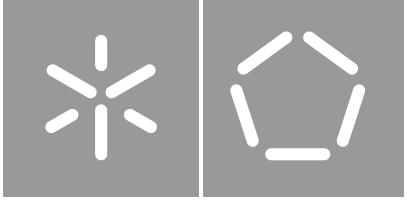
University of Minho
School of Engineering

Maria Elisa Maciel Valente

Network Anomaly Detection using Adversarial Deep Learning

January 2021



University of Minho
School of Engineering
Department of Informatics

Maria Elisa Maciel Valente

Network Anomaly Detection using Adversarial Deep Learning

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Miguel Francisco de Almeida Pereira da Rocha
Miguel Joaquim Garcia Rio

January 2021

Copyright Notice

To my family and friends

- for their love, endless support and encouragement

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Abstract

Computer networks security is becoming an important and challenging topic. In particular, one currently witnesses increasingly complex attacks which are also bound to become more and more sophisticated with the advent of artificial intelligence technologies.

Intrusion detection systems are a crucial component in network security. However, the limited number of publicly available network datasets and their poor traffic variety and attack diversity are a major stumbling block in the proper development of these systems.

In order to overcome such difficulties and therefore maximise the detection of anomalies in the network, it is proposed the use of Adversarial Deep Learning techniques to increase the amount and variety of existing data and, simultaneously, to improve the learning ability of the classification models used for anomaly detection.

This master's dissertation main goal is the development of a system that proves capable of improving the detection of anomalies in the network through the use of Adversarial Deep Learning techniques, in particular, Generative Adversarial Networks. With this in mind, firstly, a state-of-the-art analysis and a review of existing solutions were addressed. Subsequently, efforts were made to build a modular solution to learn from imbalanced datasets with applications not only in the field of anomaly detection in the network, but also in all areas affected by imbalanced data problems. Finally, it was demonstrated the feasibility of the developed system with its application to a network flow dataset.

Keywords: Network Security, Anomaly Detection, Deep Learning, Generative Adversarial Networks

Resumo

A segurança das redes de computadores tem-se vindo a tornar num tópico importante e desafiador. Em particular, atualmente testemunham-se ataques cada vez mais complexos que, com o advento das tecnologias de inteligência artificial, tendem a tornar-se cada vez mais sofisticados.

Sistemas de deteção de intrusão são uma peça chave na segurança de redes de computadores. No entanto, o número limitado de dados públicos de fluxo de rede e a sua pobre diversidade e variedade de ataques revelam-se num grande obstáculo para o correto desenvolvimento destes sistemas.

De forma a ultrapassar tais adversidades e consequentemente melhorar a deteção de anomalias na rede, é proposto que sejam utilizadas técnicas de *Adversarial Deep Learning* para aumentar o número e variedade de dados existentes e, simultaneamente, melhorar a capacidade de aprendizagem dos modelos de classificação utilizados na deteção de anomalias.

O objetivo principal desta dissertação de mestrado é o desenvolvimento de um sistema que se prove capaz de melhorar a deteção de anomalias na rede através de técnicas de *Adversarial Deep Learning*, em particular, através do uso de *Generative Adversarial Networks*. Neste sentido, primeiramente, procedeu-se à análise do estado de arte assim como à investigação de soluções existentes. Posteriormente, atuou-se de forma a desenvolver uma solução modular com aplicação não só na área de deteção de anomalias na rede, mas também em todas as áreas afetadas pelo problema de dados desbalanceados. Por fim, demonstrou-se a viabilidade do sistema desenvolvido com a sua aplicação a um conjunto de dados de fluxo de rede.

Palavras-chave: Segurança das Redes de Computadores, Deteção de Anomalias, *Deep Learning*, *Generative Adversarial Networks*

Contents

List of Figures

List of Tables

Acronyms

A | B | C | D | E | F | G | H | I | J | M | P | R | S | T | U | W | X

A

AI Artificial Intelligence.

ANN Artificial Neural Network plural.

API Application Programming Interface plural.

Avg Average.

B

BGD Batch Gradient Descent.

BP Back-propagation.

Bwd Backward.

C

CICFlowMeter Canadian Institute for Cybersecurity Flow Meter.

CICIDS2017 Canadian Institute for Cybersecurity Intrusion Detection System 2017.

CNN Convolutional Neural Network.

CWE Common Weakness Enumeration.

D

DDoS Distributed Denial of Service.

DL Deep Learning.

DoS Denial of Service.

DT Decision Tree.

E

EMD Earth Mover's Distance.

F

FTP File Transfer Protocol.

Fwd Forward.

G

GAN Generative Adversarial Network plural.

GPU Graphics Processing Unit plural.

H

HOPOPT IPv6 Hop-by-Hop Option.

HPC High-Performance Computing.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

HULK Http-Unbearable-Load-King.

I

IDS Intrusion Detection Systems.

IP Internet Protocol.

J

JSD Jensen-Shannon Divergence.

M

MbGD Mini-batch Gradient Descent.

ML Machine Learning.

P

PIN Personal Identification Number.

PSH PuSH.

R

ReLU Rectified Linear Unit Function.

S

SGD Stochastic Gradient Descent.

SMOTE Synthetic Minority Oversampling TEchnique.

SQL Structured Query Language.

SSH Secure Shell.

T

Tanh Hyperbolic Tangent Function.

TCP Transmission Control Protocol.

TTUR Two Time-scale Update Rule.

U

UDP User Datagram Protocol.

URG URGent.

W

WGAN Wasserstein Generative Adversarial Network plural.

WGAN-GP Wasserstein Generative Adversarial Network - Gradient Penalty.

X

XSS Cross Site Scripting.

Introduction 1

> The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him; not on the chance of his not attacking, but rather on the fact that we have made our position unassailable.
>
> *The Art of War, Sun Tzu [35]*

This chapter introduces the essential topics of this master's dissertation. It starts by presenting the context and motivation that led to the development of this work, followed by the objectives to be achieved. This chapter finalises with the presentation of the document structure.

1.1 Context and Motivation

We are living in the age of data and communication, where almost everything is connected by technology. Personal, commercial, military, and government information flow through networking infrastructures at every second, allowing to communicate and collaborate in an infinite number of ways [51]. Data connectivity grants prosperity of information and knowledge, but sensitive information, such as confidential files, requires extra security to avoid it falling into the wrong hands.

Today's era is the most powerful on a technological level. However, one must keep in mind that with great power must also come great responsibility. As the internet evolves, and computer networks become more extensive and prominent, data security becomes one of the most important aspects to consider in data communication [97]. Accordingly, network security must follow such evolution, preserving the confidentiality, integrity, and availability of system resources and information.

"Security is a journey, not a destination" [31]. Network architectures are complex and face ever-changing threatening environments [26]. Ciphers, that until a few years ago would take decades to decipher, can now be broken in a few minutes using High-Performance Computing (HPC) [53]. Malicious agents are always trying to find and exploit vulnerabilities creating new attacks that are bound to become more and more sophisticated with the advent of Artificial Intelligence technologies. It is, therefore, necessary to fight back with the same weapons.

Intrusion Detection Systems (IDS) are essential for improving the overall security of computer systems. Still, most techniques used in today's IDS are unable to address the progressive and complex

nature of attacks on computer networks [99]. Efficient adaptive methods, such as deep learning, can improve the response to dynamic and sophisticated attacks. However, detecting network anomalies using deep learning techniques requires large sets of data. Just as there is no knowledge without information, there is no deep learning without data.

Unfortunately, network flow datasets are usually not publicly available and not shared with the research community to comply with privacy concerns. Furthermore, datasets which contain realistic user behaviour and up-to-date attack scenarios are very scarce [79] and, in most of the cases, suffer from multi-class imbalanced data issues.

In such a context, this project aims to address the sparse availability of data by building a generative adversarial deep learning model responsible for producing large sets of network flow data while improving the detection of anomalies and disguised attacks.

1.2 Objectives

Having in mind the previous context and motivation, the main purpose of this dissertation is to develop algorithms and tools to allow the detection of network anomalies/attacks using adversarial deep learning methods. To accomplish this goal, it is necessary to:

- Review the state-of-the-art in computer networks attacks and anomaly detection based on traffic flow, as well as in adversarial deep learning;
- Study existing datasets on anomaly detection and software libraries for deep adversarial learning and their application for anomaly detection;
- Develop tools to enable training and validating deep learning models for the identification of different anomalies from network flow data;
- Develop tools to enable the generation of new data for different network anomalies/attacks and for all different fields that suffer from data imbalance problems;
- Develop a complete adversarial deep learning pipeline able to generate network anomaly detection classifiers and data to challenge these classifiers;

1.3 Document Structure

This dissertation is organised in six chapters. This first one, an introductory chapter that contextualises and motivates on the subject in study, and five more chapters:

- *Chapter 2: Literature Review*

  This chapter describes theoretical and scientific concepts related to this dissertation, as well as state-of-the-art technologies. A basic introduction of network security is provided, the fundamentals of deep learning are explained, and related works are reviewed.

- *Chapter 3: The Problem and Its Challenges*

  In this chapter, it is presented the problem in study and the proposed approach - solution. Alongside this, the challenges that will be faced during the development of this project are highlighted.

- *Chapter 4: Development*

  This chapter presents the system architecture and explains the details of its development. The specific methods and tools behind the system implementation are also described.

- *Chapter 5: Results*

  In this chapter, the proposed system is tested using an imbalanced network flow dataset (CICIDS2017). The obtained results are presented and discussed.

- *Chapter 6: Conclusion*

  This chapter provides the final review and conclusions of the overall work, along with the improvements that may be explored in the future.

Literature Review 2

This chapter describes the theoretical and scientific concepts related to this dissertation, as well as the state-of-the-art technologies. Section 2.1 exposes the security concepts (2.1.1), the two general categories of attacks (2.1.2) and two approaches for detecting intrusions (2.1.3). Section 2.2 presents the history behind deep learning and its fundamental concepts, such as artificial neural networks (2.2.1), artificial neurons (2.2.2), network architectures (2.2.3), gradient descent (2.2.4), and back-propagation (2.2.5). In section 2.3, the adversarial deep learning concept is introduced as well as some generative adversarial networks architectures are presented and explained. To conclude this chapter, section 2.4 brings forward some work related to the use of generative adversarial networks in the field of network anomaly detection.

2.1 Network Security

The safeguard of private information is a concern intrinsic to human beings. Confidential documents already existed long before the first computer, and physical means, such as safes, were used to store sensitive information. The introduction of personal computers in the 1980s made indispensable the use of tools for protecting files and other private information stored on computers [86]. However, it was only with the arrival of distributed systems, and with the use of networks and communications infrastructures for carrying data between devices, that network security arose to protect data during transmission [90].

Several incidents marked the history of network security. One of the most recent ones, in February of 2018, was the Distributed Denial of Service (DDoS) attack to GitHub, an online code management service used by millions of developers [18]. This attack was the most significant distributed denial of service attack recorded to date, with incoming traffic rates of 1.35 terabytes per second. The goal of such attack is to exhaust the target resources, preventing legitimate users from having access [29].

Over the years, the number of security incidents has been increasing. As shown in the figure 2.1, 2018 came in as the second most active year (missing 2017's high mark by only 3.2%), with 6515 publicly disclosed breaches reported and approximately 5 billion records exposed. The exposed records came from the business (66.2%), governmental (13.9%), medical (13.4%) and education (6.5%) sectors [85].

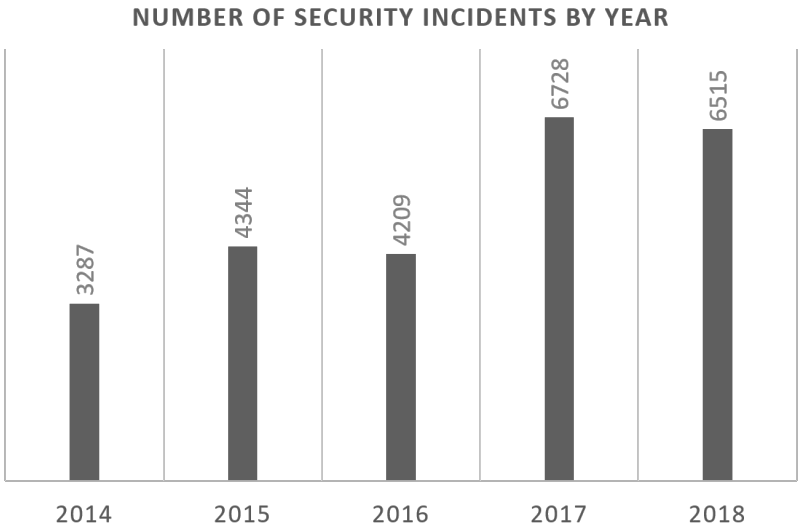


Figure 2.1: Data breach chart report - Number of security incidents between 2014 and 2018. Source: [85].

2.1.1 Security Concepts

The field of network security is in constant growth and adjustment due to the evolutionary stage of technology [51]. Although the network security field is vast, at the heart of any security concern resides three key objectives:

- *Confidentiality*

  The motivation of confidentiality is the preservation of data privacy. It assures that private or classified information is not made available or disclosed to unauthorised users.

  Data encryption is a standard method of preserving confidentiality. Still, attacks of confidentiality continue in an attempt to obtain private information like a user's credentials and credit card information [51, 90].

- *Integrity*

  Integrity refers to the ability to ensure that data is accurate and reliable. Data must be an unchanged representation of the original source information, preserving the content and source of the data [51]. Unauthorised modification or destruction of information are some of the silent attacks that compromise the integrity of the data. These attacks can occur when some data is intercepted during transmission, altered or changed, and sent to the intended receiver. The receiver thinks that the received content is as it is supposed to be, but instead, it is corrupted [31].

- *Availability*

  Availability grants access to information to authorised users. This property ensures that the information is accessible and usable upon demand for every authorised entity [31].

  The biggest threat to availability is Denial of Service (DoS) attacks that attempt to prevent legitimate users from using a service or delaying time-critical operations.

  Availability may be the most critical aspect of network security for commercial organisations. According to the security company Cloudflare, the average cost of a successful distributed DoS attack is about $100,000 for every hour the attack lasts. Besides, the dominant impact of a DoS attack on organisations is the loss of reputation, which results in a loss of customers [19].

In summary, confidentiality is a set of rules that limits access to information, integrity is the assurance that the information is trustworthy and accurate, and availability is a guarantee of reliable access to the information by authorised people. This triad constitutes the most crucial components of a security model.

2.1.2 Security Attacks

Attacks on computer networks date back to the beginning of more extensive use of network infrastructures. An attack is the usage of a technique to exploit a vulnerability and, consequently, break the security concepts [86]. There are two general categories of attacks:

- *Passive*

  A passive attack attempts to learn or make use of information residing in a network without involving any data alteration or modification on the network operational conditions [90]. Passive attacks include traffic analysis, collection of confidential information, such as authentication credential, and release of message contents. This kind of attack is challenging to detect because the attacker is merely a silent observer [51, 86].

  Although the intent of a passive attack is not to sabotage user's access to the physical infrastructure, this attack can be utilised to gather information which can later be employed in much more harmful active attacks [86].

- *Active*

  An active attack attempts to modify data or affect system resources operations. Active attacks result in a change of entities (when one entity pretends to be a different one - masquerade),

replay or modification of data, and DoS [90]. The most common active attack is the DoS. According to the report *Cyber Attack Trends Analysis* [77], 49% of organisations experienced a distributed DoS attack in 2018.

Active attacks are much more devastating to a network than the passive ones, even considering that the former are more detectable as they often affect the target in ways that raise awareness of such an attack [51, 90].

The two categories of attacks differ in several aspects, nonetheless, both are very harmful. The first, passive attack, can acquire all confidential information of a person, expose it and it could even apply an active attack to make use of it (e.g. using credit card credentials). The second, active attack, can destroy critical registers and even stop commercial websites, causing problems for users by preventing them from having access to a service. On the other hand, the organisation which provides the service is penalised with monetary losses.

2.1.3 Intrusion Detection

As already mentioned, the number of attacks, whether active or passive, has been increasing in recent years. Every year new types of attacks are generated which pose increasing difficulties. However, even if the prevention of attacks is challenging, detection could be easier. Detecting when an attack is underway or has taken place can be achieved using intrusion detection techniques [31].

Intrusion detection is the process of identifying potential incidents by dynamically monitoring events that occur on a computer system or network [80, 99]. Intrusion Detection Systems (IDS) emerge from the concept of intrusion detection. An IDS is a software application or device that monitors a network of computers by analysing data gathered by a set of sensors. Through the analysis of such data, an IDS can detect malicious activities where some examples are information theft and network protocol corruption [31, 99]. There are two approaches for detecting intrusions, namely, misuse detection and anomaly detection:

- *Misuse detection*

  Misuse detection refers to known attacks that exploit the known vulnerabilities of the system. IDS are very efficient in detecting attacks already established and consolidated in detection system databases. However, such efficiency implies to regularly update databases with the latest attacks. The database of known vulnerabilities and exploit methods can become large and

difficult to handle, slowing the IDS. Furthermore, misuse detection is ineffective in anticipating and detecting new attacks since it only detects the ones already known [31, 82].

- *Anomaly detection*

  Anomaly detection refers to the observation of unusual activity and the use of statistical techniques to detect potential intrusions. The IDS does not need to know the security vulnerabilities of a particular system as a misuse detection system does. Anomaly detection is based on the assumption that an attacker's behaviour differs from a legitimate user's behaviour in ways that can be quantified. To that, the "normal" user behaviour is established as a baseline that defines normality. If the statistical analysis of the data monitored exceeds a threshold of the baseline, an intrusion is detected. However, the exact distinction between an intruder and a regular user can sometimes overlap. If a legitimate user exhibits an unusual behaviour, he may be considered as an intruder leading to a false alarm. On the other hand, when the baseline is adjusted dynamically and automatically, a particular attacker may gradually shift the defined "normal" user behaviour until his planned attack no longer deviates from the baseline. Consequently, no alarm is generated upon an attack [31, 90].

Intrusion detection technologies are essential for improving the overall security of computer systems. Most techniques used in today's IDS, like firewalls, access control mechanisms, and encryption, are unable to address the progressive and complex nature of attacks on computer networks, making impossible the total protection of networks and systems [99].

Efficient adaptive methods, such as machine learning techniques, can improve the response to dynamic and sophisticated attacks. As a result, higher detection and lower false alarm rates can be accomplished and within an adequate window of computing and communication time [99].

2.2 Deep Learning

By means of Artificial Intelligence (AI), computer science has made progress in building intelligent machines characterised as displaying human behaviours [8, 33]. Over the years, AI has been evolving into new fields such as machine learning and deep learning and, by them, the existence of autonomous driving cars [24], voice and face recognition machines, and digital assistants such as Google Now [33] have become possible.

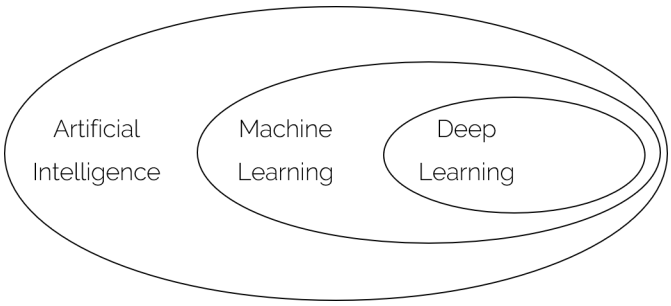


Figure 2.2: Artificial intelligence sub fields.

Artificial Intelligence, Machine Learning (ML) and Deep Learning (DL) are terms that have been used a lot these days, however, there is still some lack of knowledge about their differences:

- *Artificial Intelligence*

  Artificial intelligence was born from the wish to automate intellectual tasks usually only performed by humans. As such, AI is a general field of computer sciences that covers machine learning and deep learning, and many other approaches that don't involve any learning, as shown in figure 2.2. AI deals with intelligent agents, who have included into their programming a large set of explicit rules for manipulating knowledge. The agents are configured to perceive their environment and act autonomously, seeking to maximise the chance of achieving a certain goal, always following the established rules — e.g., chatbots [9, 17].

  AI solves problems intellectually difficult for humans, but relatively straightforward for computers. The true challenge in artificial intelligence is to solve tasks that are easy for people to perform but hard for people to describe formally, that is, tasks that are performed intuitively - e.g., speech or images recognition [33].

- *Machine Learning*

  Machine Learning began to gain popularity in the 1990s due to faster hardware availability and larger datasets. It describes computer programs that learn to solve tasks by learning from data rather than being explicitly programmed. Thus, a machine learning system is presented with many examples relevant to a task, and by them it finds a statistical structure that becomes rules for automating the task - e.g., spam filters [4, 17].

  The learning process can be supervised (the system generates rules based on a known and labelled dataset) or unsupervised (the system generates rules on unknown and unidentified data). Supervised learning is commonly used for classification and prediction and unsupervised learning to find patterns in datasets [96].

- *Deep Learning*

  Despite being a fairly old machine learning sub field, deep learning only gained prominence in early 2010s [17]. Its popularity emerged for two main reasons: it was discovered that some computational deep learning techniques, like Convolutional Neural Network (CNN), run much faster on GPUs and that the increasing amount of available training data can be used to improve the accuracy of computer vision algorithms [9, 96].

  Deep Learning is a new approach to learning representations from data. It splits the data into a hierarchy of concepts, where each concept is defined by its relation to simpler ones. This allows to build complex concepts from simpler concepts [17, 33]. A deep learning model learns the features that are important to a task by itself, without any manual selection of pertinent features as it happens in machine learning models [5, 17]. The central problem in machine learning and deep learning is to meaningfully transform data — that is, to learn useful representations of input data [17].

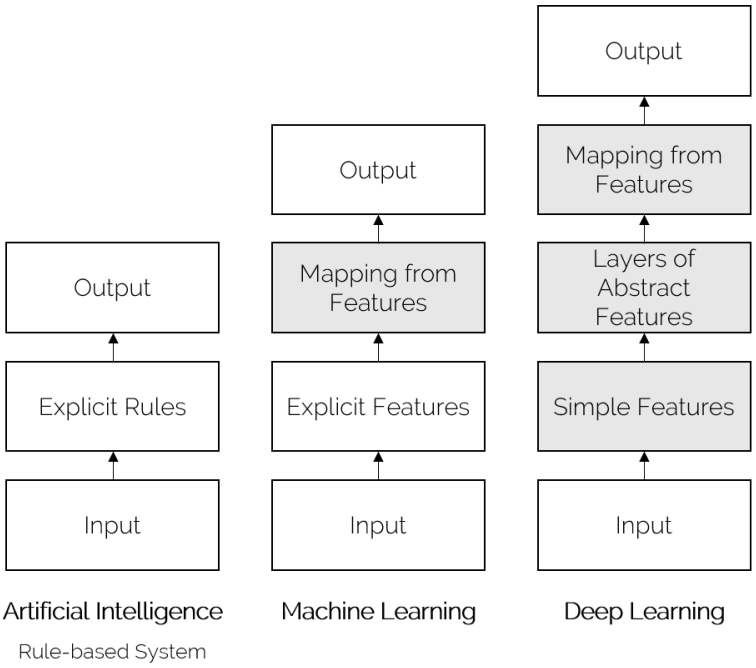


Figure 2.3: Flowcharts of the different three fields (AI, ML, and DL) described before. Gray boxes indicate components that are able to learn from data during training [9, 33].

Over time, scientists have been trying to bring computers closer to human complexity. Despite computers having a higher logical capacity than humans, such as being able to perform complex mathematical operations or store large amounts of data, humans overcome them on rather intuitive or cognitive tasks like natural language or perception [9].

The field of artificial intelligence aims to be a bridge between humans and computers, bringing genuine intelligence to machines and programs [9]. To date, this bridge has been able to give computers human capabilities such as case-based reasoning, image classification, and speech recognition at the near-human level. This has been accomplished through the field of artificial intelligence and its sub fields, in particular machine learning and deep learning that focus on transferring human perception capabilities to machines [17]. A summary of how these fields operate is shown in figure 2.3.

2.2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computer models inspired on how the human brain processes information. Thus, ANNs are adaptive systems with an interconnected structure of multiple computational units, neurons, acquiring knowledge through experience [39]. These systems resemble the behaviour of a brain in two respects:

1. Knowledge is acquired from an environment through a learning process [39];
2. Acquired knowledge is stored in connections, links or synapses, between nodes [39].

The behaviour of an ANN comes from interactions between neurons. While acquiring information through experience and observation is the most traditional way to construct ANNs, a neural network can still modify its own topology, similar to what goes on in the brain when neurons die and new synaptic connections grow [39].

2.2.2 Artificial Neurons

The artificial neuron or node, the information processing unit, is the elementary component of ANNs. It implements a mathematical function, that processes multiple input signals and provide only one output signal [9]. From figure 2.4 three basic elements of an artificial neuron can be identified.

- A *set of input connections*, each one characterised by its own weight [39, 61].
- An *adder* that reduces the input signals and respective weights to a single value, representing a linear combination. This adder also receives a bias, a constant, that helps the model better fit the given data, becoming more flexible [39]. For the training phase, the bias is associated with a trainable weight that can be modified as any other weight. Thus, since the bias is a constant value B, the bias representation in the training phase is then given by $b_k = w_{bk}B$ [20].

- An *activation function*, that the neuron applies to the result of the input sum, limiting the amplitude range of the output signal [39, 61].

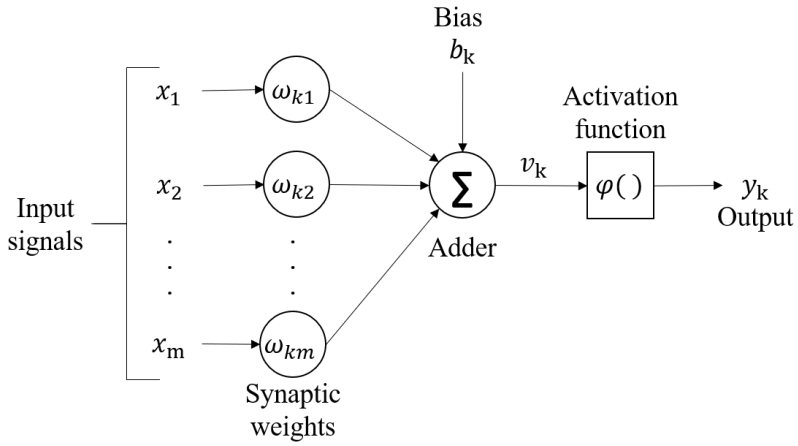


Figure 2.4: Model of a neuron k.

To describe an artificial neuron (k) in a mathematical form, the following equations should be used:

$$v_k = \sum_{i=1}^{m} x_i w_{ki} + b_k \tag{1}$$

$$y_k = \varphi(v_k) \tag{2}$$

where $x_1, x_2, ..., x_m$ are the input signals; w_{k1}, w_{k2}, w_{km} are the synaptic weights; b_k is the bias; $\varphi()$ is the activation function and y_k is the output signal.

Some of the most commonly used activation functions are for solving non-linear problems, transforming the linear result of equation 1 into a non-linear output [20, 44]. The expected output determines the type of activation function to be deployed in a given network [67]. As there are different types of deep learning architectures, there are different types of activation functions, each one targeted to different domains (object classification, segmentation, machine translation, cancer detection, and other adaptive systems) [67]. Below are presented some of the most popular activation functions.

- *Sigmoid Function*

  A sigmoid function is a limited differentiable real function, which maps to the range of values to [0,1], defined for all real input values [37, 50]. This function has been successfully used in binary classification problems and modeling logistic regression tasks, being applied mostly in

shallow networks [67]. The graph of the sigmoid function is represented in figure 2.5 and its mathematical form is given as follows:

$$f(x) = \left(\frac{1}{1 + e^{-x}} \right) \tag{3}$$

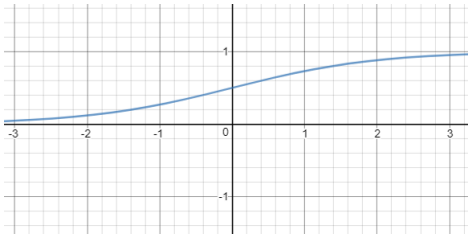


Figure 2.5: Sigmoid function graph.

- *Hyperbolic Tangent Function (Tanh)*

  The shape of the hyperbolic tangent function is similar to the S-shaped sigmoid function [20]. Tanh is a zero-centred function with its range outputs between -1 and 1, used mostly in recurrent neural networks. This function is better than sigmoid for multi-layer neural networks since it shows higher training performance [67]. The graph of the hyperbolic tangent function is represented in figure 2.6 and its mathematical form is given as follows:

$$f(x) = \left(\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \right) \tag{4}$$

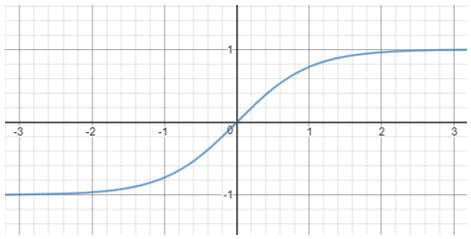


Figure 2.6: Hyperbolic tangent function graph.

- *Softmax Function*

  The softmax function computes a probability distribution from a vector of real numbers, turning the input numbers into probabilities that sum to one. Since its output is a vector that represents the probability distributions its range of values is between 0 and 1. This function is used mostly

in output layers of deep learning multi-class models where it returns the vector of probabilities of each class, being the target class the one with the highest probability [67]. The softmax function is given as follows:

$$f(x_i) = \left(\frac{e^{x_i}}{\sum_j e^{x_j}} \right) \tag{5}$$

- *Rectified Linear Unit Function (ReLU)*

  The rectified linear unit function is currently the most successful and widely-used activation function [75]. It is a simple measure that returns the input value directly (like a linear function), or 0 if the input value is less than 0 [2]. The ReLU mathematical form is given as follows:

$$f(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{6}$$

When comparing all four activation functions, ReLU is the one that offers more advantages. The most prominent one is its computational simplicity. Unlike the sigmoid, Tanh, and softmax functions that require the use of exponential equations, the rectified linear unit only requires the use of a $max()$ function [30]. In addition to its computational simplicity, ReLU also delivers better results in the training of deep neural networks than the sigmoid or Tanh activation functions.

A general problem with these two last mentioned functions is that they usually saturate, leading to vanishing gradients in deep neural networks. In sigmoid and Tanh, large input values are set to 1 and small input values (large negative values) are set to -1 for Tanh and to 0 for sigmoid [33].

Since ReLU is linear for half of the input domain and nonlinear for the other half, it preserves many of the properties that make linear models easy to optimise with gradient-based methods, and as such is commonly used in the hidden layers of deep neural networks [33].

2.2.3 Network Architectures

The manner in which the neurons are interconnected in a network structure is called topology. In a neuron connection each neuron is organised by layers, classified in three groups: *input layer* where the information is presented to the network; *hidden layer* where most of the processing is done through the weighted connections; *output layer* where the final result is concluded and presented to the user [39]. There are numerous types of topologies, each with their own potential. In general, they fall into two categories:

- *Feedforward neural network*

  The connections between nodes, in a feedforward neural network, are always in just one direction, starting from the input layer and ending in the output layer. In its simplest form a network is composed of only one input layer and one output layer (single-layer feedforward networks). However, to increase the network's ability of modelling more complex functions, one or more hidden layers are added between the input and output layers (multi-layer feedforward networks) [39]. One example of a multi-layer feedforward neural network can be seen in figure 2.7.

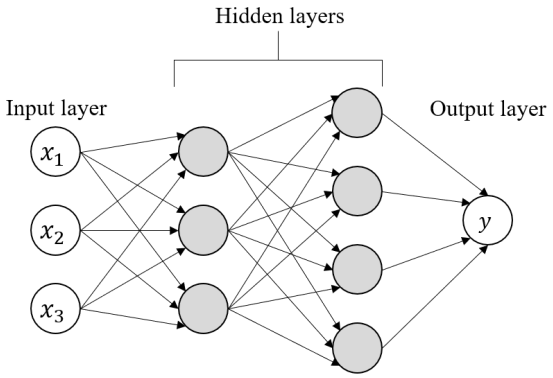


Figure 2.7: Feedforward neural network example.

- *Feedback neural network*

  A feedback neural network is a multi-layer architecture characterised by multi-directional connections, being often called of cyclic or recurrent network [39]. Recurrence exists in dynamic systems when an output of an element somehow influences the input to that same element, e.g. - information can cycle inside the network [60]. Contrary to feedforward networks, feedback networks can be adapted to past inputs, maintaining a short-term memory that is very helpful to sequential data prediction [10]. Figure 2.8 shows one example of a feedback neural network.

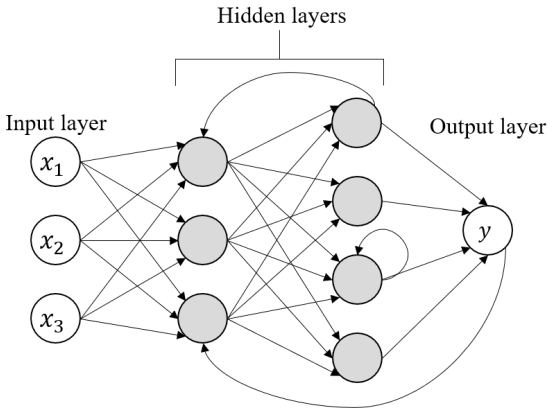


Figure 2.8: Feedback neural network example.

Choosing the right architectures for neural networks is not an easy task. The selected architecture should have enough layers to approximate the function of interest, but not too many because it will take an excessive amount of time to train and it could over-fit the model. The topology should be chosen taking into account the amount of training data (more data usually requires more layers) and the type of problem. For example, if the problem involves time series, a feedback architecture should be chosen.

2.2.4 Gradient Descent

The weights or trainable parameters, w and b, represented in the equation 1, are the most important attributes of a neuron. It is in the weights that the information learned by the network from exposure to training data is stored. As a starting point, these weights are filled with small random values. Then, in the training phase, it is measured how far the output is from what it is expected (according to a loss function), and based on that, the weights are adjusted step-by-step until the initial trainable parameters are close enough to the optimum [12, 17].

Gradient descent, one of the most popular learning algorithms to optimise neural networks is used to adjust the weights [81]. Its role is to minimise a differentiable objective function $f(x)$ by updating the parameters x in the opposite direction of the gradient $\nabla x f(x)$ [81]. Applied to a neural network, $f(x)$ would be the loss function, x would be the weights values and the gradient descent result would be the combination of weight values that yields the smallest possible loss function [17]. The gradient descent is applied in a training loop as represented in figure 2.9 and described below:

1. Initialise the weights W, define the learning rate and the number of epochs.
2. Take the training samples X and corresponding targets Y.
3. Run the network on X to obtain the predictions Y'.
4. Calculate the loss function of the network, i.e., the mismatch between Y 'and Y
5. Compute the gradient of the loss function with regard to the network's weights.
6. Move the weights slightly in the opposite direction from the gradient, reducing the loss, e.g. — W = W - learning rate * gradient.
7. Repeat the steps 3, 4, 5 and 6 until reaching the number of epochs defined.

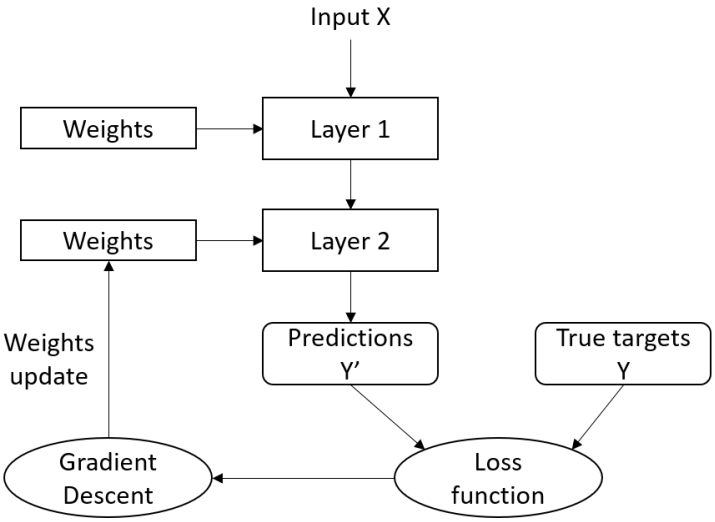


Figure 2.9: Gradient descent training loop.

When the training datasets are very large, the computational complexity of the learning algorithm becomes a critical factor. In these situations, a trade-off between accuracy and performance is required [81], leading to three distinct gradient descent variants:

- *Batch Gradient Descent (BGD)*

  The batch gradient descent, also known as vanilla gradient descent, computes the gradient of the loss function for the entire training dataset. It guarantees the convergence to the global minimum for convex error surfaces [89] and to a local minimum for non-convex [81]. As this algorithm uses the whole training dataset, its computation can be very slow.

- *Stochastic Gradient Descent (SGD)*

  The term stochastic refers to the fact of randomly pulling the data to be applied in the gradient descent [17]. This algorithm only performs one update for each example at a time and, as a result, is much faster than BGD. On the other and, its convergence to the exact minimum is not so easy as its updates are performed with a high variance [81].

- *Mini-batch Gradient Descent (MbGD)*

  This algorithm joins the best of the two other approaches (BGD and SGD) doing an update for every mini-batch of n training samples, as represented in the algorithm 1.

Algorithm 1 Mini-batch Gradient Descent

```
for each epoch do
    random shuffle of the data
    for batch in batches(data, batch_size = 50) do
        weights_grad = evaluate_gradient(loss function, batch, weights)
        weights = weights - learning rate * weights_grad
    end for
end for
```

In MbGD the computation is much faster than in BGD and the variance of the weight updates is smaller than in SGD, leading to a more stable convergence [81]. As result, a better trade-off between accuracy and performance is accomplished, making the mini-batch gradient descent the most commonly used algorithm to train neural networks [81].

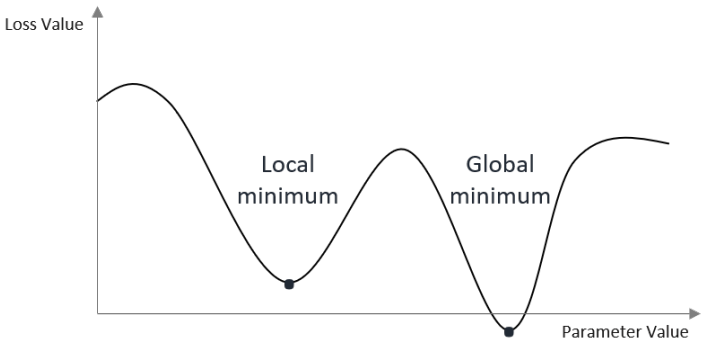


Figure 2.10: Local and global minimum of loss function.

A very important factor in gradient descent algorithms is the learning rate value. This defines the size of the step to be taken in the direction of the steepest descent [94]. If its value is too small, the probably of staying stuck in a local minimum (figure 2.10) is higher. If its value is too big, the updates can lead to random moves on the curve.

2.2.5 Back-propagation Algorithm

As previously mentioned, in a feedforward neural network architecture the information flows forward through the network like in directed acyclic graphs [5]. When this architecture has multiple layers,

the initial information x, received by the input layer, is propagated through the hidden layers until the output layer where y' is produced [33].

The Back-propagation (BP) algorithm, proposed by Rumelhart, Hinton and Williams in 1986, intends to compute the gradient of a loss function of multi-layer ANNs applying the chain rule. With the BP algorithm, the gradient is calculated from the top to the bottom layers and applied in a gradient descent algorithm to perform the learning [20, 33]. For example:

$$y' = f(x) = \varphi(w^L ... \varphi(w^2 \varphi(w^1 x + b^1) + b^2) ... + b^L) \tag{7}$$

This is the result of a L-layered neural network function which consists of many tensor operations chained together. In case of $L = 2$, the function can be written as

$$y' = f(x) = f(g(x)) \tag{8}$$

According to the chain rule, a chain of functions can be derived as follow:

$$\frac{\partial y'}{\partial x} = \frac{\partial f(x)}{\partial x} = f'(g(x)).g'(x) \tag{9}$$

Thus, the back-propagation algorithm allows the information to flow backward through the network in order to compute the contribution that each parameter had in the loss value [17, 33].

None of the algorithms previously presented guarantee full convergence to the global minimum (figure 2.10). In an attempt to improve this convergence, new methods are emerging. Momentum [81], Adagrad [62], RMSProp [52] and Adam [101] are some variants of gradient descent known as optimisation methods or optimisers that increase the speed of convergence and decrease the probability of staying blocked in a local minimum [17].

2.2.6 Properties of Deep Learning

Over the years, deep learning has yielded tremendous success in the field of AI. Near-human-level autonomous driving, near-human-level speech recognition, and improved machine translation are some of the most significant deep learning achievements to date. All these technological progress has been made thanks to the properties offered by deep learning. The three most distinctive properties are robustness, generalisation and scalability:

- *Robustness*

  Deep learning approaches do not require any manual selection of pertinent features. Instead of that, optimal features for the task are automatically learned from exposure to training data

[17] [5]. Accordingly, deep-learning models can be trained on additional data without restarting from scratch, becoming robust to the natural variations in the data and viable for continuous online learning [17].

- *Generalisation*

  Deep learning has the human ability of generalisation - reasoning about new concepts and experiences from just a single example [78]. This is converted in the faculty of facing a new concept, understanding its structure, and then being able to generate valid alternative variations of the concept [78]. Thereby, the same deep learning approach can be used in different applications or with different data types [65], being truly helpful when the problem does not have enough available data [5] or the number of parameters is substantially larger than the amount of training data [65].

- *Scalability*

  The deep learning computational requirements allow almost all deep learning models to be trained on GPUs [16] [22], taking full advantage of their parallelism capabilities [17]. Furthermore, the deep-learning models are trained by iterating over mini-batches of data [94] (maximising the GPUs parallelism potential), allowing them to be trained on datasets of arbitrary size [17], resulting in highly scalable deep learning approaches [5].

2.3 Adversarial Deep Learning

Adversarial learning emerges to study attacks on deep learning models and their respective defences. This technique shows how a malicious adversary can manipulate input data to exploit specific learning algorithm vulnerabilities and compromise the security of the deep learning system. The purpose of this technique is not to model the classification algorithm perfectly, but to identify with great certainty the malicious instances that were not previously identified as such [58].

2.3.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a recent unsupervised deep learning approach for estimating generative models, introduced by Goodfellow et al. [34] in 2014. They are a generative modelling system that, given a set of images, generates realistic synthetic images nearly identical to the real ones [33] [17]. GANs are based on an adversarial game scenario where two neural networks compete against each other [5], the generator and the discriminator, as shown in figure 2.11.

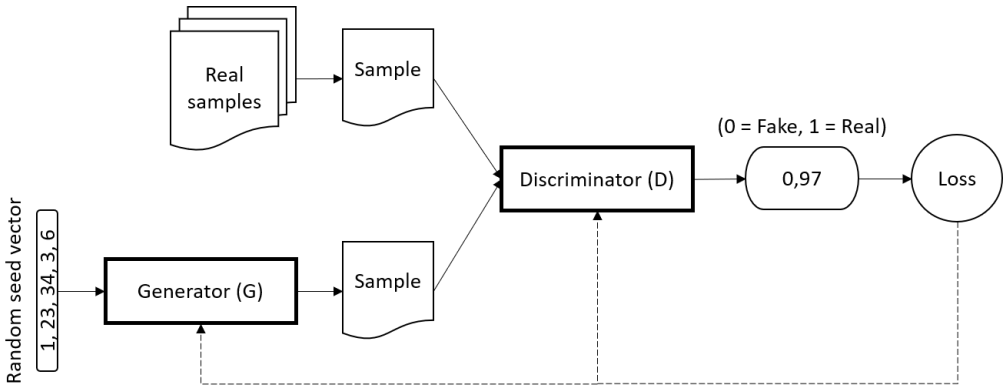


Figure 2.11: Conceptual diagram for Generative Adversarial Networks.

The generator has the function of capturing the training data distribution and generating new data, while the discriminator has to identify if a sample came from the original training data or the generated data. More specifically:

- *Generator Network*: The generator receives as input a random seed vector z (random point in latent space) and decodes it into a realistic synthetic image, similar to a real training sample [17] [55].
- *Discriminator Network*: The discriminator receives as input images from the generator network and actual training samples (although, the discriminator doesn't know the source of the images) and, for each image, indicates the probability of being from a real training example or from a synthetic sample drawn by the generator [33].

In this adversarial game scenario, it is expected that, over the iterations, the generator will become more and more competent at reproducing the training data distribution, deceiving the discriminator. Simultaneously, the discriminator will constantly adapt to the gradual improvement of the generator images, becoming increasingly expert at detecting fake samples [17]. As a result, it is a game where everyone wins, as the improvement of one network leads to the improvement of the other one.

Training process

One of the biggest challenges in the study of GANs is the training phase [6]. Unlike the other models that aim to achieve the minimum point of the cost function, in a GAN system, the main goal is to accomplish a balance between the two networks. The GAN loss function quantifies the similarity

between the generated data distribution ($G(z)$) and the real data distribution (x) through the Jensen–Shannon Divergence (JSD) [54].

As a GAN is a dynamic system, there is no fixed minimum and every step taken by the gradient descent changes the entire system [17]. The following equations represent the generator and discriminator gradients, respectively.

- *Generator gradient:*

$$-\nabla_{\theta_g}\frac{1}{m}\sum_{i=0}^{m} log(1 - D(G(z^{(i)}))) \tag{10}$$

- *Discriminator gradient:*

$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=0}^{m}[log(D(x^{(i)})) + log(1 - D(G(z^{(i)})))] \tag{11}$$

where m represents the number of samples, z the noise samples, x the real samples, D and G the discriminator and generator networks respectively. During the training phase, the generator is updated by ascending its stochastic gradient and the discriminator is updated by descending its stochastic gradient.

2.3.2 Wasserstein GAN

The Wasserstein GAN (WGAN) introduced by Arjovsky et al. [7] in 2017, is a GAN variant that uses the Earth Mover's Distance (EMD) [71] (also called Wasserstein-1 distance) to measure the difference between the true distribution and the model distribution, instead of the Jensen-Shannon divergence [54] in a standard GAN. The EMD is used to address the JSD gradient vanishing problem. With the JSD, as the difference between the real and generator distributions increases, the discriminator gets close to the ideal (it detects the fake samples very well since the generated samples are so distinct from the real ones) but, on the other hand, the generator becomes unable to improve its approximation to the real samples distribution. This happens because with an optimal discriminator the generator gradient will be zero almost everywhere (see equation 10), being unable to learn anything from the gradient descent.

The EMD has a smoother gradient that does not saturate or blow up for distributions with different supports, as it happens with the JSD. This means that WGAN learns no matter how the generator is performing. The EMD still gets signals when the difference between the two distributions is big.

In order to be able to use the earth mover's distance instead of the JSD, the discriminator network requires a small adjustment. In this new model, the discriminator outputs a scalar score rather than a probability. This score indicates how real the input images are instead of the likelihood of being real. In light of this, in a WGAN, the generator and discriminator gradients, introduced in section 2.3.1 (equations 10 and 11), are replaced by the following ones:

- *Generator gradient:*

$$g_\theta \leftarrow \nabla_\theta \frac{1}{m} \sum_{i=0}^{m} f(G(z^{(i)})) \tag{12}$$

- *Discriminator gradient:*

$$g_w \leftarrow \nabla_w \frac{1}{m} \sum_{i=0}^{m} [f(x^{(i)}) - f(G(z^{(i)}))] \tag{13}$$

where θ and w are, respectively, the generator and discriminator parameters, and f is a 1-Lipschitz function [27] [40] that follows the $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$ constraint. In order to impose this constraint and prevent against vanishing or exploding gradients, it is required to limit the maximum weight value in f. For that, a very simple clipping is applied to the discriminator weights. To better understand, see the equations below.

$$w \leftarrow w + \alpha \cdot RMSProp(w, g_w) \tag{14}$$

$$w \leftarrow clip(w, -c, c) \tag{15}$$

where α is the learning rate and c the clipping parameter.

Figure 2.12 shows how the two models, GAN with the Jensen-Shannon divergence and a WGAN with the earth mover's distance, learn to differentiate two Gaussians. As it can be seen, the WGAN gradient is smoother everywhere when compared with the GAN gradient. Whereas with GAN, the model gets quality learning only if the generator is already producing samples similar to the real ones, with WGAN, the model is able to learn even if the generator is not producing good samples.

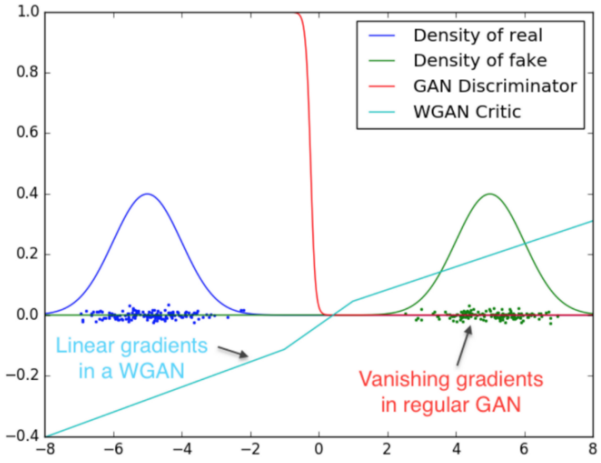


Figure 2.12: GAN and WGAN discriminators when learning to differentiate two Gaussians. The GAN discriminator (red line), saturates and results in diminishing or exploding gradients. In the WGAN discriminator (blue line), the gradient is smoother everywhere and learns better even when the generator is not producing good samples. Source: [7].

2.3.3 Wasserstein GAN - Gradient Penalty

The WGAN - Gradient Penalty, proposed by Gulrajani et al. [36], emerges to fix some of the issues of the previously presented WGAN model. The recently proposed Wasserstein GAN is making progress towards reliable GAN training. Yet, sometimes, WGAN may lead to undesired behaviour like the inability to converge or the generation of weak samples.

The challenge in WGAN is to enforce the 1-Lipschitz constraint. Weight clipping is easy, but it is not the best procedure to impose that restriction. The model performance gets very sensitive to this clipping parameter, and it is very difficult to establish the ideal clipping value [36]. If the clipping value is too large, the gradient can increase exponentially resulting in exploding gradients. If the clipping value is too small, the error signal going through each layer will be multiplied by small values during back-propagation, what can result in vanishing gradients when the number of layers is too high or the batch normalisation is not used [7].

A differentiable function is 1-Lipschitz if and only if the gradients norm is at most 1 everywhere [36]. This means that the points interpolated between real and generated data should have a gradient norm of 1 to obey the 1-Lipschitz constraint. Thus, instead of using weight clipping, WGAN-GP penalises the model if the gradient norm shifts away from the target norm value 1. The WGAN-GP discriminator loss function can be expressed as follows:

$$L = \underbrace{\mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]}_{\text{Original WGAN discriminator loss}} + \underbrace{\lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\parallel \nabla_{\hat{x}}D(\hat{x}) \parallel_2 -1)^2]}_{\text{Gradient penalty}} \tag{16}$$

where $\tilde{x} \sim \mathbb{P}_g$ ($\mathbb{P}_g$ is the model distribution defined by $\tilde{x} = G(z), z \sim p(z)$) represents a sample of generated data, $x \sim \mathbb{P}_r$ a sample of real data, and λ the penalty coefficient. $\hat{x}$ is sampled from $\tilde{x}$ and x with ϵ selected randomly between 0 and 1.

$$\hat{x} = \epsilon x + (1-\epsilon)\tilde{x} \text{ with } 0 \leq \epsilon \leq 1 \tag{17}$$

In the *"Improved Training of Wasserstein GANs"* paper [36] an experiment was made to see the differences between the two WGAN models (weight clipping and gradient penalty) during the training phase. In this study, the Swiss Roll dataset [92] was used and implemented four different models: a WGAN-GP and three WGANs with three different clipping values. The results can be seen in figure 2.13. These graphics express very well the weight clipping problem. This approach encourages the optimiser to push the absolute value of the weights towards the clipping value, which can lead to diminishing or exploding gradients depending on the c value. As can be seen in figure 2.13, as c increases from 0.001 to 0.1, the WGAN discriminator shifts from diminishing to exploding gradients.

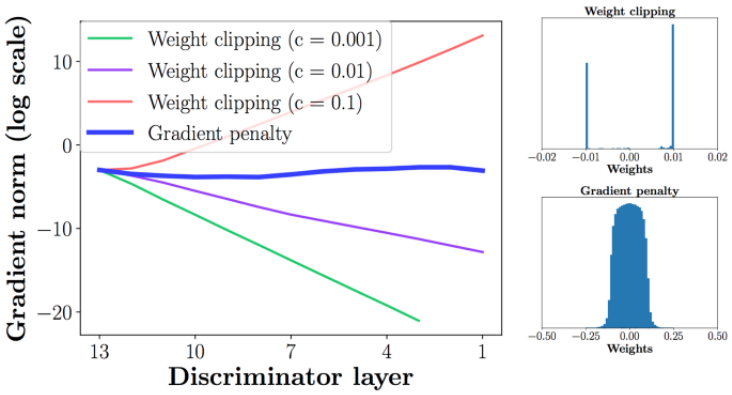


Figure 2.13: WGAN gradient penalty performance when compared with the original WGAN.

In the left graphic is shown the gradient norms of WGAN and WGAN-GP discriminators during the training on the Swiss Roll dataset. The gradient norms either explode or vanish when weight clipping is used, instead of the gradient penalty. In the right graphics, the weight clipping pushes weights towards two values (the extremes of the clipping range), unlike gradient penalty. Source: [36].

Unlike the weight clipping, the gradient penalty approach reveals an improvement in the training stability across a variety of architectures and helps in model convergence. Besides that, by imple-

menting a character-level GAN model on the Google Billion Word dataset [14], Gulrajani et al. have shown that WGAN-GP is also capable of modeling discrete distributions over a continuous latent space. However, the gradient penalty adds computational complexity.

2.4 Related Work

Anomaly detection is a significant problem that has earned the attention of many researchers. Over the years, several approaches have been studied in order to correctly detect and classify anomalous data, e.g. data which do not fit the normal data distribution.

More recently, Generative Adversarial Networks have been applied to this field [3, 84, 100]. In the original GAN formulation, the generator learns to map samples from an arbitrary latent distribution z to realistic synthetic data X', while the discriminator learns to distinguish between real and generated data. In state-of-the-art GAN-based anomaly detection algorithms, the original formulation is extended, incorporating the inverse mapping learning, which maps the data back to the latent representation z' [21]. Figure 2.14 highlights the architecture of a GAN-based anomaly detection model.

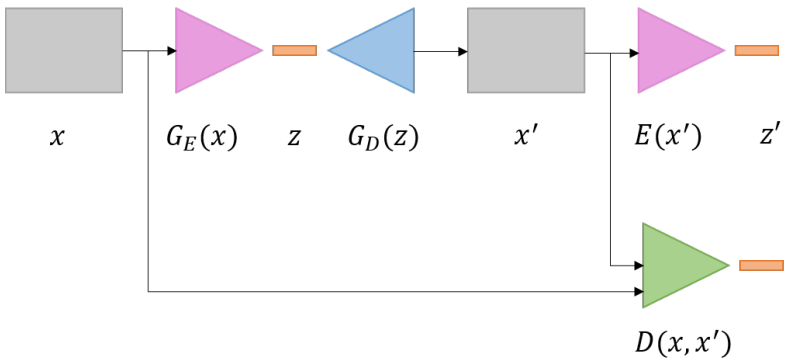


Figure 2.14: *GANnomaly* model architecture. With E being an encoder network, G_E a generator encoder and G_D a generator decoder.

During the *GANnomaly* training, the network is modelled on normal samples only. As a result, the model parameters become unsuitable for the generation of abnormal samples. Consequently, when an abnormal image is forwarded into the network, G_D is unable to reconstruct the anomalies, even though G_E has managed to map the input X to the latent vector z. These missed anomalies in the outputted sample X' lead to a vector z' that has also missed the abnormal features representation, inflicting a dissimilarity between z and z'. When such dissimilarity occurs, the model classifies X as an anomalous image [3].

Although GANs have shown successful results in the field of anomaly detection, most of the proposed architectures are solely focused on the detection of anomalies in images.

A particular work related to the content of this dissertation is the *"Flow-based Network Traffic Generation using Generative Adversarial Networks"*, proposed in 2018, by Ring et al. [79]. The problem behind this study is the current lack of available datasets of network flow that contain realistic user behaviour and updated attack scenarios. As solution for this problem, it was proposed the implementation of three synthetic flow-based network traffic generators (using GANs) to generate realistic flow-based network traffic. The biggest challenge faced in this study was the flow-based network data structure that is represented with continuous and categorical attributes, and GANs can only process continuous input attributes. In spite of all the challenges, this study concludes that GANs are well suited for generating flow-based network traffic.

3 The Problem and Its Challenges

The exponential computer networks evolution is leading to the daily growth in the number of security issues and the complexity of the latest network attacks [63, 87]. Researchers of intrusion detection systems aim to build new security approaches to keep up with the network's constant evolution. Efficient adaptive methods, such as deep learning techniques, can improve the response to dynamic and sophisticated attacks. However, the limited number of publicly available network datasets and their poor traffic variety and attack diversity are a major stumbling block in the research community as there are not enough data for an accurate models evaluation, comparison, and deployment [88].

Imbalanced classification is an emerging problem arising from a multi-class skewed distribution [1]. In many real-life situations, the amount of data produced in a multi-class event is not in the same proportion to all the classes [49, 91]. Consequently, most classification datasets do not have a uniform number of instances for each class, reducing the performance of machine learning models [68]. In this case of study, the benign network flow is much more frequent than the attack/anomalies network flow, resulting in a biased data distribution, with one of the classes containing a much smaller number of instances than the other one. As a consequence, the correct classification of the minority class is a main problem [1].

3.1 Proposed Approach - Solution

To address this problem, it is considered the use of a data level approach, oversampling. Data level approaches are pre-processing tasks, which are applied to balance the skewed distributions directly [1]. In oversampling techniques, the pre-processing tasks focus on generating artificial data for the minority classes until it is reached a desired level of balance. In this project, it is proposed the use of Generative Adversarial Network as an oversampling technique. The choice of this model lies in its ability to learn the properties of the collected data and to generate new samples with the same underlying characteristics.

The proposed approach intends to address the sparse availability of data by building a generative model responsible for producing large sets of network flow data while improving the detection of anomalies and disguised attacks. The first step will consist in building a generator for the minority classes. Resorting to a GAN both generator and discriminator will be trained to simulate traffic that

mimic an attack (minority class). Later, the discriminator will be discarded and only the generator will be used in the second step. The second step will consist in training a classification model to detect the traffic anomalies. This model will be trained with the publicly available data and the data produced by the generator in step one.

3.2 Challenges

The proposed approach will face multiple challenges, being the major ones related with the data generation.

- Generate Discrete Data Sequences

  Generative Adversarial Networks have achieved significant popularity in the field of real-valued data generation. However, using GANs to generate discrete data sequences faces some challenges. In GANs, the generator begins with random sampling and then a deterministic transformation that slightly changes the generator to make the data more realistic [98]. But, when dealing with discrete data, this "slight change" in the generator network may not have a corresponding value in the limited dictionary space [98].

- GANs Training

  The Nash equilibrium between the two non-cooperative networks proves to be very hard to achieve [83]. Non-convergence, mode collapse, and diminished gradients are some of the most frequent problems during the GANs training. Most of the recent work on this subject [41, 59, 72, 83] focuses on improving training stability. However, the GANs training phase remains an open problem.

- Generated Data Evaluation

  One of the main problems of the GAN numeric data generation is its evaluation. Even though several measures have been implemented, there is still no consensus as to which measure best captures the model strengths and weaknesses [11].

4 Development

The current chapter presents the system architecture and explains the details of its development. The global goal of this system is to improve network anomaly detection by solving the imbalanced classification problem with the use of a Generative Adversarial Network. The whole system is divided into five distinct components that can be used independently for specific tasks. This project was developed using the programming language *Python3*. The choice of this language lies in the available libraries and frameworks, the mature and supportive *Python* community, and the ease of use. The developed project can be found in the following *GitHub* repository.

4.1 System Architecture

The proposed system architecture reflects a pipeline that transforms an imbalanced dataset into a balanced one with the goal of improving the classification model's performance. Even with the focus on network anomaly detection, this system has been designed to work with every type of imbalanced dataset. An explicit and objective picture of the overall system is provided in figure 4.1.

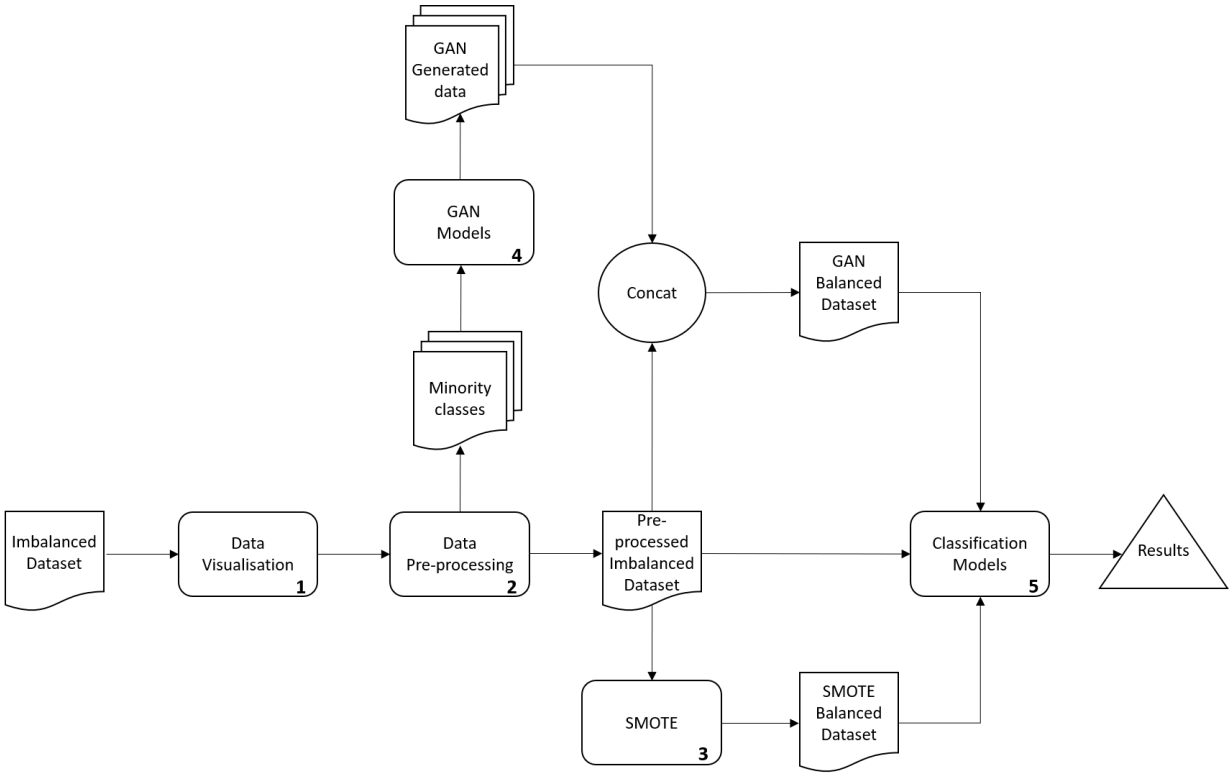


Figure 4.1: Simple representation of the overall system.

The data visualisation (1) and the data pre-processing (2) are an integral part of any machine learning project. As such, these are the first two stages of this pipeline. After these stages, the data should be in an appropriate format for the GAN and classification models, although, still imbalanced.

In order to deal with the imbalanced classification problem, it is proposed the use of a Generative Adversarial Network to oversample the minority classes. As a baseline for this experiment, it will be used a traditional data augmentation technique, the Synthetic Minority Oversampling TEchnique (SMOTE). Thus, in the stage 3, the pre-processed training set will be balanced through SMOTE.

During the stage 4, the GAN models will receive as input the pre-processed minority classes. Each GAN will be trained with just one class at a time. After the GAN training, the generator will provide the amount of data required to balance the training set. The baseline for the GAN generated data will be the Euclidean distance between the SMOTE generated data and the real data. Regarding to the detection of the attacks/anomalies, the classification models (5) will receive as input three different types of data: the imbalanced pre-processed dataset, the SMOTE balanced dataset, and the GAN balanced dataset. The main goal is to compare the classification results of each dataset and see how beneficial is the use of a Generative Adversarial Network in deep learning network anomalies detection.

Figure 4.2 provides a quick overview of what will be performed at each stage of the process. In the following subsections, all five stages will be explained in detail.

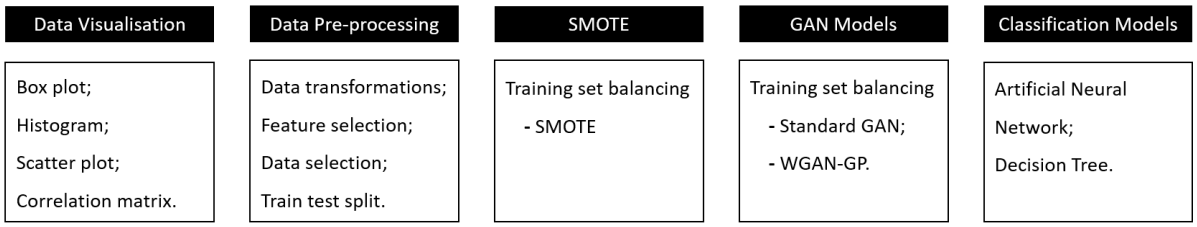


Figure 4.2: Brief summary of the system stages.

4.1.1 Data Visualisation

To better understand the behaviour of the data and the range of values that they present, it is essential to build exploratory graphical models that promote its perception. Thus, for each of the variables present in the dataset, the following graphics will be created:

- *Diagram of extremes and quartiles*

  This graphic model displays the data distribution based on a five number summary: *minimum*, *first quartile (Q1)*, *median*, *third quartile (Q3)*, and *maximum*. It is useful to identify the

variation associated to the value of each variable and to provide information of interest such as the median and outliers.

- *Histogram*

  A histogram shows the frequency of the occurrences, enabling the visualisation of the amount of data organised by bins within certain ranges and how it is distributed. This chart type displays indicators as mean and mode.

- *Scatter diagram*

  The scatter diagram will be used to perceive the relationship between an independent variable and a response variable. This graphic model is useful to identify how much one variable is affected/correlated with another.

- *Correlation matrix*

  In order to understand the relationship between all the variables and ensure a more global analysis of the data, a correlation matrix graphic will also be used. A correlation matrix is a symmetric table showing correlation coefficients between variables, coloured according to the value. The diagonal of the table is always a set of ones. Each variable (Xi) in the table is correlated with each of the other variables (Xj). This graphic allows to detect which pairs of variables have the highest correlation.

4.1.2 Data Pre-processing

The data pre-processing is one of the most important and time consuming stages. In this stage, the dataset must be transformed into a more understandable, meaningful, and useful format. All the predictors must be selected and the data problems must be solved. In this context, different data transformations will be performed:

- *Incorrect and missing data filtering*

  Incorrect and missing data are a prevailing problem in most of the real-world datasets. The reasons why they occur are numerous - from human errors to software bugs. There is no perfect way to deal with incorrect and missing values. The most common and successful methods used for this purpose are: delete the rows or columns with these values, replace them with a central value (mean, median, mode), or assign a unique discrete variable.

- *Logarithmic transformation*

  When the continuous data distribution is skewed, data transformations are implemented to make the data as "normal" as possible and therefore to improve the statistical power of the associated statistical analyses. The log transformation is used to transform skewed data to approximately conform to normality. If the original data generally follows a log-normal distribution, then the log-transformed data follows a near normal distribution.

- *Label encoding*

  Due to the network flow properties, most of the network flow datasets contain categorical values. However, the majority of machine learning classification models achieve better results with numerical inputs. As such, the label encoding is used to convert the categorical data into numerical data. This method transforms each value in a column into a number.

- *Normalisation*

  Data normalisation is performed to improve the models optimisation. If the data are in different scales, the time needed to find the optimal points for the optimisation function is much longer. However, if the dataset includes outliers in its data, the data normalisation will reduce the natural data to a very small range. In this study, it will be used the $[0, 1]$ interval for normalisation.

- *Remove duplicated rows*

  The dataset can often contain duplicate rows. Including them usually bias the fitted model. Consequently, the duplicated data should be removed from the dataset.

Regarding to the features selection, this step will take into account the observations of the visualisation stage. All the variables less than 70% correlated with each others will be considered predictors. As for the variables more than 70% correlated, it will be removed one of the variables of each pair, to avoid the problem of multicollinearity. As result, the maximum number of features will be guaranteed for data generation.

With respect to the classification models train-test split, as the dataset is imbalanced it is necessary to ensure that both train and test sets hold all the response variable classes. In this way, for each of the classes will be performed a 67-33% train-test split.

After these transformations and selection, it is essential to proceed with the extraction of data for the GAN models. Since these models will only receive and generate one minority class at a time, the training data must be separated by the classes of the response variable. In this way, the GAN models

will be able to receive different sets of data, each one representative of a different class. Once all these steps have been completed, the data will be in an appropriate format for the next stages.

4.1.3 SMOTE

Synthetic Minority Oversampling Technique is an oversampling technique used to re-balance the pre-processed training set. Rather than simply replicating the minority class instances, SMOTE's main concept is to introduce new synthetic examples [25].

These new synthetic data is generated through the interpolation of several instances of minority class that are within a defined neighbourhood. SMOTE loops through a current instance of a particular minority. One of the K closest minority class neighbours is selected at each loop iteration and a new minority instance is synthesised between the minority instance and that neighbour. Neighbours from the K nearest neighbours are selected randomly, depending on the amount of oversampling needed [13, 25].

4.1.4 Generative Adversarial Network

The main goal of this stage is to balance the pre-processed training set through the generation of data for the minority classes. In order to do so, the GAN models will receive the class-separated training data as input and will independently train the models for each class.

After the research of improving GAN models, it was decided to implement two different GANs: an Improved Wasserstein GAN (WGAN-GP) [36] and a standard GAN [34]. These GANs will have the following composition:

- *Generator*

  In both models, the generator network starts with a 70 neurons input layer that receives the noise vector z. Then, depending on the model, each generator has 1 to 3 hidden layers with 64 neurons and `ReLU` as activation function. Since the output of the generator has to match the shape of the training data, the output layer has as many neurons as the number of features of real data. The activation function used in the output layer is the `Sigmoid` once the real data values go from 0 to 1 (normalisation). In figure 4.3 can be seen the generator networks architecture.

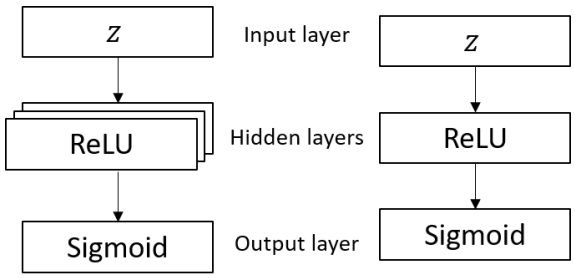


Figure 4.3: Generator networks architecture: (left) WGAN-GP, (right) GAN.

- *Discriminator*

  The discriminator network takes as input a candidate sample x (real or generated) and outputs a single value that represents how real the input images are (WGAN-GP), or the likelihood of being real (GAN). In this way, the discriminator input layer has the same number of neurons as the number of features of the original data and the discriminator output layer has only one neuron. Regarding to the hidden layers, the discriminator is composed of one or two hidden layers (depending on the model), each one with 64 neurons. In the WGAN-GP model is used the `Leaky ReLu` activation function for the hidden and output layers, while in the GAN model is used the `Leaky ReLu` function in the hidden layer and the `Sigmoid` function in the output layer. Figure 4.4 shows the discriminator networks architecture.

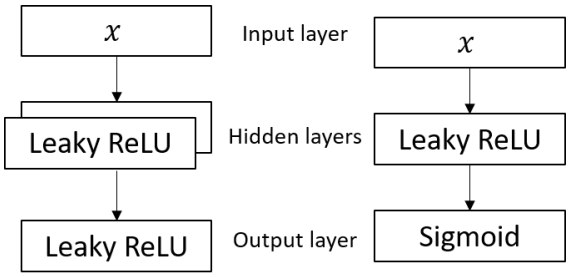


Figure 4.4: Discriminator network architecture: (left) WGAN-GP, (right) GAN.

- *Training losses*

  The biggest difference between the GAN and WGAN-GP models is the training losses definition. The loss functions reflect the distance between the real and generated data distributions. In this study, each model has two loss functions: one for the generator training and other for the discriminator training. Each loss function definition is represented below.

- GAN

  Generator loss: $\frac{1}{m}\sum_{i=0}^{m} -ln(D(G(z_i)))$

  Discriminator loss: $\frac{1}{m}[\sum_{i=0}^{m} -ln(D(x_i)) + \sum_{i=0}^{m} -ln(1 - D(G(z_i)))]$

- WGAN-GP

  Generator loss: $\frac{1}{m}\sum_{i=0}^{m} -D(G(z_i))$

  Discriminator loss:

  $[\frac{1}{m}\sum_{i=0}^{m} D(G(z_i)) - \frac{1}{m}\sum_{i=0}^{m} D(x_i)] + \lambda[\frac{1}{m}(\sqrt{\sum_{i=0}^{m} |\nabla_{\hat{x}} D(\hat{x}_i)|^2} - 1)^2],$

  where $\hat{x} = \epsilon x + (1 - \epsilon)G(z)$ with ϵ selected randomly between 0 and 1.

- *Optimisation*

  After the analysis of different GAN studies [15, 32, 36, 74], it was decided to use the optimiser algorithm `Adam` [46] to minimise the loss functions. The `Adam` recommended settings of the WGAN-GP authors [36] were used for the WGAN-GP, while the `Adam` default hyperparameters were used for the standard GAN. Thus, the `Adam` hyperparameters α, β_1, β_2 used in each model are the following:

  - GAN $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$;
  - WGAN-GP $\alpha = 0.0005$, $\beta_1 = 0$, $\beta_2 = 0.9$.

For the correct balance of the training set, it will also be required to indicate the number of samples that need to be generated, for each of the classes. The original training set will be considered balanced when all the response variable classes have the same amount of data as the class with the higher number of samples. A small example of the amount of data that should be required to be generated is provided in table 4.1.

Table 4.1: Amount of data required to be generated.

Response variable classes	Nº samples original dataset	Nº samples to generate
Class 1	100	900
Class 2	50	950
Class 3	1000	0

Once the data is generated, it must be analysed. As explained earlier, generating categorical data through GANs proves to be difficult. In this way, the categorical generated data must be processed in

order to become categorical. With regards to the evaluation of the generated data quality, in this study, the GAN generated data will be compared with the SMOTE generated data. It will be calculated the Euclidean distance between the SMOTE generated data and the real data, as well as the Euclidean distance between the GAN generated data and the real data. If the distances are similar, the GAN generated data will be approved.

4.1.5 Classification Models

In machine learning, there are several different types of models, all with different types of outcomes. Where statistical classification is concerned, the model attempts to recognise two or more defined groups, and have them classified accordingly. In this study, two classification models will be used:

- *Artificial Neural Network*

  The Artificial Neural Network, introduced in 2.2.1, is a deep learning model composed of a group of neurons layer-organised, which convert an input vector into an output. This model offers a set of properties (real-time adaptability, robustness with respect to damages and to missing data, and automatic generalisation [47]) that overcome some of the most traditional models [56]. However, the ANN main disadvantage is in its "black box" nature. Every ANN output is obtained through a complex simultaneous execution of a large number of neurons, which make the model decisions very hard to understand and explain [47].

  During the implementation of the Artificial Neural Network, many architectures were tested. The first one was a simple network composed of only one input and one output layer. Then, the consequent models were produced by adding some hidden layers and increasing the number of nodes of each layer.

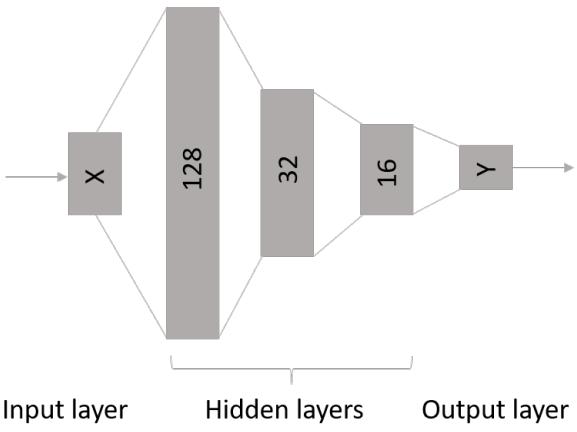


Figure 4.5: Artificial Neural Network architecture.

The final model was obtained when a balance between the number of hidden layers, nodes, batch size, and epochs was reached. Therefore, the Artificial Neural Network used in this project will be composed of one input layer, three hidden layers (with 128, 32, and 16, neurons per layer, respectively), and one output layer. The model will be trained during 30 epochs with a batch size of 1000. Figure 4.5 represents the ANN architecture.

- *Decision Tree (DT)*

  Compared with the Artificial Neural Networks, Decision Trees have the advantage of being expressed as rules, which makes this model very intuitive and easy to understand [23, 48]. In order to find solutions a Decision Tree makes sequential, hierarchical decisions about the outcome variable based on the predictor data. Once the model has been set up, it acts as a protocol in a series of "if this happens then this happens" conditions.

These models will receive four different inputs: the imbalanced, the SMOTE balanced, the GAN balanced, and the WGAN-GP balanced training sets. In order to compare equally the results obtained from the Decision Tree and Artificial Neural Network classification models, the same test set will be used for the different training sets. Note that the test set consists solely on original data, with no data augmentation.

4.2 Model Evaluation Metrics

The evaluation stage is one of the most important stages during the construction of the model as it indicates how accurately the model works. To evaluate the model efficiency, model evaluation metrics are needed. This section presents and explains all four measures that will be used to evaluate the classification models built in this project.

Note: All the variables present in the equations of this section came from the confusion matrix.

- *Accuracy*

  Accuracy is the most intuitive metric. It represents the ratio of the correctly labelled data out of the total number of input samples. Accuracy answers the following question: *How much data was correctly labelled by the model out of all data?* The accuracy formula is represented in the equation 18.

$$Accuracy = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{True Negatives} + \text{False Negatives}} \tag{18}$$

Regardless of the fact that accuracy is the most intuitive and applied metric, this measure only does a proper evaluation when there is an equal amount of samples belonging to each class [43]. For example, if in some training set there are 97% of samples of class X and 3% of samples of class Y, then, the model used in that training set can easily achieve 97% of training accuracy by only predicting every training sample as class X. Still, if the same model is tested on a test set of 70% of class X samples and 30% of class Y samples, the test accuracy would be reduced to 70%. Accuracy is an outstanding metric, but sometimes it gives the false sense of reaching high model performance.

- *Precision*

  Precision is the percentage of positive instances correctly predicted out of the total predicted as positive. In the context of this study, precision answers the following question: *How many of those that have been labelled as attack are actually attacks?* The precision formula is represented in the equation 19.

$$Precision = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \tag{19}$$

- *Recall*

  Recall is the ratio of positive instances correctly predicted by the model to all how are positives in reality. In the context of this study, recall answers the following question: *Of all the existing attack data, how many of those were correctly predicted?* The recall formula is represented in the equation 20.

$$Recall = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}. \tag{20}$$

- *F1-score*

  F1-score is the harmonic mean of precision and recall metrics. This measure should be chosen over accuracy in imbalanced classification problems [42]. The f1-score formula is represented in the equation 21.

$$F1\text{-}score = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision + Recall}} \tag{21}$$

4.3 Frameworks and Tools

Given the popularity of machine learning and, particularly, its deep learning sub-field in recent years, new frameworks and tools have been developed to support and facilitate the implementation of new models. The software available for these fields is designed to provide a higher level of abstraction and performance, along with the simplification of complex programming challenges. Most of the released frameworks, tools, and libraries are open-source, coming from academies, companies, or open-source communities [66]. Some of the most relevant tools that will be used in this dissertation are the following:

- *Jupyter Notebook*

  The *Jupyter Notebook*[1] is an open-source, browser-based tool that allows the user to create and share documents that contain live code, equations, visualisations, and narrative text [76]. The combination of the *Python* code execution with the rich text-editing functionalities allows the user to describe the research process in a simple and efficient way. A notebook also allows long experiments to be divided into smaller parts that can be independently executed [17].

- *TensorFlow*

  TensorFlow[2] is an end-to-end open-source machine learning platform, created and maintained by the Google Brain team [66, 93]. It is based on a computational graph where the nodes represent mathematical operations and the edges represent multidimensional data arrays (called tensors) that flow through the different operations [66, 70]. The *TensorFlow* can run on a wide range of systems, from mobile devices to large-scale distributed systems [66]. Given its flexible and scalable ecosystem, *TensorFlow* enables the development and training of state-of-the-art models without compromising speed or performance [93]. *TensorFlow* has available multiple-language APIs, with the *Python* API currently being the most complete and easiest to use [93].

- *Keras*

  Keras[3] is a *Python*-written open-source high level neural network library that provides simple and intuitive tools to build and train distinct deep learning models [17, 70]. It runs on top of

1 *Jupyter Notebook*
2 *TensorFlow*
3 *Keras*

other DL tools, including *TensorFlow*, *Theano*, and *CNTK*, abstracting their capabilities and hiding their complexity [66]. *Keras* was originally developed for researchers with the aim of allowing fast experimentation [17, 66]. To accomplish this purpose, *Keras* offers consistent and simple APIs, minimises the number of user actions required for common use cases, provides straightforward and actionable error messages, and also has an extensive and detailed documentation. Due to its properties, *Keras* has become the deep learning solution of choice for many university courses, researchers, and companies [45].

Besides these, also *Pandas*, *Numpy*, *Matplotlib*, *Seaborn*, and *Scikit-learn* libraries will be used. The first two libraries (*Pandas* and *Numpy*) will be used to manipulate the dataset. The *Matplotlib* and *Seaborn* libraries will be used for the construction of graphic models. Finally, the *Scikit-learn* library will be used for the Decision Tree classification model.

5 Results

The developed system aims to improve the detection of network anomalies through the use of a generative adversarial model. To test the system it is necessary to find a suitable network flow dataset. In this chapter, firstly it will be presented and described the chosen dataset and, finally the obtained results using the developed system.

5.1 Dataset

Finding a suitable dataset for this project was challenging. Due to privacy concerns, many datasets cannot be shared and those that become available are heavily anonymized and do not reflect current trends [88]. In addition, due to the absence of proper documentation and data labelling [63], most datasets cannot be understood. If the labels are missing or misleading, the dataset can not be used correctly and the results of the analysis may not be valid or reliable [28].

After evaluating the available network datasets, the Canadian Institute for Cybersecurity Intrusion Detection System 2017 (CICIDS2017) dataset [87] was considered the most suitable for this project since it fulfils all the eleven characteristics of a true intrusion detection dataset [69]: Attack Diversity, Anonymity, Available Protocols, Complete Capture, Complete Interaction, Complete Network Configuration, Complete Traffic, Feature Set, Heterogeneity, Labelling, and Metadata [28].

This generated dataset provides five days of realistic background traffic with network events produced by the abstract behaviour of 25 users. The testbed architecture covers all common and necessary equipments, (router, firewall, switch, and modem), operating systems (Windows, Ubuntu and Macintosh), protocols (HTTP, HTTPS, FTP, SSH and email protocols), and attacks (Brute Force, Heartbleed, Botnet, DoS, DDoS, Web-based and Infiltration) [87].

Although the CICIDS2017 has been considered one of the best IDS datasets, it still presents several limitations. The Panigrahi and Borah detailed analysis of the CICIDS2017 [69] revealed problems such as high data dimensionality, missing values, and high class imbalance. The study shows that the whole dataset contains 3119345 instances, 84 strongly correlated features, and 288602 missing class label, and 203 missing information instances.

5.1.1 CICIDS2017 Description

The CICIDS2017, previously presented, is a generated network traffic dataset composed of 84 features, including the target feature (Label), and 15 different class labels. This dataset was generated through the CICFlowMeter software [64] that is able to generate bidirectional flows and extract more than 80 network traffic features. All the features name can be found in table 5.1 (see Appendix 6.2 for a detailed description of the features).

The whole dataset is divided into seven smaller datasets, grouped by attack categories: `DDoS`, `Port Scan`, `Botnet`, `Infiltration`, `Web Attack`, `Brute Force`, and `DoS`. Each dataset exhibits a selected group of class labels. Of the 15 different class labels, 1 corresponds to the normal flow (`Benign`), while the other 14 correspond to the following attacks:

- `DDoS`: The attacker floods the victim's bandwidth or resources with a large network traffic generated from multiple systems;
- `Port Scan`: The attacker sends packets to a variety of destination ports of the victim's machine, in order to find an active port and exploit a known vulnerability;
- `Infiltration`: The attacker uses infiltration tactics, such as infected files, to gain full access to the victim's system;
- `Botnet`: A botnet is a set of internet-connected malware-infected devices that allow the attacker to monitor them. This dataset uses the *Python*-based Botnet Ares, which can provide remote shell, file upload/download, screenshots capture, and keylogging [87];
- `Web Attack-Brute Force`: The attacker uses the trial-and-error technique to obtain privilege information, such as the administrator's password and the Personal Identification Number (PIN);
- `Web Attack-XSS`: Cross-site Scripting (XSS) is a client-side code injection attack. The attacker can use the injected scripts to change the website's content;
- `Web Attack-SQL Injection`: The attacker injects a string of malicious SQL commands in data-driven applications to access or modify unauthorised information;
- `FTP-Patator`: The attacker uses the Patator tool to brute force an FTP login;
- `SSH-Patator`: The attacker uses the Patator tool to brute force an SSH login;

Table 5.1: CICIDS2017 features.

No.	Feature	No.	Feature	No.	Feature
1	Flow ID	29	Fwd IAT Std	57	ECE Flag Count
2	Source IP	30	Fwd IAT Max	58	Down/Up Ratio
3	Source Port	31	Fwd IAT Min	59	Average Packet Size
4	Destination IP	32	Bwd IAT Total	60	Avg Fwd Segment Size
5	Destination Port	33	Bwd IAT Mean	61	Avg Bwd Segment Size
6	Protocol	34	Bwd IAT Std	62	Fwd Avg Bytes/Bulk
7	Time stamp	35	Bwd IAT Max	63	Fwd Avg Packets/Bulk
8	Flow Duration	36	Bwd IAT Min	64	Fwd Avg Bulk Rate
9	Total Fwd Packets	37	Fwd PSH Flags	65	Bwd Avg Bytes/Bulk
10	Total Backward Packets	38	Bwd PSH Flags	66	Bwd Avg Packets/Bulk
11	Total Length of Fwd Pck	39	Fwd URG Flags	67	Bwd Avg Bulk Rate
12	Total Length of Bwd Pck	40	Bwd URG Flags	68	Subflow Fwd Packets
13	Fwd Packet Length Max	41	Fwd Header Length	69	Subflow Fwd Bytes
14	Fwd Packet Length Min	42	Bwd Header Length	70	Subflow Bwd Packets
15	Fwd Pck Length Mean	43	Fwd Packets/s	71	Subflow Bwd Bytes
16	Fwd Packet Length Std	44	Bwd Packets/s	72	Init_Win_bytes_fwd
17	Bwd Packet Length Max	45	Min Packet Length	73	Act_data_pkt_fwd
18	Bwd Packet Length Min	46	Max Packet Length	74	Min_seg_size_fwd
19	Bwd Packet Length Mean	47	Packet Length Mean	75	Active Mean
20	Bwd Packet Length Std	48	Packet Length Std	76	Active Std
21	Flow Bytes/s	49	Packet Len. Variance	77	Active Max
22	Flow Packets/s	50	FIN Flag Count	78	Active Min
23	Flow IAT Mean	51	SYN Flag Count	79	Idle Mean
24	Flow IAT Std	52	RST Flag Count	80	Idle Packet
25	Flow IAT Max	53	PSH Flag Count	81	Idle Std
26	Flow IAT Min	54	ACK Flag Count	82	Idle Max
27	Fwd IAT Total	55	URG Flag Count	83	Idle Min
28	Fwd IAT Mean	56	CWE Flag Count	84	Label

- DoS Slow Loris: The attacker uses the Slowloris tool to perform a denial of service attack. This attack works by opening simultaneous HTTP connections to a targeted Web server and keeping those connections open as long as possible;
- DoS Slow HTTP Test: The attacker uses the SlowHTTPTest tool to perform a denial of service attack; This tool simulates some application layer denial of service attacks by extending HTTP connections in different ways;
- DoS Hulk: The attacker uses the HULK denial of service tool, which generates multiple dynamic requests designed to difficult the server defences from detecting the attack;
- DoS GoldenEye: The attacker uses the GoldenEye HTTP denial of service test tool to perform a denial of service attack. The GoldenEye generates a high rate of random HTTP traffic to flood the HTTP Web server;
- Heartbleed: The attacker exploits the OpenSSL[1] vulnerability to gain access to sensitive information as usernames and passwords.

5.2 Data Visualisation

Due to the large set of variables that each dataset holds, in this section will only be presented and explained some of the DDoS dataset graphic models. However, a brief overview of the conclusions drawn during the data visualisation phase will be provided at the end of the section.

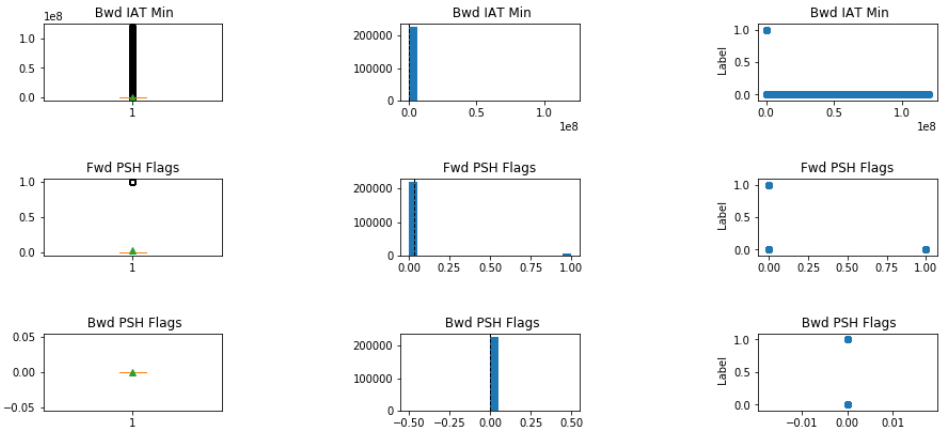


Figure 5.1: Box plot, histogram and scatter plot models of DDoS dataset. Value 0 of Label means BENIGN; Value 1 of Label means DDoS.

1 OpenSSL Cryptography and SSL/TLS Toolkit

In order to properly build the graphic models (box plot, histogram, scatter plot, and correlation matrix), it was necessary to convert the textual categorical features (e.g. `Label`) into numerical categorical features. In addition, rows with a lack of information were excluded. Some of the DDoS dataset graphics models can be seen in the figure 5.1.

Through the observation of these graphics, it was possible to detect a large disparity in the range of features values, and a presence of single unique value features (*Bwd PSH Flags*, *Fwd URG Flags*, *Bwd URG Flags*, *CWE Flag Count*, *Fwd Avg Bytes/Bulk*, *Fwd Avg Packets/Bulk*, *Fwd Avg Bulk Rate*, *Bwd Avg Bytes/Bulk*, *Bwd Avg Packets/Bulk*, and *Bwd Avg Bulk Rate*).

Figure 5.2: Correlation Matrix of DDoS dataset.

As concerns to the correlation matrix, this graphic was built without the single unique value features mentioned above, as they do not correlate with any variable. Figure 5.2, shows that many variables

are positively correlated (dark blue) and some are negatively correlated (light blue). A positive correlation indicates that the two variables move together, and the relationship gets stronger the closer the correlation gets to one. On the other hand, a negative correlation indicates that the two variables move in opposite directions, and the relationship also gets stronger the closer to -1 the correlation gets.

5.2.1 Summary

After analysing all the obtained graphics, a deeper understanding of the behaviour of the data was reached. These graphics revealed that some features exhibit a single unique value. A feature with only one unique value cannot be useful for machine learning as it has zero variance. For example, a tree-based model can never make a split on a feature with only one value since there are no groups to divide the observations into. Regarding to the distribution of the data, it should be remembered that some datasets have a very small amount of attack data when compared to the non-attack data. The following figures show the `Label` distribution in each dataset.

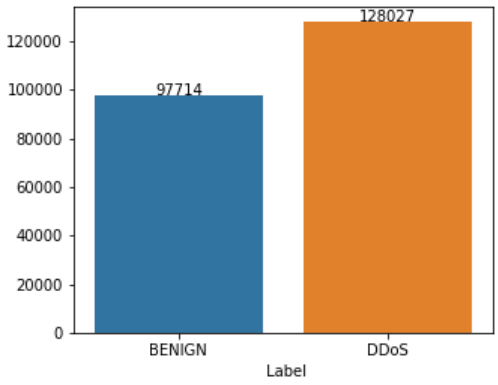


Figure 5.3: Label distribution on DDoS dataset.

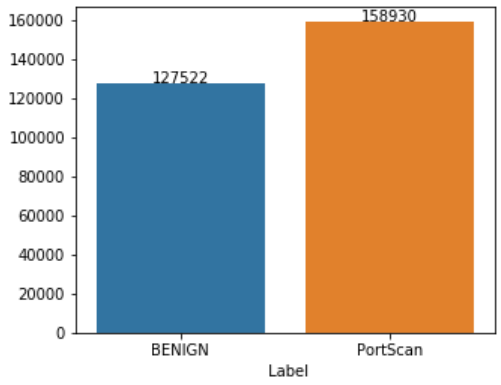


Figure 5.4: Label distribution on `Port Scan` dataset.

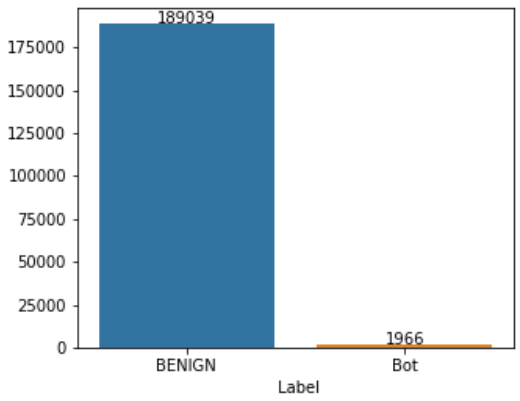


Figure 5.5: Label distribution on `Botnet` dataset.

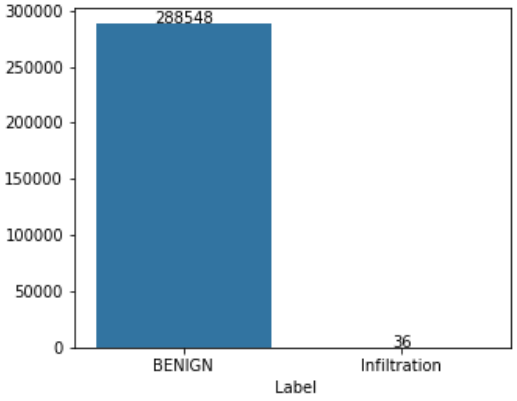


Figure 5.6: Label distribution `Infiltration` dataset.

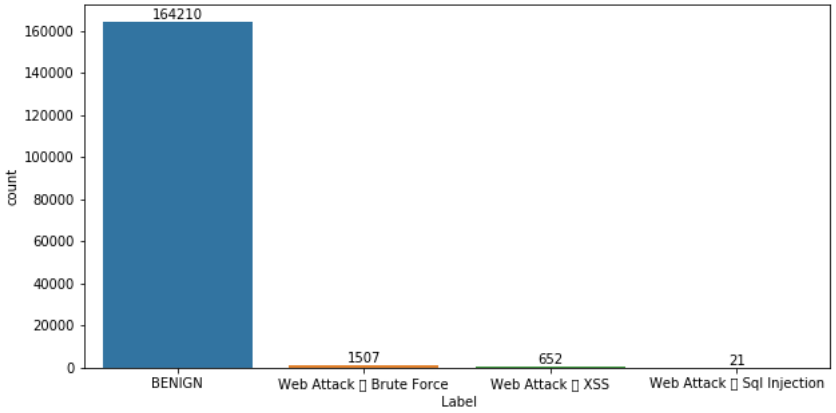


Figure 5.7: Label distribution on `Web Attack` dataset.

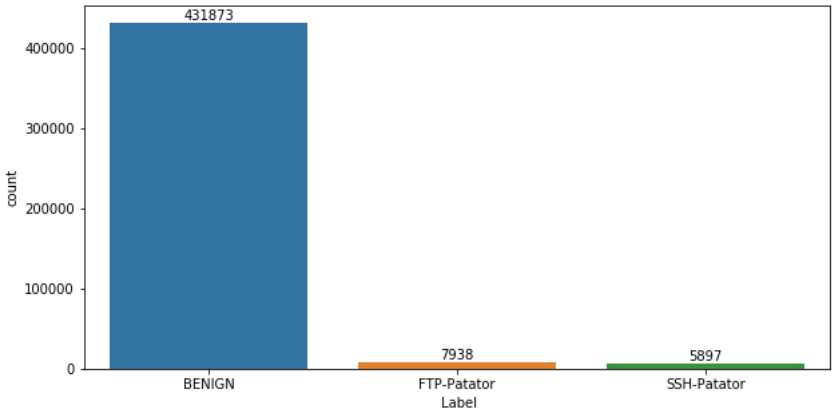


Figure 5.8: Label distribution on `Brute Force` dataset.

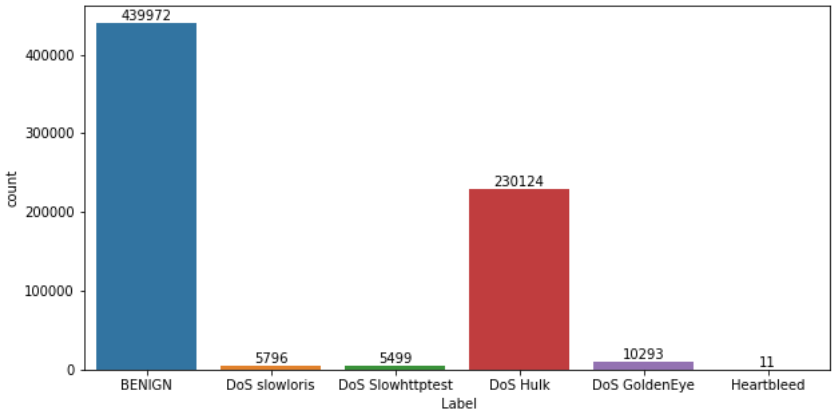


Figure 5.9: Label distribution on DoS dataset.

Of the datasets under study, only the DDoS and Port Scan datasets (figures 5.3 and 5.4) have a reasonable distribution of benign and attack data. In all the other datasets (figures 5.5 to 5.9), the amount of attack data, when compared with the amount of benign data, represents a very small fraction of the dataset. After analysing these graphics, it was decided to exclude the Infiltration dataset and delete the Heartbleed and Web Attack - SQL Injection classes from the DoS and Web Attack datasets, respectively. This decision was taken on the basis of the limited amount of data represented by these three classes.

In addition to the poor data distribution, it was also detected that several features had negative values, which were not expected. The number of negative values in each feature for each of the datasets are represented in table 5.2.

Table 5.2: Number of negative values in each feature per dataset.

	Datasets						
Features	**DDoS**	**Port Scan**	**Botnet**	**Infiltration**	**Web Attacks**	**Brute Force**	**DoS**
Flow Duration	2	36	9	4	11	17	21
Flow Bytes/s	2	36	8	4	10	0	16
Flow Packets/s	2	36	9	4	11	17	21
Flow IAT Mean	2	36	9	4	11	17	21
Flow IAT Max	2	36	9	4	11	17	21
Flow IAT Min	108	176	225	353	236	504	749
Fwd IAT Min	6	0	0	0	0	0	11
Fwd Header Length	0	0	0	0	0	11	1
Bwd Header Length	0	0	0	0	0	5	0
Fwd Header Length.1	0	0	0	0	0	11	1
Init_win_bytes_forward	32925	60210	95834	102433	79108	200128	203253
Init_win_bytes_backward	88296	76936	118403	164338	99177	256488	334494
min_seg_size_forward	0	0	0	0	0	11	1

As seen in the table above, the two variables with the highest number of negative values in all datasets are the variables *Init_win_bytes_forward* and *Init_win_bytes_backward*. Since they have such a high number of negative entries, during the pre-processing stage, these two features should be excluded from all datasets.

5.3 Data Pre-processing

After visualising the data in its raw form, there was the perception that the data should be changed to be able to create an optimal model. As such, firstly it was performed the data selection. In this

operation the most suitable predictors for each dataset were selected. Then, to achieve a better model performance, multiple data transformations were applied.

5.3.1 Data Selection

- *Predictors*

  Due to the different attack scenarios that each dataset holds, the predictors chosen for one dataset could not fit well for the others. As such, a group of features was independently selected for each dataset. Table 5.3 present the different groups of predictors.

Table 5.3: Predictors per dataset.

	Datasets					
Predictors	**DDoS**	**Port Scan**	**Botnet**	**Web Attacks**	**Brute Force**	**DoS**
Source Port	X	X	X	X	X	X
Protocol	X	X	X	X	X	X
Total Fwd Packets		X				
Total Length of Fwd Packets	X	X	X	X	X	X
Fwd Packet Length Max					X	X
Fwd Packet Length Min	X		X	X	X	X
Flow Bytes/s	X	X	X	X	X	X
Flow Packets/s	X	X	X	X	X	X
Flow IAT Mean	X			X		X
Fwd IAT Max		X	X	X	X	X
Fwd IAT Min	X	X	X	X	X	X
Bwd IAT Total	X	X	X	X	X	X
Fwd PSH Flags	X	X	X	X	X	X
Bwd Packets/s	X	X	X	X	X	X
Fwd Header Length	X		X	X	X	
PSH Flag Count				X	X	X
FIN Flag Count	X	X	X	X	X	X
RST Flag Count	X	X	X	X	X	X
ACK Flag Count	X	X	X	X	X	X
URG Flag Count	X	X	X	X	X	X
Down /up Ratio	X	X	X	X	X	X
Avg Bwd Segment Size	X	X	X	X	X	
act_ata_kt_wd			X		X	X
min_eg_ize_orward	X	X	X	X	X	X
Active Mean	X	X	X	X	X	X
Active Std	X	X	X	X	X	X
Idle Std	X	X	X	X	X	X

- *Response Variable*

  In all the datasets, the response variable is the feature `Label`. However, the values that this feature can assume depends on the dataset. As it can be seen on table 5.4, only the class `Benign` appears in all the datasets, whereas the others, appear in distinct ones.

Table 5.4: *Label* classes per dataset.

	Datasets					
Label Classes	**DDoS**	**Port Scan**	**Botnet**	**Web Attacks**	**Brute Force**	**DoS**
BENIGN	X	X	X	X	X	X
DDoS	X					
PortScan		X				
Botnet			X			
Web Attack Brute Force				X		
Web Attack XSS				X		
FTP-Patator					X	
SSH-Patator					X	
DoS Slowloris						X
DoS Slowhttptest						X
DoS Hulk						X
DoS GoldenEye						X

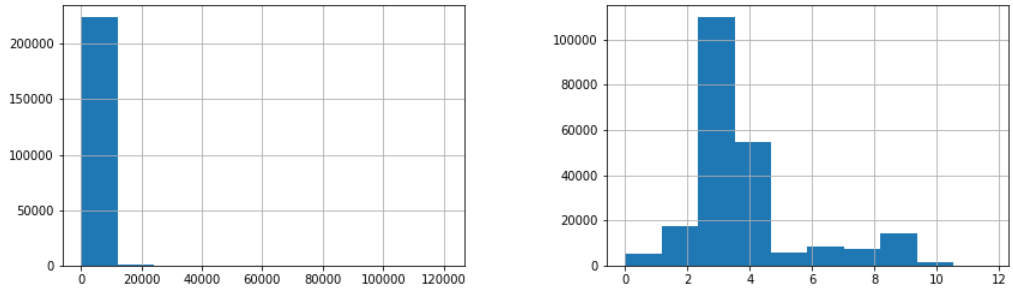
5.3.2 Data Transformation

- *Incorrect and missing data filtering*

  To filter the incorrect and missing data, several strategies could be used. As already introduced in section 5.2.1, the first filtering used was the removal of two columns corresponding to the variables *Init_win_bytes_forward* and *Init_win_bytes_backward*, since they presented large amounts of negative values. To all the other incorrect and missing data, the strategy was slightly different. Instead of removing the columns, rows were removed.

- *Logarithmic transformation*

  In this study, the log transformation was applied to all the features whose value distribution resembles a logarithmic distribution. As can be seen in figure 5.10, the application of log transformation in the feature *Total Length of Fwd Packets* reduced the x axis values from $[0, 120000]$ to $[0, 12]$, leading to a more normal distribution of the data.



(a) Before (b) After

Figure 5.10: Logarithmic transformation in feature *Total Length of Fwd Packets*.

- *Normalisation*

  In this study, the interval used for normalisation, in all predictors, was $[0, 1]$. The figure 5.11 was produced after the application of the normalisation process to the data represented in 5.10b. In this case the x axis values were reduced from $[0, 12]$ to $[0, 1]$.

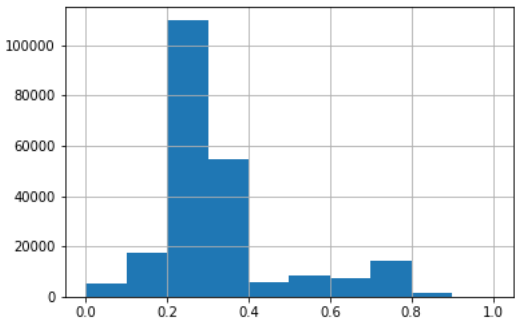


Figure 5.11: Histogram of feature *Total Length of Fwd Packets* after normalisation.

- *Label encoding*

  The label encoding was used in the data visualisation phase to map each type of IP (*Source IP* and *Destination IP*) into a specific class, assigning it a corresponding integer. Label encoding was also used in the response variable *Label*.

- *Duplicated rows*

  After applied the previous transformations, it was detected the existence of duplicated rows. As such, all the duplicated rows were removed.

5.4 Generative Adversarial Network

In this experiment, two distinct GAN models were tested: the standard GAN and the WGAN-GP. Both models revealed challenges during the training phase. The most common problem was the failure to converge. Such type of loss occurs whenever the discriminator and the generator are unable to find a point of balance between the two networks. Usually, this happens when the generator outputs poor quality samples that are easily detected by the discriminator. In cases where this problem prevailed, the noise vector z was increased in an effort to improve the GANs convergence. Figures 5.12 and 5.13 highlight this training issue, while figures 5.14 and 5.15 illustrate a stable training.

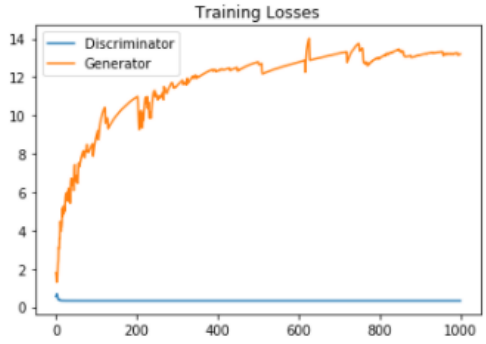


Figure 5.12: GAN - Convergence Failure.

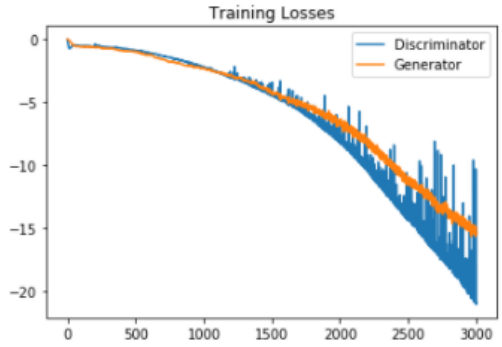


Figure 5.13: WGAN-GP - Convergence Failure.

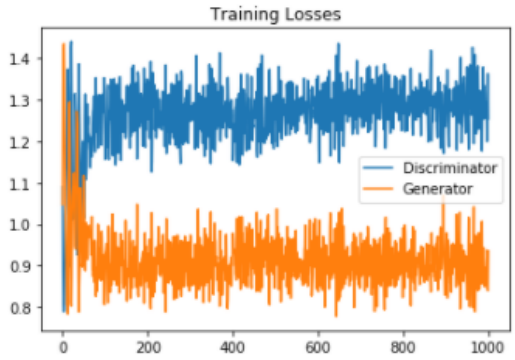


Figure 5.14: Stable GAN.

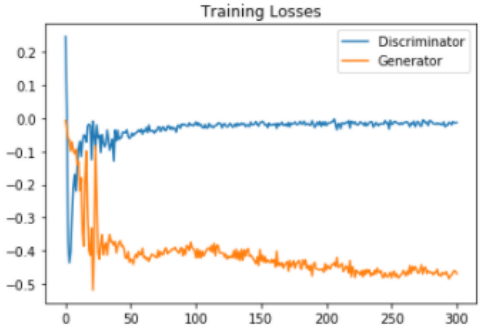


Figure 5.15: Stable WGAN-GP.

5.4.1 Generated Data Pre-processing

After analysing the generated data in its raw form, there was a perception that the values assigned to the binary features were, in most cases, floats close to zero or one. As such, with the goal of improving the quality of the generated data, all the binary features values were replaced by 0 (if generated

value < 0.5), or 1 (if generated value ≥ 0.5). Besides these, also the values of the categorical variable `Protocol` were replaced.

5.4.2 Generated Data Evaluation

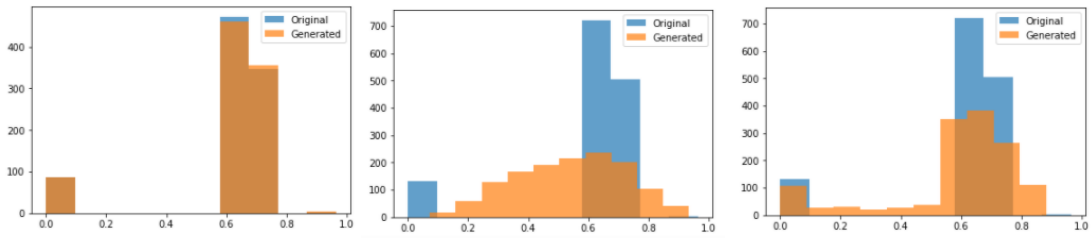
In order to evaluate the distribution of the generated data, the Euclidean distance between the real and generated data was calculated for each feature independently. The baseline for this evaluation method was the Euclidean distance between the SMOTE generated data and the real data. Table 5.5 presents the Euclidean distance results of the `Botnet` generated data.

Table 5.5: Euclidean distances between the original pre-processed `Botnet` class data and the SMOTE (Baseline), GAN, and WGAN-GP generated data, in each attribute.

Features	Baseline	GAN	WGAN-GP
Source Port	0,000710	0.001963	0.002072
Protocol	0,0	0.004466	0.000259
Total Length of Fwd Packets	0.001289	0.005037	0.002436
Fwd Packet Length Min	0,000004	0.001190	0,000022
Flow Bytes/s	0,000710	0.001925	0.001888
Flow Packets/s	0,000588	0.006132	0.001348
Fwd IAT Max	0.000679	0.009847	0.002079
Fwd IAT Min	0.001232	0.006516	0.002881
Bwd IAT Total	0.000724	0.003940	0.002247
Fwd PSH Flags	0,0	0,0	0,0
Fwd Header Length	0,000342	0.007198	0.001112
Bwd Packets/s	0,000602	0.005333	0.001384
FIN Flag Count	0,0	0,0	0,0
RST Flag Count	0,0	0,0	0,0
ACK Flag Count	0,001101	0.008474	0.003957
URG Flag Count	0.001101	0.007636	0.004409
Down/Up Ratio	0,000575	0.001748	0.001100
Avg Bwd Segment Size	0,000037	0.001270	0.000261
act_data_pkt_fwd	0,000012	0.003585	0,000033
min_seg_size_forward	0,000472	0.003030	0.001606
Active Mean	0,000656	0.000620	0.000620
Active Std	0,000620	0.000585	0.000585
Idle Std	0,000584	0.000570	0.000570

Network traffic is prone to variation. The number of packets in each flow, the flow duration, the length of the packets, and almost all of the network traffic features are subject of drift, from flow to flow. The exact replication of the real data distribution is therefore not desirable. Consequently, Euclidean distances with value zero are only expected for binary or categorical features.

Through the observation of table 5.5, it was possible to see that the WGAN-GP generated data presents a lower Euclidean distance than the GAN generated data. Still, both models present values slightly higher than the baseline. This slight increase in the Euclidean distances is due to the fact that the generative models do not generate the exact values present in the real data, but rather a large variety of values close to the real ones. This can be easily perceptive in figure 5.16.



(a) SMOTE generated data. (b) GAN generated data. (c) WGAN-GP generated data.

Figure 5.16: Real data distribution (Original) versus generated data distribution (Generated) in feature *Bwd IAT Total*.

These histograms display the distribution of the generated and real data in the feature *Bwd IAT Total*, for each model. Besides only being presented the graphics for this feature, the same behaviour occurs in all the other features. The attack data generated by SMOTE (figure 5.16a) follow a distribution very similar to the original data. On the other hand, the GAN generated data (figure 5.16b) trace a very different distribution. Lastly, the data generated by the WGAN-GP (figure 5.16c) lists some of the key points of the original distribution, but with some deviations. These graphs support the results obtained in the evaluation of the Euclidean distance. The greater Euclidean distance, the greater the difference between the two data distributions. Revealing that the WGAN-GP model has generated a more realistic data than the standard GAN model.

5.5 Classification Models Results

In this section, the results obtained from the Artificial Neural Network and Decision Tree classification models to the datasets under analysis will be presented. The tables presented below show the evaluation metrics to all the *Label* classes. The class *BENIGN* represents the non-attack data while

all the others represent the respective network attack. The support column indicates the number of instances that each class has in the test set.

5.5.1 Original Training Set

The following tables (5.6 to 5.11) present the classification results when the classification models were trained with the pre-processed imbalanced training set.

- **DDoS Dataset**

Table 5.6: Result of the imbalanced DDoS dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99870	0.99801	0.99835	30682
DDoS	0.99857	0.99906	0.99882	42668
			Accuracy: 0.99862	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99938	0.99935	0.99936	30682
DDoS	0.99953	0.99955	0.99954	42668
			Accuracy: 0.99947	

- **Port Scan Dataset**

Table 5.7: Result of the imbalanced Port Scan dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99852	0.99989	0.99921	37788
Port Scan	0.99986	0.99801	0.99893	28160
			Accuracy: 0.99909	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99984	0.99992	0.99988	37788
Port Scan	0.99989	0.99979	0.99984	28160
			Accuracy: 0.99986	

- **Botnet Dataset**

Table 5.8: Result of the imbalanced Botnet dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99873	0.99950	0.99912	55998
Bot	0.93171	0.84327	0.88528	453
			Accuracy: 0.99825	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99977	0.99984	0.99980	55998
Bot	0.97996	0.97130	0.97561	453
			Accuracy: 0.99961	

- **Web Attack Dataset**

Table 5.9: Result of the imbalanced Web Attack dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99854	0.99958	0.99906	47937
W.A. Brute Force	0.66467	0.88446	0.75897	502
W.A. XSS	1.00000	0.00461	0.00917	217
			Accuracy: 0.99396	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99954	0.99971	0.99962	47937
W.A. Brute Force	0.86089	0.85060	0.85571	502
W.A. XSS	0.67907	0.67281	0.67593	217
			Accuracy: 0.99671	

- **Brute Force Dataset**

Table 5.10: Result of the imbalanced Brute Force dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99926	0.99916	0.99921	123231
FTP-Patator	0.97955	0.99659	0.98800	2643
SSH-Patator	0.97255	0.95573	0.96407	1965
			Accuracy: 0.99844	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99986	0.99989	0.99987	123231
FTP-Patator	0.99886	0.99886	0.99886	2643
SSH-Patator	0.99439	0.99288	0.99363	1965
			Accuracy: 0.99976	

- **DoS Dataset**

Table 5.11: Result of the imbalanced DoS dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99550	0.99328	0.99439	128235
DoS GoldenEye	0.96571	0.97725	0.97145	3429
DoS Hulk	0.99015	0.99426	0.99220	72295
DoS Slowhttptest	0.90751	0.94217	0.92452	1833
DoS SlowLoris	0.94549	0.88404	0.91373	1923
			Accuracy: 0.99189	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99601	0.99589	0.99595	128235
DoS GoldenEye	0.99182	0.98979	0.99080	3429
DoS Hulk	0.99347	0.99353	0.99350	72295
DoS Slowhttptest	0.98644	0.99236	0.98939	1833
DoS SlowLoris	0.98809	0.99220	0.99014	1923
			Accuracy: 0.99490	

After analysing all the binary and multi-class classification results, two appreciations can be taken: the first one is the impact of data distribution on machine learning models; the second one is the crucial use of different model evaluation metrics.

As it can be seen in the section 5.2.1, the DDoS and Port Scan datasets are the only ones with a balanced distribution of attack and non-attack data. In these two datasets both models (ANN and DT) had similar results, predicting both classes accurately. On the other hand, in the Botnet, Web Attack, Brute Force, and DoS datasets (imbalanced datasets) just the non-attack data (Benign)

was accurately predicted in both models. Although, if the model evaluation only took into consideration the accuracy metric, these last models would have been considered as extremely good, given the high accuracy. It is in these cases that the F1-score is so important. When there is a large data disparity, only the Precision, Recall, and F1-score metrics can correctly evaluate the model results. Besides that, looking at the entire classification results, it could be concluded that the Decision Tree model works better than the ANN model when the data follow a skewed distribution.

Since the DDoS and `Port Scan` datasets have already a balanced distribution, these two datasets will not be taken into consideration from now on.

5.5.2 SMOTE Balanced Training Set

The following tables (5.12 to 5.15) also result from the original test set classification. However, at this time, the models were trained with the SMOTE balanced data. It was used the *imbalanced-learn*[2] *Python* package to balance the training set through the SMOTE.

- **Botnet Dataset**

Table 5.12: Result of the SMOTE balanced Botnet dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99998	0.99614	0.99806	55998
Bot	0.67665	0.99779	0.80642	453
			Accuracy: 0.99616	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99995	0.99993	0.99994	55998
Bot	0.99119	0.99338	0.99228	453
			Accuracy: 0.99988	

- **Web Attack Dataset**

Table 5.13: Result of the SMOTE balanced Web Attack dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99996	0.99497	0.99764	47937
W.A. Brute Force	0.59459	0.52590	0.55814	502
W.A. XSS	0.30739	0.72811	0.43228	217
			Accuracy: 0.98894	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99985	0.99962	0.99974	47937
W.A. Brute Force	0.95766	0.94622	0.95190	502
W.A. XSS	0.84188	0.90783	0.87361	217
			Accuracy: 0.99866	

2 *imbalanced-learn*

- **Brute Force Dataset**

Table 5.14: Result of the SMOTE balanced Brute Force dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99996	0.99738	0.99867	123231
FTP-Patator	0.97344	0.99849	0.98581	2643
SSH-Patator	0.88533	0.99796	0.93828	1965
			Accuracy: 0.99741	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99995	0.99992	0.99994	123231
FTP-Patator	0.99962	0.99962	0.99962	2643
SSH-Patator	0.99543	0.99746	0.99644	1965
			Accuracy: 0.99987	

- **DoS Dataset**

Table 5.15: Result of the SMOTE balanced DoS dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99912	0.98925	0.99416	128235
DoS GoldenEye	0.94169	0.99854	0.96929	3429
DoS Hulk	0.98695	0.99777	0.99233	72295
DoS Slowhttptest	0.96038	0.99182	0.97585	1833
DoS SlowLoris	0.90141	0.99844	0.94745	1923
			Accuracy: 0.99248	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99867	0.99856	0.99861	128235
DoS GoldenEye	0.99593	0.99825	0.99709	3429
DoS Hulk	0.99776	0.99775	0.99775	72295
DoS Slowhttptest	0.99509	0.99564	0.99536	1833
DoS SlowLoris	0.99430	0.99740	0.99585	1923
			Accuracy: 0.99823	

With regard to the results presented by the Artificial Neural Network model, it is shown that with the use of a balanced training set through SMOTE, there is a significant improvement in terms of the Recall evaluation metric. This indicates that, when trained with this data, the model is able to identify a greater number of attacks comparing with the imbalanced data. However, this improvement in the Recall is followed by a decrease in Precision, i.e., despite the fact that there has been a rise in the number of correctly classified attack instances, true positives, there has also been a rise in the number of false positives, which, in most cases, results in a decrease in the F1-score.

With respect to the classification results of the Decision Tree model, improvements are highlighted in both Recall, Precision, and F1-score evaluation metrics. When trained this model with the SMOTE balanced data, it is revealed an increase in the number of detected attacks as well as in the percentage of correctly classified instances, which leads to a more accurate classification.

5.5.3 GAN Balanced Training Set

At this point in the study, the classification models were trained with the GAN balanced data. Once the training phase was completed, the models were tested with the original test data.

- **Botnet Dataset**
  - Standard GAN generated data

    In this case, the data used to train the classification models were balanced by the standard GAN model. Tables 5.16, 5.18, 5.20, and 5.22, represent all cases where this happens.

Table 5.16: Result of the GAN balanced Botnet dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99896	0.99952	0.99924	55998
Bot	0.93602	0.87196	0.87827	453
			Accuracy: 0.99849	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99982	0.99987	0.99985	55998
Bot	0.98444	0.97792	0.98117	453
			Accuracy: 0.99970	

  - WGAN-GP generated data

    In this case, the data used to train the classification models were balanced by the WGAN-GP model. Tables 5.17, 5.19, 5.21, and 5.23, represent all the cases where this happens.

Table 5.17: Result of the WGAN-GP balanced Botnet dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99939	0.99845	0.99892	55998
Bot	0.82806	0.92494	0.87393	453
			Accuracy: 0.99786	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99980	0.99987	0.99984	55998
Bot	0.98441	0.97572	0.98004	453
			Accuracy: 0.99968	

- **Web Attack Dataset**
  - Standard GAN generated data

Table 5.18: Result of the GAN balanced Web Attack dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99935	0.99952	0.99944	47937
W.A. Brute Force	0.69540	0.96414	0.80801	502
W.A. XSS	0.53333	0.03687	0.06897	217
			Accuracy: 0.99486	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99958	0.99973	0.99966	47937
W.A. Brute Force	0.86815	0.85259	0.86030	502
W.A. XSS	0.68493	0.69124	0.68807	217
			Accuracy: 0.99683	

- WGAN-GP generated data

Table 5.19: Result of the WGAN-GP balanced Web Attack dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99946	0.99935	0.99941	47937
W.A. Brute Force	0.67265	0.97012	0.79445	502
W.A. XSS	0.00000	0.00000	0.00000	217
			Accuracy: 0.99459	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99958	0.99973	0.99966	47937
W.A. Brute Force	0.86735	0.84661	0.85685	502
W.A. XSS	0.67117	0.68664	0.67882	217
			Accuracy: 0.99675	

- **Brute Force Dataset**

  - Standard GAN generated data

Table 5.20: Result of the GAN balanced Brute Force dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99974	0.99935	0.99955	123231
FTP-Patator	0.98835	0.99546	0.99189	2643
SSH-Patator	0.97392	0.98830	0.98106	1965
			Accuracy: 0.99910	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99988	0.99985	0.99987	123231
FTP-Patator	0.99886	0.99886	0.99886	2643
SSH-Patator	0.99238	0.99389	0.99314	1965
			Accuracy: 0.99974	

  - WGAN-GP generated data

Table 5.21: Result of the WGAN-GP balanced Brute Force dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99968	0.99972	0.99970	123231
FTP-Patator	0.99320	0.99546	0.99433	2643
SSH-Patator	0.98977	0.98473	0.98724	1965
			Accuracy: 0.99940	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99989	0.99987	0.99988	123231
FTP-Patator	0.99924	0.99773	0.99849	2643
SSH-Patator	0.99239	0.99542	0.99390	1965
			Accuracy: 0.99976	

- **DoS Dataset**
  - Standard GAN generated data

Table 5.22: Result of the GAN balanced DoS dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99887	0.98900	0.99391	128235
DoS GoldenEye	0.97313	0.98221	0.97765	3429
DoS Hulk	0.98230	0.99881	0.99049	72295
DoS Slowhttptest	0.97216	0.97163	0.97190	1833
DoS SlowLoris	0.97274	0.98336	0.97802	1923
			Accuracy: 0.99209	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99611	0.99611	0.99611	128235
DoS GoldenEye	0.99416	0.99271	0.99343	3429
DoS Hulk	0.99368	0.99364	0.99366	72295
DoS Slowhttptest	0.98644	0.99182	0.98912	1833
DoS SlowLoris	0.99427	0.99324	0.99376	1923
			Accuracy: 0.99513	

  - WGAN-GP generated data

Table 5.23: Result of the WGAN-GP balanced DoS dataset classification.

Artificial Neural Network				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99595	0.99321	0.99458	128235
DoS GoldenEye	0.98588	0.93642	0.96051	3429
DoS Hulk	0.98913	0.99660	0.99285	72295
DoS Slowhttptest	0.95170	0.97818	0.96476	1833
DoS SlowLoris	0.97785	0.94124	0.95919	1923
			Accuracy: 0.99284	

Decision Tree				
Label Classes	**Precision**	**Recall**	**F1-score**	**Support**
BENIGN	0.99612	0.99589	0.99600	128235
DoS GoldenEye	0.98838	0.99213	0.99025	3429
DoS Hulk	0.99358	0.99364	0.99361	72295
DoS Slowhttptest	0.98483	0.99182	0.98831	1833
DoS SlowLoris	0.99272	0.99220	0.99246	1923
			Accuracy: 0.99497	

Looking at the obtained results, both the data generated by the standard GAN and the data generated by the WGAN-GP reveal improvements in the detection of attacks compared to the original data. Nevertheless, the Decision Tree classification model continues to present better classification results than the Artificial Neural Network model.

Since the quality of the data produced by the standard GAN has been shown to be inferior to the quality of the data generated by the WGAN-GP, it would be expected that worse results would be obtained with the use of the first-mentioned data. However, this was not the case. Contrarily to what was anticipated, in some datasets, the training set balanced by the standard GAN has revealed better classification results than the training set balanced by WGAN-GP.

5.6 Comparative Analysis

In order to compare the previously presented classification results, the following tables (5.24 and 5.25) were built. The values presented in these tables were calculated by the difference between the results obtained from the balanced training data and the results obtained from the original training data. Through these tables, it is possible to identify which pre-processed training data reaches the best results for each classification model.

Table 5.24: Comparison between the outcomes of the ANN classification model. Difference between the 3 distinct balanced data results and the original imbalanced data results. Red means negative difference (worse results), green means positive difference (better results).

	SMOTE		GAN		WGAN-GP	
Label Classes	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Botnet	0,25506 ↓	0,15452 ↑	0,00431 ↑	0,02869 ↑	0,10365 ↓	0,08167 ↑
Web Attack Brute Force	0,07008 ↓	0,35856 ↓	0,03073 ↑	0,07968 ↑	0,00798 ↑	0,08566 ↑
Web Attack XSS	0,69261 ↓	0,7235 ↑	0,46667 ↓	0,03226 ↑	1 ↓	0,00461 ↓
FTP-Patator	0,00611 ↓	0,00190 ↑	0,00880 ↑	0,00113 ↓	0,01365 ↑	0,00113 ↓
SSH-Patator	0,08859 ↓	0,04223 ↑	0,00137 ↑	0,03257 ↑	0,01722 ↑	0,02900 ↑
DoS GoldenEye	0,02402 ↓	0,02129 ↑	0,00742 ↑	0,00496 ↑	0,02017 ↑	0,04083 ↓
DoS Hulk	0,00320 ↓	0,00351 ↑	0,00785 ↓	0,00455 ↑	0,00102 ↓	0,00234 ↑
DoS Slowhttptest	0,05287 ↑	0,04965 ↑	0,06465 ↑	0,02946 ↑	0,04419 ↑	0,03601 ↑
DoS SlowLoris	0,04408 ↓	0,11440 ↑	0,02725 ↑	0,09932 ↑	0,03236 ↑	0,0572 ↑

Based on the observation of table 5.24, it could be concluded that, for the Artificial Neural Network model, the training set that presents the best results of Precision and Recall is the one that was generated through the standard GAN. While the data generated through SMOTE shows a greater increase in the detection of attacks (Recall), it also shows a significant drop in Precision. If the user's main goal is to increase the identification of attacks regardless of the consequences that high false-positive rates may have, SMOTE is the approach that should be used to balance the dataset since it is the one that presents the most significant improvement in Recall. On the other hand, if the users want to boost the identification of attacks without compromising the precision of the classified instances, the best choice is to balance the training set using a standard GAN.

Contrary to what was observed in table 5.24, table 5.25 reveals that the training set balanced by SMOTE is the one that presents the best results of Precision and Recall for the Decision Tree classification model. Once again the data generated by the GAN model yields better results than the

data obtained by the WGAN-GP model. Although, for the Decision Tree model, it wasn't able to beat the results of the SMOTE generated data.

Table 5.25: Comparison between the outcomes of the DT classification model. Difference between the 3 distinct balanced data results and the original imbalanced data results. Red means negative difference (worse results), green means positive difference (better results), blue means zero difference (same results).

	SMOTE		GAN		WGAN-GP	
Label Classes	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Botnet	0,01123 ↑	0,02208 ↑	0,00448 ↑	0,00662 ↑	0,00445 ↑	0,00442 ↑
Web Attack Brute Force	0,09677 ↑	0,09562 ↑	0,00726 ↑	0,00199 ↑	0,00646 ↑	0,00399 ↓
Web Attack XSS	0,16281 ↑	0,23502 ↑	0,00586 ↑	0,01843 ↑	0,00790 ↓	0,01383 ↑
FTP-Patator	0,00076 ↑	0,00076 ↑	=	=	0,00038 ↑	0,00113 ↓
SSH-Patator	0,00104 ↑	0,00458 ↑	0,00201 ↓	0,00101 ↑	0,00200 ↓	0,00254 ↑
DoS GoldenEye	0,00411 ↑	0,00730 ↑	0,00234 ↑	0,00292 ↑	0,00344 ↓	0,00234 ↑
DoS Hulk	0,00429 ↑	0,00422 ↑	0,00021 ↑	0,00011 ↑	0,00011 ↑	0,00011 ↑
DoS Slowhttptest	0,00865 ↑	0,00300 ↑	=	0,00054 ↓	0,00161 ↓	0,00054 ↓
DoS SlowLoris	0,00621 ↑	0,00520 ↑	0,00463 ↑	0,00104 ↑	0,00463 ↑	=

These findings confirm that balancing the training set is advantageous for both classification models. In general, the training set balanced by the standard GAN is what shows a more consistent improvement in the results of classification, when considering both models (ANN and DT). These conclusions reveal to be outstanding since the data generated by the standard GAN model was the one with a less similar distribution to the real data. Leading to the assumption that with more diversified data, the model's ability to detect new attacks is enhanced.

5.7 Discussion

In addition to the use of Generative Adversarial Networks to improve the detection of network anomalies, this research also noticed that the Decision Tree model was able to achieve better classification results for the CICIDS2017 dataset than the Artificial Neural Network model.

The data properties could be the reason for ANN's poor results. As CICIDS2017 is a network flow dataset, many of the variables are categorical. Decision Trees can handle these variables very well, however, the Artificial Neural Networks expect all input values to be numerical [73]. When trained with categorical values, the ANN model misunderstands the data to be in some order, $0 < 6 < 17$

(`Protocol` values: HOPOPT(0), TCP(6), UDP(17)), which is not true. During the implementation of the pre-processing stage this was not taken into account. Through the use of the label encoding technique, only the textual categorical variables were transformed into numerical values. Even though, this technique may not be the best for the ANN models since it uses number sequencing.

For that reason, a warning note is left. To achieve state-of-the-art results on network anomalies detection using an Artificial Neural Network classification model, it is recommended the use of different encoding techniques [38, 73] in all the categorical variables (numerical or textual).

6 Conclusion

During the course of this work, detailed conclusions on the outcomes achieved at each stage of the project were presented. As such, only the conclusions that can be drawn from the overall work will be addressed in this chapter. In addition, it will be described the contributions to the network security and adversarial deep learning fields, along with work improvements that may be explored in the future.

6.1 Conclusions

The presented work is intended to improve the detection of anomalies in the network through the use of adversarial deep learning methods. Due to the sparse availability of network flow data and its multi-class imbalanced problem, the proposed solution focuses on generating new synthetic data from the current one in order to increase the learning ability of the classifiers.

The final system results in a pipeline that receives an imbalanced dataset as input and, through the use of Generative Adversarial Networks as an oversampling technique, makes it balanced. Alongside this, the traditional data augmentation technique, SMOTE, is applied to the imbalanced dataset. The pipeline is finalised with the ANN and DT classification models that are independently trained with each data (imbalanced data, GANs balanced data, and SMOTE balanced data).

The application of the proposed system to the CICIDS2017 dataset revealed that the use of Generative Adversarial Networks to balance the training set has increased the efficiency of classification models in the detection of anomalies in the network. Moreover, it was observed that the standard GAN model produced samples of lower quality than the samples produced by the WGAN-GP, and even so, showed a greater improvement in the results of the classifiers, leading to the assumption that with more diversified data, the model's ability to detect emerging and evolving attacks is enhanced.

Despite the great results obtained with the balanced data from the GANs, the SMOTE balanced data was the one that showed the greatest increase in Recall. The biggest difference between these two techniques was in the similarity to reality. During the training process, the GAN models presented numerous limitations that proved to be detrimental to the quality of the data generated. Although the data produced by the GANs were close to the real, they were not yet at the level of the data generated by the SMOTE.

With regard to the classification models, the results obtained revealed that the Decision Tree model was the best performing classifier in the detection of network anomalies. These results are supported by the properties of the data under study. Many of the attributes present in the network flow data are categorical (for example, protocol, source port, etc.). The Decision Tree model is better suited to categorical data than the Artificial Neural Network model, making it one of the best choices for network anomaly detection classifiers.

Lastly, it can be concluded that GANs have far more to offer than just image generation. This project has shown that Generative Adversarial Networks are also capable of generating continuous and categorical data of good quality, disclosing an additional value for processes of data augmentation.

6.2 Prospect for Future Work

Despite the good results obtained with the developed system, it would be useful to improve the quality of the data generated by GANs. As such, new approaches that promote GANs training stability should be studied and tested. The Two Time-scale Update Rule (TTUR) proposed by Heusel et al. [41] will be one of the approaches to be considered as future work, as it improves the convergence between the two networks during the training process.

In terms of data pre-processing, new approaches to deal with categorical variables should be considered, e.g. one hot label encoding. Besides this, it would be interesting to explore ANN hyper-parameter optimization with Genetic Algorithms [57, 95]. In this optimization technique, the specific problem is pictured in a chromosome composed of genes. Each gene represents a characteristic of the problem, e.g. number of neurons, layers, epochs, batch size, activation function. After specifying the problem, the algorithm will perform several iterations (each iteration consists of: training, selection, crossover, and mutation) with different chromosomes configuration. Once the Genetic Algorithm has completed all iterations the results will be presented as a population of solutions.

With these improvements in data pre-processing and hyperparameter optimization, an increase in the identification of network anomalies via the Artificial Neural Network model is expected.

Bibliography

[1] L. Abdi and S. Hashemi. "To Combat Multi-Class Imbalanced Problems by Means of Over-Sampling Techniques". In: *IEEE Transactions on Knowledge and Data Engineering* 28.1 (2016), pp. 238–251.

[2] Abien Fred Agarap. "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375* (2018).

[3] Samet Akcay, Amir Atapour Abarghouei, and Toby P. Breckon. "GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training". In: *CoRR* abs/1805.06725 (2018). arXiv: 1805.06725.

[4] Alex Smola and S.V.N. Vishwanathan. "Introduction to Machine Learning". In: *University Press, Cambridge* (October 1, 2010), p. 213.

[5] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. "The history began from alexnet: A comprehensive survey on deep learning approaches". In: *arXiv preprint arXiv:1803.01164* (2018).

[6] Martin Arjovsky and Léon Bottou. "Towards principled methods for training generative adversarial networks". In: *arXiv preprint arXiv:1701.04862* (2017).

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[8] Anurag Bhardwaj, Wei Di, and Jianing Wei. *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling*. Packt Publishing Ltd, 2018.

[9] Sebastian Blank. *Deep Learning Fundamentals*. [Accessed January 9, 2019]. url: https://www.inovex.de/blog/deep-learning-fundamentals/.

[10] Mikael Boden. "A guide to recurrent neural networks and backpropagation". In: *the Dallas project* (2002).

[11] Ali Borji. “Pros and Cons of GAN Evaluation Measures”. In: *CoRR* abs/1802.03446 (2018). arXiv: 1802.03446.

[12] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT’2010*. Springer, 2010, pp. 177–186.

[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[14] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. “One billion word benchmark for measuring progress in statistical language modeling”. In: *arXiv preprint arXiv:1312.3005* (2013).

[15] Ting Chen, Xiaohua Zhai, Marvin Ritter, Mario Lucic, and Neil Houlsby. “Self-supervised gans via auxiliary rotation loss”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12154–12163.

[16] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. “Project Adam: Building an Efficient and Scalable Deep Learning Training System”. In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 571–582. isbn: 978-1-931971-16-4.

[17] F Chollet. *Deep Learningwith Python*. Manning Publications Co., 2018, p. 447. isbn: 9783958458406.

[18] Cloudflare. *Famous DDoS Attacks*. [Accessed October 10, 2019]. url: https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/.

[19] Cloudflare. *Proteger contra ataques de DDoS Mitigação ilimitada de DDoS para manter o desempenho e a disponibilidade*. [Accessed October 10, 2019]. url: https://www.cloudflare.com/pt-br/ddos/.

[20] Graupe Daniel. *Principles of artificial neural networks*. Vol. 7. World Scientific, 2013.

[21] Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelfi. “A survey on GANs for anomaly detection”. In: *arXiv preprint arXiv:1906.11632* (2019).

[22] Diego Pereira Domingos. “Utilização de GPU para Sistemas de Paralelismo Massivo Resumo”. In: *unicamp* (2012), pp. 1–9.

[23] Stephan Dreiseitl and Lucila Ohno-Machado. “Logistic regression and artificial neural network classification models: a methodology review”. In: *Journal of biomedical informatics* 35.5-6 (2002), pp. 352–359.

[24] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. “Deep reinforcement learning framework for autonomous driving”. In: *IS and T International Symposium on Electronic Imaging Science and Technology* (2017), pp. 70–76. issn: 24701173. arXiv: 1704.02532.

[25] Alberto Fernández, Salvador Garcia, Francisco Herrera, and Nitesh V Chawla. “SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary”. In: *Journal of artificial intelligence research* 61 (2018), pp. 863–905.

[26] Forcepoint. *What is Network Security? Network Security Defined, Explained, and Explored*. [Accessed October 10, 2019]. url: https://www.forcepoint.com/cyber-edu/network-security.

[27] Cameron Freer, André Nies, Frank Stephan, et al. “Algorithmic aspects of Lipschitz functions”. In: *Computability* 3.1 (2014), pp. 45–61.

[28] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. “An evaluation framework for intrusion detection dataset”. In: *2016 International Conference on Information Science and Security (ICISS)*. IEEE. 2016, pp. 1–6.

[29] GitHub. *February 28th DDoS Incident Report*. [Accessed October 11, 2019]. url: https://github.blog/2018-03-01-ddos-incident-report/.

[30] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: JMLR Workshop and Conference Proceedings, Nov. 2011, pp. 315–323.

[31] Dieter Gollmann. “Computer security”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.5 (2010), pp. 544–554.

[32] Ian Goodfellow. “NIPS 2016 tutorial: Generative adversarial networks”. In: *arXiv preprint arXiv:1701.00160* (2016).

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[35] Samuel B Griffith. *Sun Tzu: The art of war*. Vol. 39. Oxford University Press London, 1963.

[36] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. “Improved training of wasserstein gans”. In: *Advances in neural information processing systems*. 2017, pp. 5767–5777.

[37] Jun Han and Claudio Moraga. “The influence of the sigmoid function parameters on the speed of backpropagation learning”. In: *International Workshop on Artificial Neural Networks*. Springer. 1995, pp. 195–201.

[38] John T Hancock and Taghi M Khoshgoftaar. “Survey on categorical data for neural networks”. In: *Journal of Big Data* 7 (2020), pp. 1–41.

[39] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1994. isbn: 0023527617.

[40] Juha Heinonen. *Lectures on Lipschitz analysis*. 100. University of Jyväskylä, 2005.

[41] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems*. 2017, pp. 6626–6637.

[42] L. A. Jeni, J. F. Cohn, and F. De La Torre. “Facing Imbalanced Data–Recommendations for the Use of Performance Metrics”. In: *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*. 2013, pp. 245–251.

[43] Brendan Juba and Hai S Le. “Precision-recall versus accuracy and the role of large data sets”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 4039–4048.

[44] Bekir Karlik and A. Vehbi Olgac. “Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks”. In: *International Journal of Artificial Intelligence And Expert Systems (IJAE)* 1 (2019). issn: 17426596.

[45] Keras. *Keras Simple. Flexible. Powerful.* [Accessed November 2, 2020]. url: https://keras.io/.

[46] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[47] Igor Kononenko and Matjaž Kukar. “Chapter 11 - Artificial Neural Networks”. In: *Machine Learning and Data Mining*. Ed. by Igor Kononenko and Matjaž Kukar. Woodhead Publishing, 2007, pp. 275–320. isbn: 978-1-904275-21-3.

[48] Vijay Kotu and Bala Deshpande. “Chapter 4 - Classification”. In: *Predictive Analytics and Data Mining*. Ed. by Vijay Kotu and Bala Deshpande. Boston: Morgan Kaufmann, 2015, pp. 63–163. isbn: 978-0-12-801460-8.

[49] Bartosz Krawczyk. “Learning from imbalanced data: open challenges and future directions”. In: *Progress in Artificial Intelligence* 5.4 (2016), pp. 221–232.

[50] David Kriesel. *A brief introduction on neural networks*. Citeseer, 2007.

[51] Gulshan Kumar, Amanjeet Kaur, and Sania Sethi. “Computer Network Attacks - A Study”. In: *International Journal of Computer Science and Mobile Applications* 2.January 2014 (2014), pp. 24–32.

[52] Thomas Kurbiel and Shahrzad Khaleghian. “Training of Deep Neural Networks based on Distance Measures using RMSProp”. In: *CoRR* abs/1708.01911 (2017). arXiv: 1708.01911.

[53] Kurt Langfeld and David Mcmullan. “Code Breaking”. In: (2008).

[54] J. LIN and S. K. M. WONG. “A NEW DIRECTED DIVERGENCE MEASURE AND ITS CHARACTERIZATION”. In: *International Journal of General Systems* 17.1 (1990), pp. 73–81.

[55] Ming-Yu Liu and Oncel Tuzel. “Coupled generative adversarial networks”. In: *Advances in neural information processing systems*. 2016, pp. 469–477.

[56] David J Livingstone, David T Manallack, and Igor V Tetko. “Data modelling with neural networks: advantages and limitations”. In: *Journal of computer-aided molecular design* 11.2 (1997), pp. 135–142.

[57] Sehla Loussaief and Afef Abdelkrim. “Convolutional neural network hyper-parameters optimization based on genetic algorithms”. In: *International Journal of Advanced Computer Science and Applications* 9.10 (2018), pp. 252–266.

[58] Daniel Lowd and Christopher Meek. “Adversarial learning”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2005), pp. 641–647.

[59] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. “Unrolled generative adversarial networks”. In: *arXiv preprint arXiv:1611.02163* (2016).

[60] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černock, and Sanjeev Khudanpur. “Recurrent neural network based language model”. In: *Eleventh annual conference of the international speech communication association*. 2010.

[61] Mohammad Mohammadhassani, Hossein Nezamabadi-Pour, Mohd Zamin Jumaat, Mohammed Jameel, and Arul M.S. Arumugam. “Application of artificial neural networks (ANNs) and linear regressions (LR) to predict the deflection of concrete deep beams”. In: *Computers and Concrete* 11.3 (2013), pp. 237–252. issn: 15988198.

[62] Mahesh Chandra Mukkamala and Matthias Hein. “Variants of rmsprop and adagrad with logarithmic regret bounds”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2545–2553.

[63] Joshua Ojo Nehinbe. “A critical evaluation of datasets for investigating IDSs and IPSs researches”. In: *2011 IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS)*. IEEE. 2011, pp. 92–97.

[64] University of New Brunswick - Canadian Institute for Cybersecurity. *Applications - CICFlowMeter | CIC-AB*. [Accessed October 26, 2020]. url: https://www.unb.ca/cic/research/applications.html#CICFlowMeter.

[65] Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nati Srebro. “Exploring Generalization in Deep Learning”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5947–5956.

[66] Giang Nguyen, Stefan Dlugolinsky, Martin Bobák, Viet Tran, Álvaro López García, Ignacio Heredia, Peter Malík, and Ladislav Hluchý. “Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey”. In: *Artificial Intelligence Review* 52.1 (2019), pp. 77–124.

[67] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. “Activation functions: Comparison of trends in practice and research for deep learning”. In: *arXiv preprint arXiv:1811.03378* (2018).

[68] Douwe Osinga. *Deep Learning Cookbook: Practical Recipes to Get Started Quickly*. O’Reilly Media, Inc., 2018.

[69] Ranjit Panigrahi and Samarjeet Borah. “A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems”. In: 7 (Jan. 2018), pp. 479–482.

[70] Aniruddha Parvat, Jai Chavan, Siddhesh Kadam, Souradeep Dev, and Vidhi Pathak. “A survey of deep-learning frameworks”. In: *2017 International Conference on Inventive Systems and Control (ICISC)*. IEEE. 2017, pp. 1–7.

[71] O. Pele and M. Werman. “Fast and robust Earth Mover’s Distances”. In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 460–467.

[72] Ben Poole, Alexander A Alemi, Jascha Sohl-Dickstein, and Anelia Angelova. “Improved generator objectives for gans”. In: *arXiv preprint arXiv:1612.02780* (2016).

[73] Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. “A comparative study of categorical variable encoding techniques for neural network classifiers”. In: *International journal of computer applications* 175.4 (2017), pp. 7–9.

[74] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[75] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Searching for activation functions”. In: *arXiv preprint arXiv:1710.05941* (2017).

[76] Bernadette M Randles, Irene V Pasquetto, Milena S Golshan, and Christine L Borgman. “Using the Jupyter notebook as a tool for open science: An empirical study”. In: *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. IEEE. 2017, pp. 1–2.

[77] Check Point Research. “Cyber Attack Trends Analysis Key insights to gear up for in 2019”. In: Volume 01 (2019).

[78] Danilo Jimenez Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. “One-shot generalization in deep generative models”. In: *arXiv preprint arXiv:1603.05106* (2016).

[79] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. “Flow-based network traffic generation using Generative Adversarial Networks”. In: *Computers & Security* 82 (2019), pp. 156–172.

[80] Craig H Rowland. *Intrusion detection system*. US Patent 6,405,318. June 2002.

[81] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747.

[82] Jake Ryan, Meng-Jang Lin, and Risto Miikkulainen. “Intrusion detection with neural networks”. In: *Advances in neural information processing systems*. 1998, pp. 943–949.

[83] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. “Improved techniques for training gans”. In: *Advances in neural information processing systems*. 2016, pp. 2234–2242.

[84] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. “Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery”. In: *CoRR* abs/1703.05921 (2017). arXiv: 1703.05921.

[85] Risk Based Security. *Data Breach QuickView Report Year End 2018 - Data Breach Trends*. Tech. rep. Risk Based Security, Inc., Feb. 2019, pp. 1–19.

[86] Jochen Seitz. *Fundamentals of Network Security*. Vol. 24. 2002, pp. 89–90. isbn: 1580531768.

[87] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. “Toward generating a new intrusion detection dataset and intrusion traffic characterization.” In: *ICISSP*. 2018, pp. 108–116.

[88] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. “Toward developing a systematic approach to generate benchmark datasets for intrusion detection”. In: *computers & security* 31.3 (2012), pp. 357–374.

[89] Ida G Sprinkhuizen-Kuyper and Egbert JW Boers. *The Shape of the Error Surfaces of some simple Neural Networks*. Rijksuniversiteit Leiden. Vakgroep Informatica, 1995.

[90] William Stallings. *Network Security Essentials 5Th Edition*. 2015, pp. 2–5. isbn: 9780136108054.

[91] Jerzy Stefanowski. “Neighbourhood Sampling in Bagging for Imbalanced Data”. In: *Neurocomputing* (2015), pp. 184–203.

[92] Dinoj Surendran. *Swiss Roll Dataset*. [Accessed October 07, 2020]. May 2004. url: http://people.cs.uchicago.edu/~dinoj/manifold/swissroll.html.

[93] TensorFlow. *Why TensorFlow*. [Accessed November 2, 2020]. url: https://www.tensorflow.org/about.

[94] Paolo Viviani, Maurizio Drocco, Daniele Baccega, Iacopo Colonnelli, and Marco Aldinucci. "Deep Learning at Scale". In: *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. Feb. 2019, pp. 124–131.

[95] Michael D Vose. *The simple genetic algorithm: foundations and theory*. MIT press, 1999.

[96] Hans-Dieter Wehle. "Machine Learning, Deep Learning, and AI: What's the Difference?" In: *Conference: Data Scientist Innovation Day* (July 2017).

[97] T. Andrew Yang. "Computer Security and Impact on Computer Science Education". In: *Journal of Computing Sciences in Colleges* 4.May 2001 (2001), pp. 233–246. issn: 1937-4771.

[98] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. "Seqgan: Sequence generative adversarial nets with policy gradient". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[99] Mahdi Zamani and Mahnush Movahedi. "Machine Learning Techniques for Intrusion Detection". In: (2013), pp. 1–11. arXiv: 1312.2177.

[100] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. "Efficient GAN-Based Anomaly Detection". In: *CoRR* abs/1802.06222 (2018). arXiv: 1802.06222.

[101] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. "A Sufficient Condition for Convergences of Adam and RMSProp". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

Appendix

CICIDS2017 Dataset Features Description

Table 6.1: CICIDS2017 dataset features description.

#	Attribute	Description	Type	Example
1	Flow ID	Flow identifier	categorical	192.168.10.5-104.16.207.165-54865-443-6
2	Source IP	Source IP address	categorical	104.16.207.165
3	Source Port	Source Port	categorical	443
4	Destination IP	Destination IP address	categorical	192.168.10.5
5	Destination Port	Destination Port	categorical	54865
6	Protocol	Protocol number HOPOPT(0) TCP(6) UDP(17)	categorical	6
7	Time stamp	Flow time stamp	categorical	07/07/2017 03:30:00
8	Flow Duration	Duration of the flow in Microsecond	int	1293792
9	Total Fwd Packets	Total packets in the forward direction	int	3
10	Total Backward Packets	Total packets in the backward direction	int	7
11	Total Length of Fwd Packets	Total size of packet in forward direction	int	26
12	Total Length of Bwd Packets	Total size of packet in backward direction	int	11607

Continued on next page

Table 6.1 – continued from previous page

#	Attribute	Description	Type	Example
13	Fwd Packet Length Max	Maximum size of packet in forward direction	int	20
14	Fwd Packet Length Min	Minimum size of packet in forward direction	int	0
15	Fwd Packet Length Mean	Mean size of packet in forward direction	float	8.67
16	Fwd Packet Length Std	Standard deviation size of packet in forward direction	float	10.26
17	Bwd Packet Length Max	Maximum size of packet in backward direction	int	5840
18	Bwd Packet Length Min	Minimum size of packet in backward direction	int	0
19	Bwd Packet Length Mean	Mean size of packet in backward direction	float	1658.14
20	Bwd Packet Length Std	Standard deviation size of packet in backward direction	float	2137.30
21	Flow Bytes/s	Number of flow packets per second	float	8991.40
22	Flow Packets/s	Number of flow packets per second	float	7.73
23	Flow IAT Mean	Mean time between two packets sent in the flow	float	143754.67
24	Flow IAT Std	Standard deviation time between two packets sent in the flow	float	430865.81
25	Flow IAT Max	Maximum time between two packets sent in the flow	int	1292730
26	Flow IAT Min	Minimum time between two packets sent in the flow	int	2

Continued on next page

Table 6.1 – continued from previous page

#	Attribute	Description	Type	Example
27	Fwd IAT Total	Total time between two packets sent in the forward direction	int	747
28	Fwd IAT Mean	Mean time between two packets sent in the forward direction	float	373.5
29	Fwd IAT Std	Standard deviation time between two packets sent in the forward direction	float	523.97
30	Fwd IAT Max	Maximum time between two packets sent in the forward direction	int	744
31	Fwd IAT Min	Minimum time between two packets sent in the forward direction	int	3
32	Bwd IAT Total	Total time between two packets sent in the backward direction	int	1293746
33	Bwd IAT Mean	Mean time between two packets sent in the backward direction	float	215624.33
34	Bwd IAT Std	Standard deviation time between two packets sent in the backward direction	float	527671.93
35	Bwd IAT Max	Maximum time between two packets sent in the backward direction	int	1292730
36	Bwd IAT Min	Minimum time between two packets sent in the backward direction	int	2

Continued on next page

Table 6.1 – continued from previous page

#	Attribute	Description	Type	Example
37	Fwd PSH Flags	Number of times the PSH flag was set in packets traveling in the forward direction (0 for UDP)	int	0
38	Bwd PSH Flags	Number of times the PSH flag was set in packets traveling in the backward direction (0 for UDP)	int	0
39	Fwd URG Flags	Number of times the URG flag was set in packets traveling in the forward direction (0 for UDP)	int	0
40	Bwd URG Flags	Number of times the URG flag was set in packets traveling in the backward direction (0 for UDP)	int	0
41	Fwd Header Length	Total bytes used for headers in the forward direction	int	72
42	Bwd Header Length	Total bytes used for headers in the backward direction	int	152
43	Fwd Packets/s	Number of forward packets per second	float	2.32
44	Bwd Packets/s	Number of backward packets per second	float	5.41
45	Min Packet Length	Minimum length of a packet	int	0
46	Max Packet Length	Maximum length of a packet	int	5840
47	Packet Length Mean	Mean length of a packet	float	1057.55
48	Packet Length Std	Standard deviation length of a packet	float	1853.44
49	Packet Length Variance	Variance length of a packet	float	3435230.67
50	FIN Flag Count	Number of packets with FIN	int	0

Continued on next page

Table 6.1 – continued from previous page

#	Attribute	Description	Type	Example
51	SYN Flag Count	Number of packets with SYN	int	0
52	RST Flag Count	Number of packets with RST	int	0
53	PSH Flag Count	Number of packets with PUSH	int	1
54	ACK Flag Count	Number of packets with ACK	int	0
55	URG Flag Count	Number of packets with URG	int	0
56	CWE Flag Count	Number of packets with CWE	int	0
57	ECE Flag Count	Number of packets with ECE	int	0
58	Down/Up Ratio	Download and upload ratio	int	2
59	Average Packet Size	Average size of packet	float	1163.3
60	Avg Fwd Segment Size	Average size observed in the forward direction	float	8.67
61	Avg Bwd Segment Size	Average number of bytes bulk rate in the forward direction	float	1658.14
62	Fwd Avg Bytes/Bulk	Average number of bytes bulk rate in the forward direction	float	0
63	Fwd Avg Packets/Bulk	Average number of packets bulk rate in the forward direction	float	0
64	Fwd Avg Bulk Rate	Average number of bulk rate in the forward direction	float	0
65	Bwd Avg Bytes/Bulk	Average number of bytes bulk rate in the backward direction	float	0
66	Bwd Avg Packets/Bulk	Average number of packets bulk rate in the backward direction	float	0
67	Bwd Avg Bulk Rate	Average number of bulk rate in the backward direction	float	0
68	Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction	int	3

Continued on next page

Table 6.1 – continued from previous page

#	Attribute	Description	Type	Example
69	Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction	int	26
70	Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction	int	7
71	Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction	int	11607
72	Init_Win_bytes_forward	The total number of bytes sent in initial window in the forward direction	int	8192
73	Init_Win_bytes_backward	The total number of bytes sent in initial window in the backward direction	int	229
74	Act_data_pkt_fwd	Count of packets with at least 1 byte of TCP data payload in the forward direction	int	2
75	Min_seg_size_forward	Minimum segment size observed in the forward direction	int	20
76	Active Mean	Mean time a flow was active before becoming idle	int	0
77	Active Std	Standard deviation time a flow was active before becoming idle	float	0
78	Active Max	Maximum time a flow was active before becoming idle	int	0
79	Active Min	Minimum time a flow was active before becoming idle	int	0

Continued on next page

Table 6.1 – continued from previous page

#	Attribute	Description	Type	Example
80	Idle Mean	Mean time a flow was active before becoming idle	int	0
81	Idle Std	Standard deviation time a flow was idle before becoming active	float	0
82	Idle Max	Maximum time a flow was idle before becoming active	int	0
83	Idle Min	Minimum time a flow was idle before becoming active	int	0
84	Label	Flow label	categorical	DDoS