future malware. Measuring accuracy alone shows that the added samples may have very little effect on future malware prediction on a temporal dataset and the use of the AUT metric gives a clearer measure of the effects of the added samples. The AUT and accuracy scores were all reported without a measure of spread such as standard deviation due to the fact that training a Resnet took a considerable amount time; running an experiment over several runs would have been infeasible in the time available. In order to remedy this, all the experiments were seeded with a value of 0 using the PyTorch Lightning $\text{seed\_everything}()$ function. However, different accuracy scores were obtained over several runs and hence accuracy and AUT scores could be greater or lower than reported in the results. Nonetheless the results still give a valid representation of the effects of future malware prediction and, if a very accurate measure were required, then the experiments would need to be run a number of times to gain this higher degree of certainty.

Following on from this, the temporally consistent dataset used a fixed split date for all of the experiments. Usually it would be the case that the several date splits are used to show how the accuracy of a classifier changes over several training and test sets. Again, this was not possible with the time constraints as a GAN can take days to run before it converges. However, this could be considered with more time and a powerful GPU to reduce GAN training times.

A key observation obtained in the first two experiments is that adding a high number of samples (10,000) did not worsen the classification performance when it could be expected that this amount would significantly imbalance the dataset. The main reasoning behind this is the use of dropout layers in the Resnet18 model. Dropout is where units in the neural networks are dropped randomly along with their connections which prevents co-adaption between units. Consequently, dropout is equivalent to minimising a regularised network which in turn penalises overfitting. This in turn prevents a large number of malware samples from degrading the network performance. However, if another CNN classifier does not use dropout, then imbalancing the dataset with a large number of generated samples may cause poorer results.

Addressing the key hypothesis of this paper, the results show that the use of GANs to generate samples do have an effect on future malware prediction. However, it is not known what kind of malware are detected when extra samples are used to train a classifier model. It may be the case that new malware that is detected after adding more samples is very similar to previous malware and significantly different malware is never picked up. This raises the question as to whether training a classifier with added samples actually helps predict new variants of malware or whether it helps detect malware seen in the training set that may be slightly modified. Furthermore, this work focuses on the binary class problem of detecting malware or benign rather than the prediction of several malware variants, so it may be the case that future malware prediction could be better or worse with a different type of classification problem.

6.1 Project Planning and Time Management

In order to plan effectively at the start of the project a PERT chart (shown in Figure 6.1) was created to give an outline of the key tasks and the amount time required to undertake each of them. The individual boxes indicate the project milestones with the estimated length of time it should take to complete. Taking the duration of the project to be 13 weeks, the critical path of the PERT graph is 9 weeks. This left 4 weeks of contingency that could be used as floats for each of the milestones within the project and to start writing up the project report. The time taken to complete the milestones later on in the graph is less certain and, therefore, the extra float could have been used at these points. The milestones indicated in the dotted box indicate that these components follow an iterative process and hence the workflow could switch between each component depending on how well a model is working.

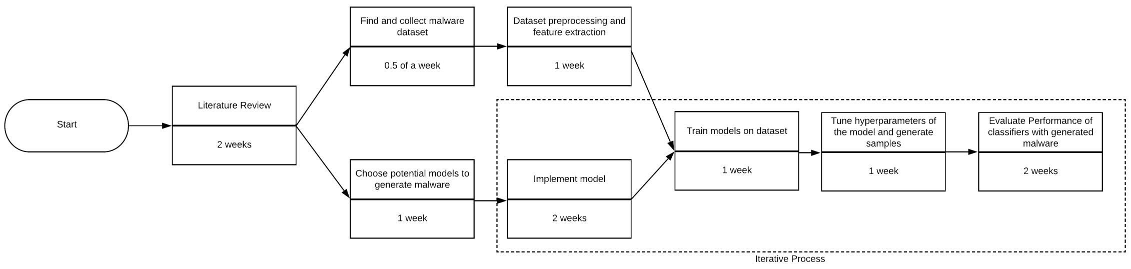

Start
Literature Review
2 weeks
Find and collect malware dataset
0.5 of a week
Dataset preprocessing and feature extraction
1 week
Choose potential models to generate malware
1 week
Implement model
2 weeks
Train models on dataset
1 week
Tune hyperparameters of the model and generate samples
1 week
Evaluate Performance of classifiers with generated malware
2 weeks
Iterative Process

Figure 6.1: A PERT Chart showing the outline of the project plan. The oval on the left hand side indicates the start of the project with the two branches showing the milestones that could be worked on at the same time. Each milestone is dependent on the previous milestone being completed. The dotted box shows an iterative process where at any point the project could go back to a previous milestone.

6.2 Problems faced

One of the problems face during the project was the training of GANs. GANs are known for their unstable training process and the generator and discriminator must be kept synchronised to prevent mode collapse. Mode collapse occurs when the generator only provides a limited variety of samples as the generator will converge to only produce a single output. If the synchronisation is broken then the generator will converge to an optimal image x^* that consistently fools the discriminator. The extreme case is where the mode collapses to a single point and hence the gradient of the discriminator's loss function will be zero and no gradient descent will occur. At this point any latent vector that is passed to the generator will always produce the same image. This issue was mainly prevalent in BEGAN but GANs using the Wassertein distance metric did not encounter these problems. Using trial and error by modifying image prepossessing and model hyper parameters, the majority of GAN models successfully trained without mode collapse.

The GAN training time was another problem that had to be addressed. On average a GAN would take at least a day of training for convergence with more complex GANs taking longer. This was overcome by using several GPU enabled machines and training several models in parallel. This also used up some of the 4 week float time so the overall project was not significantly delayed.

Chapter 7

Conclusion

This work explored the use of generative models for future malware prediction. The overall results show that samples from GAN models do have an impact in predicting future malware. Datasets without temporal constraints overstated the effect of adding extra generated samples and the increase in accuracy was not as significant when adding generated samples to datasets with temporal constraints. The AUT metric increased by 6.8% when malware samples were added to the temporal dataset indicating that there was a measurable benefit to adding the generated malware samples. In addition, differentiable augmentation was applied to the training of a DCGAN which enabled it to learn the distribution of malware images with very few images. The CNN trained over the imbalanced datatset resulted in lower accuracy however the added malware samples corrected the dataset imbalance and subsequently improved the final accuracy by 3%. Further work is needed to confirm the validity results and to give more insight of the benefit gained from using generative models.

All code for the report can be found at `https://github.com/xandernewton/Future-Malware-Prediction-Through-Generative-Modelling`

Chapter 8

Future Work

Future work in this area of research could include determining a more accurate way of quantifying the effect of adding malware samples to the original dataset with temporal constraints along with determining what kind of samples are detected when more extra samples added. This could be achieved by examining the activation maps of the malware images at the different layers of a CNN classifier to see what regions contribute to the classification of a malware sample. This analysis should also be extended to mutliclass datasets where the aim is to predict malware families. The exploration of differentiable augmentation would also be very useful for these types of datasets where there may be a low number of samples for a specific malware variant. An alternative option to differentiable augmentation is to use a conditional GAN (CGAN) which uses class labels to generate images for a specific classes. The multiclass classification problem would be possible with the MC-dataset-multiclass dataset used in this work. However, other common datasets such as MalImg and the Microsoft Malware Classification dataset cannot be used due to the lack of timestamps for the malware samples which enable the dataset to be temporally consistent.

Due to memory limitations, the malware image size was fixed at 64x64 although larger image sizes may help boost classifier performance as the increase in image size will give a clearer structure of a malware executable and reveal details that may have been lost due to downscaling an image. More memory will also allow training of large GAN models such as BigGan and StyleGan which could result in very high quality malware image samples. Generative modelling is currently a very active research area and new GANs are released frequently as well as new methods to train them. Therefore, new state of the art GANs may further improve on image quality. An important metric of GAN image quality is the Fréchet Inception Distance (FID) which allows for a comparison between images produced by different GANs. FID is calculated using the statistics from a pretrained InceptionV3 network. Malware images are significantly different to images used to train publicly available Inception models and thus open source FID libraries would not be suitable for calculating FID of malware images. Instead further work could train an InceptionV3 model with malware images which can then be subsequently open sourced to enable comparable FID scores across further research. Other generative models which could be explored include, Fully Visible Belief Nets inlcuding PixelCNN (van den Oord et al. (2016)) or new advances in VAEs such as VQ-VAE-2 (Razavi et al. (2019)).

Other future work could also investigate the methodology used to train GANs and CNN classifiers. This work used the default hyper-parameters for the GAN models which could be further refined to give improved image quality for malware images. In addition, Pendlebury et al. (2019) proposed the TESSERACT algorithm to find the optimal percentage of malware to use in the training set as well as using delay strategies such as incremental retraining and active learning to improve the AUT scores. These methods could added to the training of the CNN classifier to gain more optimal future malware prediction performance.

Bibliography

Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static PE machine learning malware models via reinforcement learning. *CoRR*, abs/1801.08917, 2018. URL `http://arxiv.org/abs/1801.08917`.

David Berthelot, Thomas Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks, 2017.

Violet Blue. Cryptolocker's crimewave: A trail of millions in laundered bitcoin, Dec 2013. URL `https://www.zdnet.com/article/cryptolockers-crimewave-a-trail-of-millions-in-laundered-bitcoin/`.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2018.

Roland Burks, Kazi Aminul Islam, Yan Lu, and Jiang Li. Data Augmentation with Generative Models for Improved Malware Detection: A Comparative Study. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2019*, pp. 0660–0665. Institute of Electrical and Electronics Engineers Inc., oct 2019. ISBN 9781728138855. doi: 10.1109/UEMCON47517.2019.8993085.

Raphael Labaca Castro, Corinna Schmitt, and Gabi Dreo. AIMED: Evolving malware with genetic programming to evade detection. *Proceedings - 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering, TrustCom/BigDataSE 2019*, pp. 240–247, 2019. doi: 10.1109/TrustCom/BigDataSE.2019.00040.

Eduardo de O. Andrade. MC-dataset-multiclass. 3 2018. doi: 10.6084/m9.figshare.5995468.v1. URL `https://figshare.com/articles/dataset/MC-dataset-multiclass/5995468`.

WA Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning Cited by*, 3, 2019.

Daniel Gibert, Carles Mateu, · Jordi Planes, and · Ramon Vicens. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15:15–28, 2019. doi: 10.1007/s11416-018-0323-0. URL `https://doi.org/10.1007/s11416-018-0323-0`.

Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges, 2020. ISSN 10958592.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Michael Howard, Avi Pfeffer, Mukesh Dalai, and Michael Reposa. Predicting signatures of future malware variants. In *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, volume 2018-Janua, pp. 126–132. IEEE, oct 2017. ISBN 978-1-5386-1436-5. doi: 10.1109/MALWARE.2017.8323965. URL `https://ieeexplore.ieee.org/document/8323965/`.

Weiwei Hu and Ying Tan. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. feb 2017. URL `http://arxiv.org/abs/1702.05983`.

Vaidas Juzonis, Nikolaj Goranin, Antanas Cenys, and Dmitrij Olifer. Specialized Genetic Algorithm Based Simulation Tool Designed For Malware Evolution Forecasting. *Annales UMCS, Informatica*, 12(4):23–37, 2013. ISSN 1732-1360. doi: 10.2478/v10065-012-0031-1.

Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. June 2020a.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2020b. doi: 10.1109/cvpr42600.2020.00813. URL `http://dx.doi.org/10.1109/cvpr42600.2020.00813`.

Masataka Kawai, Kaoru Ota, and Mianxing Dong. Improved MalGAN: Avoiding Malware Detector by Leaning Cleanware Features. In *1st International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2019*, pp. 40–45. Institute of Electrical and Electronics Engineers Inc., mar 2019. ISBN 9781538678220. doi: 10.1109/ICAIIC.2019.8669079.

Jin-Young Kim, Seok-Jun Bu, and Sung-Bae Cho. Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. *Information Sciences*, 460-461:83–102, sep 2018. ISSN 00200255. doi: 10.1016/j.ins.2018.04.092. URL `https://linkinghub.elsevier.com/retrieve/pii/S0020025518303475`.

Yan Lu and Jiang Li. Generative Adversarial Network for Improving Deep Learning Based Malware Classification. In *Proceedings - Winter Simulation Conference*, volume 2019-Decem, pp. 584–593. Institute of Electrical and Electronics Engineers Inc., dec 2019. ISBN 9781728132839. doi: 10.1109/WSC40007.2019.9004932.

Zahra Moti, Sattar Hashemi, and Amir Namavar. Discovering Future Malware Variants By Generating New Malware Samples Using Generative Adversarial Network. In *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*, number Iccke, pp. 319–324. IEEE, oct 2019. ISBN 978-1-7281-5075-8. doi: 10.1109/ICCKE48569.2019.8964913. URL `https://ieeexplore.ieee.org/document/8964913/`.

L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: Visualization and automatic classification. In *ACM International Conference Proceeding Series*, 2011. ISBN 9781450306799. doi: 10.1145/2016904.2016908.

Ian Goodfellow Openai. NIPS 2016 Tutorial: Generative Adversarial Networks. Technical report. URL `http://www.iangoodfellow.com/slides/2016-12-04-NIPS.pdf`.

Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. Tesseract: Eliminating experimental bias in malware classification across space and time. *Proceedings of the 28th USENIX Security Symposium*, pp. 729–746, 2019.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2, 2019.

Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft malware classification challenge, 2018.

Mihaela Rosca. VAEs and GANs. CVPR18, June 2018.

C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

Anh Pham Tuan, An Tran Hung Phuong, Nguyen Vu Thanh, and Toan Nguyen Van. Malware Detection PE-Based Analysis Using Deep Learning Algorithm Dataset. 6 2018. doi: 10.6084/m9.figshare.6635642.v1. URL `https://figshare.com/articles/dataset/Malware_Detection_PE-Based_Analysis_Using_Deep_Learning_Algorithm_Dataset/6635642`.

Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis, 2019. ISSN 01674048.

Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders, 2016.

Jiqing Wu, Zhiwu Huang, Janine Thoma, Dinesh Acharya, and Luc Van Gool. Wasserstein divergence for gans. *Lecture Notes in Computer Science*, pp. 673–688, 2018. ISSN 1611-3349. doi: 10.1007/978-3-030-01228-1_40. URL `http://dx.doi.org/10.1007/978-3-030-01228-1_40`.

Yan Wu, Jeff Donahue, David Balduzzi, Karen Simonyan, and Timothy Lillicrap. LOGAN: Latent optimisation for generative adversarial networks. December 2019.

Fang Yong, Zeng Yuetian, Li Beibei, Liu Liang, and Zhang Lei. Malware dataset from virusshare used in article ”deepdetectnet vs rlattacknet: An adversarial method to improve deep learning-based static malware detection model”, Feb 2020.

Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, Adobe Research, and Song Han. Differentiable Augmentation for Data-Efficient GAN Training. Technical report. URL `https://github.com/mit-han-lab/data-efficient-gans`.

Appendix A

Model Parameters

Parameters	DCGAN	DCGAN+DiffAug	BEGAN	WGAN-DIV	WGAN-DIV-LOGAN
Batch Size	64	64	64	64	64
Learning Rate	0.0002	0.0002	0.0002	0.0002	0.00005
Optimiser	Adam	Adam	Adam	Adam	Adam
b1 (decay first order)	0.5	0.5	0.5	0.5	0.5
b2 (decay second order)	0.999	0.999	0.999	0.999	0.99
Size of latent space	128	128	62	100	100
Image size	64	64	64	64	64
Image channels	3	3	3	3	3
Augmentation	N/A	color+translation+cutout	N/A	N/A	N/A
Number of critics	N/A	N/A	N/A	5	5
Clip Value	N/A	N/A	N/A	0.01	N/A
Latent Optimisation Method	N/A	N/A	N/A	N/A	natural gradient descent

Table A.1: Hyperparameters of each GAN model

Appendix B

Design File Outline

```
ROOT
|
|__scripts
|   |__vgg.py - Implementation of the VGG network
|   |__vae.py - Implementation of a vanilla VAE
|   |__resnet.py - Implementation of a resnet18 network
|   |__inceptionV3.py - Implementation of a InceptionV3 newtork
|   |__get_PE_timestamp.py - Retrieves timestamps from the PE
|   |  header of an windows executable
|   |__GAN_2_64.py - Implementation of a vanillia GAN network
|   |__binary_to_image.py - Converts binary files to images
|   |__AUT.py - Calculates the AUT score given a resnet checkpoint
|   |  and test set data path
|   |__Accuracy.py - Calculates the accuracy and classification
|      metrics given a resnet checkpoint and test set data path
|__PyTorch-GAN
|   |__implementations
|   |__sample.py - Code to generate new samples from a GAN
|       |__DiffAugment_pytorch.py - Code from the official
|       |  Differentiable Augmentation Github repo
|       |__utils.py - Functions shared between each of the GAN
|          implementations
|           |__dcgan - Contains the code dcgan model
|           |__began - Contains the code began model
|           |__wgan_div - Contains the code wgan_div model
|           |__DCGANDiffAug - Contains the code dcgan with diff aug
|__notebooks
    |__verify_dataset.ipynb - Notebook for ensuring temporal
    |  consistency of a dataset
    |__preprocessing.py - Notebook to use virus total API to get
    |  dates and split into temporal datasets
    |__malware_date_split.ipynb - Notebook for splitting a dataset
    |  by a specified date
    |__downsample.ipynb - Notebook to downsample the dataset
```