City, University of London

MSc in Cyber Security

project report

2021 - 2022

Using Generative Adversarial Networks to improve malware detection in the IoT focused intrusion detection dataset IoT-23

Cameron Swart

Supervised by: Nikos Komninos

Declaration

Abstract

Machine learning systems were often faced with the issue of imbalanced datasets. A possible method of tackling this issue came about with the introduction of Generative Adversarial Networks by Ian Goodfellow *et al* (2021) in which they introduced a method where two neural networks compete against one another in a zero sum game: a Generator *G* attempts to 'fool' the Discriminator *D* by creating synthetic data that is capable of being identified as real by *D*, each time the generator loses, its weights are updated and thus after a number of rounds of this game the generator can hopefully generate data that the discriminator identifies as real; and at this point we have a generator of synthetic data. This project explores the use of that synthetic data to augment a class imbalanced training dataset IoT-23 to improve unseen data classification performance of an artificial neural network-based IDS. This system will contribute to the relatively small but growing field of research around network data generation. The results of the project show promise, with an overall improvement of F1 score of 9% when compared to the standalone IDS results and a steadily increasing individual class F1 results over the course of all scenarios tested, with the biggest improvement in individual F1 score being 34% for C&C heartbeat.

Tabel of Contents

Chapter 1: Introduction and objectives

1.1. Introduction

The question, ‘will a system be breached?’ has been replaced by the statement ‘when the system is breached’. This switch in mindset towards Cybersecurity marked a change in tactics for both the defender and the attacker: it is a game of cat and mouse where the adversary is constantly evolving their attack parameters and the defender is constantly blocking attacks. Simply put, an attacker only needs to be lucky once whereas the defender needs to lucky every time**.** SonicWall reports that IoT targeted malware rose by 228% in the US and 134% in the UK across all industry sectors, thus the importance of the systems tasked with defending critical infrastructure needing to constantly evolve within the threat landscape. Machine learning defense solutions have been on a steady rise over the past two decades with the pace of research exploding in the past decade, however an issue that constantly arises is extreme imbalance.

One solution to this is the recent introduction of Generative Adversarial networks (GAN) where two models compete against one another with one attempting to fool the other, this game is used to generate synthetic data. My project is an adoption of this game where I task the GAN to produce synthetic network flow data with the objective of using that synthetically generated data to augment the training portion of the IoT-23 dataset that will be used to train an artificial neural network with the aim of improving the model’s ability to identify minority samples in the test set. Based on this, my research question is: “Can synthetically generated samples from a GAN be used to overcome sample imbalance issues in a modern IoT IDS dataset”. The results of my research will be classifications that highlight the improvements or the drawback of implementing synthetically generated samples. I expect some improvements in the IDS’s ability to detect the unseen test set.

1.2. Aim

The aim of this project is to determine whether a simple generative adversarial network can generate quality synthetic examples of real network data to improve the classification performance of an IDS.

1.3. Objectives

The research question I chose to answer is the following: **Can synthetically generated samples from a GAN be used to overcome sample imbalance issues in a modern IoT IDS dataset?**

I aim to complete the following objectives over the course of my research:

- Review the literature relating to generating purely tabular data through generative adversarial networks
- Develop a framework were training data can be synthetically augmented using a generative adversarial network
- Use the GAN architecture to produce synthetic data samples that improve the detection performance of a classifier
- Identify limitations of using synthetic data within a multiclass classification scenario

1.4. Beneficiaries

The application of GAN stretches to an array of different fields: The main beneficiaries of this project are researchers exploring the potential use of GAN, The majority of GAN research is currently focused on image generative. My project will add to the finite pool of tabular data GAN research. Additional beneficiaries include researchers/industry working with models affected by class imbalance.

Secondary beneficiaries include:

- Machine learning based intrusion detection system researchers.
- Researchers hoping to share privacy sensitive datasets to other researchers without incurring privacy restrictions.
- Companies looking for ways to sell privacy sensitive data
- Me - machine learning and this area of data science is of great interest to me This project was an enjoyable learning experience.

1.5. Deliverables

- The first deliverable will be the findings of the research being the applicability of GAN generated data being usable for training data augmentation.
- Add to the limited pool of research surrounding the use of GANs on non-graphical implementations.
- Providing a framework/method of using privacy sensitive data / distribution of privacy sensitive data e.g., personal ad data without the infringement of personal data protection laws.

1.6. Report structure

Chapter 1 – Introduction, here a summary of the project including aims, objectives, and planned outcomes

Chapter 2 – Context, a look at the relating research surrounding the project's keys areas

Chapter 3 – Methods, the code and function of the overall program is explained in this section with explanations of the evaluation methods

Chapter 4 – Results, here the results of the different testing scenarios are explored and possible reasons for results are explained

Chapter 5 – Discussion, in this section I look back on the objectives and review their success against the outcome of the project.

Chapter 6 - Evaluation, Reflections and Conclusions, the project is evaluated by section and experience is explained.

Chapter 2: Context

This chapter contains literature relevant to the project, it is organized in the following sections.

- Generative adversarial networks
- Class imbalance
- GAN generated synthetic data
- IoT-23 dataset
- Literature review conclusion

2.1. Intrusion detection systems

Intrusion detection systems are rooted in the signature detection approach, through the continual upkeep of vast archives of known transgressions. This relies on matching the observed network behavior with known patterns (Pawlicki, *et al* 2022). Much work in this field makes use of the well-known KDDcup99 [2] dataset as illustrated in the paper by Kunal et al (2019). The popularity of the dataset however has a few shortcomings. Eugen (2010) describes how extreme class imbalances in the dataset poses a significant challenge to machine learning.

A large portion of research that applies deep learning detection methods is focused around the KDDcup99 and NSL KDD datasets. These are often used as benchmark examples in papers that explore different kinds of detection algorithms. The KDD99 dataset contains simulated attacks that fall into one of four categories: (1) DOS which relates to attacks such as SYN flood and ICMP flood; (2) R2L which relates to methods of attack involving unauthorized access from a remote machine such as credential stuffing; (3) U2R which relates to unauthorized access to local superuser (root) privileges through methods such as 'buffer overflow' attacks; and (4) Probing which relates to reconnaissance and surveillance techniques such as port scanning (Gaurav Meena, Ravi Raj Choudary, 2017). A number of issues however stem from the dataset. An issue is the number of duplicate records. It has been found that about 78% of the train set and 75% of the test set contain duplicate records. The NSL KDD dataset was created using select records from the KDDcup99 dataset to improve on its many issues (Sarika Choudharu, Nishtha Kesswani, 2020). While these datasets are used as benchmarks in a large majority of deep learning IDS papers, (Yin *et al*, 2017; Staudemeyer 2015*;* Bhupendra Ingre, Anamika Yadav, 2015) it is widely agreed that these benchmark datasets do not reflect the modern malware types that are mostly used in today's world due to the lack of modern low footprint type attacks and lack of modern normal traffic scenarios (Nour Moustafa, 2016).

2.2.Generative Adversarial Networks (GANs)

A GAN is a relatively new technique in machine learning, designed in 2014 by Ian Goodfellow *et al*, it works by having two neural networks compete in a game where the objective is to beat one another. A GAN is made up of two components, a generator whose job is to "generate" synthetic examples of data to fool the second component, the discriminator which needs to be able to distinguish the difference between the synthetic data and the real data, this process is repeated until the generator is able to "beat" the discriminator.

The process behind GANs is the zero-sum game played between two players, the generator model takes a random input vector and generates a sample, the discriminator is then fed combinations of real samples from the dataset and synthetic samples from the generator, the discriminator must then say whether a sample is fake or not, both models trying to outsmart the other and "win" the game. For every success the discriminator has, the weights of the generator are adjusted to edge closer to creating better synthetic samples and the discriminator's weights are adjusted to better spot the synthetic samples (Ian Goodfellow *et al* 2014).

Since the original paper, there have been many alterations of the GAN. One example is a DuelGan mode, introduced by Wei *et al* (2021) where, alongside *G* there is *D1* and *D2,* while the generator performs its usual operation and competes with *D*. Authors introduced a further game between *D1* and *D2* where both models are encouraged to disagree with one another as to the type of data being observed. Results show that this method alleviates the issue of early mode collapse by preventing the two models from converging too fast. This method increases the diversity of generated samples as the generator is forced to work harder to appease both *D1* and *D2*.

2.3.Class imbalance

The issue of class imbalance within datasets refers to the difference between number of class samples within the dataset. A class imbalance within a dataset can greatly affect classification and, in effect, deteriorate model performance (Junjie Hu *et al* 2018). Research focused on class imbalance mainly seeks to solve this issue through manipulation of class distributions (Guo *et al 2008)*. This method has stood the test of time, showing that both over-sampling the minority class and under sampling the majority class were very effective methods of dealing with the problem (Nathalie Japkowicz 2018).

Although it is noted that simply under sampling the majority class can lead to a loss of useful information by removing significant patterns (Shaza *et al* 2013).

Previous methods have been devised to alleviate the problem of class imbalance, studies using methods such as over-sampling and under-sampling. Techniques such as Synthetic Minority Oversampling Technique (SMOTE), which combines over and under sampling, locates data close to the input minority class and randomly sample data from within that range (JooHwa Lee & KeeHyun Park 2019)

2.4. GAN generated synthetic data

GAN generated data has already shown a lot of promise; a study by JooHwa Lee & KeeHyun Park (2021) presented a GAN-RandomForest system. Their research compared the classification results of the GAN generated data on a random forest classifier to SMOTE generated data on a RF classifier They found that evaluation data from the GAN-RF out-performed the RF operating on its own. They also found that the GAN-RF out-performed SMOTE-RF when evaluating imbalanced class classification (JooHwa Lee & KeeHyun 2021).

Another implementation used a system similar to that in the previously mentioned paper where authors used a GAN to augment several banking-based datasets. Their study again used a RandomForest model as the baseline classifier and a GAN to bring the data distribution up so to balance out the dataset. In this paper, the generated data was used to augment the entire dataset including the testing set. Their results saw an improved classification of the original minority samples in all three datasets tested with an overall improvement in F1 scores of 15%, 8% and 5% (Shu *et al*, 2020).

Work by Shahrair *et al* (2020) demonstrated that live network traffic capture for IDS training can be further improved through GAN generated data; the IDS performed better when the synthetic data was combined with the captured network traffic. Results from their research showed improved classification scores for all targeted classes, with the largest increase being 25% better FF1, and the lowest improvement was 8%. However, in their further testing they found that their proposed G-IDS was computationally expensive and time consuming requiring a smaller dataset for training (Shahrair *et al* 2020).

Research into botnet detection using GAN architecture to enhance the trained model's performance caried out by Yin *et al* (2018) made the observation that their GAN model, when used to augment the original detection model was able to detect a class of botnet that was not in the training data and only existed in the test set. This method improved detection performance and decreased the false positive rate of the detection model (Yin *et al* 2018).

Medical researchers Diamant *et al* (2018) use GAN generated synthetic data to improve CNN classification performance in the detection of liver lesions; their research indicated that the generated images used to augment the dataset increased model specificity by 4% and sensitivity by 7% (Diamant *et al* 2018).

A side benefit of the GAN generation process is the ability to anonymise privacy sensitive data, Yoon *et al* (2020) developed a framework to generate synthetic data to minimize re-identification, while used in a medical image generative setting the process can be applied to personal information sensitive IDS datasets (Yoon *et al* 2020).

2.5. IoT-23 Dataset

A relatively new dataset that comprises a benign network traffic from three real IoT devices Somfy door lock, Philips hue and an Amazon Echo and various malware scenarios captured using a Raspberry Pi. There have been a few papers documenting different ML methods to classify malware. In the paper by Nicolas-Alin Stoian (2020) five different methods were compared with random forest (RF) having the best classification performance of 99.5% testing accuracy. Of the methods used, a neural network was implemented in their paper with an accuracy and F1 score of 0.66 and 0.52. The dataset is used as is without extensive under-sampling or data manipulation outside of standard cleaning and column dropping. Of the columns dropped the paper states that ts, uid, id.orig_h, local_orig, local_resp, missed bytes, tunnel parents were dropped due to missing data and lack of correlation to the target variable i.e., “label”.

The IoT-23 dataset builds on the drawbacks of previous popular datasets, it introduces modern versions of malware that are specifically targeted toward IoT devices, Shahid *et al* (2020) note the added benefit of real IoT devices being used for the benign samples rather than using simulated normal network data.

While classifying intrusions into their respective categories provides invaluable */information of a possible attack taking place it is also impetrative to simply have knowledge of an attack taking place. In the paper by Bains *et al* (2021) authors converted each malware capture into a binary format, 0 represented benign (normal) traffic and 1 represented an attack. Compared to other papers Bains *et al* (2020) there was no under-sampling used instead authors ran each segment of malware capture as an individual instance giving them 20 separate classification results. For detection, four different ML models were used, (1) decision tree, (2) logistic regression, (3) random forest classifier and (4) an artificial neural network, given the task of binary classification the NN achieved a accuracy and F1 rating of 99.9% and 99.9% respectively.

The restructuring of the IoT-23 dataset is documented by Strecker *et al* (2020) where researchers optimized the feature set specifically for their respective ML models. They found that the tested models (RF, KNN, SVM) all used the same base set of features: uid, resp_ip_bytes, resp_pkts, orig_ip_pkts, history, resp_bytes, and duration however it was noted that RF malware classification worked better with additional features: conn_state, proto, orig_bytes.

2.6. Literature review conclusion

To summarise, the studies observed that incorporate GAN's have for the most part been focused on image generation. Of the GAN's that generate synthetic tabular data, JooHwa Lee & KeeHyun Park (2021) demonstrated that a RF classifier had a 7% improvement in F1-score and Shahrair *et al* (2020) implemented a G-IDS system where a hybrid dataset was tested incrementally upon their NN based IDS system. Notes from this paper to take-away are that incorporating a GAN generated sample into an IDS targeted dataset does improve test prediction rates. I plan on a similar method used in this study, their training data was broken up into 20%, 40%, 60%, 80% and 100% sections. This gave feedback on a GAN performance with regards to a growing dataset, in my case I will be using a similar distribution of the target samples to train the GAN. I will expand on this decision in the methods section of this paper.

Moreover, an added benefit of a GAN approach to generating synthetic data samples for a privacy sensitive type of dataset is that it is proven to be capable of generating non-personally identifiable information (non-PII). Yoon *et al* (2020) used a GAN framework to anonymise privacy sensitive medical data. While their study generated medial imagery the process behind the generation is the same if used for tabular data. However this kind of assumption is still up for debate as noted in a paper by Bridgland *et al* (2019) where authors generated tabular medical records and trucking data from original records. They point out that only a very small amount of data would be needed to make it personally identifiable so while image generation in a medical sense would be capable of a higher degree of privacy when it comes to tabular data, there may need to be further methods implemented upon the generated data to ensure that it does not contain personally identifiable information.

Chapter 3: Methods

This chapter contains the methods used throughout the project. As a note there will be screenshots of code. While most of the code was written inside the juypter notebook I have placed the code into the pycharm IDE so to gather screenshots with line numbers which will make descriptions easier to follow. This section of the paper is structured as follows

- **Research strategy**
- **System overview**
- **Environment setup**
    - Hardware
    - Software
- **Dataset**
    - IoT-23
    - Pre-processing
    - Dataset exploration
    - Feature reduction
- **Intrusion detection system**
- **Generative adversarial network**
    - Generator
    - Discriminator
    - GAN training
- **Evaluation methods**

3.1.Research strategy

The research in this project hopes to provide a strategy for synthetic data generation that has real world applicability, therefore I followed the design and build (D/B) or design and create strategy to develop the overall methods used within the project. Throughout the project I followed a modified Waterfall project management methodology called incremental build model, the incremental build model methodology in simple terms is a structured approach where each stage of a project is broken down into small sections to be designed implemented and tested incrementally. As with the waterfall methodology, this means each stage of development cannot start until the previous has been completed. This is applied in my project as it was broken down into stages that must be completed before the following stages could begin. This methodology has its shortcomings as described by Ganis (2012); projects following a waterfall methodology suffer from section "freezing" where returning to a previously completed section for adjustments can be time consuming and difficult due to the nature of development. This is an issue I encountered during my project, often as I understood sections better, I realised that adjustments needed to be made to sections completed earlier on in the project which was difficult due to the nature of development causing later sections to be “frozen”, as I could not continue until a solution was in place and this often caused a ripple effect of problems down the line. During the project, however, I become better at making modular functions that I could quickly implement and test without too much modification to earlier sections of code.

3.2. Environment setup

3.2.1. Hardware

I will be using my own desktop for training, testing and evaluation of the models, the specs are as follows.

Intel Core i7 11700k (running at 4.6Ghz)

32gb RAM

RTX 3070ti

3.2.2. Software

All the code for my project will be written using python within the juypter notebook IDE, the models will utilize the Keras machine learning library and several other libraries noted in table 1.

Library	Use
Pandas	Dataset manipulation, data structures
Numpy	Mathematical operations on arrays
Matplotlib	Creating visualisations of model performance
Scikit-learn	Scaling, feature selection, evaluation metrics, label encoding
Keras	Deep learning library

Table 1 - Main python libraries used

3.3. Program overview

With regards to the objective of this project, I have set out to design and build a program where an imbalanced IDS dataset can be augmented with synthetic data samples generated by a GAN. To test the quality of the data once it has been generated the data is placed back into the training set and the IDS is trained on the augmented dataset with the aim of improving classification performance of poorly classified samples due to class imbalance. The layout of the program and the order of processing can be seen in figure XX where firstly the IDS module will be trained on 80% of the original IoT-23 dataset and once the model is fully trained it is tested on the remaining 20% of the data. This unseen data will test how well the model learned to recognise the classes present in the training set. Once testing has been completed we will have the classification results, these will be checked and any underperforming classes that fall below the 0.70 F1 score requirement will be selected for synthetic augmentation through GAN oversampling. There, the GAN will be subject to different percentages of the training data to assess the algorithms performance on a growing sample size; the different sizes are the proportions used by Shahrair *et al* (2020) where the data will be segmented into sizes of 20%, 40%, 60%, 80% and 100%. Where their paper differs from my own project is the training and testing method; firstly their method of testing the quality of the data after every GAN training iteration by feeding the data into a hybrid dataset for classification was hugely time consuming as the many early training epochs resulted in poor quality data whereas the system I have used in my project saves time by training the GAN on all classes independently of one another and tests the quality of the data once at the end of training rather than repeatedly throughout training. A possible drawback to this is unlike the paper by Shahrair *et al* (2020). I won't be able to identify the individual samples that do not improve classification performance as I am not testing each batch of synthetic data for its quality of representation towards the class it is synthetically representing.

The code for the GAN I have used in this project is a converted GAN for image generation that is from a blog post written by Chris Nicholson (2020). It was adapted for tabular data generation by a GitHub user Paakki (2020). I have placed a link for the referenced repository in the referencing section, and the sections of code I have written myself will be highlighted when it comes to the explanation of them. There has been very little literature that explores tabular data generation, Lei Xu & Kalyan Veeramachaneni (2020) propose a system where data is generated using a long, short term memory network (LSTM) and the discriminator is a MLP, their target dataset consisted of a variety of datatypes, this complex mix of datatypes such as continuous and categorical data required the generator to be constructed from a LSTM. Further work based their discriminator off the architecture used by Xu *et al* (2020). However, rather than using a LSTM as the generator, I instead opted for a second MLP as the generator module, as the data type distribution is like the distribution of the IoT-23 dataset. I opted for the MLP for both *D* and *G*. The overall structure of the system is shown in figure 1.

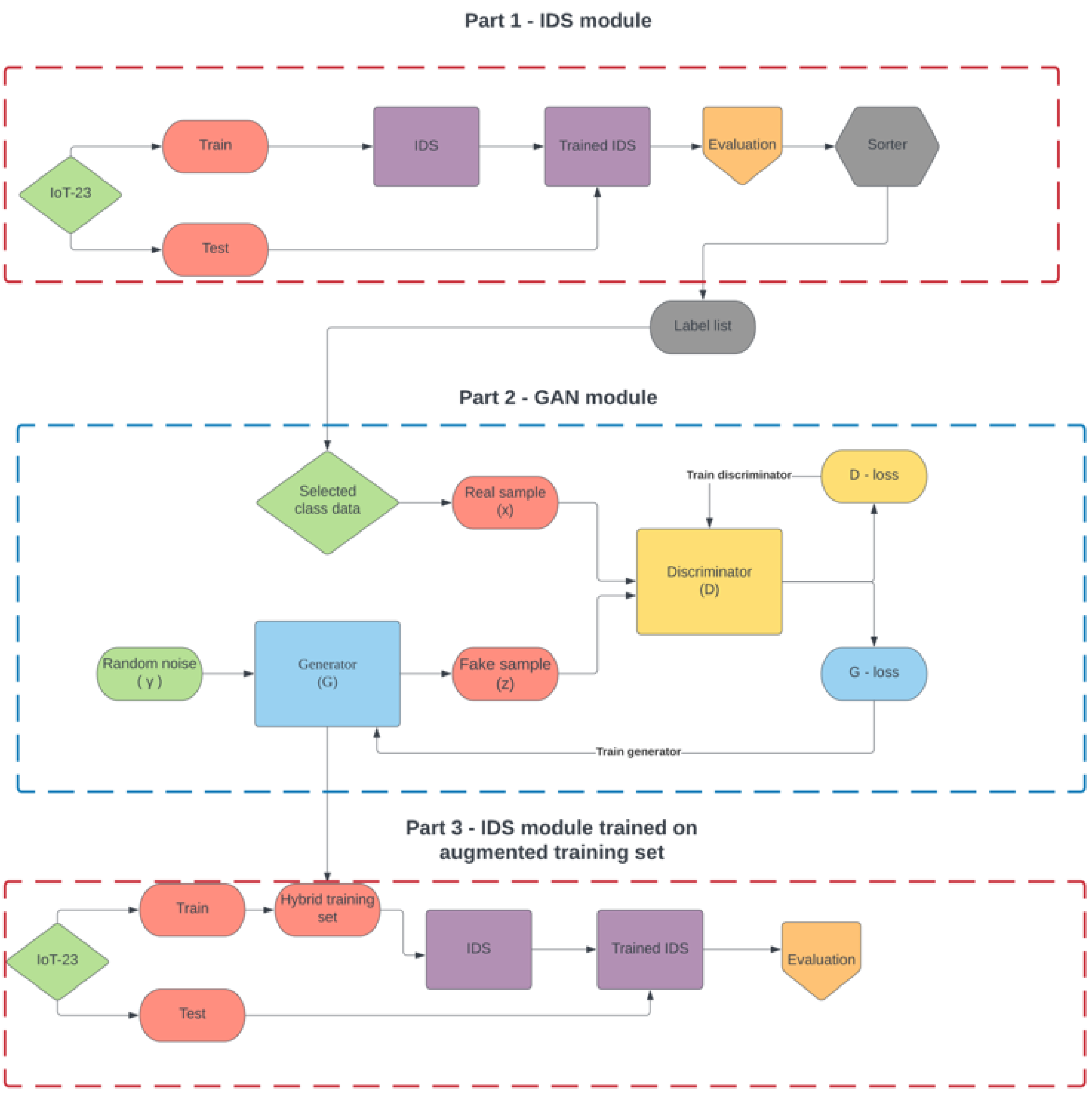


Figure 1 - Program diagram

3.4. IoT-23

The dataset I have used in my project is the IoT-23, created by Sebastian Garcia *et al* (2020) at Avast AIC laboratory. It represents a shift in mindset towards trying to understand the behaviour of IoT targeted malware. Published in January 2020 it is a relatively new dataset which is still being studied and explored. Consisting of twenty-three captures of different IoT traffic, malicious captures consist of twenty captures with 16 malware classes. Benign traffic is from three real IoT devices: a Philips HUE smart LED lamp, an Amazon echo and a Somfy smart door lock. This is significant because rather than simulated regular network traffic in this dataset, the network captures are from the actual devices. Once imported and prior to any cleaning or feature reduction, the dataset consists of 10400775 rows and 21 columns.

Exploration

The dataset contains 20 different malware captures and has 16 different classes of malware, while some malware labels can be represented by the same class for example C&C heartbeat this just means that both attacks are present in the same class, table XX describes the labels.

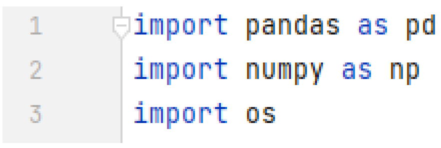
Attack type	**Description**
Malicious Attack	An attack from an infected device to another host where it attempts to exploit a vulnerability
Benign	Regular network traffic between network connected devices
C&C	Infected device is connected to a command-and-control server
DDoS	A distributed denial of service attack is being executed by an infected device
C&C File Download	A file is being downloaded to an infected device from a C&C server
C&C Heartbeat	Packets sent from are being used to keep a track on the infected host by the C&C server
File Download	A file is being downloaded to an infected device
PartOfAHorizontalPortScan	The connections are used to perform a horizontal port scan to gather information to perform further attacks
Okiru	The connections have characteristics of a Okiru botnet
C&C Torii	The connections have characteristics of a Torri botnet
C&C-HeartBeat-FileDownload	Similar to C&C heartbeat but instead of packets a file is being sent from the infected source to the C&C
Okiru-Attack	The attack is recognised as belonging to the Okiru family, but the method of attack is difficult to identify

C&C-Mirai	Attack performed by a Mirai botnet

Table 2 - Label descriptions

Cleaning

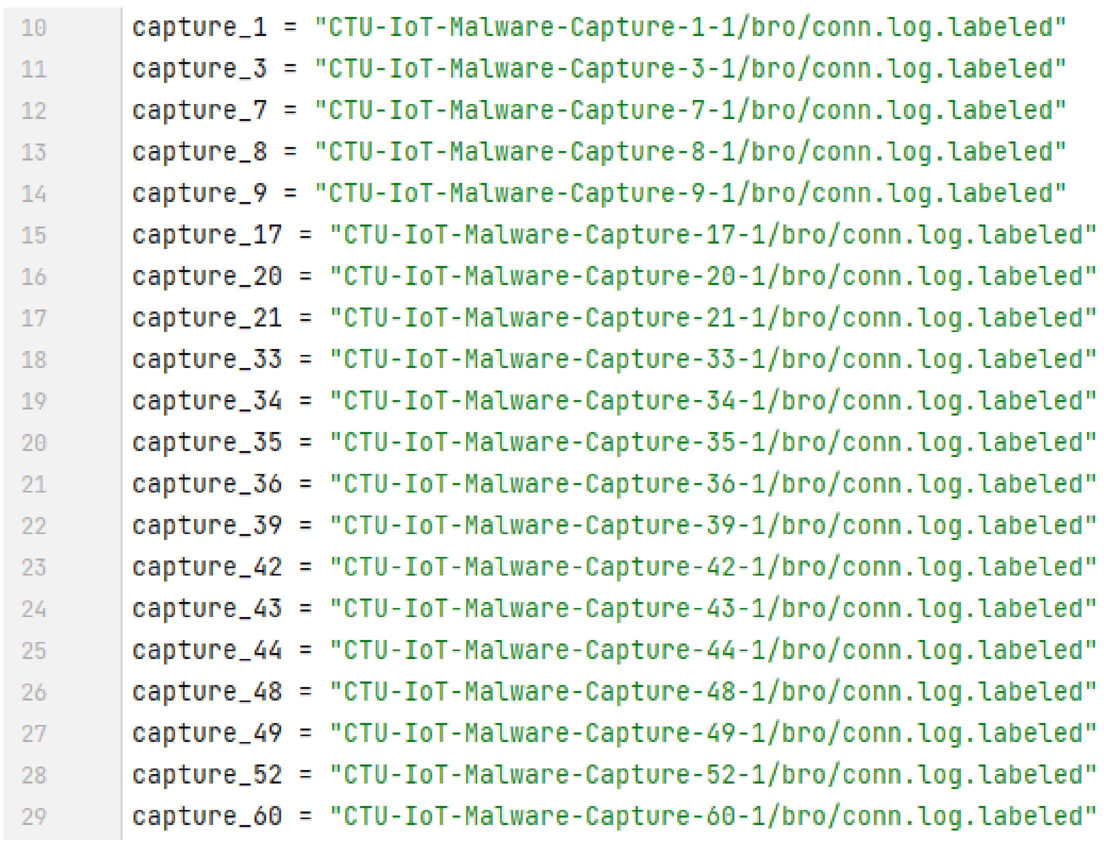
To start with there were several techniques incorporated into the cleaning process to ensure the dataset was suitable for classification. First, I imported the necessary python libraries: Pandas, NumPy and os (First two relate to data manipulation while os is simply to change the working directory to the location of the captures).



```python
import pandas as pd
import numpy as np
import os
```

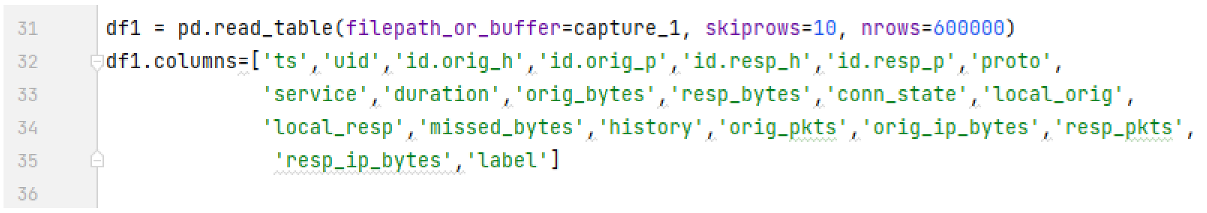
Figure 2 - Dataset cleaning libraries

The files that make up the IoT-23 dataset are a set of connection log files created by the network traffic analyser Zeek, firstly the files needed to be and combined into one Pandas data frame this is shown in figures 3 and 4.



```python
capture_1 = "CTU-IoT-Malware-Capture-1-1/bro/conn.log.labeled"
capture_3 = "CTU-IoT-Malware-Capture-3-1/bro/conn.log.labeled"
capture_7 = "CTU-IoT-Malware-Capture-7-1/bro/conn.log.labeled"
capture_8 = "CTU-IoT-Malware-Capture-8-1/bro/conn.log.labeled"
capture_9 = "CTU-IoT-Malware-Capture-9-1/bro/conn.log.labeled"
capture_17 = "CTU-IoT-Malware-Capture-17-1/bro/conn.log.labeled"
capture_20 = "CTU-IoT-Malware-Capture-20-1/bro/conn.log.labeled"
capture_21 = "CTU-IoT-Malware-Capture-21-1/bro/conn.log.labeled"
capture_33 = "CTU-IoT-Malware-Capture-33-1/bro/conn.log.labeled"
capture_34 = "CTU-IoT-Malware-Capture-34-1/bro/conn.log.labeled"
capture_35 = "CTU-IoT-Malware-Capture-35-1/bro/conn.log.labeled"
capture_36 = "CTU-IoT-Malware-Capture-36-1/bro/conn.log.labeled"
capture_39 = "CTU-IoT-Malware-Capture-39-1/bro/conn.log.labeled"
capture_42 = "CTU-IoT-Malware-Capture-42-1/bro/conn.log.labeled"
capture_43 = "CTU-IoT-Malware-Capture-43-1/bro/conn.log.labeled"
capture_44 = "CTU-IoT-Malware-Capture-44-1/bro/conn.log.labeled"
capture_48 = "CTU-IoT-Malware-Capture-48-1/bro/conn.log.labeled"
capture_49 = "CTU-IoT-Malware-Capture-49-1/bro/conn.log.labeled"
capture_52 = "CTU-IoT-Malware-Capture-52-1/bro/conn.log.labeled"
capture_60 = "CTU-IoT-Malware-Capture-60-1/bro/conn.log.labeled"
```

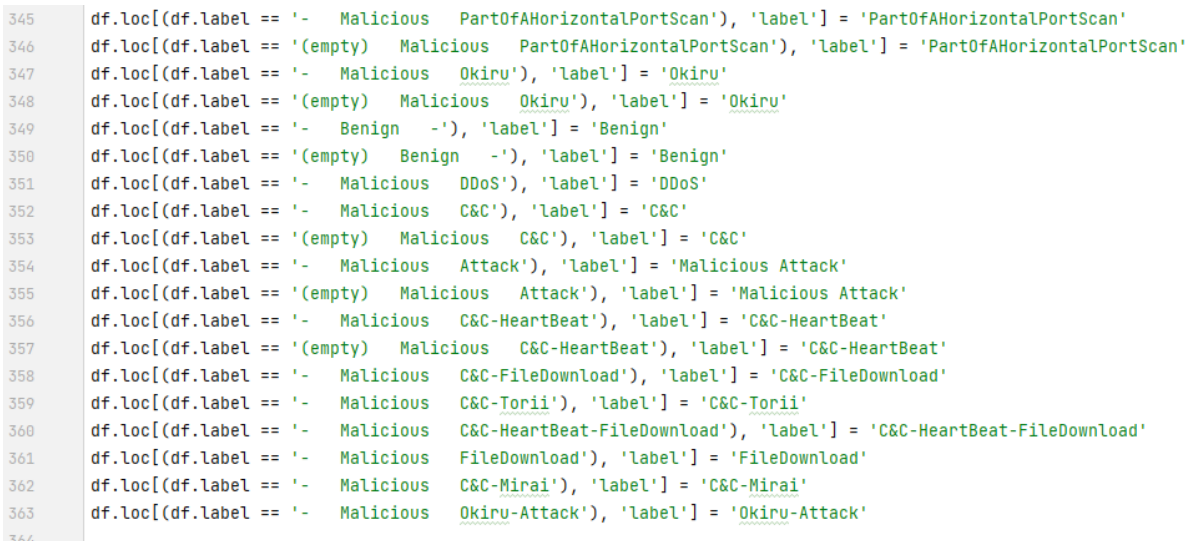
Figure 3 - Malware capture files



```
31    df1 = pd.read_table(filepath_or_buffer=capture_1, skiprows=10, nrows=600000)
32    df1.columns=['ts','uid','id.orig_h','id.orig_p','id.resp_h','id.resp_p','proto',
33                 'service','duration','orig_bytes','resp_bytes','conn_state','local_orig',
34                 'local_resp','missed_bytes','history','orig_pkts','orig_ip_bytes','resp_pkts',
35                  'resp_ip_bytes','label']
36
```

Figure 4 - Pandas import method

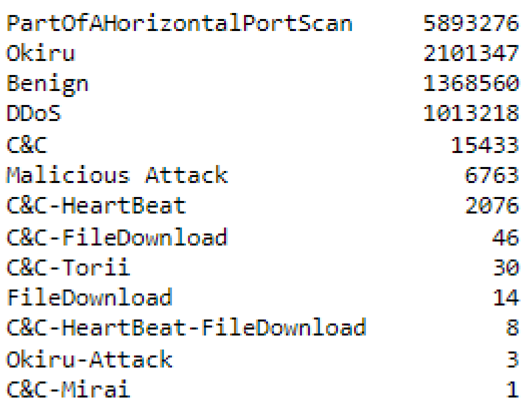
Each log file is imported individually with columns added post import, figure 4 shows the code used to import the first log file, line 31 imports the file as a table and line 32 assigns the column name for each.



```
345  df.loc[(df.label == '-   Malicious   PartOfAHorizontalPortScan'), 'label'] = 'PartOfAHorizontalPortScan'
346  df.loc[(df.label == '(empty)   Malicious   PartOfAHorizontalPortScan'), 'label'] = 'PartOfAHorizontalPortScan'
347  df.loc[(df.label == '-   Malicious   Okiru'), 'label'] = 'Okiru'
348  df.loc[(df.label == '(empty)   Malicious   Okiru'), 'label'] = 'Okiru'
349  df.loc[(df.label == '-   Benign   -'), 'label'] = 'Benign'
350  df.loc[(df.label == '(empty)   Benign   -'), 'label'] = 'Benign'
351  df.loc[(df.label == '-   Malicious   DDoS'), 'label'] = 'DDoS'
352  df.loc[(df.label == '-   Malicious   C&C'), 'label'] = 'C&C'
353  df.loc[(df.label == '(empty)   Malicious   C&C'), 'label'] = 'C&C'
354  df.loc[(df.label == '-   Malicious   Attack'), 'label'] = 'Malicious Attack'
355  df.loc[(df.label == '(empty)   Malicious   Attack'), 'label'] = 'Malicious Attack'
356  df.loc[(df.label == '-   Malicious   C&C-HeartBeat'), 'label'] = 'C&C-HeartBeat'
357  df.loc[(df.label == '(empty)   Malicious   C&C-HeartBeat'), 'label'] = 'C&C-HeartBeat'
358  df.loc[(df.label == '-   Malicious   C&C-FileDownload'), 'label'] = 'C&C-FileDownload'
359  df.loc[(df.label == '-   Malicious   C&C-Torii'), 'label'] = 'C&C-Torii'
360  df.loc[(df.label == '-   Malicious   C&C-HeartBeat-FileDownload'), 'label'] = 'C&C-HeartBeat-FileDownload'
361  df.loc[(df.label == '-   Malicious   FileDownload'), 'label'] = 'FileDownload'
362  df.loc[(df.label == '-   Malicious   C&C-Mirai'), 'label'] = 'C&C-Mirai'
363  df.loc[(df.label == '-   Malicious   Okiru-Attack'), 'label'] = 'Okiru-Attack'
```

Figure 5 - Renaming and grouping labels

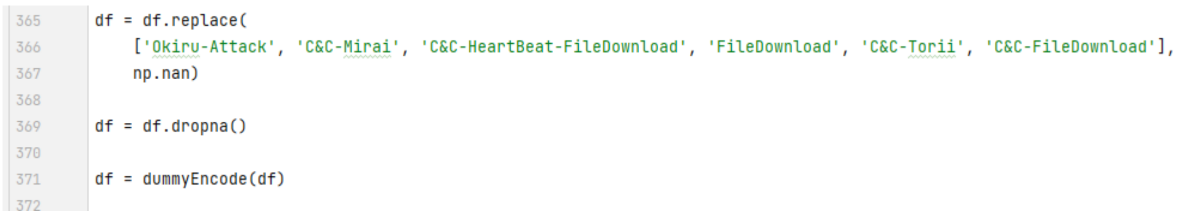
Once imported, I needed to clean up the label names and combine the labels that are the same, this process is shown in figure 5 from line 345 to 363.



```
PartOfAHorizontalPortScan     5893276
Okiru                         2101347
Benign                        1368560
DDoS                          1013218
C&C                             15433
Malicious Attack                 6763
C&C-HeartBeat                    2076
C&C-FileDownload                   46
C&C-Torii                          30
FileDownload                       14
C&C-HeartBeat-FileDownload          8
Okiru-Attack                        3
C&C-Mirai                           1
```

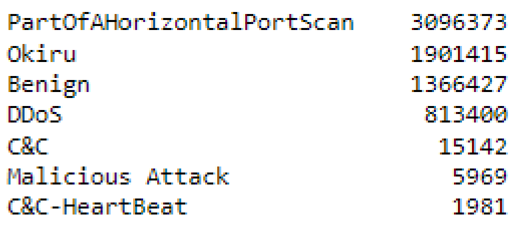
Figure 6 - total number of labels

Figure XX shows the label occurrences for each sample. At first glance, we can see an extreme class difference between majority classes and minority classes, and due to this I dropped the following labels Okiru-Attack, C&C-Mirai, C&C-HeartBeat-FileDownload, FileDownload, C&C-Torii, C&C-FileDownload. I did this due to there not being enough occurrences within the dataset for the models to learn from. This process is shown in figure 7. Similar decisions were made in the paper by Abdalgawad *et al* (2021). Furthermore, during GAN testing I found that samples generated from classes with less than a thousand instances did not result in quality generated data for further model training. This is also noted in the papers by Shahriar *et al* (2020) and Abdalgawad *et al* (2021). Figure 7 shows the final set of labels used.



```
df = df.replace(
    ['Okiru-Attack', 'C&C-Mirai', 'C&C-HeartBeat-FileDownload', 'FileDownload', 'C&C-Torii', 'C&C-FileDownload'],
    np.nan)

df = df.dropna()

df = dummyEncode(df)
```

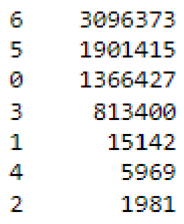
Figure 7 - dropping of labels



```
PartOfAHorizontalPortScan    3096373
Okiru                        1901415
Benign                       1366427
DDoS                          813400
C&C                            15142
Malicious Attack                5969
C&C-HeartBeat                   1981
```

Figure 8 - final set of labels

Finally, the labels needed to be encoded for the models to process the predictions. I used label encoding provided by sklearn to convert the string names to integer values, the conversions are shown in figure 9



```
6    3096373
5    1901415
0    1366427
3     813400
1      15142
4       5969
2       1981
```

Figure 9 - encoded labels

Feature reduction

The features in the IoT-23 dataset are described in table 3, these are the original column names prior to any reduction.

Column	Description
Ts	Time of capture in unix time code
Uid	Capture ID
id_orig.h	Source ip
id_orig.p	Source port
id_resp.h	Destination IP
id_resp.p	Destination port
proto	Transaction protocol: icmp, udp, tcp
service	Application protocol: dhcp, dns, http, irc, ssh, ssl
duration	Total duration of flow
orig_bytes	Number of bytes the originator sent
resp_bytes	Number of payload bytes the responder sent
conn_state	Connection state
local_orig	T if the connection originated locally and F if it originated remotely
local_resp	T if the connection originated locally and F if it originated remotely
missed_bytes	Number of bytes missed in content gaps, representative of packet loss
history	History of the state of the connection
orig_pkts	Number of packets that the originator sent
orig_ip_bytes	Number of IP level bytes that the originator sent
resp_pkts	Number of packets that the responder sent
resp_ip_bytes	Number of IP level bytes sent from the responder
label	Sample label, malicious set or benign

Table 3 - Feature descriptions

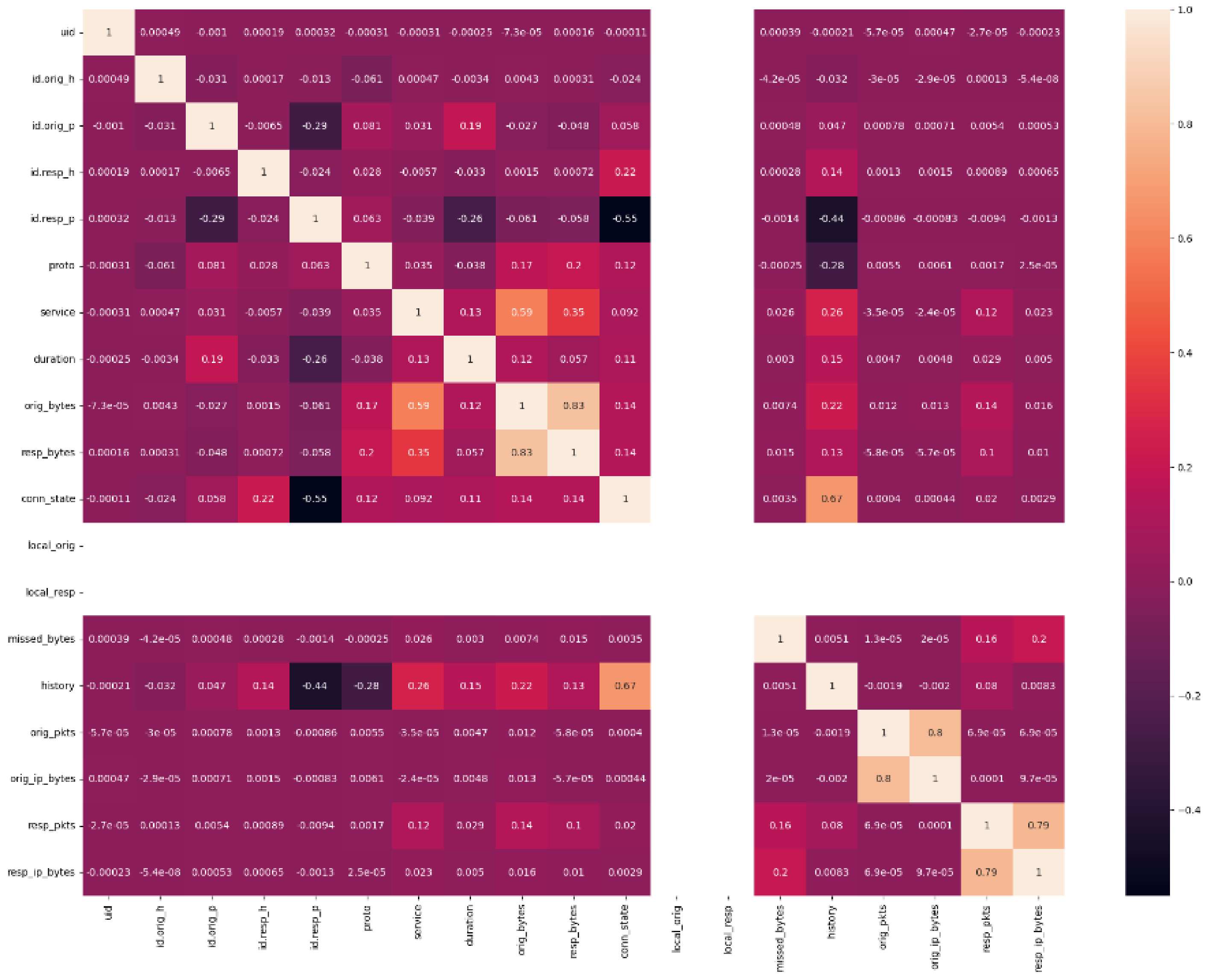
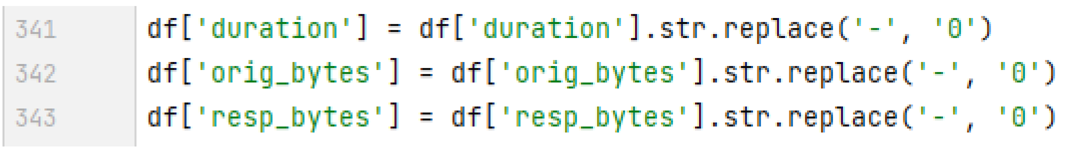


Figure 10 - correlation heatmap

Using a correlation heatmap, I firstly established that the columns 'local_orig' and 'local_resp' can be dropped from the feature set as they have too few occurrences to be of any use during classification. Related research by Abdalgawad *et al* (2021) note that based on the correlation graph 'orig_ip_bytes' and 'orig_pkts' correlate as do 'resp_pkts' and 'resp_ip_bytes'; hence the correlating features 'orig_ip_bytes' and 'resp_ip_bytes' were dropped. Similar actions were taken by Strecker *et al* (2021). The columns 'uid', 'ts' and 'id.orig_h' were also dropped due to weak correlation to the target column a decision also taken by Stoian *et al* (2020) and like similar papers using the IoT-23 dataset I dropped the 'history' column.

The final set of column names are: 'id.orig_p', 'id.resp_h', 'id.resp_p', 'proto', 'service', 'duration', 'org_bytes', 'resp_bytes', 'conn_state', 'missed_bytes', 'orig_pkts', 'label', 'resp_pks'.

The next step was to clean up the columns, the features ‘orig_bytes’, ‘resp_bytes’ and ‘duration’ contained nil values. If I were to drop these rows I would have significantly less rows in the dataset. This issue is noted in the paper by Liang and Vankayalapati (2021); their solution was to replace these missing values with 0 which is displayed in lines 341-343 in figure 12.

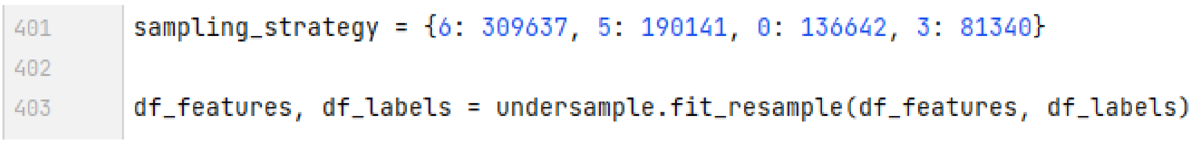


```
341    df['duration'] = df['duration'].str.replace('-', '0')
342    df['orig_bytes'] = df['orig_bytes'].str.replace('-', '0')
343    df['resp_bytes'] = df['resp_bytes'].str.replace('-', '0')
```

Figure 11 - cleaning column NaN values

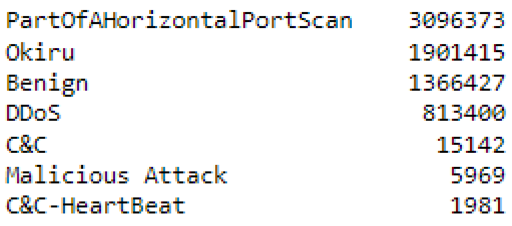
The IoT-23 dataset contains several categorical features namely: ‘service’, ‘proto’, and ‘conn_state’ which require encoding into numerical format as required by a neural network to learn. The methods available to encode categorical features include one-hot encoding and dummy variable encoding. Based on previous work on the Iot-23 dataset by Victor Oha (2022), I chose to use dummy encoding due to the quantity of sub-categorical features that would be created had I gone with one-hot encoding. This is a common issue as one-hot encoding with many categories may result in increasing the dimensionality of encodings resulting in problems of parallelism and multicollinearity (Satyam Kumar, 2021). This is noted by Zhang *et al* (2019).

As in a lot of IDS targeted datasets there is the issue of class imbalance. Nicolas-Alin Stoian (2020) & Abdalgawad *et al* (2021) note that extreme class imbalance is a serious issue when it comes to the classification of minority samples. While this is the focus of this project, I decided to under sample the majority classes. Stoian noted that their neural network testing on the IoT-23 dataset suffered greatly during training and testing due to the extreme imbalance. Although they did not implement any under sampling techniques which resulted in poor classification results of the minority classes, they note in the results analysis that the model had a bis for the classes with the largest number of samples belonging to that class. Whereas Abdalgawad *et al* (2021) used Random UnderSampling to reduce the size of the majority classes down to 10% of their former size, this proved significant in the model’s capability to learn from the minority classes, however in the previously mentioned paper researchers used synthetic minority oversampling to increase the class occurrences of the minority classes whereas in this project I use the synthetic data generated from the GAN to augment the training data set. Following on from the proportions used in the paper by Abdalgawad *et al* (2021), I implemented Random UnderSamping by a ratio of 10%, therefore each of the four majority classes were down sampled to 10% of their former number. Shown in figure 13 is the class occurrences prior to under sampling. Figure 14 is the class occurrences following under sampling and figure 12 is the code that identifies the labels that need to be under sampled by their encoded value.



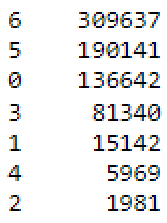
```
sampling_strategy = {6: 309637, 5: 190141, 0: 136642, 3: 81340}

df_features, df_labels = undersample.fit_resample(df_features, df_labels)
```

Figure 12 - sampling strategy



```
PartOfAHorizontalPortScan    3096373
Okiru                        1901415
Benign                       1366427
DDoS                          813400
C&C                            15142
Malicious Attack                5969
C&C-HeartBeat                   1981
```

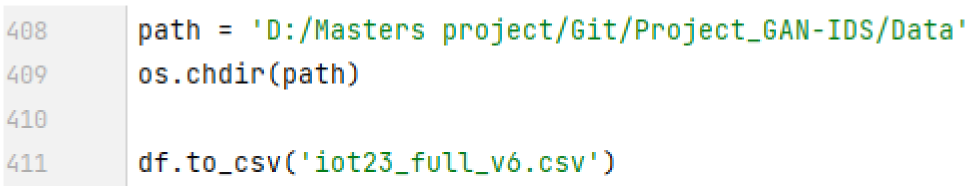
Figure 13 - pre-under sampled class values



```
6    309637
5    190141
0    136642
3     81340
1     15142
4      5969
2      1981
```

Figure 14 - post resampling class occurrences

Finally, the dataset is exported as a comma separated values file, this is shown in figure XX.



```
path = 'D:/Masters project/Git/Project_GAN-IDS/Data'
os.chdir(path)

df.to_csv('iot23_full_v6.csv')
```

Figure 15 - Saved cleaned dataset

Data split

To get training and testing data from our overall data we need to address a few points, firstly how do we ensure that samples representing all classes are present in each split and secondly how do know the correct proportions to split by. Thirdly, how do we introduce reproducibility into the testing to get a measure of how model parameters are influencing classification performance. Since I am using an unbalanced dataset the need for all classes to be present in each split is paramount to the model's training performance. To do so I performed a stratified split on the data rather than a random split so that all minority classes were present in each portion of the split. When deciding on the split proportions I chose to follow the Pareto principle (Thomas Brock, 2022) which is the observation that most things in life are not distributed evenly therefor the proposed split of 80:20 is how I chose to portion my data. Finally, as for reproducibility the 'train_test_split' function from sklearn provides us with a 'random state' parameter allowing us to control the shuffle applied to the data. Setting this to '42' ensures that for different testing iterations the data split is the same.

Min-max normalization

Following data splitting the training and testing feature sets are scaled using min max normalization, a technique where the function 'MinMaxScaler' performs a linear transformation on the feature data, this scales the data to the range of (0,1), the formular for this transformation is shown in figure 16. This process is common in machine learning as variables that are measured at different scales tend to not contribute equally during model training and the learned weights may be created with bias.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Figure 16 - min-max normalization formula

3.5. Intrusion detection system (IDS)

A standalone network intrusion detection system (IDS) observes network flows for suspicious activity, the case I am presenting is a bare bones research orientated setup where a machine learning model is trained on a portioned set known as the training set and this model once completing its training is tested upon an unseen set of data known as the test set, my project is testing whether a GAN can generate synthetic training data to augment the class number of the training set put forward to the IDS. Therefore, the IDS is a simple but effective neural network that has a good ability to classify sample of high occurrence and a difficult time classifying samples of low occurrence.

In broad terms an IDS can be either a physical device or software that is placed in strategic points on the network. Its job is to monitor given sections to look out for any malicious activity or any form of policy violations within the network and respond to the given issue or threat. Where it could be, an alert is triggered or set rules are in place for how the IDS deals with the issue. The most common forms of IDS range from off the shelf anti-virus systems to tiered monitoring systems that provide large companies with protection capabilities of tens of thousands of incoming data streams. The most common forms of IDS come in the following:

Network based intrusion detection (NIDS): Incoming network traffic for network segments is analysed for suspicious activity

Host-based intrusion detection system (HIDS): Here the system monitors network traffic on the device itself or a specific device within a network

We can break the types of IDS down into a further three subcategories.

- **Signature-based**: This form of IDS works by looking for the known signs of an attack, hence the name signature based. The system has a database of known attack types and the signs of network behaviours that relates to such attacks. The original terms come from anti-virus software that would refer to detected attack patterns as 16 signatures. As mentioned this is the method of operation for classical IDSs and while it works well in detecting known attacks, it is more or less impossible for these systems to detect new or unknown threats
- **Anomaly-based**: This is a newer direction in the IDS area of development where attacks are detected using machine learning. A system creates a model of trusted activity to use as the comparative trust model. New incoming data will be compared to this trusted model with deviations being rejected as malicious. Although machine learning methods of approach have been known to suffer from false positive responses to unseen benign data the main strength of anomaly-based methods of detection is the ability to detect unknown/novel and zero-day threats, we usually want to find a good balance between false positives and false negatives since any missed detections could be detrimental.
- **Hybrid Detection**: Here the IDS uses a mix of anomaly and signature-based methods of detection which allows for detection of known and more potential attacks while having a lower error rate than using either of the two solutions in isolation.

The system I will be using for my project is a type of anomaly detection whereby I will use a machine learning algorithm to learn to associate patterns in network flows with the type of data they represent i.e., a type of malware or normal network traffic, this will be completed by an artificial neural network (ANN).

3.6. Artificial neural network (ANN)

An artificial neural network (ANN) is named as such due to the design and process of operation, the human brain which is made up of approximately 85 billion neurons. Input signals are received via the dendrite of a neuron from environmental stimulation, the signal is processed within the cell body and transmitted down the axon to the output terminal.

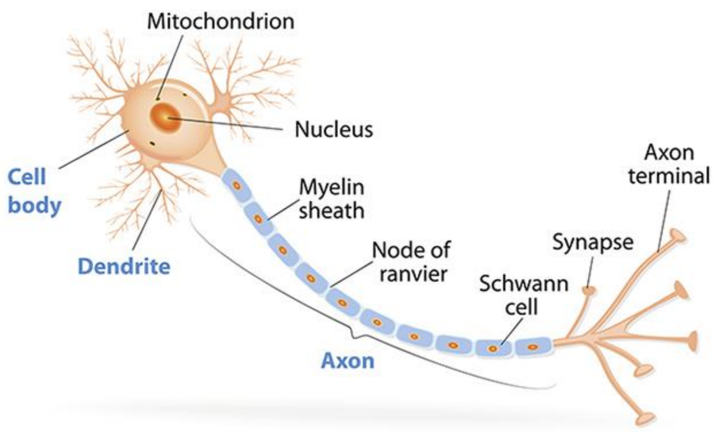


Figure 17 - diagram of brain neuron

Within our ANN a single neuron works in a very similar way, like the dendrite of a neuron our artificial neurons receive input signals though feature variables which are weighted according to its importance. These signals are processed through an activation function which is the mathematical function that transforms the input to the required output. In my project I use several different activation functions depending on the stage of the task and output requirements of the input data.

The structure of the ANN regarding the number hidden layers within the model can depend on the complexity of the data that is to be learned upon and the activation function used, in my case a single hidden layer is suitable.

Layer 1 – (, 12)	input layer
Layer 2 – (, 15)	hidden layer, activation = ReLu
Layer 3 – (, 7)	output layer , activation = SoftMax

The first layer of the NN is the input layer in my IDS. I use twelve nodes which corresponds to the number of feature columns x_1 to x_i. These are passed onto an input function *u* which computes the weight sum of all values received plus the bias, the formular is shown in figure 18.

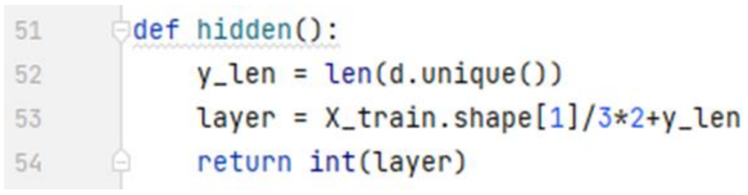
$$u\ (x) = \sum_{i=1}^{n} (x_i * w_i) + bias$$

Figure 18 - Input function

Where:

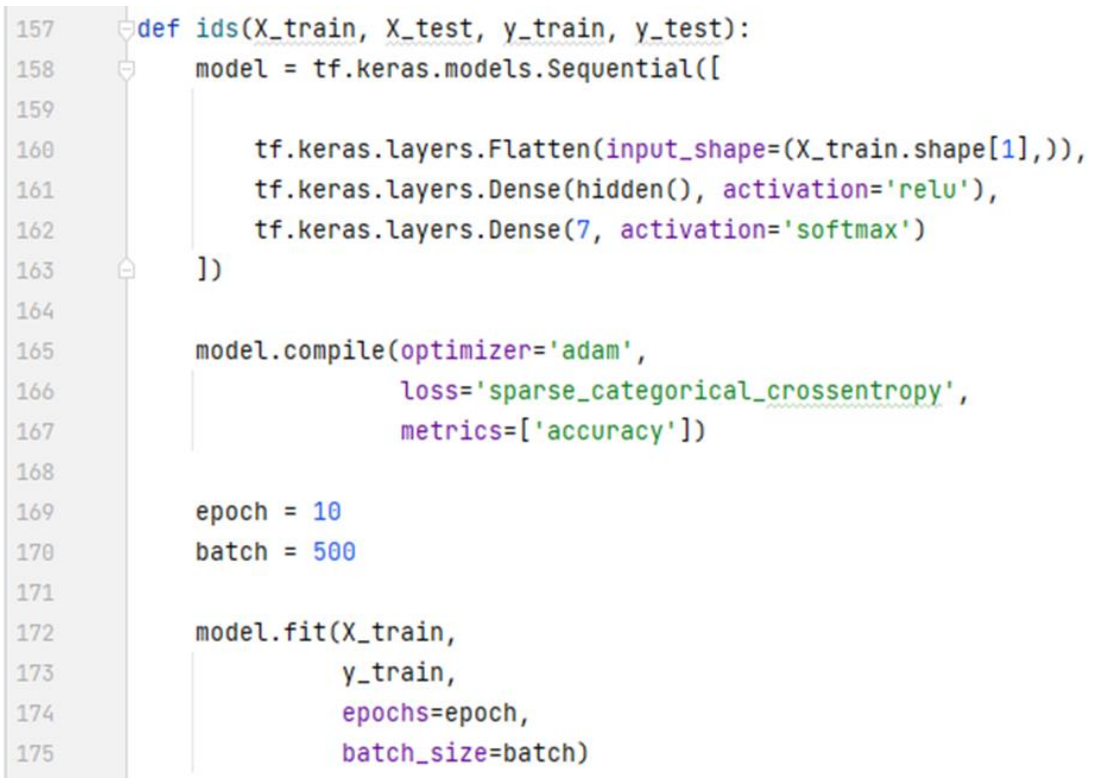
- *n* is the number of neurons connected to the target neuron
- x_i relates to the information from neuron *i*
- w_i is the weight given to the connection between the target neuron and neuron *i*
- The bias is the state of the target neuron.

Layer two or hidden layer 1 is the only hidden layer. From previous research there have been very few implementations of a multilayer perceptron used in conjunction with the IoT-23 dataset. In the paper by Nicolas-Alin Stoian (2020), the authors use a single hidden layer in their ANN, and their results indicate that while majority classes are correctly classified to a high degree the model has difficulty classifying the minority classes. Through testing and previous work with ANN's for intrusion detection I decided to use a single hidden layer. To decide on the number of neurons in the hidden layer I used a simple calculation, 2/3 the size of the input layer plus the number of nodes in the output layer resulting in 15 nodes, to prevent constantly needing to manually adjust the hidden layer values I used a function to perform this calculation, it is described in figure 19 lines 51 to 54 which is called on line 161 of figure 20.



```
51  def hidden():
52      y_len = len(d.unique())
53      layer = X_train.shape[1]/3*2+y_len
54      return int(layer)
```

Figure 19 - hidden layer node function



```python
def ids(X_train, X_test, y_train, y_test):
    model = tf.keras.models.Sequential([

        tf.keras.layers.Flatten(input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(hidden(), activation='relu'),
        tf.keras.layers.Dense(7, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    epoch = 10
    batch = 500

    model.fit(X_train,
              y_train,
              epochs=epoch,
              batch_size=batch)
```

Figure 20 - IDS code

Upon further testing this structure provided suitable classification results for the initial malware classification phase. The activation function determines the value of the output based upon the combined weighted sum of the input; in my case I use a rectified linear activation function (ReLU). ReLU activation is, despite its name, a non-linear function that can be mathematically expressed as *f(x) = max(0,x);* the output of this function is the maximum value between 0 and the input value *x*, and graphically ReLU can be represented as figure XX

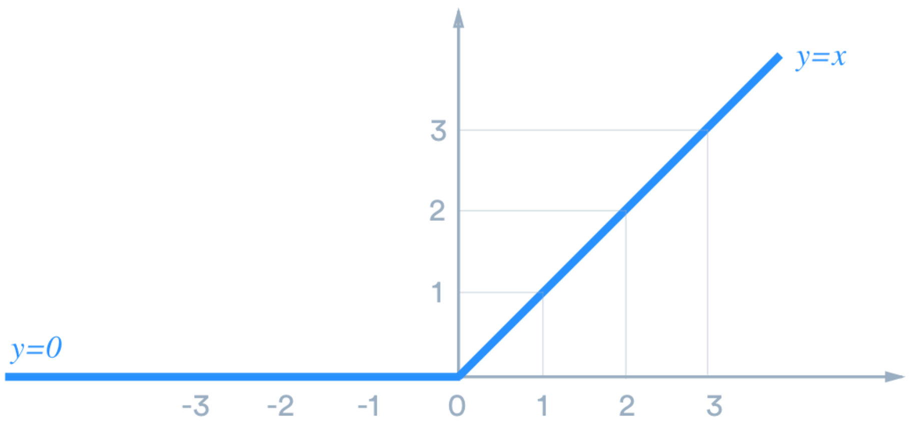


Figure 21 - ReLu activation function

The final layer is the output layer. Here the model sorts the predictions from the hidden layer into their respective class predictions**.** I used the SoftMax activation function, it is commonly used in multiclass classification as it transforms the raw outputs of the hidden layer into a vector of probabilities relating to which class each sample belongs to. Described in figure 22 where *z* is the vector of inputs which is a set of *K* real numbers, the function normalizes the input into a probability distribution consisting of *K* probabilities proportional to the exponentials of the input numbers.

For I = 1,…., K

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 22 – softmax activation formula

SoftMax activation is used as the model is solving a multinomial probability problem, i.e., what is the probability that each sample belongs to class *y*? The number of nodes in the output later is 7 as this corresponds to the number of classes in the dataset.

3.7. Generative Adversarial Networks (GANs)

A GAN is a relatively new technique in machine learning, designed in 2014 by Ian Goodfellow *et al*. As mentioned above, it works by having two neural networks compete in a game where the objective is to beat one another. A GAN is made up of two components, a generator whose job is to "generate" synthetic examples of data to fool the second component the discriminator that needs to be able to distinguish the difference between the synthetic data and the real data. This process is repeated until the generator can "beat" the discriminator. It can be formulated by minimax; this is a concept in game theory where the minimax is the highest value a player *A* can be sure to get without knowing the actions of player *B* and inversely it is the lowest value player *B* can force player *A* to receive when the actions of player *B* is known (Michael Maschler *et al*). Referred to as a vanilla GAN where D is trained to maximise the probability of assigning the correct label whether that be "true" or "false" for each sample, while *G* is trained to minimise its loss through minlog(1−D(G(z))). In practice however the training of *D* does not reach optimized completion while *G* is slowly optimized through each iteration.

3.7.1. Generator

As the name suggests the generator must "generate" fake data from the random input noise and as the generator learns through the results of the discriminator it incorporates the feedback and adjusts the weights of the network to minimise the loss of the output. A basic diagram of this is shown in figure 23. This is described mathematically as generator G turns random input noise $z \in \mathbb{R}^d$ from a given distribution γ into a set of generated samples $G(z)$ (Yang Wang, 2020).

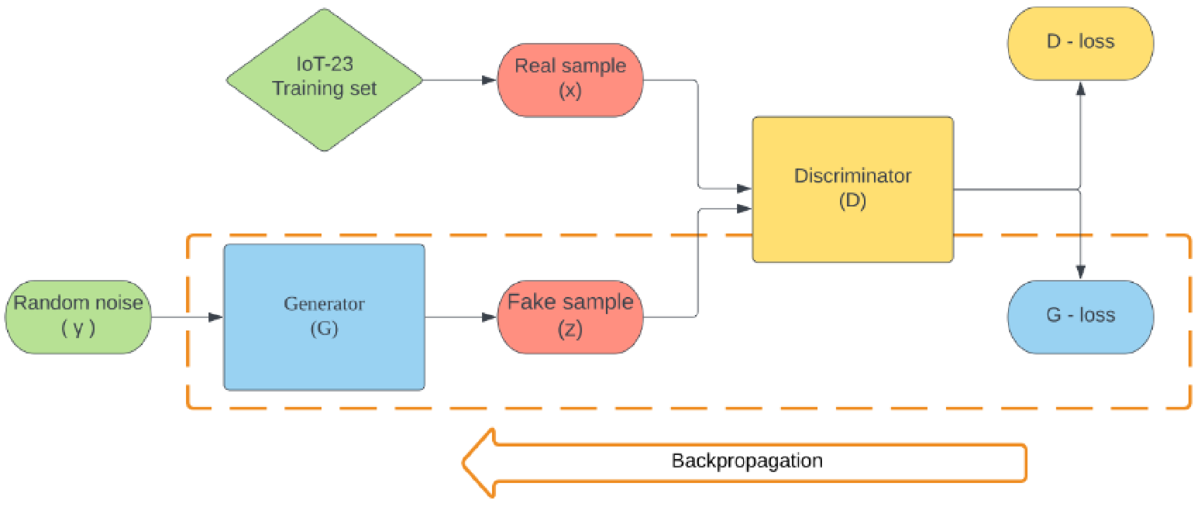


Figure 23 - Generator back propagation training

The generator G is a MLP which is a class of feed forward neural network that uses back propagation to adjust the weights of the model by approximating the non-linear relationship of the input and the output by adjusting the model's internal weight values. Back propagation in a NN is described in two stages: feedforward and backpropagation, in the feedforward stage the input layer receives the signals from the data and the effects propagate through the model's layers until an output is produced. This output is then compared to the expected output and an error signal is produced for each of the nodes and send back to update the weights of the nodes that contributed to the output. The back propagation algorithm works by finding the minimum value of the error function in weight space using gradient descent.

The activation layers are LeakyReLu with an alpha α value of 0.2. LeakyReLu is a variant of ReLU whereas instead of being 0 when $x < 0$, leaky ReLU allows for a non-zero gradient, $f(x)=1(x<0)(\alpha x)+1(x>=0)(x)$ where α small constant such as $\alpha=0.01$, put simply and as its name describes, a LeakyReLU activation function will leak some positive values to 0 if they are close enough to zero, this concept is shown in figure 24. This is to overcome the issue of a dying ReLU where neurons go into a state where they stop responding to variations in error/input.

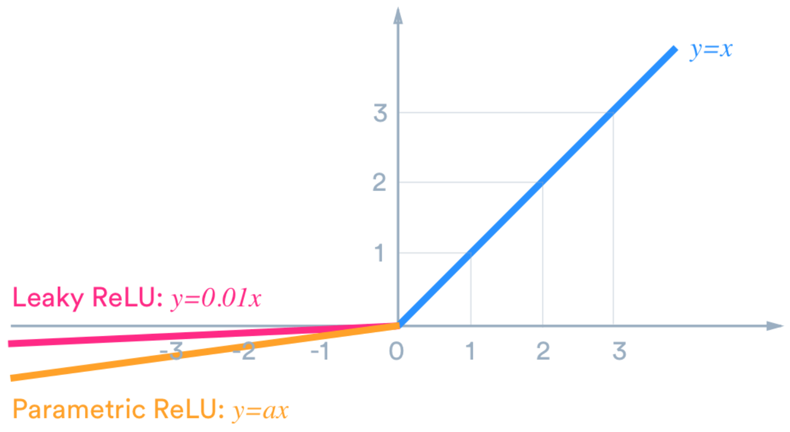
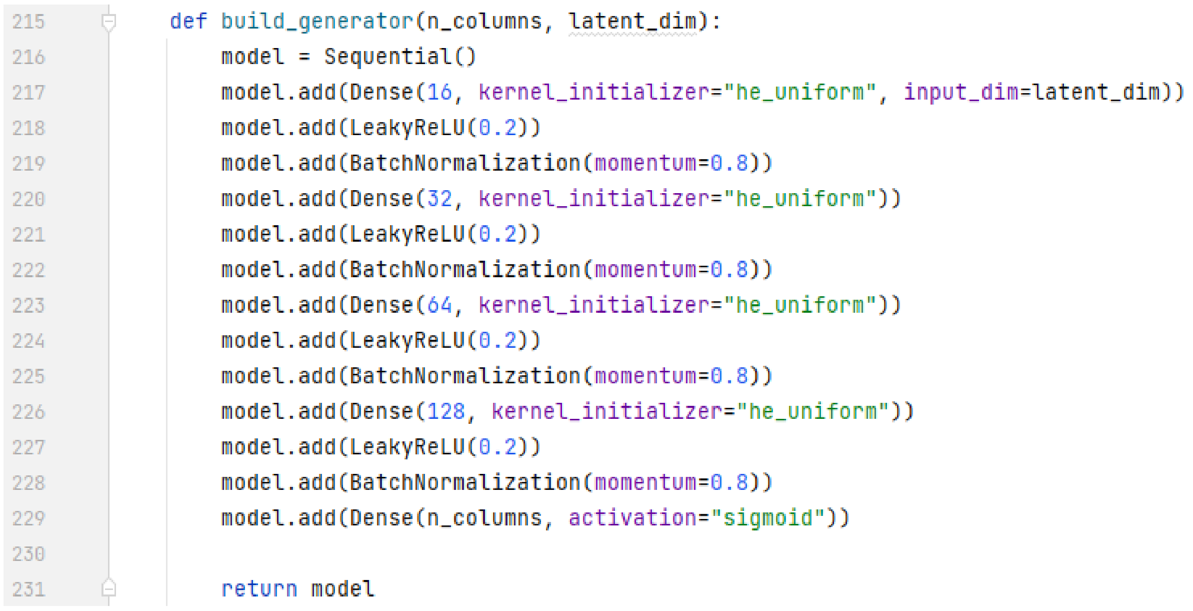


Figure 24 - LeakyReLu diagram

Batch normalization is implemented after the activation layer. It is used to achieve a stable distribution of activation values through training by maintaining the mean output close to 0 and the output standard deviation close to 1.

Layer 1 – (16 , 12)	input layer
Layer 2 – ()	activation = LeakyReLu(0.2)
Layer 3 – ()	BatchNormalization , momentum (0.8)
Layer 4 - (32)	kernel_initializer = he_uniform
Layer 5 – ()	activation = LeakyReLu(0.2)
Layer 6 – ()	BatchNormalization , momentum (0.8)
Layer 7 - (64)	kernel_initializer = he_uniform
Layer 8 – ()	activation = LeakyReLu(0.2)
Layer 9 – ()	BatchNormalization , momentum (0.8)
Layer 10 - (128)	kernel_initializer = he_uniform
Layer 11 – ()	activation = LeakyReLu(0.2)
Layer 12 – ()	BatchNormalization , momentum (0.8)
Layer 13 – ()	output layer sigmoid activation

The code for the generator stage can be seen in figure XX from line 215 – 231, this was written by Chris Nicholson (2020). It was further adapted for tabular data generation by a GitHub user Paakki (2020) and I have modified this section by adding an additional 16 node dense layer followed by additional LeakyReLu layer and BatchNormalization layer.



```
215    def build_generator(n_columns, latent_dim):
216        model = Sequential()
217        model.add(Dense(16, kernel_initializer="he_uniform", input_dim=latent_dim))
218        model.add(LeakyReLU(0.2))
219        model.add(BatchNormalization(momentum=0.8))
220        model.add(Dense(32, kernel_initializer="he_uniform"))
221        model.add(LeakyReLU(0.2))
222        model.add(BatchNormalization(momentum=0.8))
223        model.add(Dense(64, kernel_initializer="he_uniform"))
224        model.add(LeakyReLU(0.2))
225        model.add(BatchNormalization(momentum=0.8))
226        model.add(Dense(128, kernel_initializer="he_uniform"))
227        model.add(LeakyReLU(0.2))
228        model.add(BatchNormalization(momentum=0.8))
229        model.add(Dense(n_columns, activation="sigmoid"))
230
231        return model
```

Figure 25 - generator code

3.7.2. Discriminator

The task of the discriminator which is a simple classifier is to distinguish real data from fake data so following on from the generation stage the discriminator is fed combinations of real samples from the training set ***X*** i.e., real samples and generated samples from the generator $G(\boldsymbol{z})$ i.e., fake samples and must decide whether each sample x is real or not by assigning a probability for each sample $D(\mathrm{x}) \in [0,1]$ as to whether it is from the distribution of training samples (real samples) or generated samples (fake samples), a simple diagram describing the back propagation training is shown in figure 26.

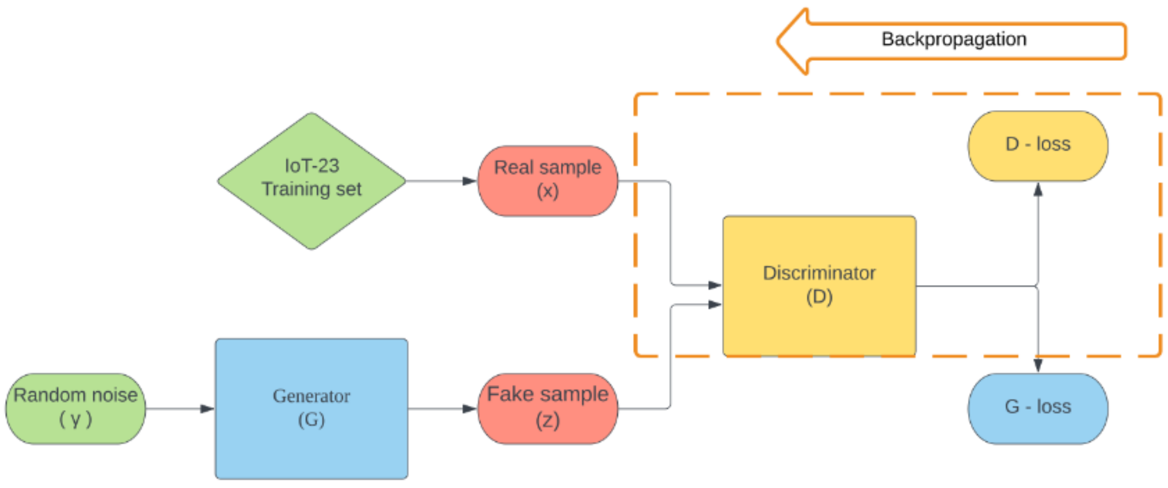
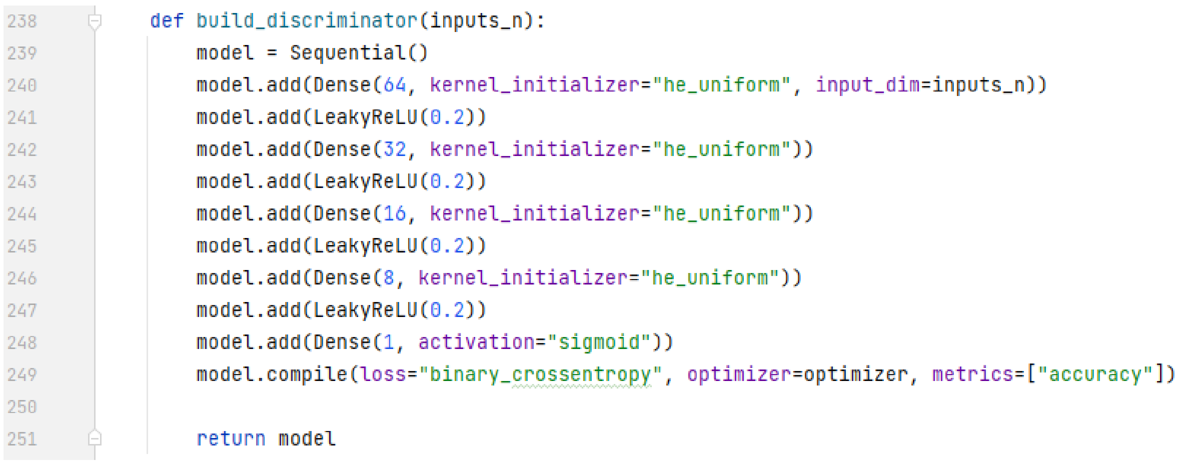


Figure 26 - Discriminator diagram

As with G the Discriminator D is a MLP. The fully connected neural network is tasked with returning an estimation as to whether x is real or fake D(x; θ^D). The chosen architecture for the discriminator uses activation layers of LeakyReLu with an alpha α value of 0.2, in terms of neuron configuration when compared to the original use case which flipped the neuron layout of the generator for use in the discriminator. Instead I chose to start with 64 neurons in the first layer rather than 128 as during testing the discriminator greatly outperformed the generator resulting in poor quality generated data. The output layer which is binary classification as the model is determining whether the data is real, or fake uses the sigmoid activation function with a single neuron.

Layer 1 – (64 , 12)	input layer
Layer 2 – ()	activation = LeakyReLu(0.2)
Layer 3 - (32)	kernel_initializer = he_uniform
Layer 4 – ()	activation = LeakyReLu(0.2)
Layer 5 - (16)	kernel_initializer = he_uniform
Layer 6 – ()	activation = LeakyReLu(0.2)
Layer 7 - (8)	kernel_initializer = he_uniform
Layer 8 – ()	activation = LeakyReLu(0.2)
Layer 9 – (1)	output layer sigmoid activation

The code shown in figure XX was written by Chris Nicholson (2020).It was further adapted for tabular data generation by a GitHub user Paakki (2020) and I adapted this section by setting the neurons in the first dense layer to have 64 neurons and this reduces by half each iteration as the dimensions of my dataset better suited this layout.



```
def build_discriminator(inputs_n):
    model = Sequential()
    model.add(Dense(64, kernel_initializer="he_uniform", input_dim=inputs_n))
    model.add(LeakyReLU(0.2))
    model.add(Dense(32, kernel_initializer="he_uniform"))
    model.add(LeakyReLU(0.2))
    model.add(Dense(16, kernel_initializer="he_uniform"))
    model.add(LeakyReLU(0.2))
    model.add(Dense(8, kernel_initializer="he_uniform"))
    model.add(LeakyReLU(0.2))
    model.add(Dense(1, activation="sigmoid"))
    model.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["accuracy"])

    return model
```

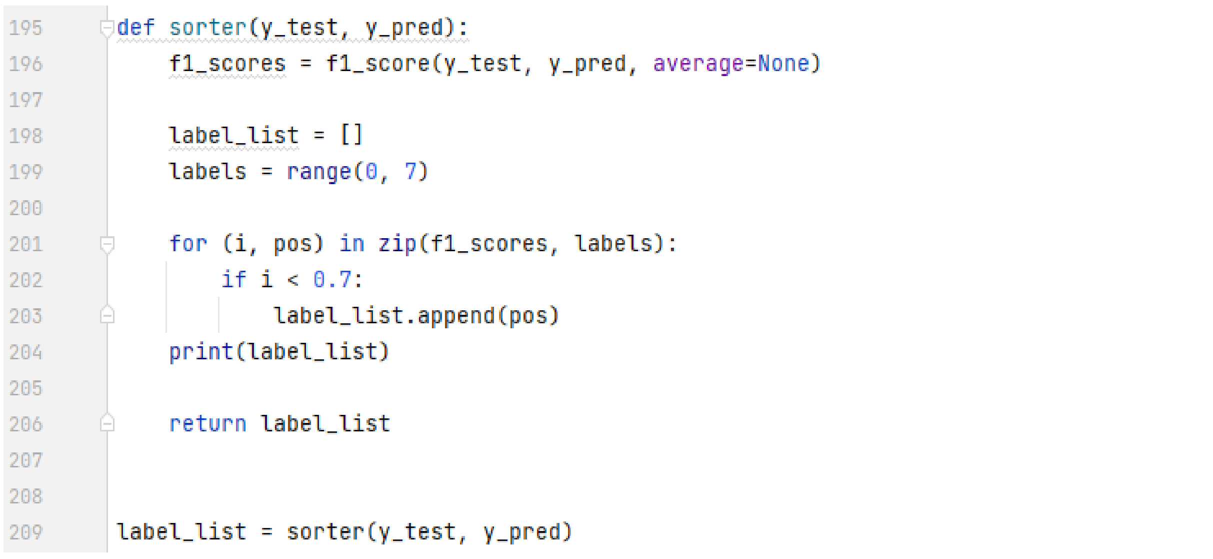
Figure 27 - Discriminator code

3.8. Sorter

Once the classification results are in from the first stage the sorter selects the underperforming samples by iterating through an array with the F1 scores. I decided on a qualifying FF1 score of < 0.70, if a FF1 score is less than 0.70 then the class label is added to the list which is put forward to the GAN for generation, the pseudocode can be seen below in figure 28 and the actual code is shown in figure 29. It is a relatively simple function designed to take an array of FF1scores, find the scores below 0.70 and append the relating encoded label to a list for use layer on.

```
**Algorithm**: sorter (y_test, y_pred)
**Input:** **y_test – array( $t_1$ - $t_n$ ),** y_pred – array( $j_1$ - $j_n$ )
**Output:** label_list
label_list = [list] – empty
F1_scores = array( $z_1 – z_n$ )
labels = range(0 to 7)
**for** ( i = 0 to i = 7 and pos = 0 to pos = 7) in F1_scores and labels
        **if**  i  is less than (an F1 score of) 0.7:
              append pos(0 – n) to [label_list]
        **end if**
  **end for**
  **return** label_list [pos < 0.70]
```

Figure 28 - psudo code for sorter

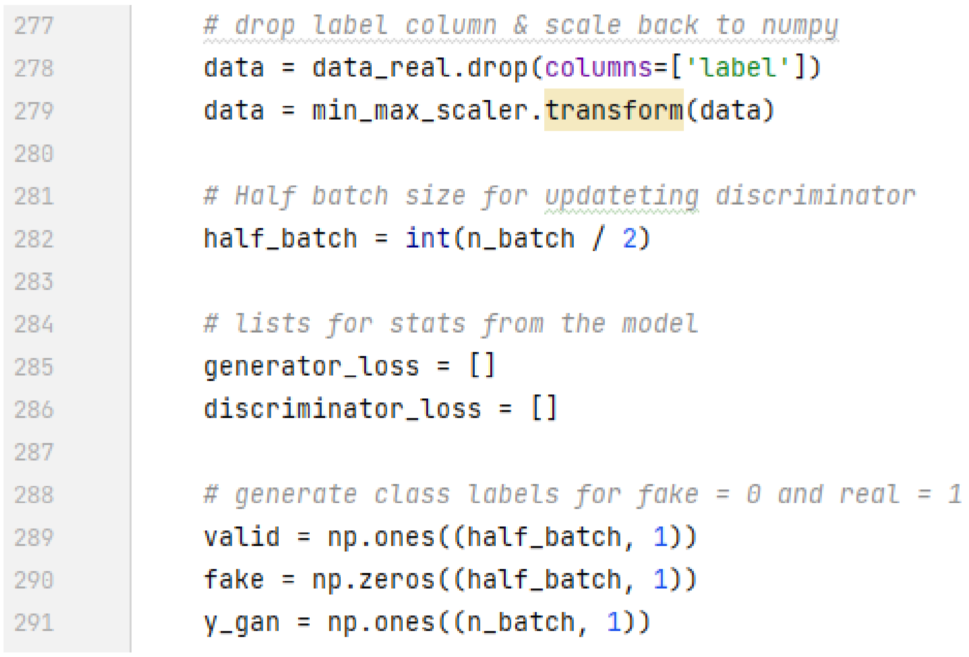


```python
def sorter(y_test, y_pred):
    f1_scores = f1_score(y_test, y_pred, average=None)

    label_list = []
    labels = range(0, 7)

    for (i, pos) in zip(f1_scores, labels):
        if i < 0.7:
            label_list.append(pos)
    print(label_list)

    return label_list


label_list = sorter(y_test, y_pred)
```

Figure 29 - label sorter code

3.9. Data generation and GAN training

The training function is where the GAN comes together. The data coming into this function is the selected training feature set, each class goes through this function one at a time, and this is to ensure the generator is updated using discriminator loss from training on a single class. If two different classes were subject to discrimination the *D* would not be able to tell them apart nor be able to classify them independently as it is a binary classifier hence each sample feature set, comes in together.

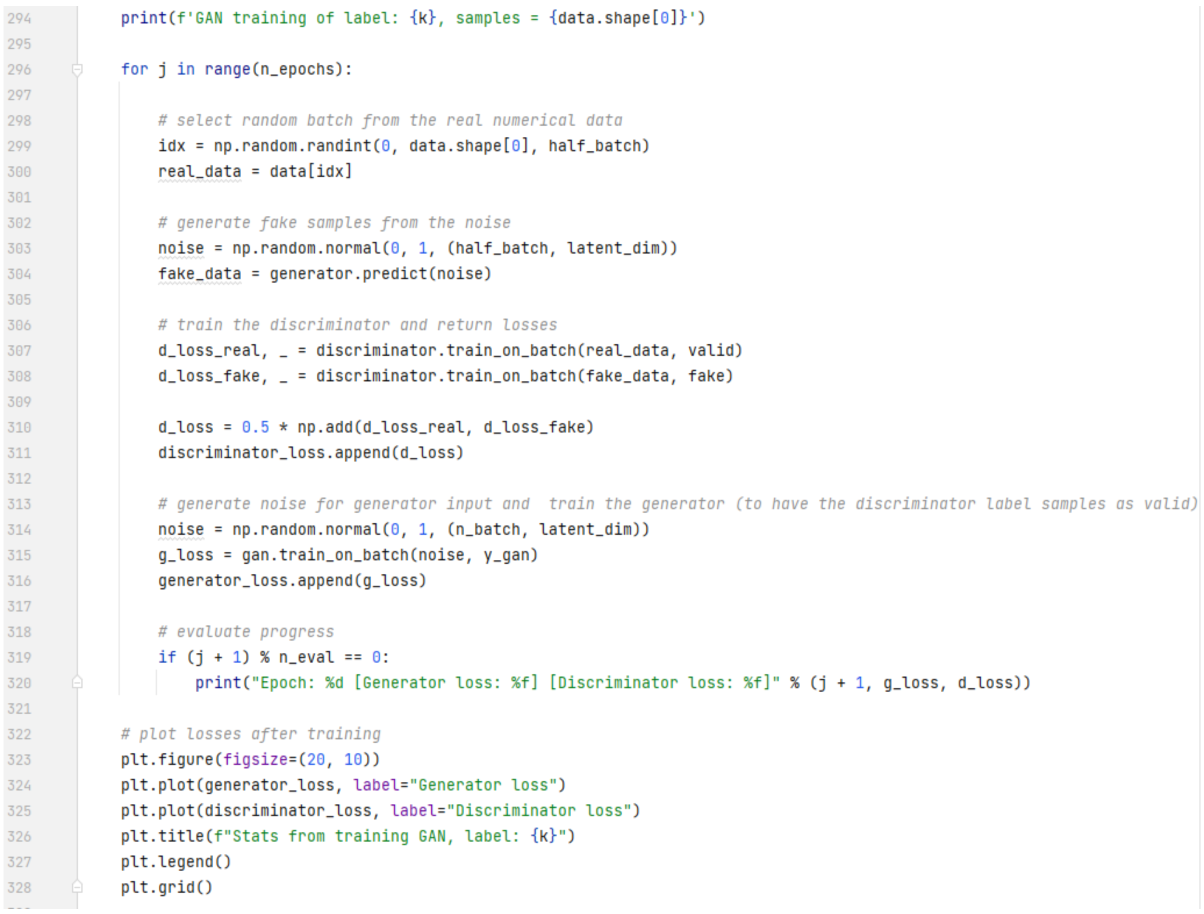


```
277         # drop label column & scale back to numpy
278         data = data_real.drop(columns=['label'])
279         data = min_max_scaler.transform(data)
280
281         # Half batch size for updateting discriminator
282         half_batch = int(n_batch / 2)
283
284         # lists for stats from the model
285         generator_loss = []
286         discriminator_loss = []
287
288         # generate class labels for fake = 0 and real = 1
289         valid = np.ones((half_batch, 1))
290         fake = np.zeros((half_batch, 1))
291         y_gan = np.ones((n_batch, 1))
```

Figure 30 - class labels

Referring to figure 30, on line 278 the label column is dropped from the dataframe as it is not to be a trainable feature and the dataframe is scaled to a numpy array. The real labels (valid) are assigned “1” on line 289 and fake labels are assigned “0” on line 290.

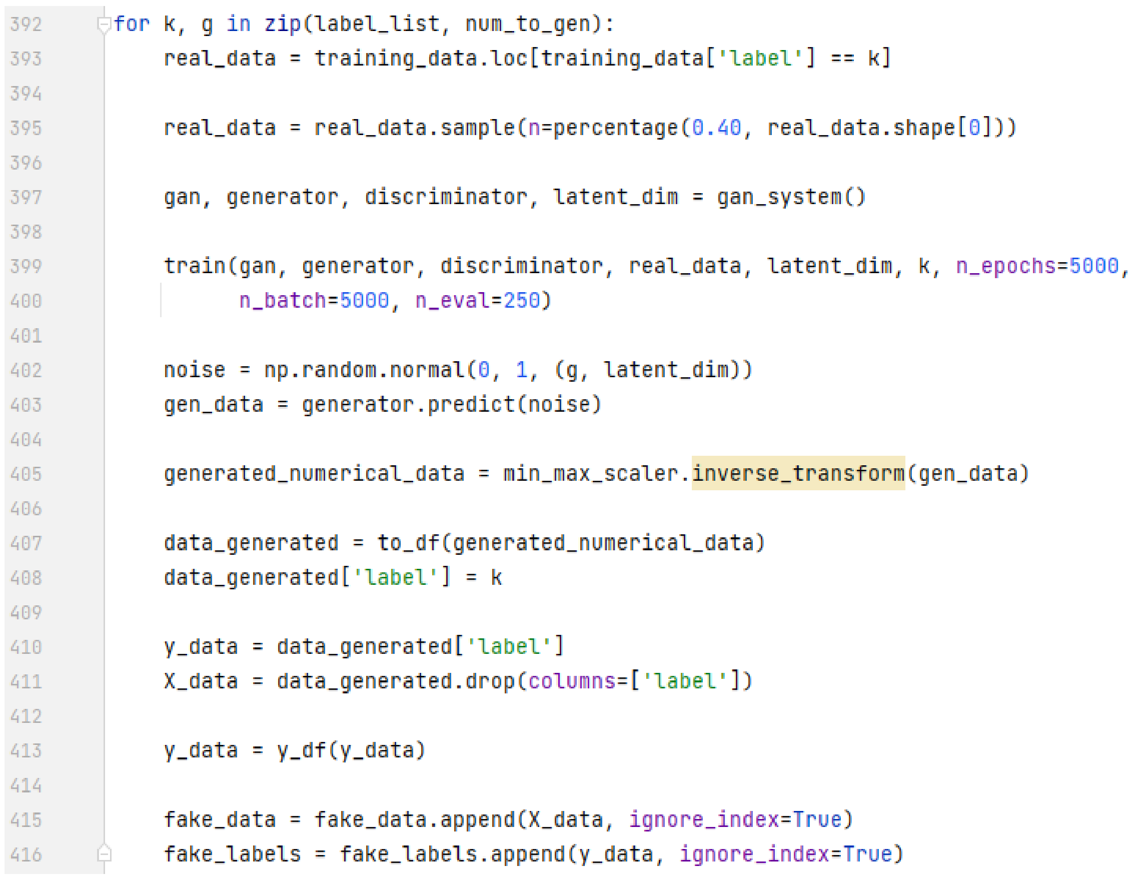
The code shown in figure 30 and 31 was written by Chris Nicholson (2020), it was further adapted for tabular data generation by a GitHub user Paakki (2020).



```
294     print(f'GAN training of label: {k}, samples = {data.shape[0]}')
295
296     for j in range(n_epochs):
297
298         # select random batch from the real numerical data
299         idx = np.random.randint(0, data.shape[0], half_batch)
300         real_data = data[idx]
301
302         # generate fake samples from the noise
303         noise = np.random.normal(0, 1, (half_batch, latent_dim))
304         fake_data = generator.predict(noise)
305
306         # train the discriminator and return losses
307         d_loss_real, _ = discriminator.train_on_batch(real_data, valid)
308         d_loss_fake, _ = discriminator.train_on_batch(fake_data, fake)
309
310         d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
311         discriminator_loss.append(d_loss)
312
313         # generate noise for generator input and  train the generator (to have the discriminator label samples as valid)
314         noise = np.random.normal(0, 1, (n_batch, latent_dim))
315         g_loss = gan.train_on_batch(noise, y_gan)
316         generator_loss.append(g_loss)
317
318         # evaluate progress
319         if (j + 1) % n_eval == 0:
320             print("Epoch: %d [Generator loss: %f] [Discriminator loss: %f]" % (j + 1, g_loss, d_loss))
321
322     # plot losses after training
323     plt.figure(figsize=(20, 10))
324     plt.plot(generator_loss, label="Generator loss")
325     plt.plot(discriminator_loss, label="Discriminator loss")
326     plt.title(f"Stats from training GAN, label: {k}")
327     plt.legend()
328     plt.grid()
```

Figure 31 - training loop

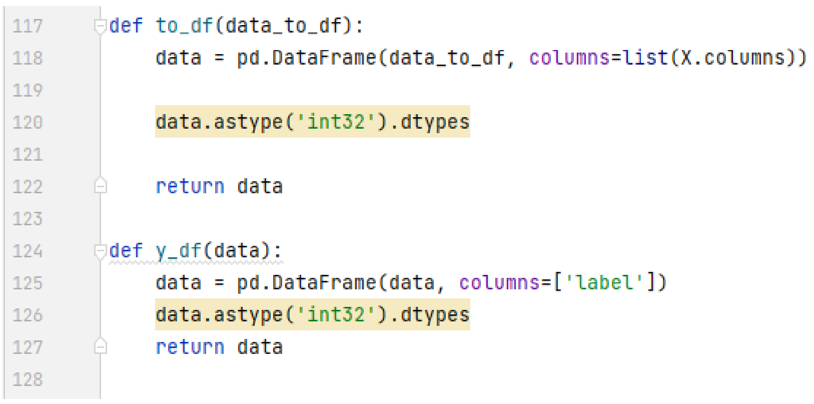
Looking to figure 31 we see the training loop, lines 299 – 300 a random selection of real data is selected, this is half the training batch as the other half Is used to create noise for the generator to generate fake data from as seen on line 303 – 304. Line 315 shows the generator loss (g_loss) as the result of generator training loss, this is appended for later graphical representation along with the discriminator loss (d_loss). This training loop is completed to 5000 loops or epochs and progress is checked every 250 epochs to get an indication of training performance.



```python
392 for k, g in zip(label_list, num_to_gen):
393     real_data = training_data.loc[training_data['label'] == k]
394 
395     real_data = real_data.sample(n=percentage(0.40, real_data.shape[0]))
396 
397     gan, generator, discriminator, latent_dim = gan_system()
398 
399     train(gan, generator, discriminator, real_data, latent_dim, k, n_epochs=5000,
400           n_batch=5000, n_eval=250)
401 
402     noise = np.random.normal(0, 1, (g, latent_dim))
403     gen_data = generator.predict(noise)
404 
405     generated_numerical_data = min_max_scaler.inverse_transform(gen_data)
406 
407     data_generated = to_df(generated_numerical_data)
408     data_generated['label'] = k
409 
410     y_data = data_generated['label']
411     X_data = data_generated.drop(columns=['label'])
412 
413     y_data = y_df(y_data)
414 
415     fake_data = fake_data.append(X_data, ignore_index=True)
416     fake_labels = fake_labels.append(y_data, ignore_index=True)
```

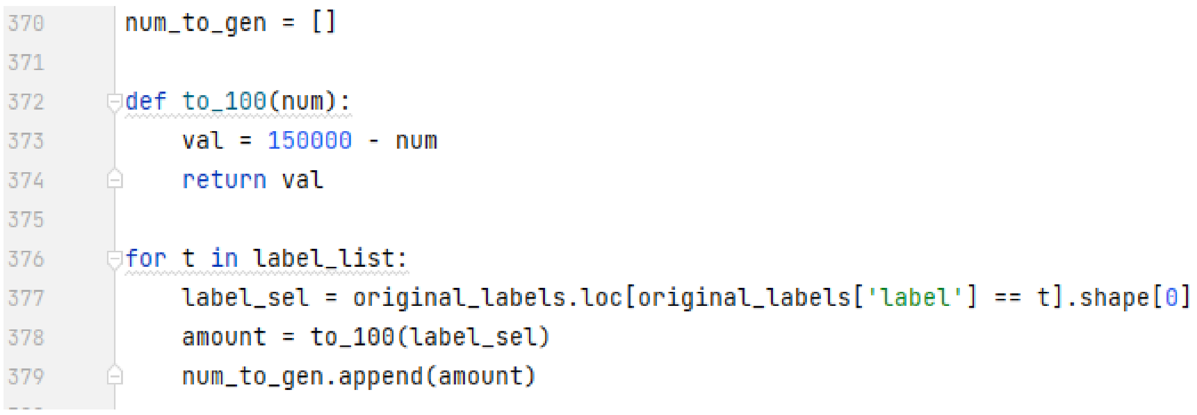
Figure 32 - label synthesising code

Figure 32 shows the code I wrote. It is the loop where real data is fed to the GAN and the trained generator *G* is used to generate the synthetic data, empty dataframes are created for the fake data and fake labels to be appended to upon creation. Specifically, lines 386 and 387 are variables of empty lists, and lines 389 and 390 turn those empty list variables into a dataframe with the required column names using special functions shown in figure 33 on lines 117 to 127. Line 392 is the beginning of the for loop where the program loops through the label list and selects data from the training set according to the encoded label type. For example, if k = 1 then the sample with the encoded label 1 is selected as "real data", from here the function in figure 32 is called on line 395 where the percentage of training data is randomly selected from the set of real training data, the example shown in figure 32 is the 40% selection variant. I decided to have each separate file working on a different percentage selection to save on computation time as I am not looking at the time taken to perform all scenarios incrementally.



```python
def to_df(data_to_df):
    data = pd.DataFrame(data_to_df, columns=list(X.columns))

    data.astype('int32').dtypes

    return data

def y_df(data):
    data = pd.DataFrame(data, columns=['label'])
    data.astype('int32').dtypes
    return data
```

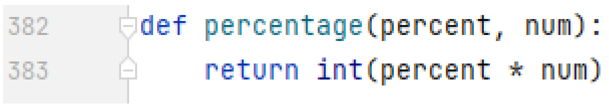
Figure 33 - Converting to pandas dataframe



```python
370  num_to_gen = []
371
372  def to_100(num):
373      val = 150000 - num
374      return val
375
376  for t in label_list:
377      label_sel = original_labels.loc[original_labels['label'] == t].shape[0]
378      amount = to_100(label_sel)
379      num_to_gen.append(amount)
```

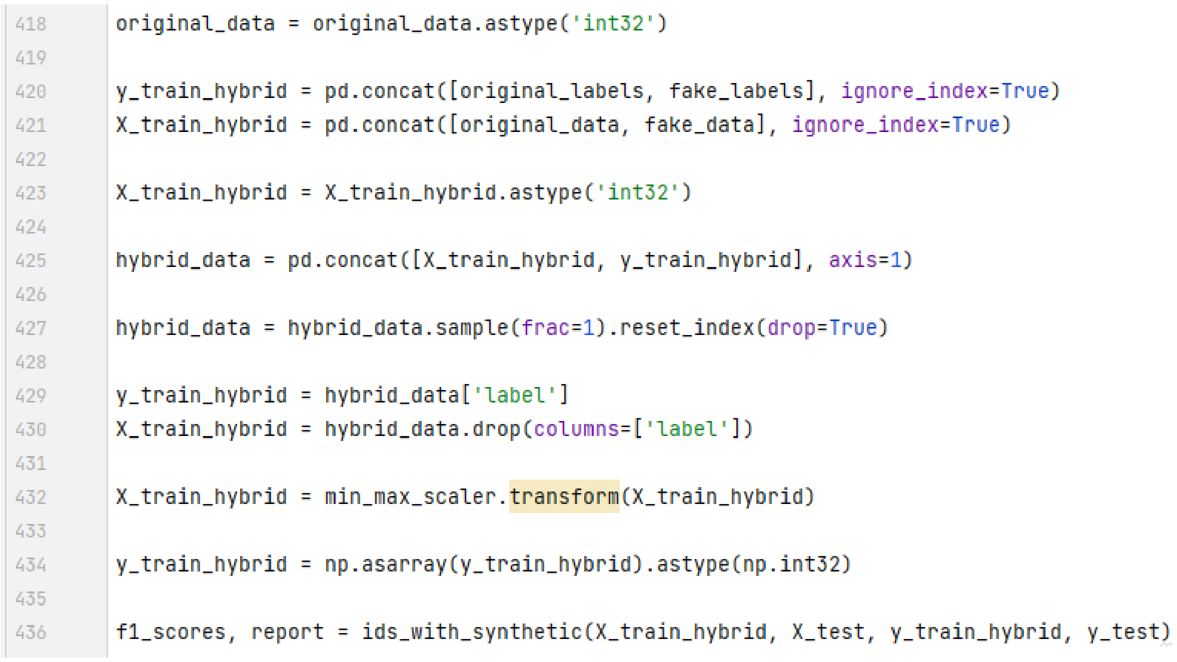
Figure 34 - Function to define number of samples to generate

The GAN training is completed in a loop, each class is trained on the GAN and once the training is complete for a single class the trained generator is called to generate enough synthetic samples to bring the class distribution to be almost equal that of the second most occurring class in the dataset i.e., 150,000. Line 402 generates the random noise vector to be fed to the generator. Similar research by Abdalgawad *et al* (2020) oversampled the minority classes to be equal to that of the majority class to balance the class distribution. Similarly, I will oversample the minority classes to be almost equal to that of second majority class, this can be seen in figure 34 on lines 372 to 379 where the required number to be generated is calculated and appended to the 'num_to_gen' list. The noise vector is fed through the trained generator on line 403 and results in synthetic data based off the current label in label list, from here the synthetic data is in an array and to append it to the original training data we need to reverse transform the array into a dataframe as shown on line 405 to 407. Line 408 a new column is temporarily added to assign the synthetic data the correct label, from here the data is separated into synthetic training and synthetic label data as X_data and y_data respectively. Finally, the generated data (X_data) and their assigned variables (y_data) are appended to the growing fake data and fake label dataframes. This process is repeated until all selected labels in label list has been generated and their synthetic samples are held in the fake data and fake label variables.



```python
def percentage(percent, num):
    return int(percent * num)
```

Figure 35 - percentage function



```python
418 original_data = original_data.astype('int32')
419
420 y_train_hybrid = pd.concat([original_labels, fake_labels], ignore_index=True)
421 X_train_hybrid = pd.concat([original_data, fake_data], ignore_index=True)
422
423 X_train_hybrid = X_train_hybrid.astype('int32')
424
425 hybrid_data = pd.concat([X_train_hybrid, y_train_hybrid], axis=1)
426
427 hybrid_data = hybrid_data.sample(frac=1).reset_index(drop=True)
428
429 y_train_hybrid = hybrid_data['label']
430 X_train_hybrid = hybrid_data.drop(columns=['label'])
431
432 X_train_hybrid = min_max_scaler.transform(X_train_hybrid)
433
434 y_train_hybrid = np.asarray(y_train_hybrid).astype(np.int32)
435
436 f1_scores, report = ids_with_synthetic(X_train_hybrid, X_test, y_train_hybrid, y_test)
```

Figure 36 - Augment the training data code

Finally looking to figure 36 here the code is written by me, and we see the augmented training data (X_train_hybrid) and augmented label data (y_train_hybrid) being created through concatenating the original labels with fake labels and original data with fake data, setting ignore index to reset the index of the new dataframe. Line 416 is the concatenation of the feature data and label data into one variable "hybrid data" because we need to reshuffle the dataframe to make the distribution of samples random because currently it is the original training data and then X lots of 150,000 examples of the synthetic data which would affect the IDS classification results and this needs to be done as one instance to ensure the index information for both label and feature data still correspond correctly. Once randomised the feature data is split away from the label data back into 'X_train_hybrid' and 'y_train_hybrid'. The feature data is transformed back into an array and scaled using min_max and the label data is transformed into an array. Now the IDS is trained on the augmented training data as shown on line 436.

The pseudocode describes the flow of the training sequence, this can be seen in figure 37.

```
Data generation
Input: real data
Output: synthetic data

For k in label list and g in num_to_gen
    h = Set % of data to select
    real_data = k label in label list
    randomly select h% of real data
    for ( j = 0 to j = 5000) in range 5000
        create g noise
        generate fake samples from noise
        train discriminator on real and fake data
        update discriminator
        update generator
    end for
    generate data using z samples of noise
    append synthetic data to training data with label k
end for
train IDS on augmented dataset
```

Figure 37 - pseudocode for data generation

3.10. Methods of evaluation

In order to evaluate the differences in classification performance, I use a combination of classification report, confusion matrix and plotted model loss, the main metric of evaluation is the F1 score, really, I am only looking to improve the individual F1 score of each class as I am measuring the improvement in individual classification performance given the increase in class size, however for the overall F1 score I decided upon macro averaging as it treats each class equally, which in the case of an IDS is important since no one class is more important than another as they are all equally important to be correctly classified

All samples being synthetically increased will be increased to 150,000 samples, this is to create a balanced dataset for classification testing, the difference will be the number of real samples the GAN is trained on. It is a similar setup to that used by Shahriar *et al* (2020) where a fixed number of samples are generated by the gan, however the data in which the GAN is trained on is increased by percentage of the training set i.e., 20%, 40%, 60%, 80% and 100%, this is to assess the GAN's performance when subject to different amount of real data to learn from.

Evaluation metrics

To assess the quality of results and how well the algorithm is in its predictions, in my project I make use of confusion matrixes and classification reports. A confusion matrix allows us to visualise the correct and misclassification results of each sample class; put simply the results can be represented by figure 38, the example shown is for binary classification of whether the model correctly predicted if the datapoint was regular datapoint or an attack datapoint.

	Predicted Benign	*Predicted Attack*
Benign actual	True Positive (TP)	False Negative (FN)
Attack actual	False Positive (FP)	True Negative (TN)

Figure 38 - confusion matrix

Accuracy is the value of correctly predicted datapoints, divided by the total number of classifications, although accuracy can fail to describe the true classification performance of the algorithm if, like in our case, it is using a highly imbalanced class dataset (Jason Brownlee, 2020)

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Figure 39 - Accuracy formula

Recall is the measure that tells us the number of predictions that were correct over the total number of class occurrences. Recall is about capturing all required classes correctly, for example if our classifier is capturing all attack datapoints but still misclassifying some benign data and attack data then we can say our recall is 1 because all required class instances have been captured.

$$Recall = \frac{TP}{TP + FN}$$

Figure 40 - recall formula

F1 is the balance between precision and recall, regarded as the 'harmonic mean' in an article by Thomas Wood (no date),

$$F1 = \frac{2 \text{ x precision x recall}}{\text{precision+recall}}$$

Figure 41 - F1 score formula

Chapter 4: Results

This section of the project outlines the findings of the research, it is broken down in the following sections

- Summary of results
- Initial intrusion detection ANN results
- Sorting algorithm samples selected
- Intrusion detection results from training on the augmented training set

Results summary

A summary of the overall classifier results of all different percentage conditions that we used to train the GAN for data generation can be seen in the table 4.

		IDS-GAN	IDS-GAN	IDS-GAN	IDS-GAN	IDS-GAN
	IDS	**20%**	**40%**	**60%**	**80%**	**100%**
Overall F1	0.74	0.80	0.80	0.80	0.79	0.83
Benign F1	0.62	0.63	0.60	0.59	0.63	0.63
C&C F1	0.20	0.47	0.46	0.45	0.55	0.54
C&CHeartbeat F1	0.48	0.64	0.70	0.73	0.48	0.77

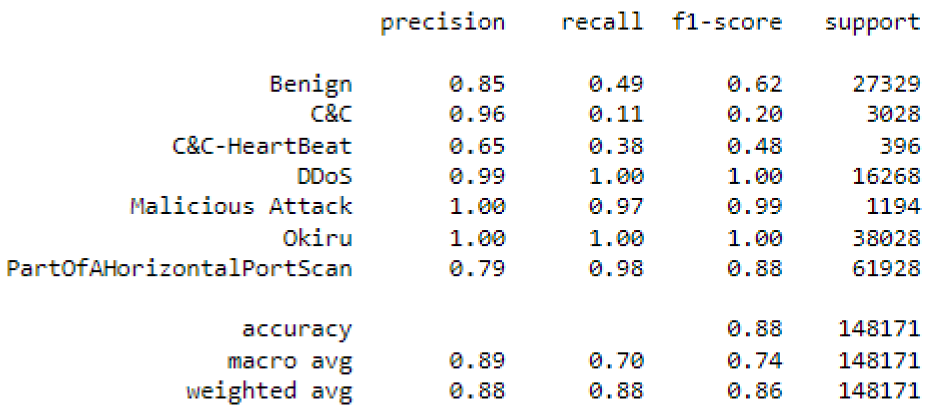
Table 4 - Results overview

Initial intrusion detection ANN classification results

As outlined in the methods section 3.6 the first stage following dataset processing is testing the ANNs classification performance on the original non-augmented dataset. To recap, the selected feature set the model is trained on is shown in table 5 below:

id.orig_p	id.resp_h	id.resp_p	proto	service	duration
org_bytes	‘resp_bytes	conn_state	missed_bytes	orig_pkts	resp_pks

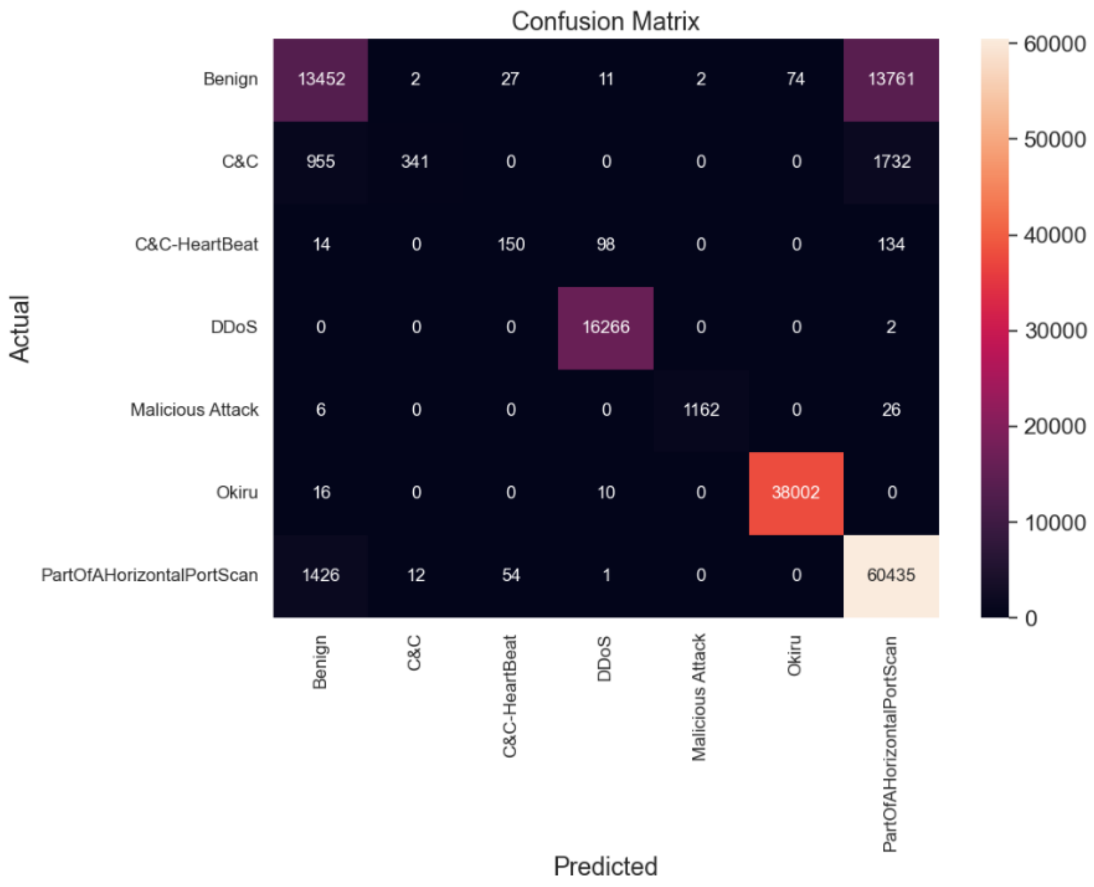
Table 5 - IoT23 feature set



```
                          precision    recall  f1-score   support

                   Benign       0.85      0.49      0.62     27329
                      C&C       0.96      0.11      0.20      3028
            C&C-HeartBeat       0.65      0.38      0.48       396
                     DDoS       0.99      1.00      1.00     16268
         Malicious Attack       1.00      0.97      0.99      1194
                    Okiru       1.00      1.00      1.00     38028
PartOfAHorizontalPortScan       0.79      0.98      0.88     61928

                 accuracy                           0.88    148171
                macro avg       0.89      0.70      0.74    148171
             weighted avg       0.88      0.88      0.86    148171
```

Figure 42 - baseline classification report

Figure 42 is the base line results of a standalone IDS with no data augmentation methods applied. Looking to the classification report we can see three classes that fall below the 0.70 F1-score: Benign, C&C and C&C-Heartbeat, these classes are selected to be synthetically generated.



Actual \ Predicted	Benign	C&C	C&C-HeartBeat	DDoS	Malicious Attack	Okiru	PartOfAHorizontalPortScan
Benign	13452	2	27	11	2	74	13761
C&C	955	341	0	0	0	0	1732
C&C-HeartBeat	14	0	150	98	0	0	134
DDoS	0	0	0	16266	0	0	2
Malicious Attack	6	0	0	0	1162	0	26
Okiru	16	0	0	10	0	38002	0
PartOfAHorizontalPortScan	1426	12	54	1	0	0	60435

Figure 43 - baseline confusion matrix

If we look to the confusion matrix shown in figure 43, we can see the performance of the classifier by samples correctly classified and samples incorrectly classified. We can observe how class imbalance is affecting the results of the IDS, the first observation is the model's bias towards the majority class. 'PartOfAhorizontalPortScan' being the class with the most instances in the dataset we see that the poorly classified classes are mainly misclassified as being 'PartOfAhorizontalPortScan', 57% of the C&C test samples are classified 'PartOfAhorizontalPortScan' with only 11% being correctly classified as C&C. Interestingly we see that the remaining 32% of the C&C test set being misclassified as Benign, I will touch on this observation later in the discussion section.

'C&C heartbeat' has the second lowest F1 score of 0.48; only 38% of the samples were correctly classified again we see most of the misclassifications for 'C&C heartbeat' being 'PartOfAhorizontalPortScan'.

Finally, the 'Benign' class which represents normal flow of data while not being on the extreme end of class imbalance, it is the third most populated class within the training set however it has the third lowest F1 score falling just under the threshold and thus being selected for generation. We see that just under half 49% of the test set is correctly classified with the majority of the misclassification again being 'PartOfAhorizontalPortScan' with small amounts being incorrectly classified as the other classes, this is the only class that has incorrect classifications as all of the other classes, in a normal IDS operating environment these would be known as false positives i.e., regular network traffic flagging as malicious which is the largest current draw back of a purely machine learning bases IDS.

Sorting stage

Following section 3.6 in the methods chapter the F1 results are currently in a list variable. To use the F1 results, the list is fed into a simple sorting function which is outlined in section 3.9. The sorting function iterates through the list looking for F1 scores that fall below the minimum F1 score requirement (0.70). Based off the initial classification results the classes that are selected are for synthetic generation are shown in table 6.

Label	F1 result	Original training set size
Benign	0.62	109313
C&C	0.20	12114
C&C Heartbeat	0.48	1585

Table 6 - labels selected following sorting function

Intrusion detection results from training on the augmented training set

The following set of results are broken into the 5 training scenarios; each of the scenarios are the same except for the percentage of data that is used to train the GAN, all models are based off the generator framework in section 3.7.1 and the discriminator framework in section 3.7.2. The training procedure is the same throughout and is outlined in section 3.9. A summary of the different training scenarios is described in table XX

	Percentage of training data fed to the GAN				
Label	**20%**	**40%**	**60%**	**80%**	**100%**
Benign	21862	43725	65588	87450	109313
C&C	2422	4845	7268	9691	12114
C&C Heartbeat	317	634	951	1268	1585

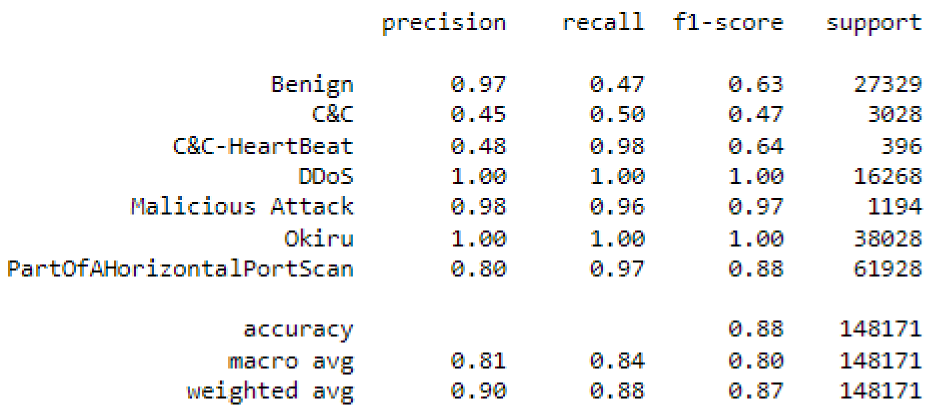
Table 7 - Data distributions by percent

GAN trained on 20% of selected class data

Label	**F1 result with 20% training data**	**Original F1 result without GAN**	**Percent difference**
Benign	0.63	0.62	-1%
C&C	0.47	0.20	+27%
C&C Heartbeat	0.64	0.48	+16%

Table 8 - Results summary for 20% scenario

This portion relates to the scenario where the GAN is exposed to just 20% of the training data for each of the select classes, a summary of the results can be seen in table 8. Firstly, looking to figure 44 we can see that the overall macro F1 score has risen from 0.74 to 0.80. For the individual classes benign rose from 0.62 F1 score to 0.63 which is a relatively small amount and looking to the confusion matrix we can see that the number of correct classifications or true positives of benign classification fell as did the number of false positives, however this means that the number of false negatives increased as seen in figure 45. This could be attributed to the number of original benign results being similar in size to the synthetic. Since there are more real samples for the IDS to distinguish as being true positive it was better at distinguishing between the synthetic examples and the real examples.



```
                          precision    recall  f1-score   support

                 Benign       0.97      0.47      0.63     27329
                    C&C       0.45      0.50      0.47      3028
          C&C-HeartBeat       0.48      0.98      0.64       396
                   DDoS       1.00      1.00      1.00     16268
       Malicious Attack       0.98      0.96      0.97      1194
                  Okiru       1.00      1.00      1.00     38028
PartOfAHorizontalPortScan       0.80      0.97      0.88     61928

               accuracy                           0.88    148171
              macro avg       0.81      0.84      0.80    148171
           weighted avg       0.90      0.88      0.87    148171
```

Figure 44 - Classification report 20% training

As for the second-class C&C we observe that the F1 score almost doubled from 0.20 to 0.47, likewise in the confusion matrix we can see that the number of correct classifications rose significantly from 341 to 1680. While this can be seen as a success since more examples of a possible connection to a command-and-control server are being detected there is also a rise in the number of C&C classes being classified as ‘benign’. This would be an issue in a real-world scenario as it would have a higher chance of bypassing an IDS system since the model thinks it is regular network traffic. Finally, for ‘C&C heartbeat’ the F1-score of this class rose from 0.48 to 0.64 and this can be attributed to the increased number of true positive predictions which increased from 150 to 379. While the number of correct classifications rose there is a small increase in the number of ‘C&C heartbeat’ samples being miss identified as Benign.

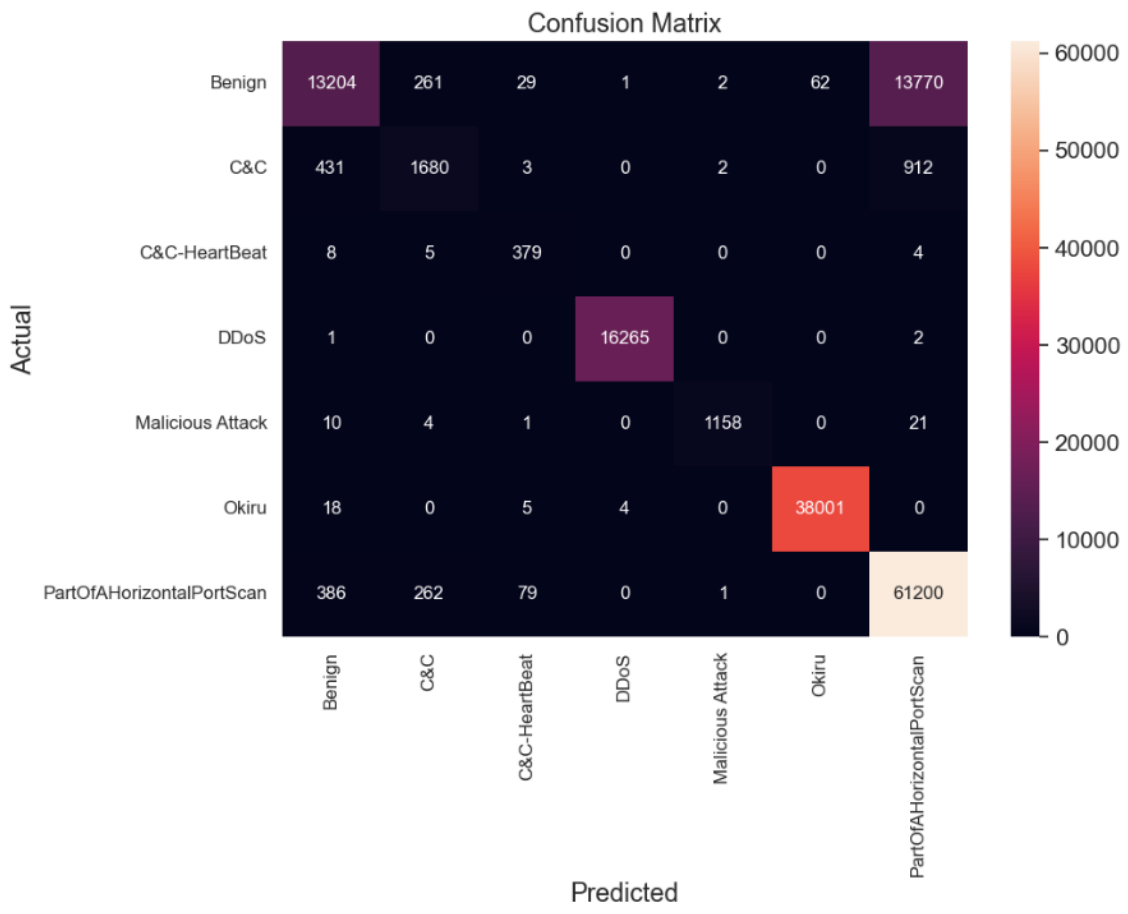


Figure 45 - Confusion matrix 20% training

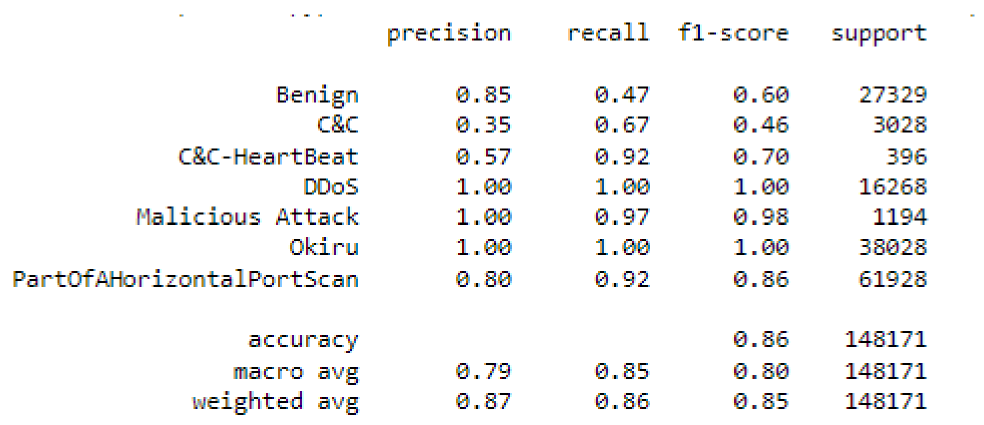
GAN trained on 40% of selected class data

Label	F1 result with 40% training data	Original F1 result without GAN	Percent difference
Benign	0.60	0.62	-2%
C&C	0.46	0.20	+26%
C&C Heartbeat	0.70	0.48	+22%

Table 9 - Results summary for 40% scenario

The following results are in reference to 40% of the original data being used for GAN training. A summary of the results can be seen in table 9. Looking to the classification report in figure 46, we can see that the overall macro F1 rose from 0.74 to 0.80 which is the same rise as the previous example of 20%. Looking to the 'benign' class we see there has been a fall in the F1 score, this can be further identified in figure 47 which shows the confusion matrix results, we see that the number of correct

‘Benign’ classifications fell from 13452 to 12723, this is not a huge drop however given the objectives of this project it is a point of concern, moreover we see a rise in the number of ‘PartOfAHorizontalPortScan’ being identified as benign. A gain this is concerning if the IDS was in a real-world environment. Interestingly we can see that the number of false positives fell when looking at the number of predicted Benign samples being identified as C&C samples, a drop from 955 to 145.



```
                           precision    recall  f1-score   support

                   Benign       0.85      0.47      0.60     27329
                      C&C       0.35      0.67      0.46      3028
            C&C-HeartBeat       0.57      0.92      0.70       396
                     DDoS       1.00      1.00      1.00     16268
         Malicious Attack       1.00      0.97      0.98      1194
                    Okiru       1.00      1.00      1.00     38028
PartOfAHorizontalPortScan       0.80      0.92      0.86     61928

                 accuracy                           0.86    148171
                macro avg       0.79      0.85      0.80    148171
             weighted avg       0.87      0.86      0.85    148171
```

Figure 46 - Classification report 40% training

As for the C&C class in this case we observe that the F1 score rose from 0.20 to 0.46, however it is a slight decrease from the previous example where the F1 score was 0.46. While there was a decrease, we can see from figure 47 that the number of correct classifications rose sharply from 341 to 2039. An interesting observation from this set of results is that the number of samples misclassified as C&C rose. ‘PartOfAHorizontalPortScan’ and “Benign” were increasingly misclassified from 12 to 2698 and 2 to 1119. Finally, for the class ‘C&C Heartbeat’ there was an increase in F1 from 0.48 to 0.70 and correct classifications from 150 to 363. Again, we see a rise in the number of ‘PartOfAHorizontalPortScan’ being misclassified as ‘C&C Heartbeat’.

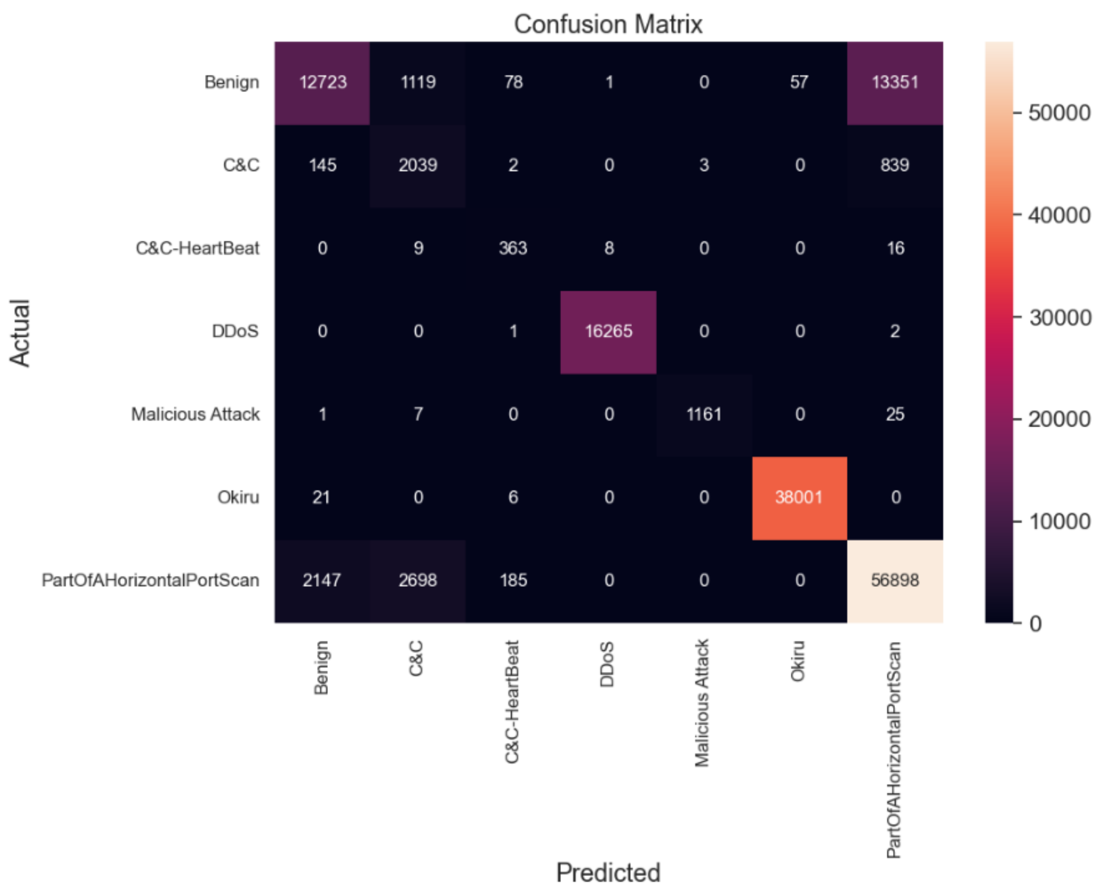


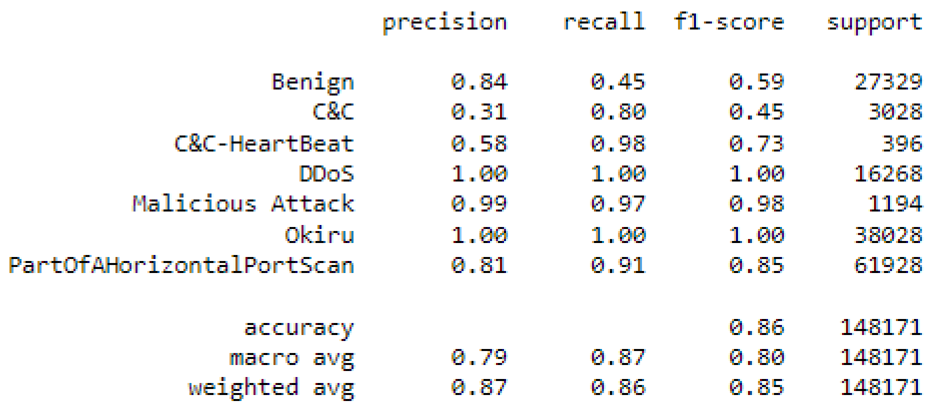
Figure 47 - Confusion matrix 40% training

GAN trained on 60% of selected class data

Label	F1 result with 60% training data	Original F1 result without GAN	Percent difference
Benign	0.59	0.62	-3%
C&C	0.45	0.20	+25%
C&C Heartbeat	0.73	0.48	+25%

Table 10 - Summary of 60% scenario

The following results are from the Gan trained on 60% of the original class data. Looking to figure 48, at first glance we see there is an increase in the overall macro F1 score from 0.74 to 0.80 which is a similar rise to the previous examples. Regarding the classes, like the previous set of results a fall in the F1 score for the 'Benign' class and as previously a rise in the number of 'Benign' samples being misclassified as 'PartOfAHorizontalPortScan'. Looking to figure 49 there is a slight drop in the number of 'PartOfAHorizontalPortScan' samples being misidentified as 'Benign'. As for 'C&C' we see an increase in F1 from 0.20 to 0.45 and an increased number of 'C&C' samples being correctly classified as such, interestingly there is also a drop in the number of 'C&C' samples being incorrectly identified as benign however there is a significant increase in the number of ‘C&C’ samples being misclassified as ‘PartOfAHorizontalPortScan’.



```
                           precision    recall  f1-score   support

                   Benign       0.84      0.45      0.59     27329
                      C&C       0.31      0.80      0.45      3028
            C&C-HeartBeat       0.58      0.98      0.73       396
                     DDoS       1.00      1.00      1.00     16268
         Malicious Attack       0.99      0.97      0.98      1194
                    Okiru       1.00      1.00      1.00     38028
PartOfAHorizontalPortScan       0.81      0.91      0.85     61928

                 accuracy                           0.86    148171
                macro avg       0.79      0.87      0.80    148171
             weighted avg       0.87      0.86      0.85    148171
```

Figure 48 - Classification report 60% training

Finally looking at ‘C&C Heartbeat’ results we can see there is again an increase in the F1 score from 0.48 to 0.73 which is the highest increase out of the scenarios and again we see another increase in the number of correctly classified samples for this class from 150 to 378. In fact there were only 9 samples missed and incorrectly classed, these were classified as ‘C&C’. As with the other scenarios there was a rise in the number of Benign samples misclassified as the class of current observation as in this case it is ‘C&C Heartbeat’.

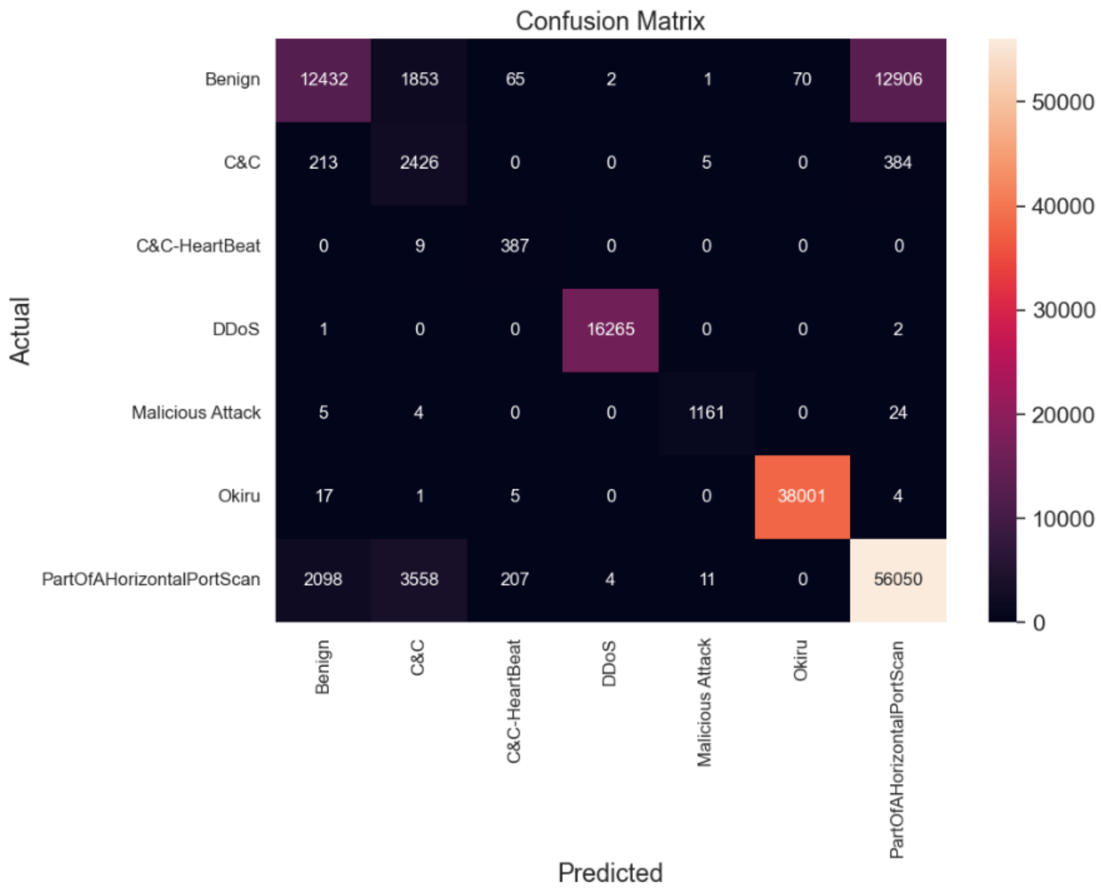


Figure 49 - Confusion matrix 60% training

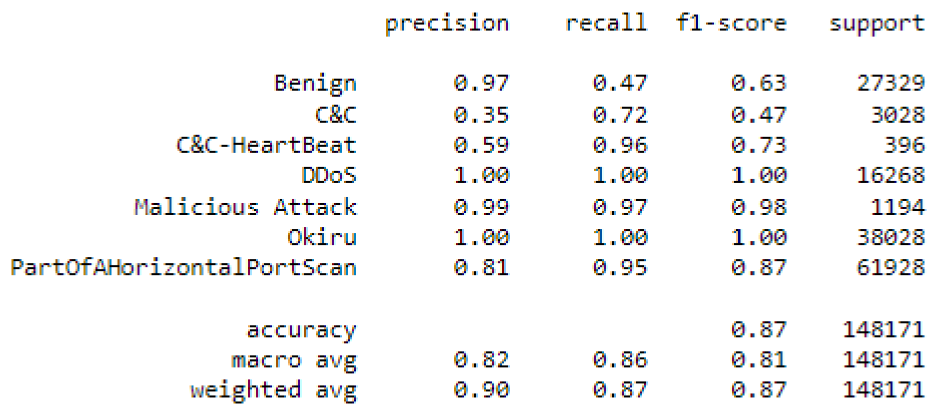
GAN trained on 80% of selected class data

Label	F1 result with 80% training data	Original F1 result without GAN	Percent difference
Benign	0.63	0.62	+1%
C&C	0.47	0.20	+27%
C&C Heartbeat	0.73	0.48	+25%

Table 11 – Results summary for 80% scenario

For the training scenario of 80% training data available to the GAN we can see there has been an improvement in the overall macro F1 performance. This rise from 0.74 to 0.81 is seen throughout the scenarios tested. As for the first class 'Benign' we observe a small improvement in the F1 score compared to the original or 0.62, which is in contradiction to the falling F1 scores as seen in previous scenarios. However, we are again seeing an increase in the number of majority class samples being misclassed as 'Benign' however regarding the other classes there is a drop in the number of malware classes being misidentified as 'Benign'.

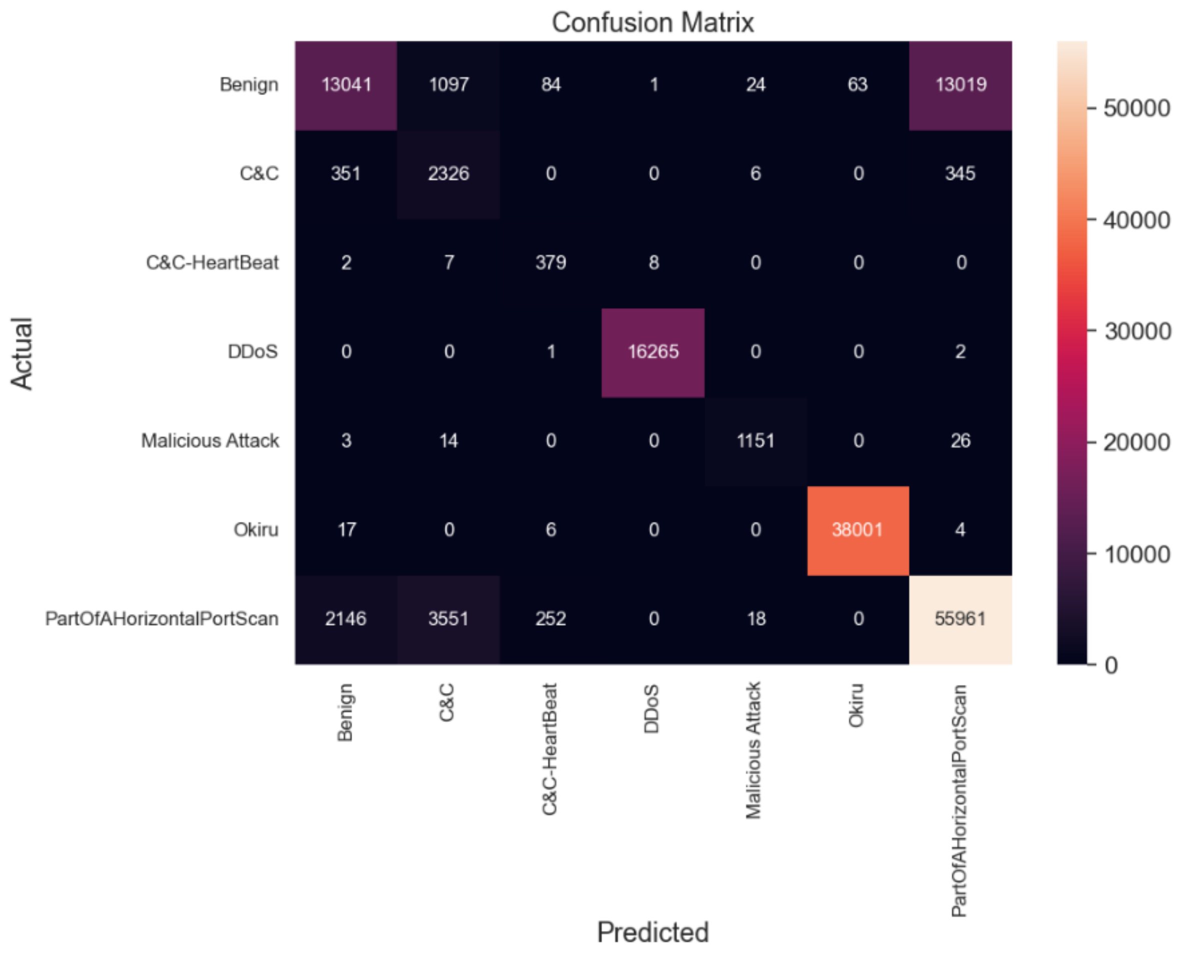
For the C&C class in this scenario we see an increase in F1 from 0.20 to 0.47 and again a large increase in the number of samples being correctly classified. As shown in figure 50 and 51, we observe the number of correct classifications to be 341 in the baseline report and 2326 in the 80% training data scenario. However, the number of samples classed as being C&C when they were part of the majority class ‘PartOfAHorizontalPortScan’ rose from 12 in the baseline report to 3551 in the 80% scenario. Similarly, there was a rise in the number of ‘Benign’ samples being misclassified as ‘C&C’ from 2 in the baseline report to 1097 in the 80% scenario.



```
                           precision    recall  f1-score   support

                   Benign       0.97      0.47      0.63     27329
                      C&C       0.35      0.72      0.47      3028
             C&C-HeartBeat       0.59      0.96      0.73       396
                     DDoS       1.00      1.00      1.00     16268
         Malicious Attack       0.99      0.97      0.98      1194
                    Okiru       1.00      1.00      1.00     38028
PartOfAHorizontalPortScan       0.81      0.95      0.87     61928

                 accuracy                           0.87    148171
                macro avg       0.82      0.86      0.81    148171
             weighted avg       0.90      0.87      0.87    148171
```

Figure 50 - Classification report 80% training

Finally for the class ‘C&C Heartbeat’ there is a rise in F1 like that of the 60% scenario.



Actual \ Predicted	Benign	C&C	C&C-HeartBeat	DDoS	Malicious Attack	Okiru	PartOfAHorizontalPortScan
Benign	13041	1097	84	1	24	63	13019
C&C	351	2326	0	0	6	0	345
C&C-HeartBeat	2	7	379	8	0	0	0
DDoS	0	0	1	16265	0	0	2
Malicious Attack	3	14	0	0	1151	0	26
Okiru	17	0	6	0	0	38001	4
PartOfAHorizontalPortScan	2146	3551	252	0	18	0	55961

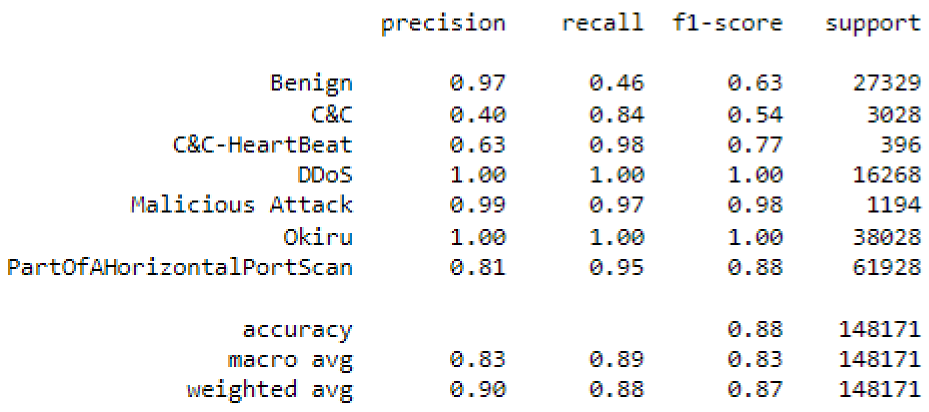
Figure 51 - Confusion matrix 80% training

GAN trained on 100% of selected class data

Label	F1 result with 100% training data	Original F1 result without GAN	Percent difference
Benign	0.63	0.62	+1%
C&C	0.54	0.20	+34%
C&C Heartbeat	0.77	0.48	+29%

Table 12 - Results summary for 100% scenario

Finally, the last scenario explored is the GAN trained on 100% of the available selected class data, we firstly observed that the macro F1 score rose from 0.74 to 0.83 as seen in Figure 52 which was the highest change of all scenarios for the macro average F1. The F1 score for 'Benign' rose for this scenario to 0.63, however we can see that correct classifications fell from 13452 to 12601 with most ‘Benign’ samples being misclassified as ‘PartOfAHorizontalPortScan’. The C&C class saw the largest increase in F1 out of all the tests, a rise from 0.20 to 0.54 with the most correct classifications out of all the completed tests, another observation is that far less C&C classes were misclassified as the majority class however there was a significant rise in the number of ‘C&C’ classes being misclassified as ‘Benign’, in the baseline results just 2 samples were incorrectly classified at ‘Benign’ however in this scenario 1337 samples were classed ‘Benign’.



```
                           precision    recall  f1-score   support

                   Benign       0.97      0.46      0.63     27329
                      C&C       0.40      0.84      0.54      3028
            C&C-HeartBeat       0.63      0.98      0.77       396
                     DDoS       1.00      1.00      1.00     16268
         Malicious Attack       0.99      0.97      0.98      1194
                    Okiru       1.00      1.00      1.00     38028
PartOfAHorizontalPortScan       0.81      0.95      0.88     61928

                 accuracy                           0.88    148171
                macro avg       0.83      0.89      0.83    148171
             weighted avg       0.90      0.88      0.87    148171
```

Figure 52 - Classification report 100% training

For the C&C Heartbeat class we observe an increase in F1 to the highest out of all scenarios of, a rise from 0.48 to 0.77 with the highest number of correct classifications out of the tested scenarios, there was also a drop in the number of ‘C&C heartbeat samples being misidentified as the majority class.

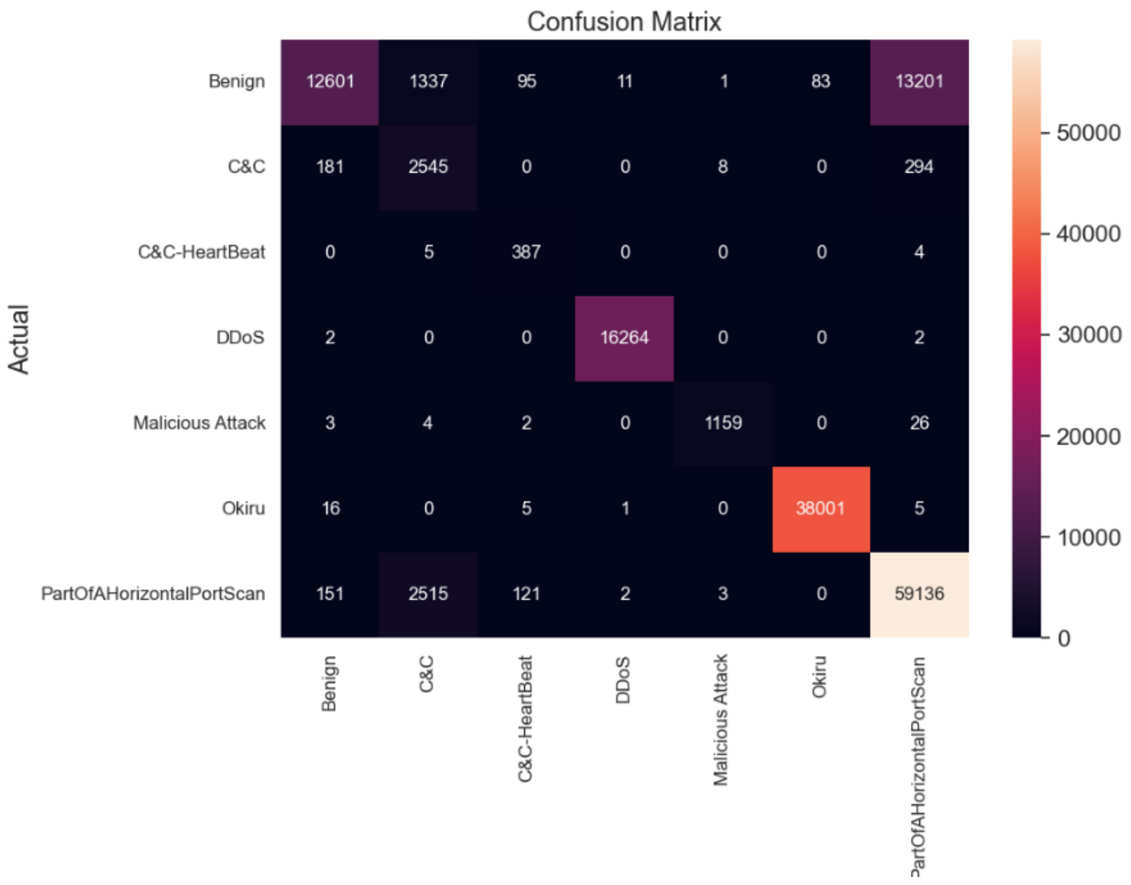


Figure 53 - Confusion matrix 100% training

Conclusion of results and further observations.

As an overall conclusion for my project, I would say that I am pleased with the results. From the different testing scenarios, I observed there was a clear correlation between the size of the label training set that was used by the GAN to train the generator, for instance on the 20% train set we saw an overall improvement macro F1 by 6% whereas on the 100% set we saw an overall improvement of macro F1 of 11% which is relatively significate given the imbalance of the training set. Likewise, the largest improvement of a single class was ‘C&C’, as expected from the 100% scenario which saw an increase of 25%.

Another observation I made during this project was that in every iteration of development, the best i.e., lowest loss for the generator and highest loss for the discriminator was around 0.70. At this stage both segments of the model reached equilibrium where their performance was relatively similar. Through different design iterations I found that I could easily make the discriminator perform far better than the generator i.e., the discriminator was very good at discerning which sample was real and which was fake, however this was obviously at a cost to the generator’s performance. I found the harmonic balance between the two was around 0.7. Shown in figures 54 and 56 are two plots of the loss incurred by the generator *G* and discriminator *D*. We see that while the target label is different and the percentage of training data supplied to the GAN is different, the balance between the two is shared at 0.7 and this was see through all plots of loss for every scenario.

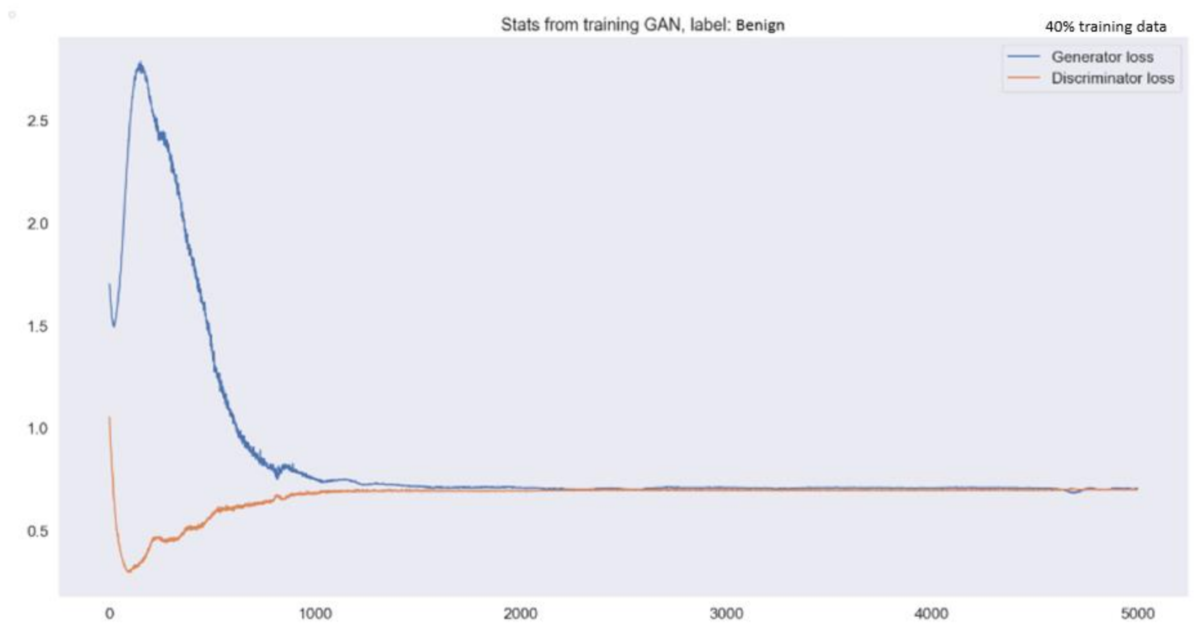


Figure 54 - training training loss 40% training

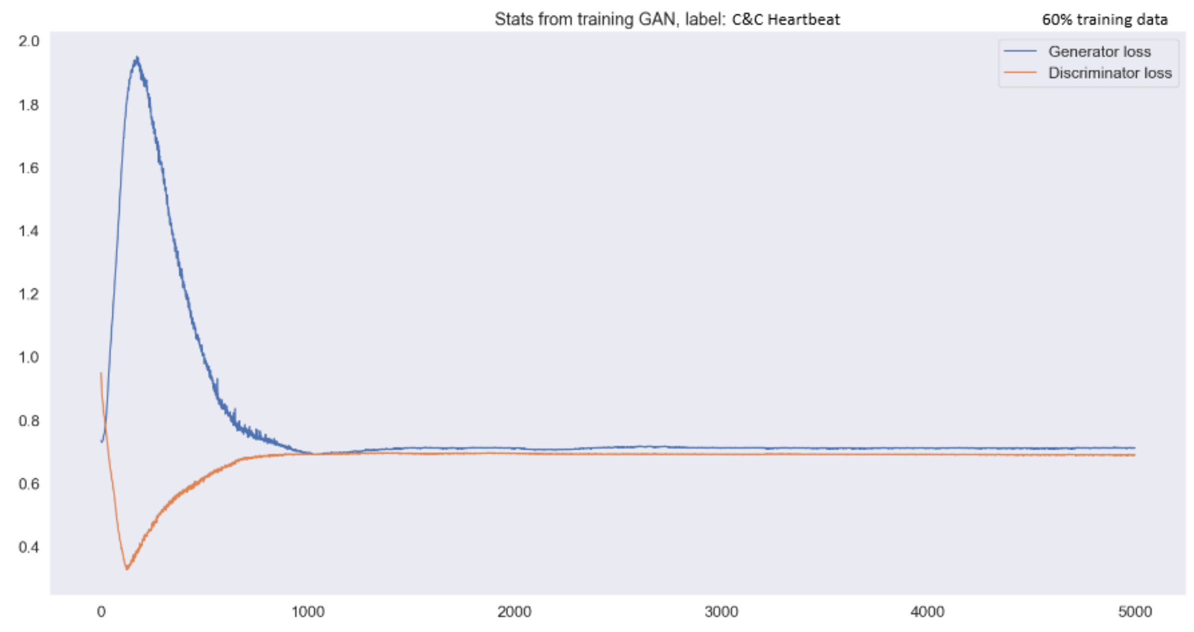


Figure 55 - training loss 60% training

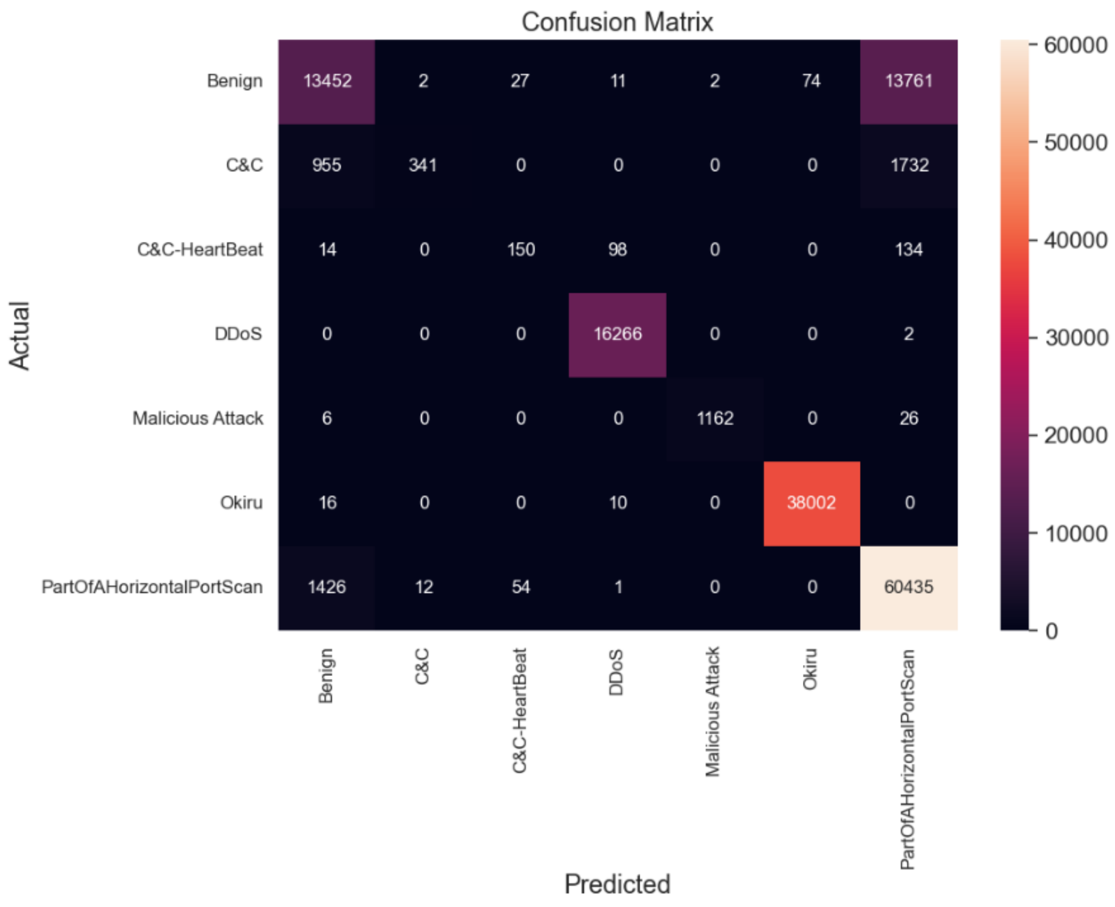


Figure 56 - baseline confusion matrix

The final observation I want to make is an issue I have found during analysis of the classification results. If we refer to the original confusion matrix as shown in figure 56, we can see that the number of classifications that were predicted ‘Benign’ that were actually ‘C&C’ as well as actually ‘C&C Heartbeat’ is relatively low which is something most IDS systems hope to achieve, as having false positives can be a nuisance. However, if we now look to figure 57 which shows the confusion matrix from the 100% training set scenario, we see the number of predicted ‘C&C’ when the samples were ‘Benign’ has risen significantly from 2 in the baseline results to 1853 in the 60% results. What I suspect is that the data generated by the GAN for the Benign class is not sufficiently representative of the real data of the class. This theory is backed up by the little to no improvement of the ‘Benign’ classified results over all the scenarios hence when it came to classifying the other classes, we see increased amounts of ‘C&C’ and ‘C&C heartbeat’ classified as ‘Benign’ since the IDS is having trouble distinguishing between ‘Benign’ and the other synthetically augmented classes. This theory may indicate that the IDS is identifying a similar pattern shared between the synthetic data samples of each class. While this is a disappointing outcome of the results it is quite interesting that even though the GAN was trained independently for each label set there is still a way of identifying of synthetic samples despite being trained off data of a different class.

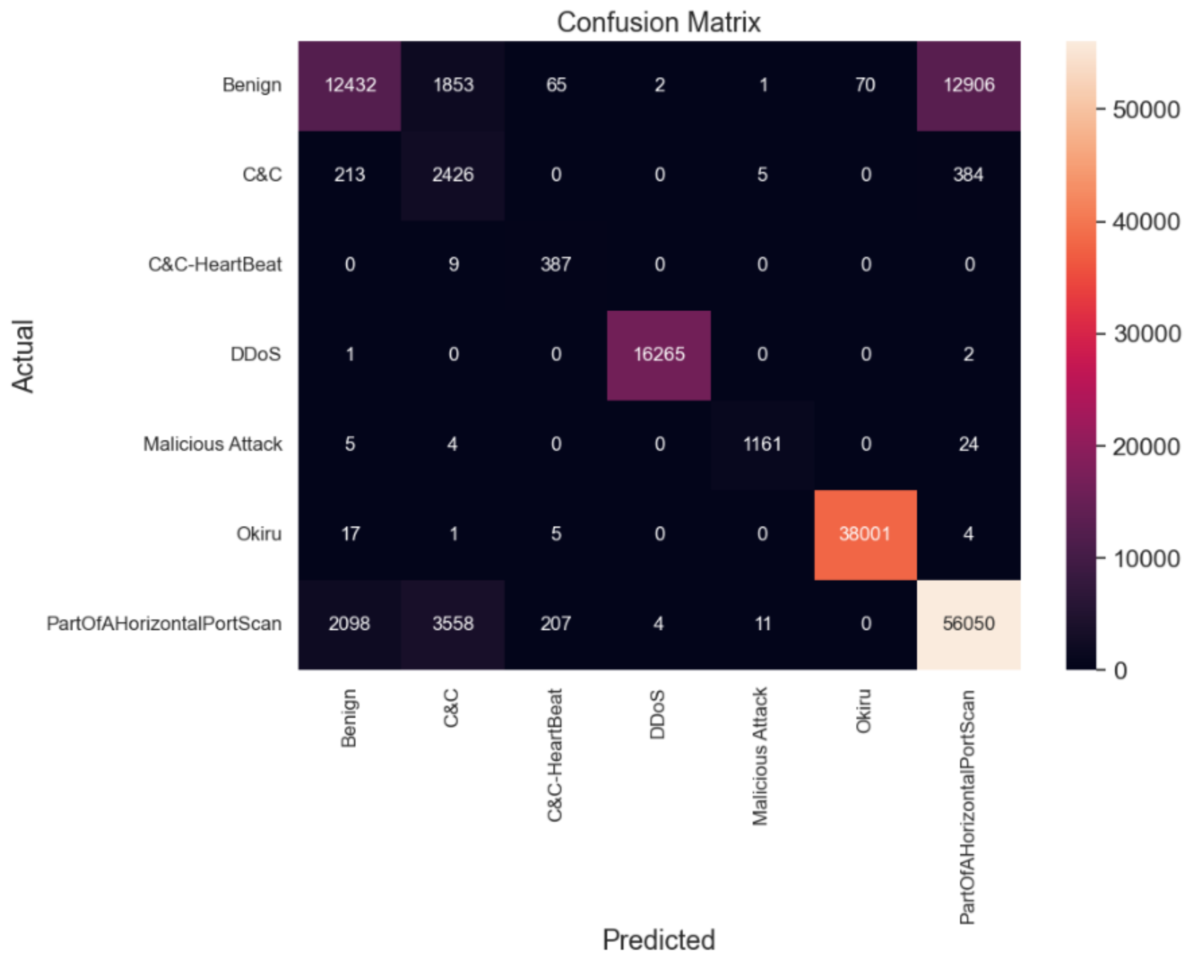


Figure 57 - confusion matrix 100%

Finally, while I have not built a custom GAN from the ground up, I have shown that a simple implementation of an existing design with a few tweaks can generate synthetic training data that can improve the classification performance unseen real data. Unlike other studies I have chosen to only balance the training set of data as I wanted to observe whether the generated data was representable of the real data it was trying to imitate. If I were to augment the whole dataset prior to train testing splitting I worry the classifier would simply learn to distinguish synthetic data as a class rather than belonging to an existing class of real data.

4.2. Comparison of results from other studies

The following results section compares the results achieved in this project to results attained in similar research. The results were obtained from papers and research using the IoT23 dataset in a manner similar to my own.

	This project	Abdalgawad *et al*		
	GAN-ANN	**RF**	**BiGAN+KNN**	**KNN+GAN**
	F1	F1	F1	F1
Benign	0.63	0.78	0.99	0.83
C&C	0.54	0.36	0.96	0.44
C&C Heartbeat	0.77	0.13	0.93	0.26

Table 13 - Classification results comparison

Research by Abdalgawad *et al* (2021) used deep learning methods to detect malware occurrences in the in the IoT-23 dataset using several machine learning methods. Compared to my own results we can see from table 13 that their Bidirectional GAN model outperformed my GAN-ANN implementation, comparing the classes selected for GAN generation from my project, in their paper the BI-GAN is used as the malware classifier, it was trained in a similar method to my own GAN but rather than using the generated data to train a different classifier researchers used a different layout to incorporate anomaly detection into the training mechanism and this proved to have good results compared to a standalone Random forest algorithm. They did however use the GAN generated data to also augment the IoT-23 dataset, however in their study they synthetically generated samples for all classes and trained the KNN model solely on the synthetic data whereas in my project just three classes were generated, and I used a mix of real and synthetic data for training. Although the KNN classifier was trained on just generated data, from the results we can see that the KNN outperformed my GAN-ANN IDS in the detection of 'Benign' samples. Where my model does better is the detection of minority class samples when compared to the KNN model, C&C classification with the GAN-ANN outperformed the KKN+GAN by 10% and C&C heartbeat by 20%, this improvement could be down to the mix of real samples within the training data, although most training samples used for those classes were still synthetic.

		Accuracy	Precsion	Recall	F1
This project	GAN-ANN	0.88	0.83	0.89	0.83
Abdalgawad *et al*	KNN				0.60
	RF				0.55
	BiGAN+KNN				0.97
Stoian	RF	1	0.88	0.85	0.84
	ANN	0.66	0.33	0.14	0.10
Dutta *et al*	DNN	0.98	1	0.98	0.87
	RF	0.98	0.92	0.98	0.89

Table 14 - Classification results comparison

Looking to table 14 we see the results from two other papers using the IoT-23 dataset. Stoian (2020) has better success using the RandomForest algorithm compared to Abdalgawad (2021), however compared to my project Stoian's (2020) implementation of the ANN performed worse than my projects GAN-ANN. Like Stoian I saw relatively poor classification results from the ANN on its own, however with the GAN supplemented training data I saw greater classification performances when compared to Stoian. The results from Dutta *et al* (2020) however saw better results with their implementation of a Deep neural network, compared to my project their model had a much higher accuracy and precision score however the F1 score from their study is close to my own with a 4% difference. The difference in results between Dutta *et al* (2020) and my project can be equated to their use of the dimensionality reduction methods Component analysis (PCA) and Autoencoder (AE) to remove inconsequential information from their dataset. In their study they note that removing the meaningless and inconsequential information from the dataset produces stronger inputs to the classifier as the effectiveness of machine learning classifiers is in direct correlation to the grade of the selected characteristics.

Chapter 5: Discussion

To reiterate the objectives that I set out in the first chapter of this project:

- Review the literature relating to generating purely tabular data through generative adversarial networks

The objective outlined above is met through the literature review I undertook and document in Chapter 2. The section on GANs and their use under different situations is explained in sections 2.2 and 2.4. An observation made during literature review is an issue that I encountered during this project being the relatively small amount of literature regarding tabular based GANs. Of the research I reviewed I had a lot of hope for my own project as there were promising results coming out of the studies. One example is the paper by Shahrair *et al* (2020) where authors used a similar system to select and synthetically generate poorly performing samples from the KDD-99 dataset, their results showed that all labels selected to be generated saw an improvement in F1 score, with the largest increase being 25% better F1 and the lowest improvement was 8%. This paper is one that I referenced a lot in this paper as the methods used by authors, I believed to be useful for my own paper; an example is their testing of increasing data sizes on their GANs performance. Their results in this regard turned out to be similar in that I also saw an improvement in the quality of data as the size of training set of each of the labels increased.

- Develop a framework where training data can be synthetically augmented using a generative adversarial network

The design of the core code for the GAN is not my own work however the implementation of generating class based synthetic data was achieved as my use case differed from the original two implementations of the code and from my own results, I judge this objective as a success. As I did not develop a system from the ground up, I could assume that this objective was not met however the work I have completed in sections 3.9 shows the support code I implemented to convert an existing GAN design for my own use. From this conversion I was able to generate synthetic data that could train a machine learning algorithm to recognise unseen data, therefor I could make the evaluation that this objective was partially met. Since the objective was stated at the beginning of the project, I would say that I possibly did not fully grasp the full extent of understanding it would require to build such a system from the ground up. In hindsight I possibly would have relaxed this objective to match my own capabilities and timeframe.

- Use the GAN architecture to produce synthetic data samples that improve the detection performance of a classifier

Referring to the results in Chapter 4, I demonstrated the system used in this project that produced sufficient results, the synthetically generated training data resulted in improved classification performance on most of the classes that were selected for synthetic generation as such the results from the different scenarios show that this technique of synthetic data generation shows promise. While I am not the first to use GANs in this manor I have added to the small field of research that is going to grow exponentially in the coming years. From my own results I have identified that even a simple implementation of a GAN can generate synthetic data for improving malware classification using machine learning, as data imbalance is a core issue for many machines learning-based systems, researchers and companies are constantly looking for ways to improve their modelling performance so the applicability of GANs is quite vast. While I did see a negative outcome during testing of the 'Bengin' classes I feel like I described the reasons for this during the evaluation stage of Chapter 4.

- Identify limitations of using the synthetic data generated by a GAN

As mentioned in the results section one of the observation I made was that while the generated data could be used to train a classification model it is possible that the model will pick up patterns in the synthetic data and equate different classes as the same if they are synthetically based. This kind of limitation may not be unique to tabular data generation and conversely it may not be a limitation in some regards for example it may be required to have a unique identifying signature for synthetically generated pictures of people as there could be multiple situations where an individual is tricked using GAN generated content, we could see a rise in requirement for not completely realistic GAN generated content.

Looking back at the project I feel there are a few aspects that I would have in hindsight done differently, firstly I believe I should have explored in detail how the type of data used for GAN training affected the quality of the generated data. This consideration would have saved me a lot of time later on in the project and fluctuating results due to datatype errors slowed progress down later in the project. Secondly I would have implemented a conditional generative adversarial network (CTGAN) as this possibly would have improved the models ability to learn from the data it was tasked to train upon, where a CTGAN is fed the training data along with the label and the discriminator has to decide if each sample is real or fake and which label each sample it belongs to, I believe that this method better suits a multiclass dataset as the generator learns to identify differences between samples rather than simply trying to game the discriminator, it adds another learning vector that would possibly result in more accurate synthetic data compared to my own implementation.

Chapter 6: Evaluation, Reflections, and Conclusions

This section contains the following details:

- Evaluation of Literature review
- Evaluation of methods
- Evaluation of results
- Significance of research
- Future research
- Reflections

Evaluation of Literature review

The literature review looked at the different areas explored in this project, starting with the foundational paper by Goodfellow *et al* (2014) where the concept of generative adversarial networks was first conceptualised and the ground-breaking GAN research that followed which all stemmed from that first paper. Next, I explored research into some of the issues surrounding class imbalance and one of the current methods of combatting data imbalance known as SMOTE. Looking further into the use of GANs to generate synthetic data I identified use cases for the data generated, the first applying to my use case of augmenting an IDS targeted dataset through synthetic data generation and the second being a secondary benefit of synthetically generated data being the ability to anonymise privacy sensitive data, a concept explored by Yoon *et al* (2020) who showed that medial imagery can be synthesised to be both anonymised and representative of a real person's medical data without revealing their identity. Finally, I reviewed research that is based upon the IoT-23 dataset, a relatively new malware detection dataset where the current papers results vary however the consensus is that the extreme class imbalance is a inhibiting factor in the classification results for the papers that tested the classification performance on ANNs.

Evaluation of methods

The project can be loosely broken down into three stages, stage one was data preparation. Thankfully there are a few papers that go into detail on their choices for feature reduction and class manipulation, however in practice this being the first task on the list resulted in some lessons being learned too late, for example upon learning how significant the datatypes had on the results and flow of the data later in the project meant I was constantly adjusting how the data was pre-processed. For the second stage being the Intrusion detection system, I had a relatively easy time completing and fine tuning this model as I have had past experience with machine learning IDS systems from my bachelor's thesis, therefore this stage flew by rather quickly and did not result is any headaches. Finally the GAN; thankfully I was using an existing design which removed a large amount of issues that could have arisen during construction, but the issues really came when adjusting the design to fit my case, as previously mentioned data types became an issue with certain python functions requiring a specific data type to function correctly and other outputting the data in a format that required further adjustments to the pre-processing stage which certainly slowed things down however thankfully python is a very modular language with plenty of existing documentation so no problem was too cumbersome.

Evaluation of results

As previously mentioned, I am overall happy with the results as I achieved the objectives that I set out to achieve. I am equally pleased with the limitations that I found with the results as I learned a lot analysing the outcomes of using synthetic data in this use case, for example I identified that using a CTGAN would have probably resulted in synthetic data that was more related to the original data since I discovered that the IDS classifier was learning to identify samples by their origin being either synthetic or the real original data. I would have liked to include other evaluation metrics that could identify further points of comparison between the synthetically generated data and the original data as I am going off misclassification rates and malware classification performance, which is good enough for my objectives however, I feel like more in-depth analysis of the generated data could have resulted in a deeper understand of the data I was creating.

Evaluation of objectives

Of the objectives I set for myself, I consider my project a success. I feel they were all realistic for the project; the first objective of reviewing project related literature was straight forward however I was relatively bogged down for some of it due to the lack of research into some areas and the overwhelming amount in other.

The second objective however I have touched on slightly however as mentioned in hindsight I feel like this was slightly unrealistic, due to the nature of the topic I selected I now believe that it was a task far greater than initially idealised, while I enjoyed and learned a lot during the process of trying to create a GAN model from the ground up I should have put that time into understanding some of the fundamentals and maths surrounding GANs. With the knowledge I have now I would have picked a different variant of GAN for the project; however, overall, I am pleased with the outcome of the project and the objective itself was still partially met.

Following on from the second objective the third one was more manageable, Once I had a simple solution in place, the testing method of having an IDS to assess the quality of the synthetic data was also a great help as in many papers I read the issues a lot of researcher into GANs found were limited methods of exploring whether their implementation and data generated was actually realistic enough to be classed useful.

As for the objective of identifying limitation of synthetic data the results obtained in chapter 4 outlined the issues I discovered using GAN generated data; the main one being the trace signatures of the data being synthetic were present in the data generated for different classes and despite the class labels being different I found that the IDS was mistaking different labels for one another based on the volume of data being synthetic opposed to real during the IDS training stage.

Significance of research

The research that I have put forward is significant in that there is not much research on the application of GAN generated synthetic data especially in the use of augmenting IDs targeted datasets. This is definitely an area of research that will expand greatly in the coming years as the techniques surrounding GANS will improve in an exponential rate with my own research being one of the few first implementations of such. Likewise, with the IoT-23 dataset there is very little research into using modern methods of deep learning with this dataset. My project is one of the first papers to use the dataset with a modern deep learning solution with most papers using traditional machine learning techniques such as RF, DT and naive bays.

Limitations of research

While this is a relatively unexplored topic the implementation, I have used is a very basic structure of a GAN and as I have touched upon, I believe I would have benefited from using a slightly newer variation CTGAN. The issues I have touched upon include the synthetic data not being sufficiently representative of some the data it is trying to imitate. Furthermore, since the system I have used is an existing design and not a novel build the research is really a proof of concept.

Future research

GANs are still in their infancy. We are in a world where Instagram filters and deep fakes are becoming more prevalent, and I hope that improvements in GAN techniques are not applied for superficial reasons. My hope is that GANs provide a solution to current issues surrounding the sharing of research data. If used correctly GANs could be the key to open sourcing highly sort after data for use in medical research and the furthering of understanding malware. Overcoming dataset imbalance is especially key in malware classification as currently dataset imbalance affects just about every dataset used to train IDS systems. With machine learning solutions being key to detecting zero-day attacks this kind of technology could be key to improving the ability of IDS systems to detect never seen before attacks as they occur.

Reflections

This project has provided me with invaluable knowledge and experience for future work. The techniques and concepts explored throughout have tested me from day one; however, I have loved every second of it, and at no point was I dismayed by the difficulties of certain aspects of the project. A few things however I believe I should have done differently, time management for one, I should have taken greater care in how long I spent on certain sections of the project as I rushed through some sections thinking they were simple only to have to go back and redo massive portions of code as the methods I had used were not technically sound or completely the correct way they should have been done. Overall, I am happy with the outcome of the project and the lessons I have learned will help me later in life and I am pleased this is the piece I am likely to end the educational stage of my life on.

References

Pawlicki, M *et al.*(2022). ‘A survey on neural networks for (cyber-) security and (cyber-) security of neural networks’, *Neurocomputing, Volume 500*,(2022) https://doi.org/10.1016/j.neucom.2022.06.002.

Meena, G *et al* (2017) ‘A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA’, *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, 2017, pp. 553-558, doi: 10.1109/COMPTELIX.2017.8004032.

Choudhary, S. Kesswani, N. (2020), Analysis of KDD-Cup’99, NSL-KDD and UNSW-NB15 Datasets using Deep Learning in IoT’ *Procedia Computer Science,Volume 167*,2020, https://doi.org/10.1016/j.procs.2020.03.367.

Yin, C *et al* (2017). ‘A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks’. in *IEEE Access*, vol. 5, pp. 21954-21961, 2017, doi: 10.1109/ACCESS.2017.2762418.

Staudemeyer, R. (2015). ‘Applying long short-term memory recurrent neural networks to intrusion detection’, *South African Computer Journal* (2015) https://doi.org/10.18489/sacj.v56i1.248

Ingre, B. Yadav, A (2015). ‘Performance analysis of NSL-KDD dataset using ANN’, *Conference: 2015 International Conference on Signal Processing And Communication Engineering Systems (SPACES)* http://dx.doi.org/10.1109/SPACES.2015.7058223

Goodfellow, I. Pouget-Abadie, J. Mirza, M. Xu, B. Warde-Farley, D. Ozair, S. Courville, A. Bengio, Y.(2014). ‘Generative Adversarial Nets’. *Proceedings of the International Conference on Neural Information Processing Systems* (NIPS 2014) Available at: https://arxiv.org/abs/1406.2661 (Accessed 5th August 2022)

Hu, J. *et al* (2018) ‘Online Nonlinear AUC Maximization for Imbalanced Data Sets’ in *IEEE Transactions on Neural Networks and Learning Systems*. https://doi.org/ 10.1109/TNNLS.2016.2610465

Guo, X. *et al* (2008) ‘On the Class Imbalance Problem’. *Fourth International Conference on Natural Computation* https://doi.org/ 10.1109/ICNC.2008.871

Japkowicz, N. (2000) ‘The Class Imbalance Problem: Significance and Strategies’. Dalhousie University, Available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.1693&rep=rep1&type=pdf (Accessed: 5 August 2022)

Abd Elrahman, A. and Abraham, A. ‘A Review of Class Imbalance Problem’. *Journal of Network and Innovative Computing.* Available at: https://ias04.softcomputing.net/jnic2.pdf (Accessed: 5 August 2022)

Wei, J. *et al* (2021) DuelGAN: ‘A Duel Between Two Discriminators Stabilizes the GAN Training’, *arXiv preprint*(2021) https://doi.org/10.48550/arXiv.2101.07524

Lee, J. and Park, K. ‘GAN-based imbalanced data intrusion detection system’. *Personal and Ubiquitous Computing (2021)* https://doi.org/ 10.1007/s00779-019-01332-y

Zhang, Z. ‘A gentle introduction to artificial neural networks’. *Ann Transl Med. 2016 Oct;4* https://doi.org/ 10.21037/atm.2016.06.20

ReLU (Rectified Linear Unit) Activation Function (no date). Available at: https://iq.opengenus.org/relu-activation/ (Accessed: 17 August)

University of California, Irvine (1999) ‘KDD Cup 1999’. Available at: http://kdd.ics.uci.edu/databases/kddcup 99/kddcup99.html (Accessed 28 April 2022)

Softmax function (2022) Available at: https://en.wikipedia.org/wiki/Softmax_function#cite_ref-FOOTNOTEGoodfellowBengioCourville2016183%E2%80%93184_8-0 (Accessed: 19 August 2022)

Cunningham, P., Cord, M., Delany, S.J. (2008). ‘Supervised Learning’. *Machine Learning Techniques for Multimedia. Cognitive Technologies*. https://doi.org/ 10.1007/978-3-540-75171-7_2

Wood, T. (no date) ‘What is the F-score?’ *DeepAi*. Available at: https://deepai.org/machine-learning-glossary-and-terms/f-score (Accessed: 19 August 2022)

Liang, Y. and Vankayalapati, N. (2021). ‘Machine Learning and Deep Learning Methods for Better Anomaly Detection in IoT-23 Dataset Cybersecurity’. *Lakehead University Canada*. Available at: https://github.com/yliang725/Anomaly-Detection-IoT23/blob/main/Research%20Paper/Research%20Paper.pdf (Accessed: 5 September 2022)

Abdalgawad, N. *et al* (2021) ‘Generative Deep Learning to Detect Cyberattacks for the IoT-23 Dataset’. *IEEE Access*. https://doi.org/ 10.1109/ACCESS.2021.3140015

Strecker, S. *et al* (2021) ‘A Modern Analysis of Aging Machine Learning Based IoT Cybersecurity Methods’. *Cornell University*. https://doi.org/10.48550/arXiv.2110.07832

Zhang, L. *et al* (2019) ‘An Improved Network Intrusion Detection Based on Deep Neural Network’ *IOP Conf. Ser.: Mater. Sci.* Accessible at: https://iopscience.iop.org/article/10.1088/1757-899X/563/5/052019 (Accessed: 4 September 2022)

Kumar, S. (2021) ‘Stop One-Hot Encoding your Categorical Features — Avoid Curse of Dimensionality’, *Towards data science*, 2nd March. Available at: https://medium.com/swlh/stop-one-hot-encoding-your-categorical-features-avoid-curse-of-dimensionality-16743c32cea4#:~:text=One%2Dhot%20encoding%20categorical%20variables,problem%20of%20parallelism%20and%20multicollinearity (Accessed: 14 September 2022)

Brock, T. (2022) ‘Pareto Principle’, *Investopedia*, 07 April. Available at: https://www.investopedia.com/terms/p/paretoprinciple.asp (Accessed 14th September 2022)

Brownlee, J (2016) ‘What is a confusion matrix in machine learning’, *Machine learning mastery*, November 18th. Available at: https://machinelearningmastery.com/confusion-matrix-machine-learning/ (Accessed 16th September 2022)

Shankarmani, R. *et al* (2012) ‘Agile Methodology Adoption: Benefits and Constraints’, *International Journal of Computer Applications (15).* https://doi.org/10.5120/9361-3698

School of Science and Technology

Project proposal

GAN generated synthetic data to overcome data imbalance issues in IDS training

Cameron Swart

210039440

MSc Cyber Security

Introduction

An overwhelmingly common issue within network-based machine learning intrusion detection systems is that the data in which we train the models upon quite often suffers from extreme class imbalance. In a regular network environment, the volume of regular non-malicious traffic is most of the traffic, therefore the nature of the IDS dataset is they are naturally imbalanced (Sofia Visa & Anca Ralescu 2005). A drawback of this is minority samples that represent attack classes are often difficult to predict as there are not enough samples for the model to learn from, especially, considering most of the dataset contains regular normal network traffic. A method to overcome these problems is to use synthetic data, the most popular method is to use synthetic minority over-sampling technique (SMOTE) to bolster the minority classes, while this can provide classification improvement issues arise in the quality of the generated data such as sample overlapping and noise interference (Jiang *et al* 2001).

Another new method of synthetic data generation is through generative adversarial networks where the generator is in competition with the discriminator by providing synthetic samples capable of fooling the discriminator into thinking they are real, once the synthetic samples have reached this threshold of credibility with the discriminator they can be routed into the dataset as new minority class samples to train the IDS model upon

Research question

These issues sustained in a modern digital society can do nothing other than evolve to incorporate new technologies and new threats, machine learning is at the forefront of this development, the data in which we train these models upon is at the epicentre, with the current issue of class imbalance currently plaguing much of the research and implementation I propose the following question:

Can synthetically generated samples from a GAN be used to overcome sample imbalance issues in IDS targeted datasets?

Objectives

To consider my project a success I aim to complete the following objectives over the course of my research

- Use the GAN architecture to produce synthetic data samples that can be used within the hybrid dataset in order improve individual class classification performance (given that sample contains synthetic data) by 25%.
- GAN produced synthetic samples provide better classification performance than SMOTE produced synthetic samples by 10-20%.

Products of work

The product of my research will be a GAN complemented IDS system where the synthetically generated samples from the GAN will be used to augment the dataset and provide meaningful minority class data samples to be used to train the IDS model.

Scope

The scope of my proposed project is the assessment of the relatively new learning framework GAN and whether the generated samples can be used to augment existing IDS datasets. I will not be producing a completely new dataset per se, however the process of augmenting the hybrid dataset and the IDS classification performance upon the hybrid dataset will be the point of comparison and assessment. During assessment and comparison phase I will compare the classification results of the GAN-IDS to the classification results of the SMOTE-IDS as such the deliveries will be the comparative classification results which will be presented quantitatively and accompanied by visualisations of classifier performance.

Beneficiaries

Those that would benefit from the findings of my project are researchers looking for a method to offset a drawback of many datasets. The implementation of GANs in many different situations not exclusively in an IDS setting has already shown great promise, so this will add to the growing list of applicable situations that GANs may be the better alternative for.

Benefits

Addresses a long-standing problem with publicly available datasets that are used for intrusion detection in that the solution is not absolute, however it does provide an alternative route to overcoming the main draw backs.

Critical context

Relevant literature

GAN generated data has already shown a lot of promise. A study by JooHwa Lee & KeeHyun Park (2021) presented a GAN-RandomForest system. Their research compared the classification results of the GAN generated data on a random forest classier to SMOTE generated data on a RF classifier, they found that evaluation data from the GAN-RF out-performed the RF operating on its own. They also found that the GAN-RF out-performed SMOTE-RF when evaluating imbalanced class classification (JooHwa Lee & KeeHyun 2021).

Work by Shahrair *et al* (2020) demonstrated that live network traffic capture for IDS training can be further improved through GAN generated data. The IDS performed better when the synthetic data was combined with the captured network traffic. In their further testing they found that their proposed G-IDS was computationally expensive and time consuming requiring a smaller dataset for training (Shahrair *et al* 2020).

Research into botnet detection using GAN architecture to enhance the trained model's performance caried out by Yin *et al* (2018) made the observation that their GAN model when used to augment the original detection model was able to detect a class of botnet that was not in the training data and only existed in the test set, this method improved detection performance and decreased false positive rate of the detection model (Yin *et al* 2018).

Medical researchers Diamant *et al* (2018) use GAN generated synthetic data to improve CNN classification performance in the detection of liver lesions, their research indicated that the generated images used to augment the dataset increased model specificity by 4% and sensitivity by 7% (Diamant *et al* 2018).

A side benefit of the GAN generation process is the ability to anonymise privacy sensitive data. Yoon *et al* developed a framework to generate synthetic data to minimize re-identification.

When used in a medical image generative setting the process can be applied to personal information sensitive IDS datasets (Yoon *et al* 2020).

Currently the main issue is the data in which we currently train the models upon is quite often highly imbalanced (Karatas 2018), Since a large quantity of the data that will be analysed by the intrusion detection system (IDS) will be benign or normal network traffic with the malicious samples being very low in occurrence, an example from popular datasets is the KDD-CUP-99 and CICIDS-2017 datasets, the former being an old but very well cited dataset with normal data representing over 50% of the entire dataset. DoS has the next highest sample size of 36% and the least occurring sample being U2R (*unauthorized access to local superuser (root) privileges, e.g., various ``buffer overflow'' attacks*) representing just 0.04% of the overall dataset. The latter CICIDS-2017 dataset suffers from the same issues, the normal class in this case encompasses 83% of the total dataset with the least occurring sample being an infiltration attack which represents 0.001% of the total dataset.

Generative Adversarial Networks (GANs)

A GAN is a relatively new technique in machine learning, designed in 2014 by Ian Goodfellow *et al*, it works by having two neural networks compete in a game where the objective is to beat one another. A GAN is made up of two components, a generator whose job is to "generate" synthetic examples of data to fool the second component the discriminator who needs to be able to distinguish the difference between the synthetic data and the real data. This process is repeated until the generator can "beat" the discriminator.

The process behind GANs is the zero-sum game played between two players, the generator model takes a random input vector and generates a sample, the discriminator is then fed combinations of real samples from the dataset and synthetic samples from the generator, the discriminator must then say whether a sample is fake or not, both models are trying to outsmart the other and "win" the game. For every success the discriminator has, the weights of the generator are adjusted to edge closer to creating better synthetic samples and the discriminator's weights are adjusted to better spot the synthetic samples (Ian Goodfellow *et al* 2014).

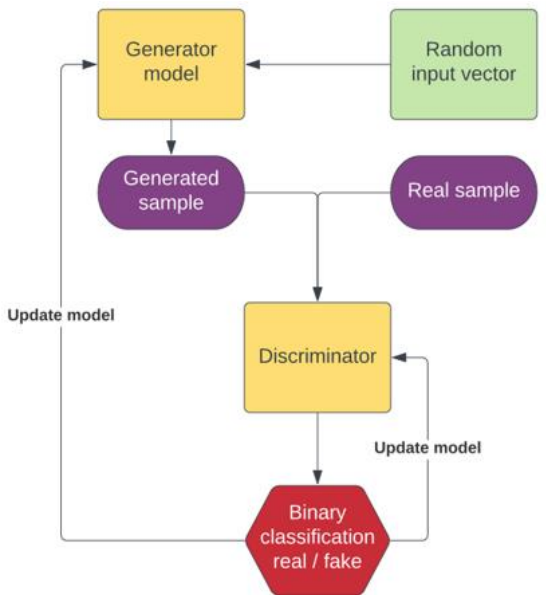


Figure 58 – Generative adversarial network

Tools & Methods

My plan is to train the IDS model on the original dataset, from here the performance of each sample will be measured and if a sample's classification performance does not meet a criterion (I plan on testing different forms of criteria i.e., precision score bellow .70 or F1 score below .70), then the sample will be flagged for processing through the GAN. Once the synthetic samples of the flagged samples generated can fool the discriminator to a specific degree, they will be funnelled into a hybrid training dataset which contains a mix of real and synthetic samples. Combined this hybrid dataset will be used to re-train the IDS model and the results to which will be assessed and compared to a classical data synthesising method SMOTE. Figure 2 shows the framework of the system. My hope is this will provide a way in which to use synthetic data without encountering the problems around currently popular data synthesising methods. SMOTE has an issue with synthetic data over-generalization as it is based upon local information, whereas GAN's learn from the overall class distribution (Xie *et al I*, 2018)

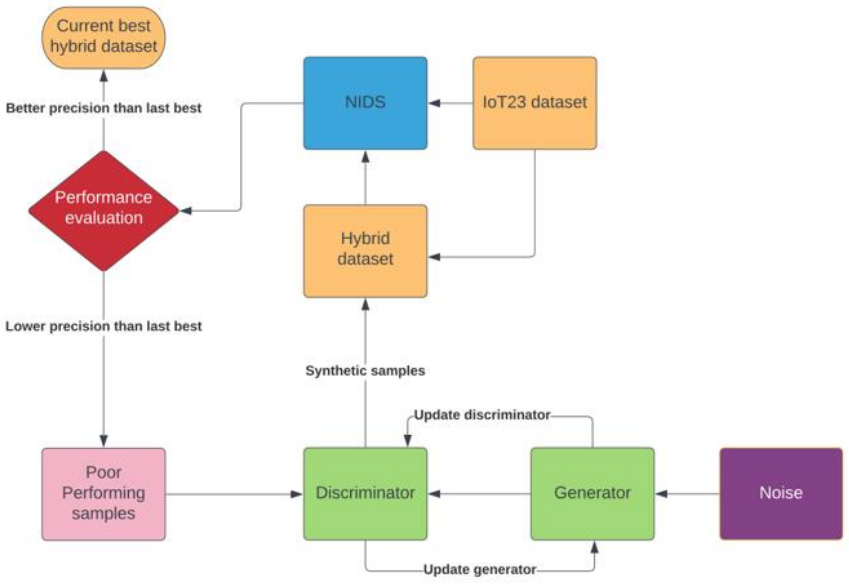


Figure 59 – Proposed system framework

Data

The dataset for this project is called IoT-23 provided by the Czech technical University. The dataset was compiled in a lab environment with no risk of personal data being included in the network log files. The IoT-23 dataset is a collection of malicious and regular network traffic created by Avast ACI laboratory. Of the malicious traffic there is a collection of different attacks represented, most of which being modern botnet attacks which are currently the biggest threat to most IoT devices. Alongside the malicious traffic in the dataset there is also regular network traffic of three IoT devices, Amazon Echo, Philips hue and a Somfy door lock. The dataset contains Nine different labels, these are described in table 1.

Label	**Description**
Benigin	Regular network traffic
C&C	Device connects to C&C server
DDoS	Distributed Denial of Service attack is being executed by the infected device
FileDownload	file is being downloaded to the infected device
Heartbeat	Packets sent on this connection are used to keep a track on the infected host by the C&C server
Okiru	Connections have characteristics of a Okiru botnet
PartOfAHorizontalPortScan	Connections are used to do a horizontal port scan to gather information to perform further attacks
Mirai	Connections have characteristics of a Mirai botnet
Torii	Connections have characteristics of a Okiru botnet

Table 15 – IoT-23 Attack samples

Intrusion detection system (NIDS)

The IDS module of the project will be made up of an artificial neural network (ANN), commonly referred to as simply as neural network (NN). Specifically, I will begin using a feed forward multi-layered perceptron (MLP), where the model will consist of three layers; input, hidden and output layer. Although each node except for the input node uses non-linear activation it is this and being able to distinguish non-linearly separable data that makes the MLP different to the single layer perceptron. You can build more complicated neural networks by adding more hidden layers for larger datasets with large amounts of categories. This is only to a certain point however, too many hidden layers can backfire and cause over fitting and drastically affect results. In a MLP information is passed through the layers in a forward direction and is transformed through each layer of the MLP. The 18 computation of each hidden layer can be defined as hi(x) = f(wi T x +bi), f is the non-linier activation function, in my case I am using SoftMax to output the predictions of each class, the activation function for SoftMax is; SoftMax(xi) = The primary goal of a feedforward network is to approximate some function f* , for example a classifier y = f * (x) will map an input x to a category or class y. A feedforward NN learns the value of the parameters θ that are mapped from y = f (x; θ) (Gupta 2017). Models are referred to as 'feed forward' because of how the information flows through the function being evaluated from x through the transitional computations used to define f to which we finally arrive at output y.

Software

All the code for my project will be written using python within he juypter notebook IDE, the models will utilize the TensorFlow machine learning library and several other libraries noted in the table below.

Library	Use
Pandas	Dataset manipulation, data structures
Numpy	Mathematical operations on arrays
Matplotlib	Creating visualisations of model performance
tensorflow	Deep learning library, models will be made using this library
Scikit-learn	Scaling, feature selection, evaluation metrics, label encoding

Table 16 - Python libraries

Hardware

I will be using my own desktop for training, testing and evaluation of the models, the specs are as follows.

Intel Core i7 11700k (running at 4.6Ghz)

32gb RAM

RTX 3070ti

Model evaluation

To assess the performance of the models I will be assessing the final IDS ability to distinguish possible attacks from the test set given whether the data used to train the IDS contains synthetic GAN created data.

Metrics

Confusion matrix: A table where the correct & incorrect predictions are presented numerically

	BENIGN PREDICTED	ATTACK PREDICTED
BENIGN ACTUAL	**True Positive (TP)**	**False Negative (FN)**
ATTACK ACTUAL	**False Positive (FP)**	**True Negative (TN)**

Table 17 - Confusion matrix table

Precision: The ratio of correct positive predictions over the total number of predicted positives

$$\text{Precision} = \frac{TP}{TP+FP}$$

Accuracy: The value of correctly predicted datapoints, divided by the total number of classifications

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Recall: The number of correct predictions over the total number of class occurrences

$$Recall = \frac{TP}{TP + FN}$$

F1: the balance between precision and recall, regarded as the harmonic mean of a model's precision and recall it is a measure of the model's accuracy

$$\text{F1} = \frac{\text{2 x precision x recall}}{\text{precision+recall}}$$

Evaluation

The overall evaluation of the two different systems will fall to the classification report which provides the overall classification performance of the two systems using the metrics outlined. Individual class performance will also be a factor for comparison. To standardise the results both systems will be subject to identical prepossessing standards such as feature selection, the IDS modules will be identical, and the only change will be the data synthesis methods i.e., SMOTE or GAN.

Risks

Risk	Likelihood (1-3)	Consequence (1-5)	Impact (L * C)	Mitigation
Dataset cannot be obtained	**1**	**3**	**3**	**Alternative data set**
Dataset too large to import/run	**2**	**4**	**8**	**Load fewer rows**
Code does not work as required	**3**	**4**	**12**	**Work on code element from start**
Models perform poorly	**2**	**4**	**8**	**Try alternative model types**
Poor classification performance	**2**	**3**	**6**	**Different model structure**
Long run time	**3**	**3**	**6**	**Reduce dataset rows/try different model structure**
Synthetic data does not improve classification performance	**2**	**4**	**8**	**Alter generator learning / alter discriminator learning**
Loss of progress/code/work	**2**	**5**	**8**	**Version control using Github**

Table 18 - Project risks

Work plan

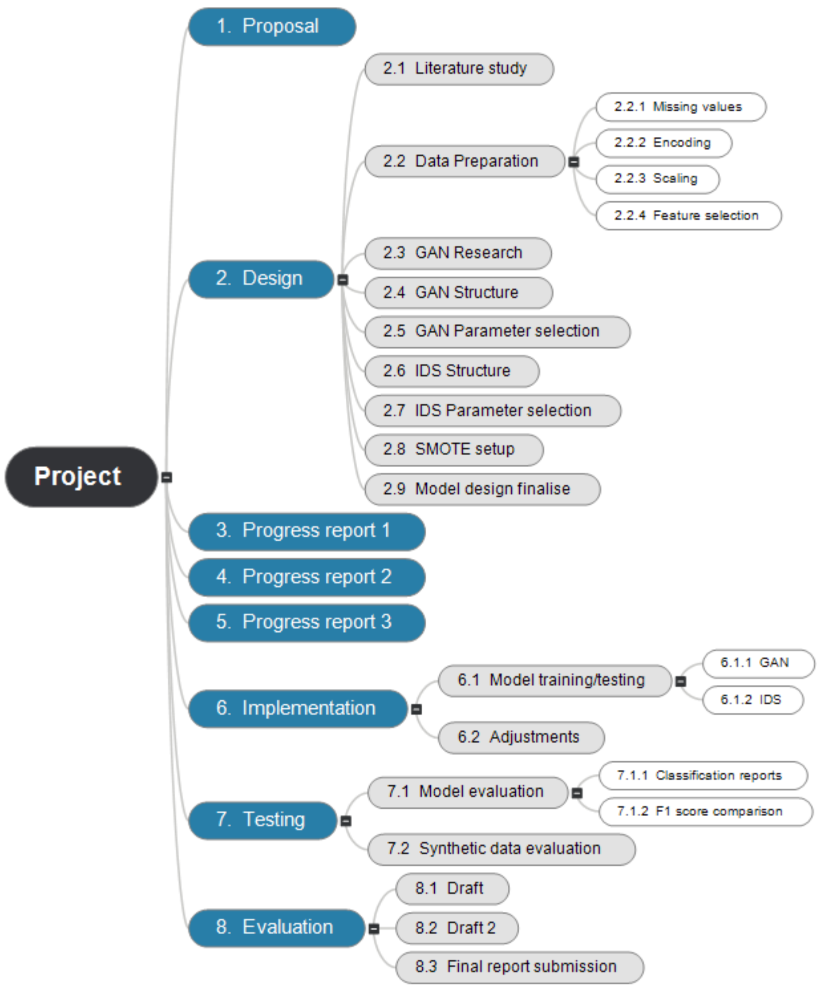
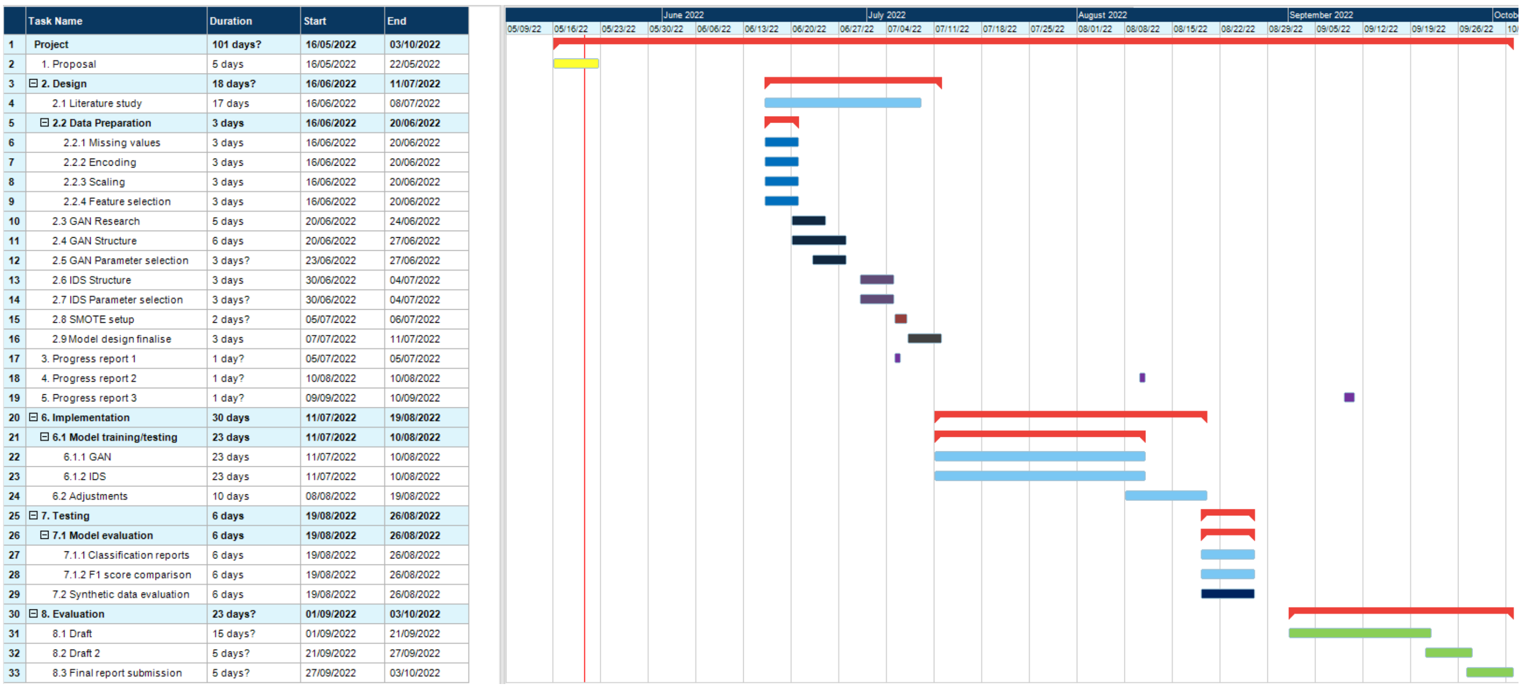


Figure 60 – Left right project break down



	Task Name	Duration	Start	End
1	**Project**	**101 days?**	**16/05/2022**	**03/10/2022**
2	1. Proposal	5 days	16/05/2022	22/05/2022
3	⊟ **2. Design**	**18 days?**	**16/06/2022**	**11/07/2022**
4	2.1 Literature study	17 days	16/06/2022	08/07/2022
5	⊟ **2.2 Data Preparation**	**3 days**	**16/06/2022**	**20/06/2022**
6	2.2.1 Missing values	3 days	16/06/2022	20/06/2022
7	2.2.2 Encoding	3 days	16/06/2022	20/06/2022
8	2.2.3 Scaling	3 days	16/06/2022	20/06/2022
9	2.2.4 Feature selection	3 days	16/06/2022	20/06/2022
10	2.3 GAN Research	5 days	20/06/2022	24/06/2022
11	2.4 GAN Structure	6 days	20/06/2022	27/06/2022
12	2.5 GAN Parameter selection	3 days?	23/06/2022	27/06/2022
13	2.6 IDS Structure	3 days	30/06/2022	04/07/2022
14	2.7 IDS Parameter selection	3 days?	30/06/2022	04/07/2022
15	2.8 SMOTE setup	2 days?	05/07/2022	06/07/2022
16	2.9 Model design finalise	3 days	07/07/2022	11/07/2022
17	3. Progress report 1	1 day?	05/07/2022	05/07/2022
18	4. Progress report 2	1 day?	10/08/2022	10/08/2022
19	5. Progress report 3	1 day?	09/09/2022	10/09/2022
20	⊟ **6. Implementation**	**30 days**	**11/07/2022**	**19/08/2022**
21	⊟ **6.1 Model training/testing**	**23 days**	**11/07/2022**	**10/08/2022**
22	6.1.1 GAN	23 days	11/07/2022	10/08/2022
23	6.1.2 IDS	23 days	11/07/2022	10/08/2022
24	6.2 Adjustments	10 days	08/08/2022	19/08/2022
25	⊟ **7. Testing**	**6 days**	**19/08/2022**	**26/08/2022**
26	⊟ **7.1 Model evaluation**	**6 days**	**19/08/2022**	**26/08/2022**
27	7.1.1 Classification reports	6 days	19/08/2022	26/08/2022
28	7.1.2 F1 score comparison	6 days	19/08/2022	26/08/2022
29	7.2 Synthetic data evaluation	6 days	19/08/2022	26/08/2022
30	⊟ **8. Evaluation**	**23 days?**	**01/09/2022**	**03/10/2022**
31	8.1 Draft	15 days?	01/09/2022	21/09/2022
32	8.2 Draft 2	5 days?	21/09/2022	27/09/2022
33	8.3 Final report submission	5 days?	27/09/2022	03/10/2022

Figure 61 – Gantt chart

References

[1] Lee, J. Park, K. GAN-based imbalanced data intrusion detection system. '*Pers Ubiquit Comput'* 25, 121–128 (2021). https://doi.org/10.1007/s00779-019-01332-y

[2] Shahriar, M. Haque, N. Rahman, M. Alonso, M. "G-IDS: Generative Adversarial Networks Assisted Intrusion Detection System," *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2020, pp. 376-385, doi: 10.1109/COMPSAC48688.2020.0-218.

[3] Yin, C. Zhu, Y. Liu, S. Fei, J. Zhang, H. "An enhancing framework for botnet detection using generative adversarial networks," *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 2018, pp. 228-234, doi: 10.1109/ICAIBD.2018.8396200.

[4] Frid-Adar, M. Diamant, I. Klang, E. Amitai, M. Goldberger, J. Greenspan, H. (2018) "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification" *Neurocomputing*, Volume 321, pp. 321-331, https://doi.org/10.1016/j.neucom.2018.09.013.

[5] Yoon, J. Drumright, L. van der Schaar, M. "Anonymization Through Data Synthesis Using Generative Adversarial Networks (ADS-GAN)," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 8, pp. 2378-2388, Aug. 2020, doi: 10.1109/JBHI.2020.2980262

[6] Karatas, G. Demir, O. Sahingoz, O. K. "Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-Date Dataset," in IEEE Access, vol. 8, pp. 32150-32162, 2020, doi: 10.1109/ACCESS.2020.2973219.

[7] University of California, Irvine (1999) 'KDD Cup 1999'. Available at: http://kdd.ics.uci.edu/databases/kddcup 99/kddcup99.html (Accessed 28 April 2022)

[8] Sharafaldin, I. Lashkari, A. Ghorbani, A. (2018) 'Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization', 4th International Conference on Information Systems Security and Privacy (ICISSP) Available at: https://www.unb.ca/cic/datasets/ids-2017.html (Accessed 28 April 2022)

[9] Garcia, S. Parmisano, A. Jose Erquiaga, M. (2020). IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]. Zenodo. http://doi.org/10.5281/zenodo.4743746

[10] Goodfellow, I. Pouget-Abadie, J. Mirza, M. Xu, B. Warde-Farley, D. Ozair, S. Courville, A. Bengio, Y.(2014). 'Generative Adversarial Nets'. *Proceedings of the International Conference on Neural Information Processing Systems* (NIPS 2014) Available at: https://arxiv.org/abs/1406.2661 (Accessed 28 April 2022)

[11] Xie, Y. Zhang, T. "Imbalanced Learning for Fault Diagnosis Problem of Rotating Machinery Based on Generative Adversarial Networks," *2018 37th Chinese Control Conference (CCC)*, 2018, pp. 6017-6022, doi: 10.23919/ChiCC.2018.8483334

[12] Gupta, T. 2017 Deep learning: feedforward neural network, *towards data science*, Available at: https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7 (Accessed 29th April 2022)