UPPSALA
UNIVERSITET

IT 22 128

Degree project 15 credits

January 2023

Exploring GANs to generate attack-variations in IoT networks

Gustaf Bennmarker

Bachelor's Programme in Computer Science

UPPSALA UNIVERSITET

Exploring GANs to generate attack-variations in IoT networks

Gustaf Bennmarker

Abstract

Data driven IDS development requires a vast amount of data to be effective against future attacks and a big problem is the lack of available data. This thesis explores the use of GANs (Generative adversarial networks) in generating attack data that can be used as a part of a training set for an IDS to improve the robustness against adversarial attacks. GAN has been used extensively in the field of image generation therefore this thesis uses image data instead of network data to easily test our methods. The image data set used was the MNIST dataset of handwritten digits where one digit class was chosen as the attack data class. The attack data class was reduced in quantity from the training set in order to replicate real world availability of attack data, different quantities were used to see how well the GAN training setup would perform with class imbalances in the training set and the generated data tested against classifiers acting as IDSs. The classifiers were added to the GAN training loop in order to bias the generator into causing misclassifications and thereby generating adversarial samples. The experiments show that in some cases the majority of the generated attack samples managed to fool the IDS while using a small amount of attack data in the training set. Adding the classifier to the training loop managed to further fool the IDS where in some instances only 2% of the generated attack data were detected. The results show that it is possible to generate data that manages to fool the IDS into being classified as something else but that there is more work to be done, getting consistent results was difficult and improving the GAN training setup by using a more advanced GAN model might solve these issues.

Faculty of Science and Technology

Uppsala University, Place of publication Uppsala

Supervisor: Andreas Johnsson Subject reader: Christian Rohner

Examiner: Johannes Borgström

Contents

1 Introduction **2**

2 Background **4**

2.1 Intrusion detection systems 4
2.2 Deep Learning 5
2.2.1 Classification in deep learning 6
2.2.2 The Generator 8
2.3 Understanding Generative adversarial networks 8
2.3.1 GAN Training 9
2.4 Conditional Generative Networks 10
2.5 GAN and IDSs 10
2.6 Tools 10

3 Methods **12**

3.1 Attack Data Availability 12
3.2 Class imbalance 12
3.3 Limited Classifiers 13
3.4 Oracle Classifier 14
3.5 CGAN model 14
3.6 CGAN Training methods 14
3.6.1 Traing with a class imbalance 14
3.6.2 Adding loss of the limited classifiers 14
3.6.3 Conditioning on the Discriminator 15

4 Results **17**

4.1 Comparing class imbalances 17
4.1.1 Comparing results against classifiers 18
4.2 Weighted Classifier Loss 19
4.2.1 Negative Classifier loss 19
4.2.2 Positive Classifier loss 21
4.3 Conditioning on the Discriminator 22

5 Discussion **25**

5.1 Answering initial question 25
5.2 Consistency in results 25
5.3 Conditioning on discriminator 25
5.4 Modifying model to use network data 26

1 Introduction

As the number of IoT devices increases in society they also become more likely to be targets for attacks and malicious activities[1]. The need for data driven Intrusion Detection Systems (IDS) development is therefore greater than ever. A promising approach [2] is the use of Machine learning where the IDS trains on the data of a network to learn to differentiate from what is normal network data and what is not. The fundamental problem with this approach is that the data used is based on previous examples making it difficult for the IDS to predict what future attacks might look like. One way to potentially solve this issue is to generate synthetic attack data with Generative adversarial networks (GAN).

GAN is a class in machine learning introduced by Ian J. Goodfellow in 2014 [3], consisting of two neural networks that are training adversarially in a min max game. The networks are known as the Generator and the Discriminator, the Generator generates fake data while the Discriminator evaluates if the data is real or not. This is done by feeding the Generator some random noise that the Generator learns to map to some data distribution. The fake data is then fed to the Discriminator along with real data of the same distribution and the Discriminator distinguishes the real data from the fake. The Discriminator is initially fed an amount of real data from the training set until it achieves an acceptable accuracy, while the Generator trains based on whether it fools the Discriminator. Backpropagation procedures are then applied independently to both networks making the Generator better at producing samples and the Discriminator better at recognising fake samples.

Machine learning is not only used to detect attacks but also to generate attacks. By slightly distorting specific parts of available data it has been shown that it is possible to make machine learning algorithms make incorrect classifications with high confidence[4]. These types of attacks are known as adversarial attacks and make it difficult to apply machine learning to security critical areas. One potential extension of GAN is to apply adversarial training procedures to the GAN training setup to not only address the challenge with data availability but to also increase the robustness of the IDS against adversarial attacks.

This thesis aims to:

- Explore the ability of a GAN to generate synthetic attack-data to diversify and enlarge the training set for IDS models.
- Extend GAN training methods towards generating adversarial samples for a more robust protection

2 Background

This section covers the theory needed to understand the applied methods used in the experiments of this thesis. This chapter starts by introducing IDS, Deep learning, Neural Networks and GAN, and then goes further into explaining the procedure of GAN training as well as GANs relationship to cybersecurity and IDSs.

2.1 Intrusion detection systems

An Intrusion Detection System (IDS) is a device or a software application that monitors a network or a system for malicious activity. We can distinguish between two IDS approaches, signature based and machine learning based.

A signature based IDS is used to identify known threats, by monitoring all packets of the network and comparing them against a database of known attack signatures, the IDS is then able to flag any packets with suspicious behavior much like an antivirus software.[5]

A machine learning based IDS trains on the data of a network to establish a normalized baseline where the baseline represents how the network behaves under normal circumstances, all network activity is then compared to that baseline. So instead of searching for known attack signatures the machine learning based IDS identifies any out of the ordinary behavior to flag potential threats. [6] In recent years studies have shown that the usage of deep learning in training IDSs have been able to further improve detection rates with an accuracy of up to 98% [7].

While deep learning has shown to be an effective way of detecting malicious activities it also has introduced issues regarding the robustness against adversarial attacks. In 2013 Christian Szegedy and his team demonstrated that by applying an imperceptible perturbation to image data they were able to cause a neural network to misclassify the image with high accuracy [8].

The intention with adversarial machine learning methods is to cause the implemented machine learning model to make misclassifications. Adversarial examples are designed to be close to the original data but with an added amount of perturbation undetectable by humans which manages to fool the model with high precision [9].

2.2 Deep Learning

Deep learning is a family of techniques for machine learning, where the word deep refers to the fact that the structure of the algorithms are often organized into many layers and the computational path will have many steps from input to output [10]. Deep learning is a widely used approach in fields such as: visual object recognition, machine translation and image synthesis. The origins of deep learning dates back all the way to the 1940s where Warren McCulloch and Walter Pitts tried to model biological neurons in the brain mathematically [10]. For this reason modern day deep learning methods are often referred to as Neural Networks even though they have little resemblance to real biological neurons.

A feedforward neural network is the simplest form of a neural network in the sense that it only has connections in one direction see Figure 1. Each of the nodes in the hidden layers computes a function and the results are then passed to the nodes/node of the next layer so information flows through the network from the input layer to the output layer without any loops as in the case of a recurrent neural network [11]. This thesis will use feedforward neural networks in its architecture.

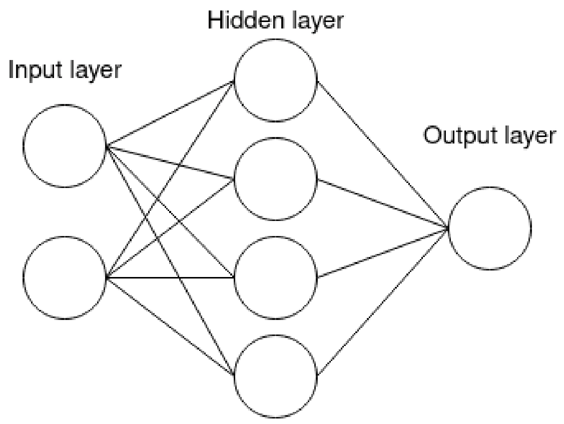


Figure 1: A simple Neural Network

2.2.1 Classification in deep learning

One of the applications of deep learning is classification tasks where data is classified into categories, classifiers have a variety of applications one of these is image classification. By feeding training data to the network during the training process the network will learn to distinguish between the different classes of data in the training set. The network's accuracy can then be tested by feeding it the test data set and comparing its predicted labels against the actual class labels of the data set. For a network with a training set of 10 data classes the classifier will have 10 output nodes, each one giving a value between 0.00 and 1.00 of how certain it is of which class that the input data is part of[12].

A simpler kind of classifier is a Binary Classifier which only has one output node representing if data is part of one class or not. To build a binary classifier that is able to let's say distinguish if an image represents a 3 or not one can use the MNIST data set as training data. The data set is a low-complexity data collection of handwritten digits (see Figure 2) consisting of 70000 28×28 pixel black and white images representing the digits zero through nine and are split into a training set of 60000 images and a test set of 10000 images. The data set has become a standard benchmark when training and testing image classification algorithms [13].

The input layer of the Binary classifier would in this case be the pixels of the image which is $28 \times 28 = 784$ where each pixel would hold a gray scale value ranging from 0.00 to 1.00 where 0.00 is black and 1.00 is white[12]. Each node in the input layer would then cause some specific pattern in the next layer which would cause some specific pattern in the next and so on until finally reaching the output layer which in this case would give a single value between 0.00 and 1.00 of how certain it is of that the image is a 3 see Figure 3.

The goal with the layered structure is for each layer to recognize some pattern within the input image. Each of the nodes in the network are connected with weights and each node of the hidden layers is assigned some bias; these weights and biases are assigned some random value when the training process starts. For the model to be able to learn these weights and biases need to be tweaked in a process called backpropagation. This is done by a loss function that maps the predictions of the model against the desired value into a real number representing how well the network is

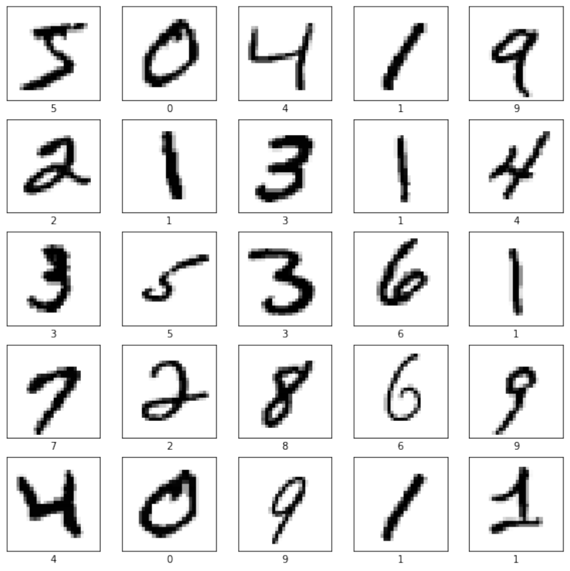


Figure 2: Sample from the MNIST dataset

performing. Backpropagation is a process where the gradient of the loss function is calculated with respect to the weights of the network with the goal of adjusting the weights to minimize the loss [12]. By using a large training set the network will see many examples of 3s and will gradually get better at classifying them.

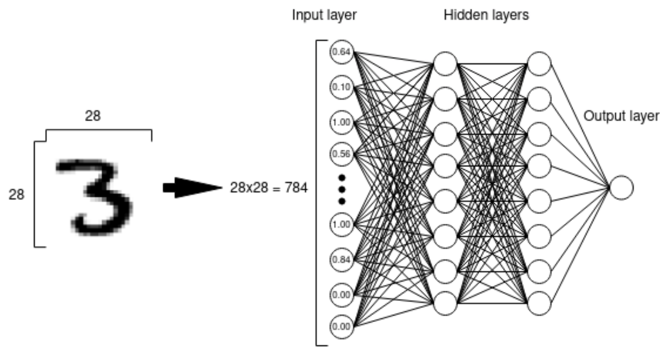


Figure 3: Simplified architecture of a binary classifier

2.2.2 The Generator

Another application for Deep Learning is to generate data, this can be achieved using a Neural Network called the Generator. To generate images of digits using the MNIST data set the input layer of the network will consist of 784 nodes as in the case of the binary classifier but since the output of the network is an image the output layer will also consist of 784 nodes [14]. The input image of the Generator is an image of random noise i.e random black and white pixels, these pixels are then shaped by going through the various hidden layers of the network until finally reaching the output layer where a new image is produced see Figure 4. For a generator model to be able to learn it needs some source of external input from another Neural Network, more about this in the next section.

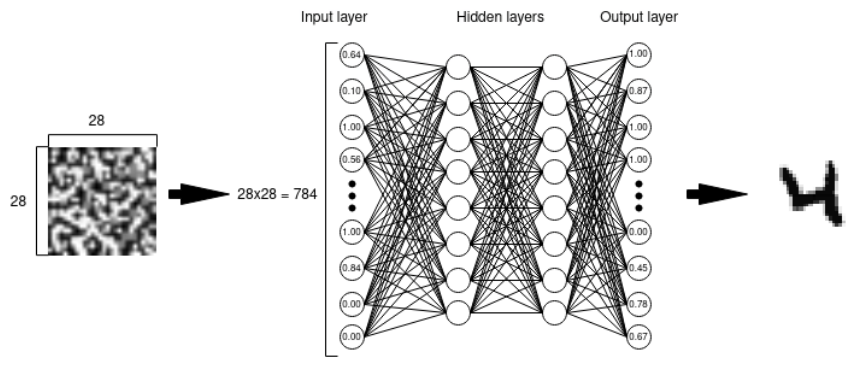


Figure 4: Simplified architecture of a Generator

2.3 Understanding Generative adversarial networks

Generative adversarial networks (GAN) is a deep learning model that is able to generate new data that has similar properties as data from the training set, the model was first introduced by Ian J. Goodfellow and his team in 2014 [3]. A GAN is made up of two neural networks: the Generator which generates data and a binary classifier called the Discriminator which distinguishes real data from generated data, these two networks train against each other to become best at their respective task. GANs have been used extensively in the field of image generation with applications such as up-scaling in video games [15].

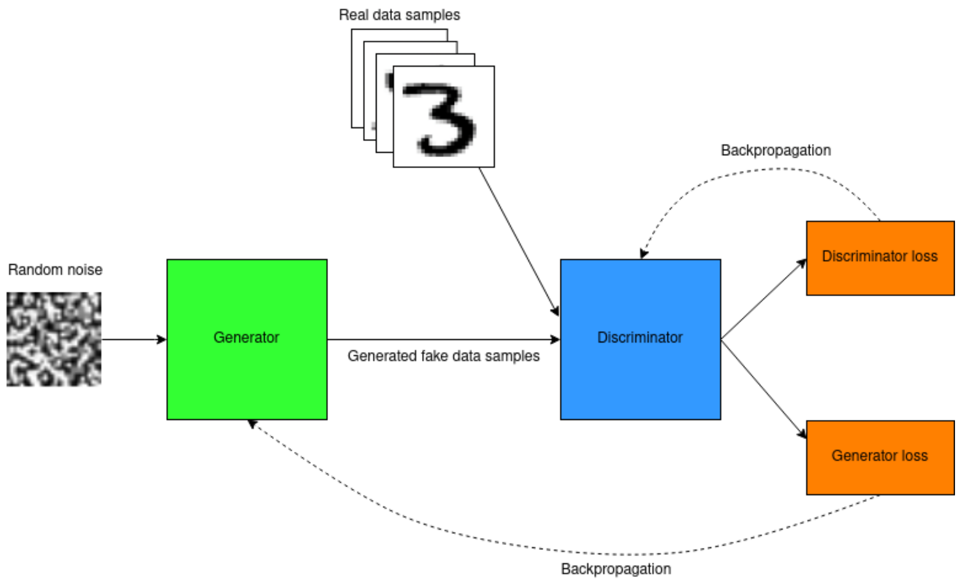


Figure 5: Overview of GAN architecture

2.3.1 GAN Training

When training a GAN the two neural networks are trained independently but interact during the training process.

When training the Discriminator the Generator feeds generated data samples to the discriminator along with real data samples from a training set, this is done randomly so the Discriminator will be unaware from which source the data came from. The discriminator then makes predictions if the images are real or fake and the loss of the Discriminator is then calculated and its weights updated through backpropagation see Figure 5.

The Generator trains in a similar manner by feeding data to the Discriminator along with real samples and from the predictions of the Discriminator the Generator loss is calculated and weights updated while the Discriminator weights are kept static. The difference in the their training is that the Discriminator is trained to maximize loss while the Generator is trained to minimize loss see Equation 1, this forms a two-player minmax game in which the losses are based upon how well the Discriminator makes it's predictions while the Generator loss is based upon how well the generated samples manages to fool the Discriminator [3].

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{z}(\mathbf{z})}[log(1 - D(G(\mathbf{z})))] \quad (1)$$

The minmax setup has though seen difficulties with getting stuck in the early stages of training and the suggested approach is to modify the generator loss to maximize $logD(G(z))$ instead of minimizing $log(1 - D(G(z)))$ [3].

The GAN model reaches an optimal condition when the Discriminator outputs 0.5 for both the real data samples as well as the generated ones meaning it can not tell them apart.

2.4 Conditional Generative Networks

A GAN can be extended to become a conditional model if both the generator and discriminator are conditioned on some extra information **y** [16]. Where **y** could be any extra information such as class labels. Adding this additional input layer (see Figure 6) would enable the Generator when fully trained to produce specific classes of data according to which labels are added to the input layer. This would add some extra control over what is generated compared to the vanilla GAN model where the output would be completely random.

2.5 GAN and IDSs

Many modern IDSs use various machine learning algorithms to detect malicious activity but lack the robustness when they are faced with previously unseen attack types. There are some examples of studies where GAN has been used as a method of generating attack data for the IDS to use as training data. One study [17] used a black-box IDS before the Discriminator in the training loop so that it received the generated data as well as training data and the predicted labels from the IDS where sent to the Discriminator which dynamical learned the IDS structure with the end goal of deceiving the IDS.

2.6 Tools

This subsection is dedicated to the tools used in this thesis. TensorFlow is an open source library for machine learning, originally developed by Google's Machine intelligence Research organization to conduct deep neural network research [18]. TensorFlow provides stable APIs for both Python and C++.

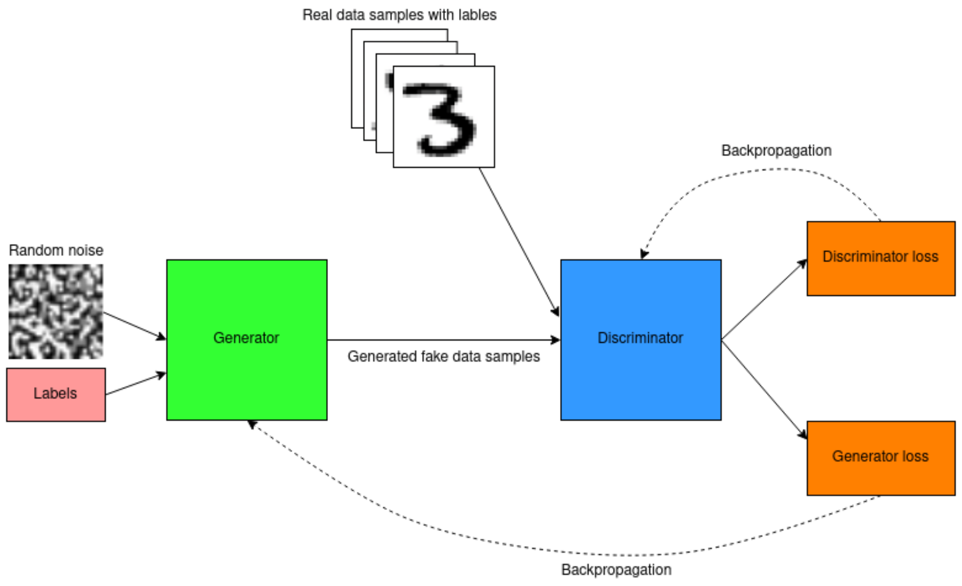


Figure 6: Overview of CGAN architecture

Keras is a deep learning API that runs on top of TensorFlow and acts as an interface to TensorFlow [19].

3 Methods

To be able to explore whether it is possible to generate attack variations with GAN this thesis is going to use the MNIST dataset to evaluate the performance of the model. Where the digit 3 is going to represent the attack data and the rest of the digits are to represent normal network data.

3.1 Attack Data Availability

Modern day IDSs require lots of data in order to learn to distinguish between different network data types and because of fact that attack data is not available in the same quantity as normal network data we will have to reduce the amount of of 3s used in the training data set and train the GAN model with a class imbalance in order to somewhat replicate availability of real world attack data.

3.2 Class imbalance

The MNIST data set consists roughly of about 6000 images from each digit class see Figure 7, in order to replicate real world availability of attack data the amount of available 3s in the training set was reduced see Figure 8. Reducing the attack data too much could cause the Generator to not have enough training samples of the attack class to generate synthetic samples with similar enough properties as of the attack class. While reducing the attack data too little could mean that more data is required than what would be available in a real world scenario. So the optimal class imbalance for the training setup would be to have as little attack data as possible without sacrificing the quality of the generated data. To explore which quantity would be optimal, four different quantities were chosen:

- 20% of all available 3s
- 10% of all available 3s
- 2% of all available 3s
- 1% of all available 3s

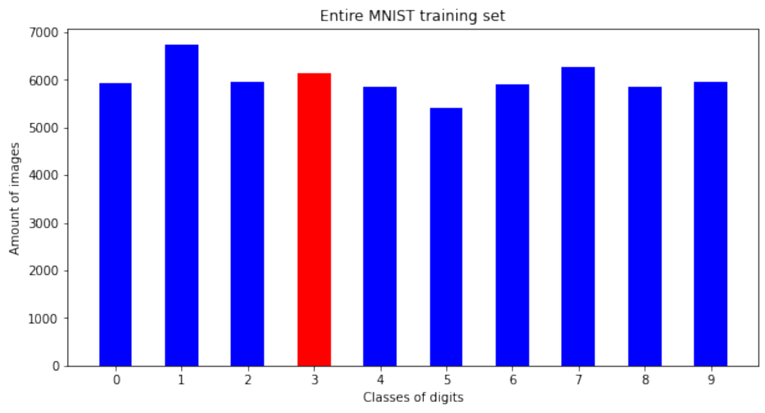


Figure 7: Quantities of each class of the MNIST training set

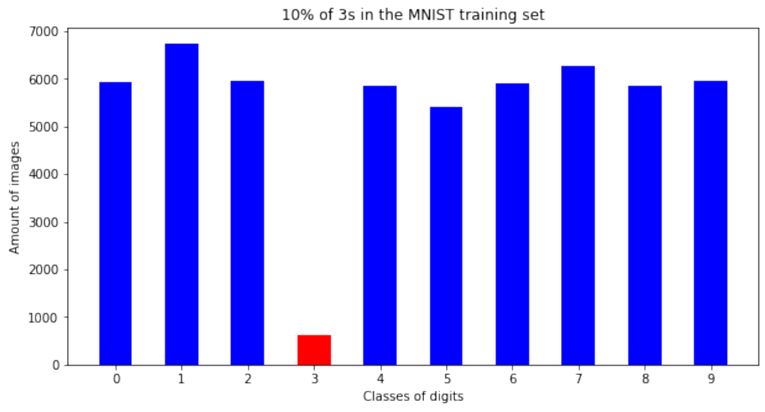


Figure 8: MNIST training set with 3s reduced to 10%

3.3 Limited Classifiers

By training IDSs with limited attack data we could study the impact of the class imbalance on their performance. These IDSs used a convolutional neural network architecture and trained to classify each digit class. All of the classifiers used a class imbalance in their training data with a reduced quantity of attack data (3s). Four limited classifiers (IDSs) were used:

- $L20$ was trained with 20% of 3s
- $L10$ was trained with 10% of 3s
- $L02$ was trained with 2% of 3s

- $L01$ was trained with 1% of 3s

3.4 Oracle Classifier

In order to assure some measurement of quality control of the generated data an additional classifier was trained with the entire MNIST data set i.e all available attack data (3s) see Figure 7. This ensures that the classifier will have a good recognition of all data classes including the attack data (3s) and will be used to see if the imbalanced training data will cause the generator to generate data that is not considered to be part of the intended class. This classifier will henceforth be referred to as the Oracle Classifier.

3.5 CGAN model

To be able to control what type of image class and in what quantity that image class is generated, a conditional GAN model was used. The code base of the CGAN was taken from Keras website [20]. Where all image class labels were taken in as an extra parameter to make the generator aware of which class of images it was producing.

3.6 CGAN Training methods

Below are the methods used for the GAN training.

3.6.1 Traing with a class imbalance

To evaluate how the model would perform with a class imbalance of different levels of 3s. The model was trained with the full data set for 100 epochs and the Generator model was saved, this step was repeated 10 times to get a robust result when evaluating the results. This same process was then repeated but with a reduced number of 3s in the training set to 20%, 10%, 2% and finally 1%. When training was done all the saved Generator models were tested against the Oracle Classifier.

3.6.2 Adding loss of the limited classifiers

In order to extend the GAN training setup to generate adversarial samples, the limited classifier was added to the training loop seen in Figure: 9. As the limited classifiers cross-entropy loss function will output higher values for misclassifications made by the limited classifier and by using a negative

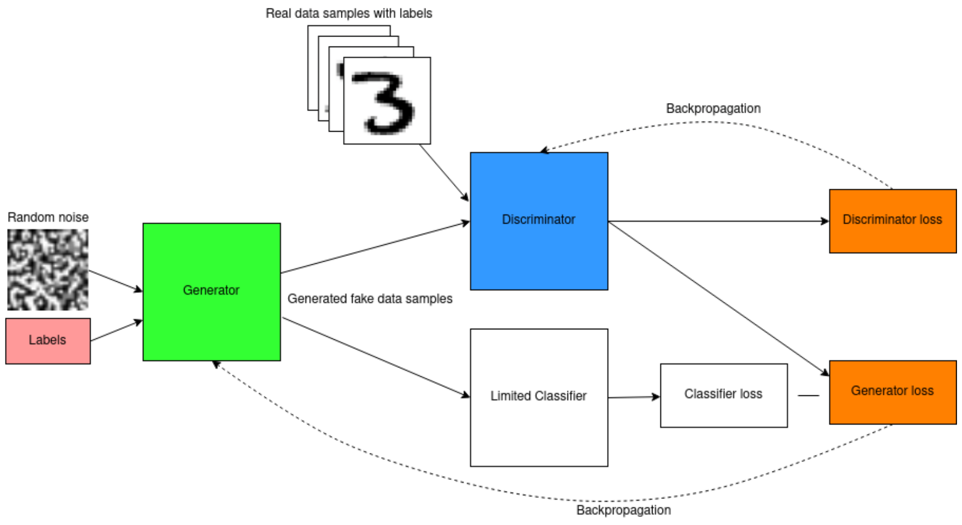


Figure 9: Overview of the CGAN architecture with the classifier loss

sign in the function seen in Equation 2 will bias the generator to cause misclassifications. With this reasoning the opposite should also be true by using a positive sign to bias the generator to generate data that is correctly classified by the limited classifier. Both methods were used in the experiments to evaluate if our reasoning would hold. The weight was added to scale the impact of the added loss as a too large impact could cause the generator to converge to output data with too much perturbation.

Misclassification:

$$GeneratorLoss - (weight \times ClassifierLoss) \tag{2}$$

Data generation:

$$GeneratorLoss + (weight \times ClassifierLoss) \tag{3}$$

3.6.3 Conditioning on the Discriminator

A side effect of adding the limited classifier to the training loop is that it adds a lot of noise to all generated classes, a solution to this would be to use the Discriminator as a filter for the generated images. The motivation for this is that during training of a normal GAN it is expected for the predictions of the Discriminator to get worse as training converges with them ideally being random outputs between the fake and real images. But with the added classifier loss the Generator would not be able to converge ideally which would cause the Discriminator to not converge to output random values but would be able to provide information for filtering samples. Therefore it would be expected for the Discriminator to become better at recognising real from fake samples as the weight of the loss function is increased.

4 Results

This section contains the results of the experiments. First we introduce the result of the initial experiment testing the different class imbalances. Then we move on to test the Generators against the Limited Classifiers as well as the Oracle Classifier with the standard GAN training setup, and finally we introduce the results of the GAN setup with the limited classifier loss included in the GAN training setup with and without conditioning the results on the discriminator.

All experiments were preformed on an NVIDIA GeForce RTX 3080 Ti GPU provided by Uppsala University.

4.1 Comparing class imbalances

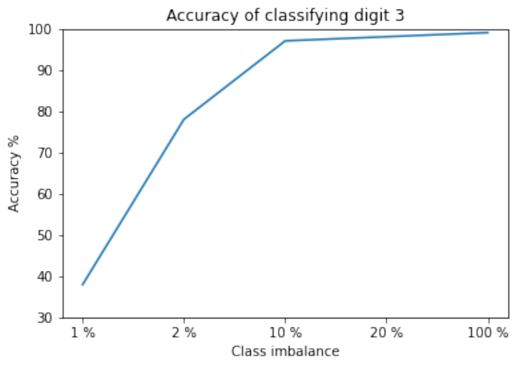
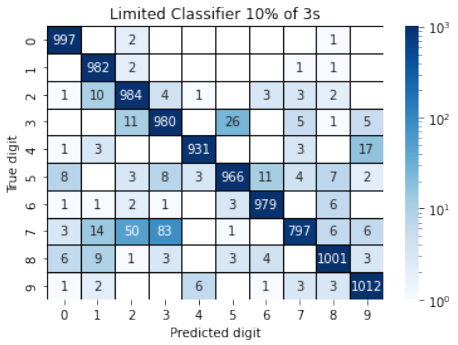


Figure 10: Accuracy of the generators tested against the Oracle Classifier

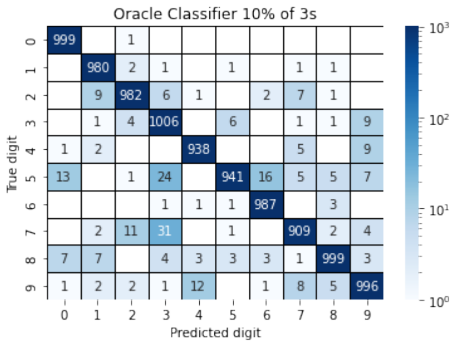
In the initial experiment the GAN model was trained with the class imbalances of: 100%, 20%, 10%, 2% and 1% of 3s where training was repeated 10 times for each training setup. This resulted in 10 Generators from each class imbalance all of which were tested against the Oracle Classifier by generating 1000 3s with each Generator. In Figure 10 we see the results of those experiments. Here we see a very high accuracy in both the 100% Generator as well as the 20% Generator, it is only around the 10% Generator that the accuracy drops. Since the lower class imbalances are more inline with real world data availability the next experiments will focus on the GANs with class imbalances of 10%, 2% and 1%.

4.1.1 Comparing results against classifiers

In the next experiment each of the Generators were tested against the Oracle Classifier as well as their respective Limited Classifier: $L10$, $L02$, $L01$. To illustrate the results a confusion matrix is used with the generated digit on the y-axis and the predicted digit on the x-axis. For the 10% class imbalance GAN in Figure11b it is noted that the Oracle Classifier manages to classify almost all of the generated 3s as 3s while $L10$ (see Figure 11a) also classifies a majority of 3s correctly.



(a) Limited classifier 10% 3s



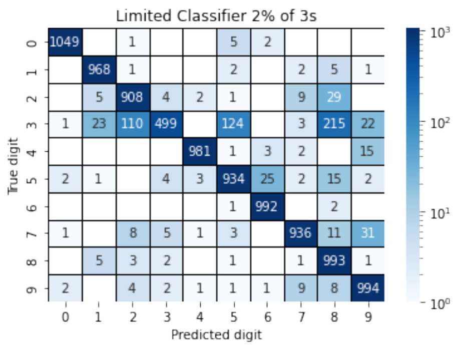
(b) Oracle classifier

Figure 11

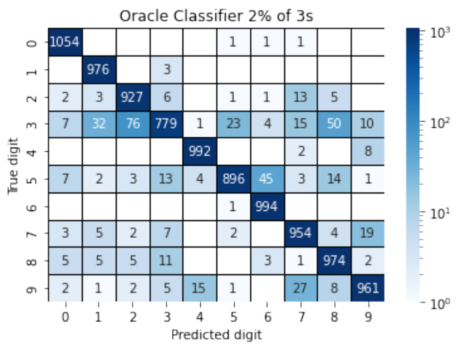
The Generator trained with 2% of 3s the Oracle Classifier classifies about 80% of the 3s correctly which gives a good indication that the quality of the generated samples are high see Figure 12b. While $L02$ is having a higher misclassification rate with only about 50% being classified correctly, the rest are classified as different classes with the majority being 8s.

The 1% generator the Oracle Classifier classifies only about 37% of 3s correctly see Figure 13b. While $L01$ see Figure 13b classifies the majority of generated 3s as other digits see Figure 13a.

The conclusion from this experiment is therefore that the optimal class imbalance for the GAN training setup is the one where 2% of 3s were used. The reasoning for this is that the Generator could still produce a majority of 3s that was usable and to be considered as actual 3s by the Oracle Classifier while still managing to produce 3s that adds enough variation to fool $L02$. Hence the rest of the experiments will only use the class imbalance of 2% of 3s.

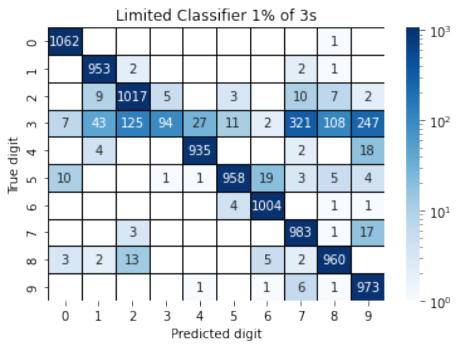


(a) Limited classifier 2% 3s

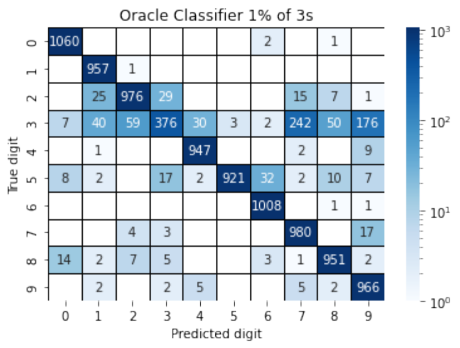


(b) Oracle classifier

Figure 12



(a) Limited classifier 1% 3s



(b) Oracle classifier

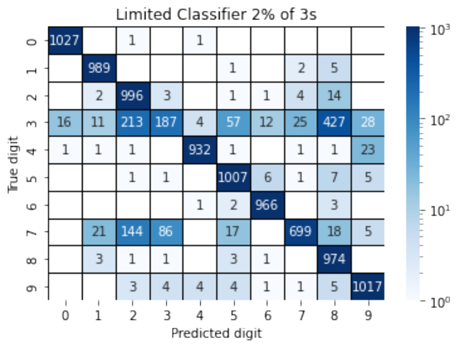
Figure 13

4.2 Weighted Classifier Loss

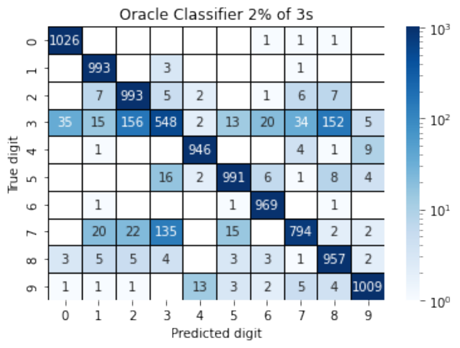
In this section we present the result of the experiments where the Limited 2% classifier (*L*02) was used in the training loop of the GAN see Figure 9. The loss was multiplied by a weight to control how much influence it would have on the loss of the Generator. The weights used in the experiment were: 0.001 and 0.0001 the classifier loss was then tested by using a negative sign see Equation 2 as well as a positive sign see Equation 3.

4.2.1 Negative Classifier loss

The goal of the experiments in this section is to bias the generator into causing *L*02 to misclassify the generated 3s. Initially the weight 0.001 was used and we can note an increase in misclassifications of 3s by *L*02 Figure 14a compared to the previous experiment. A slight increase in misclassifications can also be seen by the Oracle Classifier with 548 classified 3s see Figure 14b.



(a) Limited classifier 2% 3s



(b) Oracle classifier

Figure 14

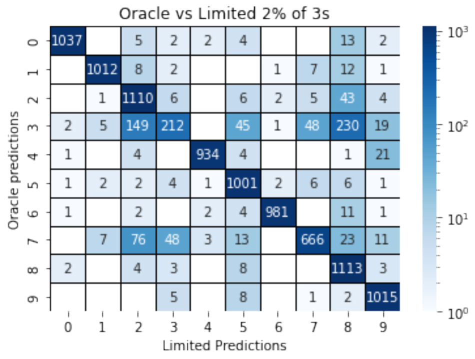
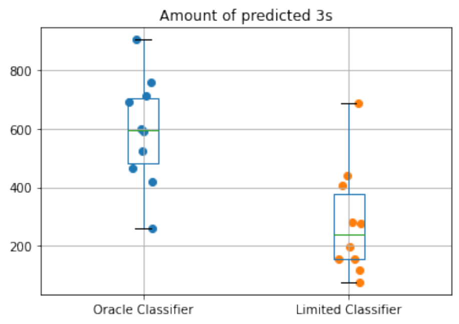
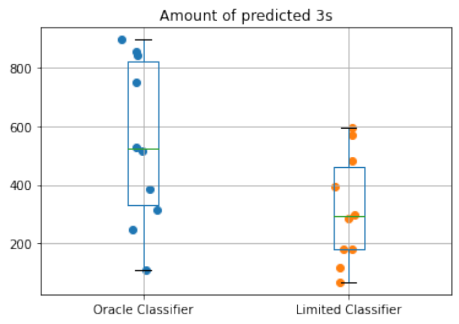


Figure 15: The oracle plotted against the limited classifier

To get a better understanding of how these 548 3s are getting classified by *L*02 a confusion matrix was made with the Oracle Classifier predictions on the y-axis and *L*02 on the x-axis seen in Figure 15. Here it can be noted that the majority of 3s are misclassified as 8s by *L*02 with 230 samples while only 212 are classified in accordance with the predictions of the Oracle Classifier. Looking at the other axis it can also be noted that *L*02 classified 48 samples as 3s which was classified as 7s by Oracle Classifier.



(a) Weight: 0.001



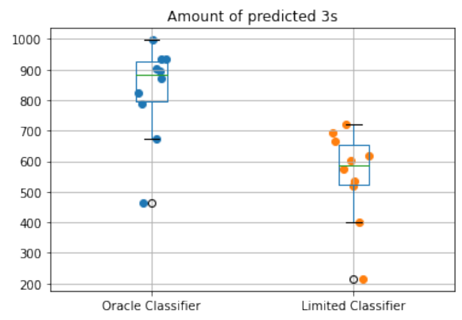
(b) Weight: 0.0001

Figure 16

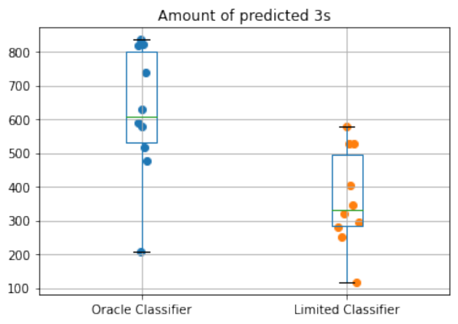
The experiment was repeated 10 times and the amount of predicted 3s were recorded in 16a. The experiments showed a big variance in the consistency of the results where it was difficult to replicate similar classification rates between iterations. The experiments made with the smaller weight of: 0.0001 would in some iterations show similar results as in Figure 15 but with a slightly smaller rate of misclassifications by $L02$. But this model also showed similar inconsistencies between iterations as with the bigger weight see Figure 16b. It can also be noted that the spread of classifications from Oracle Classifier is also large in both cases, especially in the case of the smaller weight in 16b.

4.2.2 Positive Classifier loss

The goal of the experiments in this section is to find out if adding a positive sign in the loss function would bias the generator into generating samples that are correctly classified by $L02$. The results showed a decrease in misclassifications by $L02$ as well as by the Oracle Classifier. From Figure 17a it can also be noted that the amount of recognised 3s have increased in both classifiers, although the experiments using a smaller weight of: 0.0001 Figure 17b have lower rates of recognised 3s compared to those using the 0.001 weight. As misclassifications by $L02$ is something that is preferable in this context, it was decided to only use the negative classifier loss in the next experiments.



(a) Weight: 0.001



(b) Weight: 0.0001

Figure 17

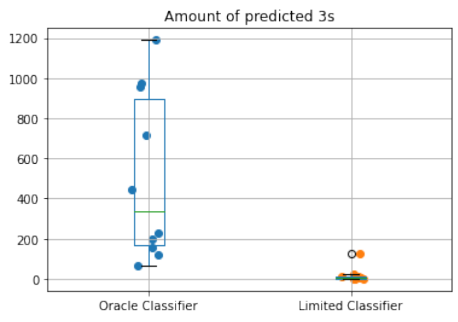
4.3 Conditioning on the Discriminator

The weighted classifier loss would in some instances create lots of noise in the generated data especially when using a larger weight. By feeding the discriminator 10 000 real images as well as 10 000 fake images the discriminator would make predictions on which images were real or fake. By only using the fake images that the Discriminator predicted to be real the idea was to filter out the most noisy data, this was done in every experiment in this section.

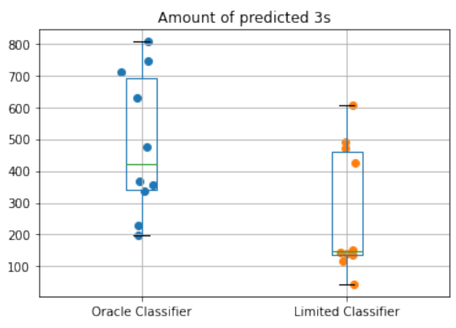
Three different weights were used: 0.0001, 0.001, 0.01 and the smaller weights showed similar results as in the previous experiments Figure 18c and Figure 18b. But in the results from the 0.01 weight Figure 18a it can be noted that the amount of predicted 3s has declined to a consistent low level for $L02$ previous unseen in the past experiments while the Oracle Classifiers prediction rates are more inline with the previous results.

The confusion matrix in Figure 19a where the large weight of 0.01 was used shows a significant reduction in the classification of 3s for $L02$ where only 6 data samples are classified in accordance with the Oracle Classifier with a big majority getting classified as 8s. It can also be noted that 1998 samples are classified as 6s while only 76 samples are classified as 9s indicating that the large weight might cause perturbation not only in the attack data class but in other classes as well.

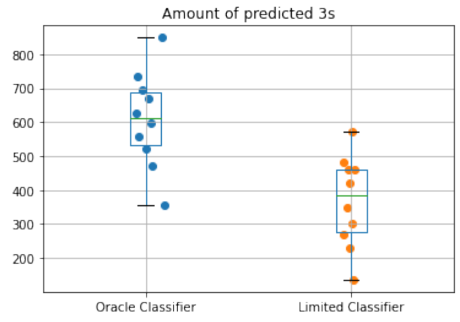
After conditioning the generated data on the Discriminator in Figure 19b we see that about 50% of the data have been filtered by the Discriminator



(a) Weight: 0.01

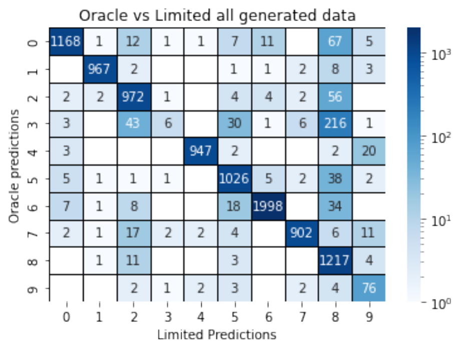


(b) Weight: 0.001

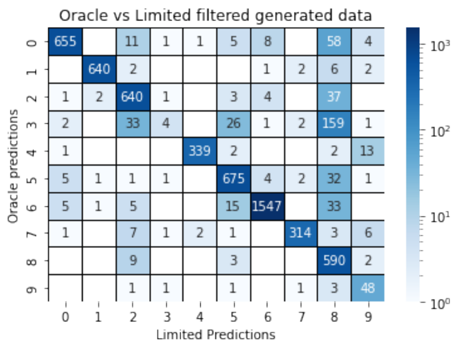


(c) Weight: 0.0001

Figure 18: Filtered generated data



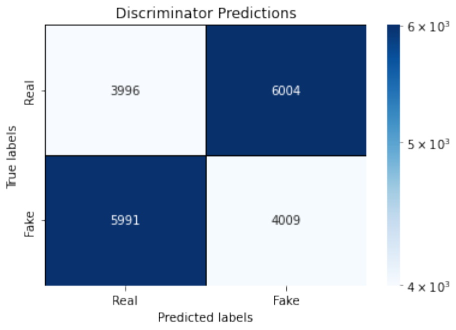
(a) All generated data



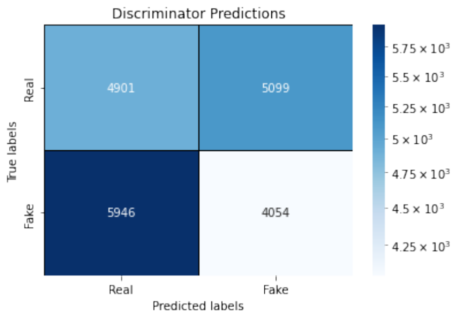
(b) Filtered generated data

Figure 19

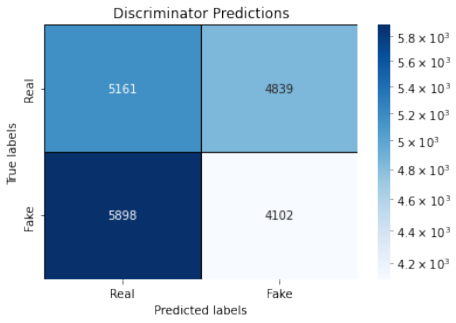
in most classes with some exceptions, the amount of 6s for instance are still very high with 1547 samples. We also still see a majority of misclassifications by $L02$ with only 4 data samples being classified in accordance with the Oracle Classifier. To put this amount in relation to the Oracles predictions of 228 3s only about 2% of the samples are classified in accordance with the Oracle Classifier by $L02$.



(a) Weight: 0.01



(b) Weight: 0.001



(c) Weight: 0.0001

Figure 20: Filtered generated data

The classifications of the discriminator were also plotted to get a visualization of its predictions and from Figure 20a it is seen that the Discriminator training seem to have converged as it normally would with a normal GAN training setup as the predictions between real and fake samples are close to a 50/50 split. This pattern was also noted in the experiments with smaller weights seen in Figures 20b and 20c contradicting our expectations.

5 Discussion

5.1 Answering initial question

The initial premise of this thesis was to explore if GAN could be used to generate attack data that could be used as training data for an IDS. That question has only been partly answered as the experiments have shown that some of the GAN models were able to generate data that could not be classified correctly by an IDS (*L*02) but was still considered to be part of the attack data class. This has shown that it can be possible to generate data that would be able fool an IDS. The next step would be to use the generated data to train an IDS and to see if the robustness of its predictions would increase. This could have been done during this thesis but due to time constraints it was not.

5.2 Consistency in results

The results from the experiments were very varied and it was difficult to get consistent results from the models that were trained. There are many improvements that can be made to the model used in the experiments; the usage of a Wasserstein GAN instead of a vanilla GAN could be one of those improvements. A Wasserstein GAN is an improved GAN model with improved training stability which addresses issues such as mode collapse where the Generator maps several inputs to the same output as an example [21]. Therefore the usage of a Wasserstein GAN could possibly also address the issues seen in this thesis with inconsistent results and noise in generated data.

5.3 Conditioning on discriminator

The idea of using the Discriminator as a filter was that due to that it was expected for the Generator not to converge ideally during training with the introduced classifier loss and that the same would follow with the Discriminator as their progress in training is dependent on one another. In the experiments of this thesis this showed not to be the case as the Discriminator showed a seemingly random output even when the weight was increased. This might have been due to restrictions in the particular GAN training setup that was used in the thesis but due to time constraints could not be investigated further.

5.4 Modifying model to use network data

As some of the results have shown the data generation was successful and in some cases very good at deceiving the IDS (*L*02) this would have been very interesting to see in a real environment using network data instead of images. The obvious next step apart from improving the GAN model would therefore be to modify it to use real network data to see if the results could be replicated and hopefully improved.

References

[1] T. Seals, "IoT Attacks Skyrocket, Doubling in 6 Months," 2021. [Online]. Available: https://threatpost.com/iot-attacks-doubling/169224/

[2] A. S. Dina and D. Manivannan, "Intrusion detection based on Machine Learning techniques in computer networks," in *Internet of Things*, vol. 16, 2021. [Online]. Available: https://doi.org/10.1016/j.iot.2021.100462

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[4] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," *2017 IEEE Symposium on Security and Privacy (SP)*, 2017. [Online]. Available: https://doi.org/10.1109/SP.2017.49

[5] B. Lutkevich, "Intrusion detection system," 2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/intrusion-detection-system

[6] N-able, "Intrusion Detection System (IDS): Signature vs. Anomaly-Based," 2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/intrusion-detection-system

[7] A. Shenfield, D. Day, and A. Ayesh, "Intelligent intrusion detection systems using artificial neural networks," *ICT Express*, vol. 4, 2018. [Online]. Available: https://doi.org/10.1016/j.icte.2018.04.003

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014. [Online]. Available: https://doi.org/10.48550/arXiv.1312.6199

[9] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, 2019. [Online]. Available: https://doi.org/10.1109/TNNLS.2018.2886017

[10] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Pearson, 2022, ch. 22.0.

[11] ——, *Artificial Intelligence: A Modern Approach.* Pearson, 2022, ch. 22.1.

[12] M. Nielsen, "Neural Networks and Deep Learning," 2019. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap1.html#a_simple_network_to_classify_handwritten_digits

[13] "MNIST dataset," 2019-09-22. [Online]. Available: https://deepai.org/dataset/mnist

[14] M. Alhamid, "Generative Adversarial Networks GANs: A Beginner's Guide," 2020-07-18. [Online]. Available: https://towardsdatascience.com/generative-adversarial-networks-gans-a-beginners-guide-f37c9f3b7817

[15] T. Thompson, "How AI upscaling can help remaster game art," 2021. [Online]. Available: https://www.gamedeveloper.com/art/how-ai-upscaling-can-help-remaster-game-art

[16] S. Dobilas, "cGAN: Conditional Generative Adversarial Network — How to Gain Control Over GAN Outputs," 2022-08-01. [Online]. Available: https://towardsdatascience.com/cgan-conditional-generative-adversarial-network-how-to-gain-control-over-gan-outputs-b30620bd

[17] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection," *Advances in Knowledge Discovery and Data Mining. PAKDD 2022. Lecture Notes in Computer Science*, vol. 13282, 2022. [Online]. Available: https://doi.org/10.1007/978-3-031-05981-0_7

[18] "Tensorflow." [Online]. Available: https://github.com/tensorflow/tensorflow

[19] "About Keras." [Online]. Available: https://keras.io/about/

[20] S. Paul, "Conditional GAN," 2021-07-15. [Online]. Available: https://keras.io/examples/generative/conditional_gan/

[21] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017. [Online]. Available: https://proceedings.mlr.press/v70/arjovsky17a.html