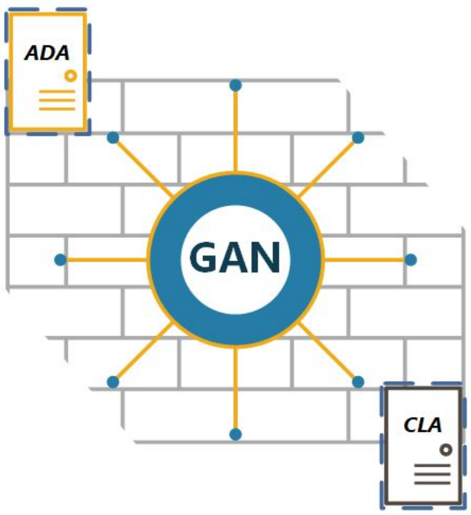
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΙΧΝΕΥΣΗ ΕΙΣΒΟΛΩΝ ΜΕ ΤΗ ΧΡΗΣΗ ΒΑΘΙΑΣ ΚΑΙ ΜΗ ΚΕΝΤΡΙΚΟΠΟΙΗΜΕΝΗΣ ΜΑΘΗΣΗΣ

του

Ηλία Σινιόσογλου
1074



Επιβλέπων Καθηγητής: Επίκουρος Καθηγητής Παναγιώτης Σαρηγιαννίδης

ΙΟΥΝΙΟΣ 2020, ΚΟΖΑΝΗ

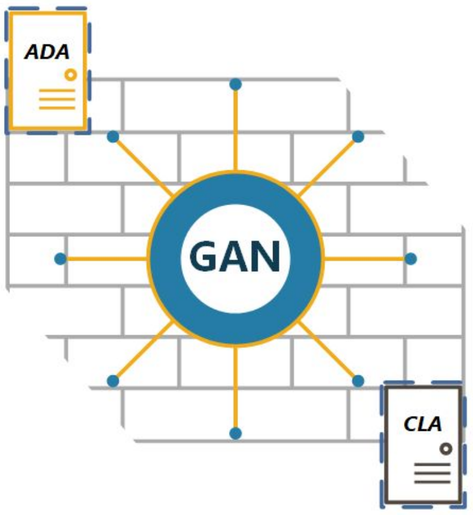
UNIVERSITY OF WESTERN MACEDONIA
DEPARTMENT OF COMPUTER SCIENCE AND TELECOMMUNICATION ENGINEERING

Diploma Thesis

INTRUSION DETECTION USING DEEP AND FEDERATED LEARNING

by

ILIAS SINIOSOGLOU
1074



Supervisor: Assistant Professor Panagiotis Sarigianidis

JUNE 2020, KOZANI

ACKNOWLEDGMENT

The last five years, in which I have completed the span of my degree studies, have been eventful and colorful with unexpected iridescence moments and have been full of knowledge. Finishing this long journey and stepping onward to an unexplored future, I find myself thankful to the Institution, the University of Western Macedonia, and its faculty that guided me on this course.

As such, I would firstly like to thank my supervisor, Assistant Professor Panagiotis Sarigianidis, for his continuous support and guidance that he offered for this work and beyond. Secondly I would like to thank Assistant Professor Stamatia Bibi for her insightful input and comments in this work. Moreover, I would like to express my deep gratitude to Professor Vasilis Argyriou for his critical input and reference in the field of this thesis.

Finally, I want to give my appreciation to everyone that helped me and supported me through my journey.

Abstract

In the frantic momentum that today's technology is rapidly progressing, both people, atomically, and organizations are trying to keep up with the latest and safest ways to procure their everyday interactions with the digital world. More often than not, though, malicious entities, those being socioeconomical or deterministic elements, try to undermine the normal way that systems work and to actively exploit users or companies by overcoming the security mechanisms that protect their privacy and function.

To tackle this problem and ensure the protection of the privacy and safe operation of digital services, the field of cybersecurity strives to minimize the invasion of malicious entities into personal or commercial networks. One of the implemented techniques used in the field of cybersecurity are the Intrusion Detection Systems (IDS) that stand to recognize and in extend prevent an attack to protect digital resources. By using already known digital system interaction and through the methods of Machine Learning and Deep Learning, build and train models that detect such attacks. A problem that arises, though, with the anonymization of the data used to train IDS systems. Federation Learning, a novel decentralized method of training and communicating deep learning models addresses this problem by decoupling the data from the training process of centralized systems. This work delves into the building of Intrusion Detection systems with deep learning algorithms and adapting them to the powerful collective process of Federation Learning.

Abbreviations

ID - Intrusion Detection

IDS - Intrusion Detection System

AI - Artificial Intelligence

ML - Machine Learning

DL - Deep Learning

FL - Federated Learning

LAN - Local Area Network

MAN - Metropolitan Area Network

WAN - Wide Area Network

PLC - Programmable Logic Controller

RTU - Remote Terminal Unit

HMI - Human Machine Interface

MAC - Media Access Control

IPS - Intrusion Prevention System

IDPS - Intrusion Detection and Prevention System

NIDS - Network Intrusion Detection System

NBA - Network Behavior Analysis

AD - Anomaly Detection

AC - Anomaly Classification

API - Application Programming Interface

TP - True Positive

TN - True Negative

FP - False Positive

FN - False Negative

FID - Fréchet Inception Distance

GAN - Generative Adversarial Network

RNN - Recurring Neural Network

ACC - Accuracy

TPR - True Positive Rate

FPR - False Positive Rate

F1 - F1-Score

ABOD - Angle-Based Outlier Detection

Iforest - Isolation Forest

PCA - Principal Component Analysis

MCD - Minimum Covariance Determinant

LOF - Local Outlier Factor

LDA - Linear Discriminant Analysis

NB - Naive Bayes

SVM - Support Vector Machine

SVM RBF - Radial Basis Function Support Vector Machine

MLP - Multilayer Perceptron

AdaBoost - Adaptive Boosting

DNN - Deep Neural Network

ANN - Artificial Neural Network

IoT - Internet of Things

TABLE OF CONTENTS

Page

ACKNOWLEDGMENT iii

ABSTRACT iv

ABBREVIATIONS v

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF EQUATIONS xvi

CHAPTER

1 Introduction 1
- 1.1 A short story on Intrusion Detection 1
- 1.2 Motivation 2
- 1.3 Thesis Structure 3

2 Network Elements 4
- 2.1 Computer Networks Categories 4
- 2.2 Industrial Networks 6
- 2.3 Network Traffic 7
  - 2.3.1 Common Protocols 7
  - 2.3.2 Industrial Protocols 9
    - 2.3.2.1 Modbus Protocol 10
  - 2.3.3 Network Flows 11
- 2.4 Industrial Operational Data 11
- 2.5 Malicious Activity 12
  - 2.5.1 Indicators 13

2.5.1.1 Network Traffic Indicators 13

2.5.1.2 Operational Data Traffic Indicators 13

2.5.2 Attack Taxonomy 14

2.5.3 Repercussions 15

3 Introduction To Intrusion Detection Methodology 16

3.1 Intrusion Detection Systems 16

3.1.1 Network Interaction 18

3.2 Intrusion Detection Methods 18

3.2.1 Signature Based Intrusion Detection 18

3.2.2 Anomaly Detection & Anomaly Classification 19

3.2.3 Stateful Protocol Analysis 19

3.2.4 Used in this Work 20

4 Deep Learning 21

4.1 Summary 21

4.2 Machine Learning 21

4.2.1 Intro To Machine Learning 22

4.2.2 Machine Learning Categories 22

4.3 Deep Learning & Intrusion Detection 23

4.4 Artificial Neural Networks 24

4.4.1 Architecture 24

4.4.1.1 Neuron 24

4.4.1.2 Multi-layer Networks 25

4.4.1.3 Activation Functions 26

4.4.1.4 Loss Functions 27

4.4.1.5 Forward Propagation 27

4.4.1.6 Back Propagation 27

4.4.1.7 Optimizers 28

4.4.2 Training & Testing 28

4.4.3 Metrics 30

4.4.3.1 TP/TN - FP/FN 30

4.4.3.2 Accuracy 31

4.4.3.3 Precision 31
4.4.3.4 Recall 31
4.4.3.5 F1-Score 32
4.4.3.6 FID Score 32
4.5 Federated Learning 33
4.6 Tools and Frameworks 34

5 ADA-CLA-GAN Design & Implementation 38
5.1 Design 38
5.1.1 Basis 38
5.1.1.1 Literature Review 38
5.1.1.2 The GAN Architecture 39
5.1.1.3 The Autoencoder Architecture 40
5.1.1.4 The AnoGan Architecture 40
5.1.2 ADA-GAN Design and Specifications 41
5.1.2.1 Input Layer 42
5.1.2.2 Generator-Decoder Module 43
5.1.2.3 Discriminator-Encoder Module 44
5.1.3 CLA-GAN Design and Specifications 45
5.1.3.1 Input Layer 47
5.1.3.2 Generator-Decoder Module 47
5.1.3.3 Discriminator-Encoder Module 49
5.2 Implementation 49
5.2.1 Development Environment 50
5.2.1.1 Language 50
5.2.1.2 Test Bed 51
5.2.2 ADA-GAN 51
5.2.2.1 Class Organization 51
5.2.2.2 Modulation 53
5.2.2.3 Data Preprocessing 54

5.2.2.4 ADA-GAN Initialization 54
5.2.2.5 ADA-GAN Training 55
5.2.2.6 ADA-GAN Testing 56
5.2.2.7 Produced Model 57
5.2.3 CLA-GAN 57
5.2.3.1 Class Organization 58
5.2.3.2 Data Preprocessing 60
5.2.3.3 CLA-GAN Initialization 60
5.2.3.4 CLA-GAN Training 60
5.2.3.5 CLA-GAN Testing 61
5.2.3.6 Produced Model 61

6 ADA-CLA-GAN Experimental Results 63
6.1 Evaluation 63
6.1.1 Dataset 63
6.1.1.1 Anomaly Detection Data 64
6.1.1.2 Anomaly Classification Data 65
6.1.2 Metrics Comparison 66
6.1.2.1 ADA-GAN Comparison 67
6.1.2.2 CLA-GAN Comparison 69
6.2 Experimental Configuration 73
6.2.1 ADA-GAN Configuration 73
6.2.1.1 Number of Epochs 73
6.2.1.2 Batch Size 73
6.2.2 CLA-GAN Configuration 73
6.2.2.1 Number of Epochs 73
6.2.2.2 Batch Size 74

7 Federated Learning Extension - Design & Implementation 75
7.1 Design 75
7.1.1 Basis 75

7.1.1.1 Orchestrator 76

7.1.1.2 Worker 77

7.1.2 Federated Testbed 77

7.1.3 Literature Review 78

7.2 Implementation 79

7.2.1 Federated Learning Convention 79

7.2.2 Pre-Federation Training 80

7.2.3 Dataset Federation 80

7.2.4 Federated Model Training 80

7.2.5 Results 81

8 Conclusion & Future Work 82

8.1 Conclusion 82

8.2 Future Work 84

LIST OF TABLES

6.1 Attack Classes 66

6.2 PDP01 Anomaly Detection Results 67

6.3 PDP02 Anomaly Detection Results 67

6.4 PDP03 Anomaly Detection Results 68

6.5 PDP04 Anomaly Detection Results 68

6.6 ADA-GAN Metric Statistics 69

6.7 PDP01 Anomaly Classification Results 70

6.8 PDP02 Anomaly Classification Results 70

6.9 PDP03 Anomaly Classification Results 71

6.10 PDP04 Anomaly Classification Results 71

6.11 CLA-GAN Metric Statistics 72

7.1 ADA-GAN Federated Results 81

LIST OF FIGURES

2.1 Local Area Network - (LAN) 5
2.2 Metropolitan Area Network - (MAN) 5
2.3 Wide Area Network - (WAN) 6
2.4 Industrial Network Architecture 7
2.5 OSI Network Model 9
2.6 Modbus payload architecture 11
2.7 Network Flow Model 12
3.1 Intrusion Detection System 17
3.2 Network Intrusion Detection System 18
4.1 Biological Neuron 25
4.2 Artificial Neuron 25
4.3 3-layered Perceptron 26
4.4 Forward Propagation 28
4.5 Backward Propagation 29
4.6 Anomaly Detection Neural Network - Loss History 30
4.7 Federated Learning Flow 33

4.8 Tensorflow Library 35

4.9 Keras API 35

4.10 Scikit-learn Library 36

4.11 Pytorch Library 36

4.12 NumPy Library 36

4.13 Pandas Library 37

4.14 Matplotlib Library 37

4.15 PySyft Library 37

5.1 ADA-GAN Architecture 42

5.2 Uniform Noise Distribution 43

5.3 Generator-Decoder Structure 44

5.4 Discriminator-Encoder Structure 45

5.5 CLA-GAN Architecture 46

5.6 Normal Noise Distribution 47

5.7 Generator-Decoder Structure 48

5.8 Discriminator-Encoder Structure 50

5.9 Python Language 51

5.10 Ubuntu OS 51

5.11 ADA-GAN Class 52

5.12 ADA-GAN Class Python 53

5.13 ADA-GAN Class Mounting 54

5.14 Generator Initialization 55

5.15 Discriminator Initialization 55
5.16 ADA-GAN Training 56
5.17 ADA-GAN Testing 57
5.18 CLA-GAN Class 59
5.19 CLA-GAN Class Python 59
5.20 Generator Initialization 60
5.21 Discriminator Initialization 60
5.22 CLA-GAN Training 61
5.23 CLA-GAN High-level Evaluation 62
6.1 Normal 64
6.2 Abnormal 64
6.3 PDP03 Readings 64
6.4 Normal 65
6.5 Abnormal 65
6.6 PDP04 Readings 65
6.7 ADA-GAN Accuracy Comparison 68
6.8 ADA-GAN F1-Score Comparison 68
6.9 ADA-GAN TPR Comparison 69
6.10 ADA-GAN FPR Comparison 69
6.11 CLA-GAN Accuracy Comparison 71
6.12 CLA-GAN F1-Score Comparison 71
6.13 CLA-GAN TPR Comparison 71

6.14 CLA-GAN FPR Comparison 71

6.15 Anomaly Detection F1-Score Curve - PDP04 74

7.1 Federated Flow 76

7.2 Federated Topology 78

7.3 Dataset Federation - Pytorch API 80

LIST OF EQUATIONS

4.1 General Accuracy 31

4.2 Classification Accuracy 31

4.3 Precision 31

4.4 Recall/Sensitivity 32

4.5 F1-Score 32

4.6 Fréchet Inception Distance 32

5.1 GAN Mathematical Model 40

5.2 Autoencoder Output Formula 40

5.3 Adversarial Loss 41

5.4 Tanh Equation 43

5.5 General Binanry Cross-Entropy Formula 44

5.6 Sigmoid Equation 45

5.7 Class Identification Formula 47

5.8 Categorical Crossentropy 48

Dedication

This thesis is dedicated to my mother and father who persevered through my fantastical daydreaming, my sister that kept a straight face through my unending whining, my friends without whom I would still be struggling with basic math and lastly my professors who taught me to strive for success even in the twilight of uncertainty.

Chapter One

Introduction

1.1 A short story on Intrusion Detection

Intrusion Detection (ID), as defined in [1], is the ceaseless effort to detect incoming attempts of intrusion in a system. In the world of computer science, Intrusion Detection Systems are specialized systems or services that work towards uncovering intrusive actions, by a third malicious party, with the goal of exploiting of infliction damage to a system, an individual or an organization.

The concept of IDS's was introduced by James P. Anderson, in 1980, in his paper "How to use accounting audit files to detect unauthorized access" some years after the USAF introduced the problem of digital information security and privacy concerns in 1972. The first rule based IDS was later developed in 1984-1986 by Dorothy Denning and Peter Neumann.

Today, the field of Intrusion Detection constitutes a rapidly advancing topic in the field of cybersecurity and many infrastructures, either personal, commercial or industrial, depend heavily on it. Using continuously evolving new technologies in almost every field of computer science, such as Data Science, Logistics, Machine Learning (ML), Deep Learning (DL), Software Defined Networking and many more, the field of Intrusion Detection is building an arsenal of defending against malicious attacks that undermine the normal function of digital services. Intrusion Detection Systems are described in depth in section 3.1: Intrusion

Detection Systems.

1.2 Motivation

In recent years, the abundance of information traded daily on both private and public networks, as well as the constant evolution of mechanisms for invading and intercepting this information, have made intrusion detection systems necessary to ensure the privacy of users' data. The techniques that malicious adversaries use to infiltrate and exploit private data and services are ever-evolving and harder to detect and prevent.

The use of machine learning, specifically neural networks, in intrusion detection systems is a field that has developed greatly in recent years due to its versatile method of producing results. However, the mechanism of their training and operation often raises questions about the privacy and concealment of the data required for this process. This problems tackles an emerging field of machine learning and decentralized ML and DL data transfer method, Federated Learning.

In this dissertation, an intrusion detection system using neural networks and the Federated Learning technique is investigated. This system aims to optimize the method detecting anomalies in an industrial network running the modbus protocol and subsequently classifying them while ensuring the privacy of users' data during the optimization of the intrusion detection deep learning model produced through non-centralized Learning. The objectives of this work are to i) develop a novel method of detecting and classifying anomalies in a network running the modbus protocol, ii) develop, train and evaluate a deep neural network using the aforementioned method, iii) extent the designed architecture to accommodate Federated Learning techniques and finally iv) test the produced system in a unified testing network topology.

1.3 Thesis Structure

From this point on, this work is organized as follows. Chapters 2-4 give an extensive background to the architectures, tools and methods used to produce this work. Specifically, Chapter 2 described the network elements, protocols and tools that contribute or are connected with this work, Chapter 3 defines the Intrusion Detection field and its characteristics, Chapter 4 gives an in depth description of the Deep Learning technology and methods. Chapter 5 defines the basic concept of this work and depicts the architectural design and principles. Moreover it illustrates the development of the mentioned system and each of its separate branches. Chapter 6 evaluates this work and produces its results. Additionally, Chapter 7 outlines the implemented extension of the proposed architecture to the Federated Learning Environment. Finally, Chapter 8 concludes this work and presents possible future extensions.

Chapter Two

Network Elements

Computer Networks form a fundamental pillar of today's communication and global interconnection. They contribute in the transfer of enormous amounts of information and it is that through them a lot of aspects of modern life have advanced so rapidly and successfully, as they offer multilateral communication of that information all around the world. Modern computer networks consist of a plethora of hardware and subsequently software elements that communicate with each other.

2.1 Computer Networks Categories

There are a lot of different categories of computer networks, the major and most important of which are:

- **Local Area Network (LAN):** These networks are typically small and encapsulate a relatively small number of devices. They are implemented in small topologies like homes, offices, schools, etc. and are defined by a small internal speed. These networks are mostly isolated from the wide web and contain personal devices and devices of interest. These networks are not restricted to wired connections only but can contain wireless elements also.

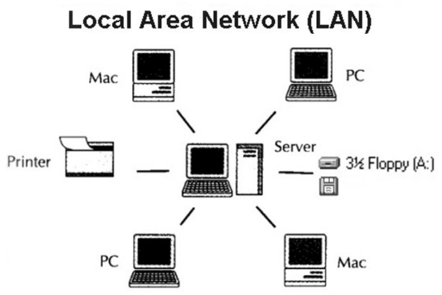


Figure 2.1 Local Area Network - (LAN) [2]

- **Metropolitan Area Network (MAN):** MAN's are bigger than LAN networks that connect smaller networks together. They are defined by a bigger speed and bandwidth and usually connect a amalgam of LAN networks in a specific area. MAN networks are usually wide node networks in a wide area like a part of a city.

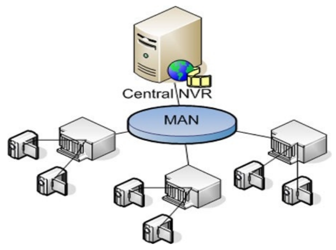


Figure 2.2 Metropolitan Area Network - (MAN) [3]

- **Wide Area Network (WAN):** WAN networks are larger than the aforementioned categories. They connect all other categories of networks over great distances that can cover from a town to a whole continent and more.

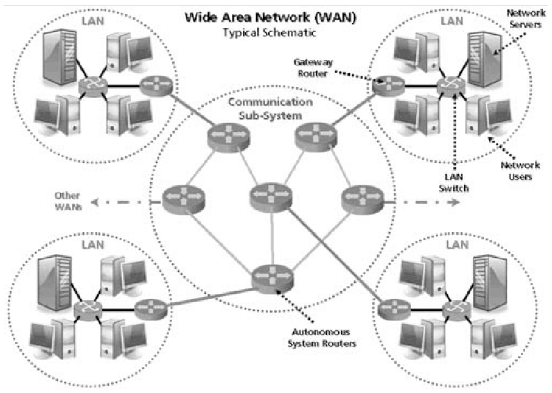


Figure 2.3 Wide Area Network - (WAN) [4]

2.2 Industrial Networks

There is a more strictly defined supercategory of LAN and WAN Networks that is the Industrial Network. Industrial Networks can span the distance of both LAN and WAN networks and can contain both categories in a unified network. As the name implies, they are designed for commercial use in industries and organizations. They differ from normal networks in the fact that most of the devices connected in the network are of industrial use, such as, industrial sensors, field devices, controllers, servers, data centers and so on, intended for industrial use and that they are designed to transfer in real time huge amounts of information and accommodate a big number of these devices over great distances. This means that the basic interactions within the network, but also the information transfer and the access to the network components are strictly defined and specific to their function in contrast to a personal network.

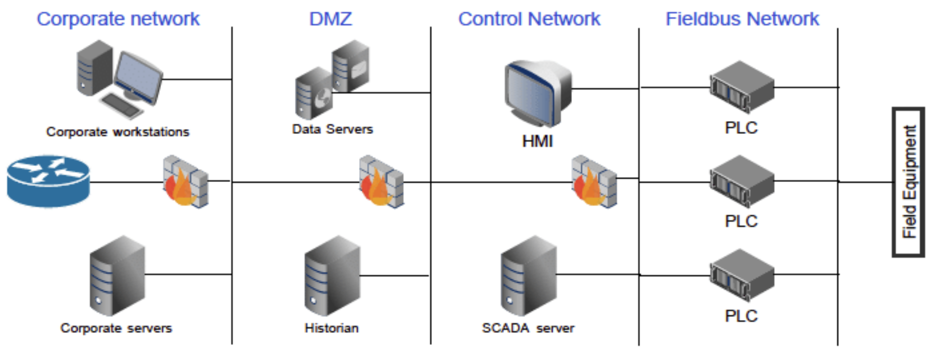


Figure 2.4 Industrial Network Architecture [5]

2.3 Network Traffic

Network Traffic is defined as the measure of the transferred information in a network in a given time. Specifically, the total number of exchanged packets of the devices in the network. The network traffic consists of packets of transmitted or received information of any given device in the network, utilizing a specific communication protocol in a given network stack layer.

2.3.1 Common Protocols

Network protocols constitute the basic communication interfaces in a computer network. There are a lot of network communication protocols used to transfer information in a network. Each protocol is communicated in a specific layer of the network communication architecture as can be seen in figure 2.5 which depicts the Open Systems Interconnection (OSI) network model. OSI is only one of the commonly used network models but there are more, each used in a different implementation scheme. Some of the common protocols used in most of the computer networks are:

- **Internet Protocol (IP):** It is the predominant network protocol used to facilitate other transport protocols on top of it. Its function is to interface and route information across networks.

- **Transmission Control Protocol (TCP):** TCP is a message transfer protocol used for short information exchanges in a network. It is the dominant transfer protocol used over networks and the world wide web. It is a connection oriented protocol running on top of the IP protocol and is located in the transport network stack layer.
- **User Datagram Protocol (UDP):** UDP is a transfer protocol located in the transport network layer. It is a connectionless protocol and thus unreliable and is responsible for transferring continuous bursts of data like realtime data, media data and so on.
- **Hypertext Transfer Protocol (HTTP):** HTTP is a generic information exchange protocol in the application layer. It is used by the majority of services allowing the independent construction of system with a collaborative information exchange baseline.
- **File Transfer Protocol (FTP):** FTP is a protocol running on top of TCP/IP to provide file transfer between system over the network.
- **Domain Name System (DNS):** DNS is the protocol responsible for translating and matching domains to ip addresses so that the exchanging information can be routed to the correct source.
- **Address Resolution Protocol (ARP):** ARP is a link layer protocol used for resolving internal network ip addresses to MAC addresses for information exchange in a network.
- **Simple Network Management Protocol (SNMP):** SNMP is an application layer protocol responsible for transferring management information on top of TCP/IP.
- **Internet Control Message Protocol (ICMP):** The ICMP is an internet layer protocol responsible for control information messages. Its role is to diagnose and inform about network communication issues and errors.

There are a lot more crucial network protocols that are not mentioned. The mentioned protocols are the most commonly interacted protocols for malicious intents and are mentioned as a preamble.

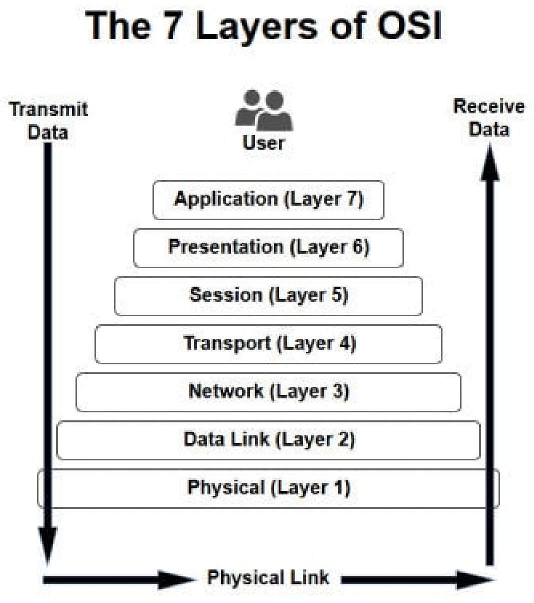


Figure 2.5 OSI Network Model [6]

2.3.2 Industrial Protocols

In Industrial Networks, except the common network protocols, there is also a wide range of specialized industrial protocols that target and orchestrate information packing and routing between industrial devices and service that function outside the scope of normal data communication. Explicitly, they handle structured information payloads that refer to specific processed and organization patterns, handling realtime data transferring, event notifying, error posting and more. Even though there are protocols running in different network layers, with the evolution of networks, most of these protocols run on top of the application network layer. Some examples of industrial protocols are:

- Modbus

- DNP3
- RS-232
- Backnet
- IEC 104

2.3.2.1 Modbus Protocol

Modbus [7] is an open protocol, widely used in industrial applications. It is a simple protocol, developed to be used with PLCs to control industrial network entities over serial communication and later RTUs developing also a Transmission Control Protocol (TCP) wrapper scheme to be used over modern networks. In this work the ModbusTCP module is utilized.

The basic Modbus entities in a network are i) Modbus Clients, ii) Modbus Servers and iii) Modbus Slaves. The Client is a remote query terminal, such as an Human-Machine Interface (HMI), requesting information from the Modbus servers and sending control information to them. Servers usually represent either PLC or RTU controllers in the network. Those controllers supervise Modbus Slave entities, such as Acquisition Blocks, that oversee the field devices. Each Server can have multiple Slaves with unique slave IDs attached to them. In the Modbus protocol the data are stored in four tables in each device. Each table corresponds to the Discrete inputs and outputs (COILS) and Numerical inputs and outputs (REGISTERS) respectively, in Modbus memory addresses.For the request and responses specific Function Codes (FC) serve to signal the devices about the payload of the incoming package. The most common function codes are a) Read Coil Status (FC01), b) Read Input Status (FC02), c) Read Holding Registers (FC03), d) Read Input Registers (FC04), f) Force Single Coil (FC05), g) Preset Single Register (FC06), h)Force Multiple Coils (FC15), i) Preset Multiple Registers (FC16).

A Modbus TCP packet consist of the TCP protocol wrapper and the Modbus packet.The Modbus packet encapsulates information about a) the Slave ID, b) the Function Code, c) the

Starting Address of the address frame requested for information or change, d) the number of addresses following the Starting Address.

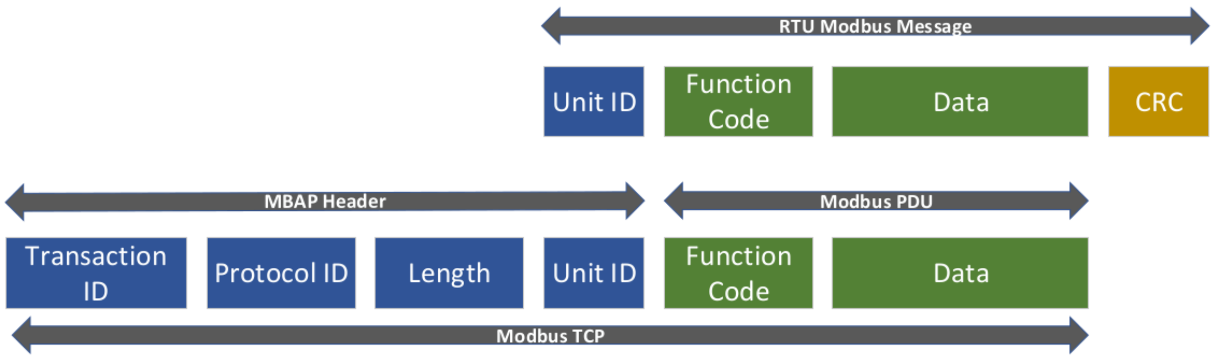


Figure 2.6 Modbus payload architecture

2.3.3 Network Flows

As network flows are defined a number of aggregated computer network traffic, packet transmissions with a sum of mutual characteristics [8]. Usually, these mutual elements are a) the Source IP, b) the Destination IP, c) the Source Port and d) the Destination Port. Depending on the level of abstraction that the flow is extracted from, a flow can be classified as one-way or as two-way flow [9]. Moreover, individual network flows are described, usually, by the same protocol. As will be seen in section 6.1.1.2 - Anomaly Classification Data, Modbus network flows from industrial applications are used to provide the data needs for this work. Figure 2.7, shows the structure of a network flow model.

2.4 Industrial Operational Data

One category of the network elements of Industrial Networks contains the field equipment, as shown in 2.4. The field devices represent serve as the edges of an industrial establishment and are either passive monitoring devices (e.g., sensors, meters, etc) or functional devices (e.g., Actuators). Commonly, the field equipment measures the performance of its designed task.

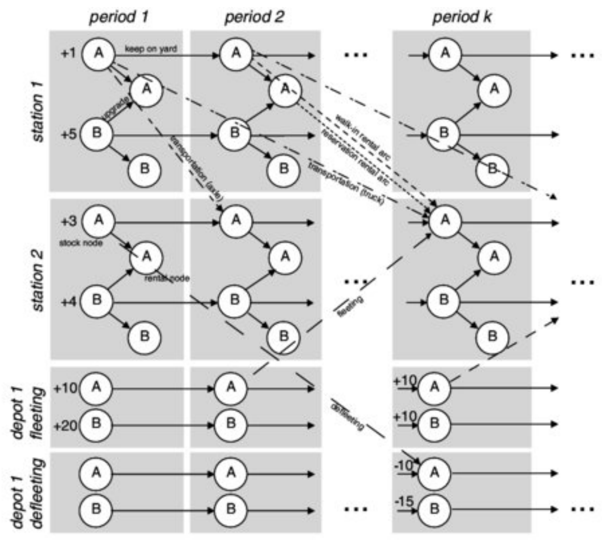


Figure 2.7 Network Flow Model [10]

Especially in the case of monitoring devices, they return a large number of task related data. In Industrial facilities these data can be voltage measurements, water/liquid levels, pressure and other invaluable information for the auditing of the normal operation of the said facility. This information circulates through the network, using some of the protocols described above back to the main facility systems to be checked and, in case of an abnormality, for a correct action to be taken accordingly. These data belong to the time-series category, meaning that they are time-dependant. In contrast with flow-based data that rely on the statistical information offered by a given frame of information, the time-series data offer time correlated arrangements that can be used for identifying patterns in the data.

2.5 Malicious Activity

In a computer network, malicious activity is every process that takes place in that network contrary to its normal and established function, caused by a third party. Commonly, and depending to the network, a third party might try to exploit information and processes withing the structure. These actions can have catastrophic consequences both to the network and the person or organization that holds it. The same can be said about the Industrial

Operational Data. A third party, using the network as a channel, can target the operation of a facility's device in order to cause malevolent abnormalities.

2.5.1 Indicators

2.5.1.1 Network Traffic Indicators

Malicious network activity comes in all shapes and forms. Since computer network architecture is multilayered, there are different attacks and approaches to infiltrate and exploit different parts of a network [11]. From individual packets to aggregated flows, the indicators may vary. Another factor that plays a part for the indication of abnormal activity in a network is the perspective of the third party's view of the network, i.e., from within the network, from affiliate networks or outside of the network. Nevertheless, the most common indicators for anomalous activity are actions that contrast the normal activity of the network. These, actions may be an atypical communication from unrelated devices, a weirdly elevated number of transmitted or received packets in total or from a specific protocol, authorization access of key elements and more. Section 3.2 describes frequently used intrusion detection methods based on key indicators.

2.5.1.2 Operational Data Traffic Indicators

When talking about malicious activity indicators in the data send by the field devices it is meant the abnormal readings that a monitoring device obtains. If there is a surge in the activity of the equipment that contrasts its normal behaviour that is a serious indication that something is wrong. Be that it is caused by a malfunction in the equipment or, somehow, a third party found its way into the network and meddled with the function of the devices, the system should be checked to avoid possibly catastrophic results. Anomaly Detection in Operational Data is often used as a mean to indirectly detect Intrusions in an Industrial Network.

2.5.2 Attack Taxonomy

There is a big variety of cyber attacks in different fields of computer science. These attacks are the means to infiltrate and in extend exploit a system or a service. In networking most attacks have a goal of getting access to the network and intercept information withing that network or manipulate a process of a single or a group of network elements. Depending on their target, method or outcome these attacks are separated in some major attack categories [12] and [13].

- **Viruses:** Malicious programs that intercept or damage information and interfere with services and propagate through infected files
- **Worms:** Self-propagating malware that spreads through a network without the need of infected files
- **Denial of service:** Process through which a target service becomes unavailable due to resource depletion
- **Network attacks:** Network implemented attack that exploit resources within the network
- **Physical attacks:** Attacks that damage physical hardware
- **Password attacks:** Process aiming to uncover a password and obtain access to information
- **Information gathering attacks:** Indirect process through which an attacker gathers information used to implement other attacks or directly exploit individuals

Though Intrusion Detection is implemented in almost all of the attack classes mentioned, in this work the attack category of interest is the Network Attack category. Network attacks can be branched into sub categories, describing specific methodologies.

2.5.3 Repercussions

With the rapid advance of the digital world and the global interconnection through the web as well as the increasing dependency on virtual means of controlling processes that have an impact on the physical world, the real world implications of a possible malicious exploitation become increasingly worrisome. Especially, in Critical Infrastructures (ICs) like electrical grids or gas pipelines, constituted by Industrial networks, that depend heavily on the precise and timely transfer of control and data of interest information, the consequences of a successful cyber attack can bring catastrophic results. An example of these repercussions took place in Western Ukraine on December 23rd, 2015 when the electrical network was disabled leaving a big number of commercial areas and critical services without power [14].

Chapter Three

Introduction To Intrusion Detection Methodology

As mentioned before, the rapid advancements in modern technological means and methodologies, as well as the volatile evolution of malevolent efforts to intrude and exploit these means make the need for state-of-the-art security measures in order to fortify the continuation of the privacy and integrity of digital services ever more essential. As such, the field of Intrusion Detection takes over the role of timely identifying abnormal phenomenae in order to consequently prevent and secure the target systems. Thus, ID is defined as the process of monitoring and identifying events in computer systems and networks and analyzing them in an effort to identify anomalous signs indicating possible intrusions [15].

3.1 Intrusion Detection Systems

The work in [16], defines an IDS as a software or hardware system that, using the methodologies defined in the field of ID, automates the process of monitoring and analyzing events in a computer system or network towards identifying a possible intrusion. Usually, IDSs extent or rather incorporate the function of Intrusion Prevention Systems (IPSs) and are called Intrusion Detection and Prevention Systems (IDPSs). There are four main IDPS categories.

These are:

- **Network-Based:** They monitor network traffic and network elements of interest in a variety of network topologies and deployments
- **Host-Based:** As the name implies they monitor single hosts of interest, auditing behaviours of key system elements, processes and services within that host. Host-Based systems are the most common category of IDS for securing targeted systems
- **Network Behavior Analysis (NBA):** Specified for monitoring network behaviours and policy violations by analyzing flows or key network-specific features
- **Wireless:** Specialized in analyzing wireless network traffic and protocols specifically used in wireless communication

It is important to note that in this work an IDS is not either designed or implemented. Rather than the system, intrusion detection methodologies, as described below, used in an IDS are developed. Nevertheless, it is deemed essential to delineate the structure and workings of IDSs since they constitute the elements that adapt and utilize the proposed techniques.

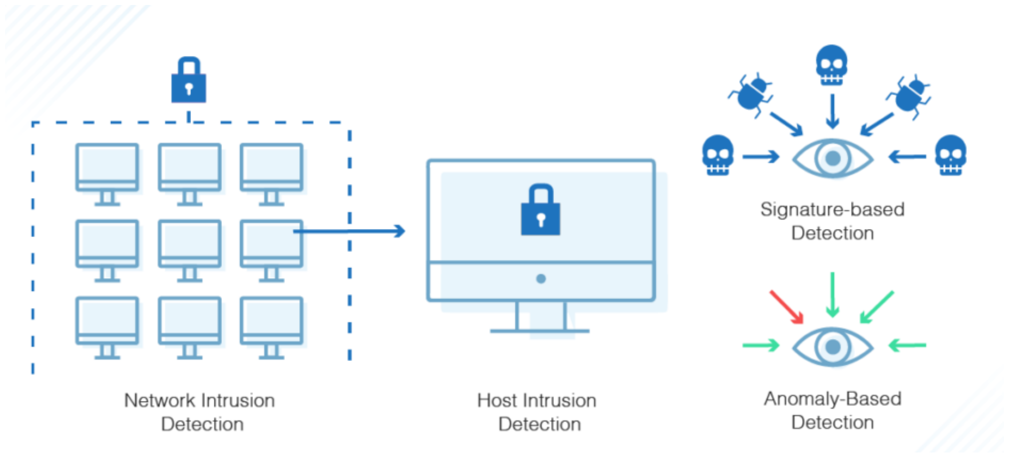


Figure 3.1 Intrusion Detection System [17]

3.1.1 Network Interaction

IDSs, most of the time are used in parallel with the network in which they are installed. This happens due to the fact that some IDSs don't or can't analyze all of the observed traffic of the network and if they can it is logical that they create a form of a bottleneck in that network. Thus, traffic mirroring and packet sampling techniques are often used to provide the necessary data to the detection system.

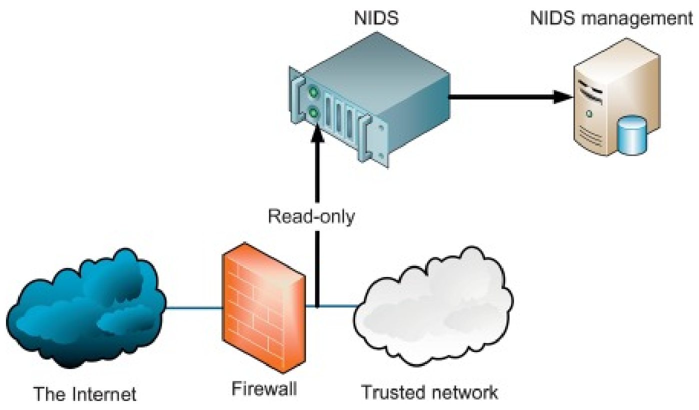


Figure 3.2 Network Intrusion Detection System [18]

3.2 Intrusion Detection Methods

The authors in [19] define the major ID methodologies that are used in modern IDPS systems. There are three significant ID methods used to identify impending or in motion attacks in a system or network, namely, a)Signature Based Intrusion Detection, b) Anomaly Detection & Anomaly Classification and c) Stateful Protocol Analysis. These methods are described below.

3.2.1 Signature Based Intrusion Detection

Signature Based Intrusion Detection is a method of recognizing known and documenting attack patterns used by malicious elements. This is the simplest of the ID methodologies and relies heavily on already documented patterns and deviations. Due to this fact it is ineffective in recognizing novel or zero-day attacks.

3.2.2 Anomaly Detection & Anomaly Classification

Anomaly Detection (AD) is the process of observing the function of a system and detecting atypical to that system behavior. Anomaly detection systems work by registering a system's normal behaviour definitions and then using that definition to compare ongoing operations. If the current behaviour does not conform with the normal behaviour definition then it is considered an anomaly and is treated as such. Anomaly based detection is a very powerful tool that can detect novel and previously unseen attacks. One of the most major step backs of this method, though, is that if the normal definition is contaminated with malicious samples then it becomes vulnerable to that category of malicious attacks.

Even though Anomaly Classification (AC) isn't included in the ID techniques, it can be said that it is a different method of anomaly detection. It is part of the classification problems category in the field of machine learning as the name reveals. It works by, in contrast with Anomaly Detection, learning to distinguish between the different classes of intrusions by assimilating profiles of all the provided attacks. This means that this process gives the ability to not only detect an attack but also clarify it in a specific category. It works in a similar manner as the Anomaly Detection methodology but in a more strictly defined frame. Due to this fact, this method is vulnerable to unknown definitions and usually this results in wrongly classifying a novel attack to a defined category, even if this category is the "Normal" one.

3.2.3 Stateful Protocol Analysis

Stateful Protocol Analysis is a technique that utilizes predefined profiles, usually vendor based in contrast to Anomaly based methods that are host or network-specific, describing benign activities and behaviours. It compares these predefined profiles with currently observed behaviours and detects those who significantly deviate from the generally accepted model or protocol correlated with a specific profile. It has the ability to distinguish between levels of authorization, defined in the corresponding profile, and thus is effective in universally used systems or services. The downside of this method is that, if the standard describing the profile of a certain behaviour or protocol is not correctly defined or complete, there is a high risk of bypassing its alertness.

3.2.4 Used in this Work

In this work, both an Anomaly Detection and Anomaly Classification methods for intrusion detection, primarily in Industrial Networks and Critical Infrastructures, are designed, researched and developed. These implementations rely on the Deep Learning and Federated Learning technologies. The reason behind the choice of these two methodologies is that they offer a flexible way to recognize both new and documented attacks using novel machine learning approaches that do not rely on predefined standards.

Chapter Four

Deep Learning

4.1 Summary

Deep Learning (DL), in the science of Artificial Intelligence (AI), is a sub-category of Machine Learning that specializes in the accumulation of big data into models that target distinct implementations. Specifically, it solves the problem of computer knowledge representation. It uses complex structures to represent data into simpler representations [20]. DL takes its principles from the workings of the human brain, mimicking the way that neurons handle information. Structures named artificial neural networks learn from large amounts of data, aiming to solve complex problems that normal ML implementations can't. In this chapter, the technology of Deep Learning is described in-depth, since it constitutes the main tool and focus of this work.

4.2 Machine Learning

Machine Learning is an application of the algorithms and techniques of the science of AI in order to produce systems capable of learning and adapting in an independent manner, without being explicitly programmed to do so and by providing them with data in the form of abstract observations.

4.2.1 Intro To Machine Learning

The field of Machine Learning started its upbringing in the 1950s by Arthur Samuel of IBM. Until the change of the century a big portion of the foundations had been laid in the fields of computer science and mathematics to support its life-changing ideals. It wasn't though until the bloom of the data revolution, magnifying the transfer and handling of huge amounts of data, that ML started to be applied in complex real-world scenarios and get on its way to commercial use.

As mentioned, ML is the science of creating self-sufficient and independent systems that are able to learn from experience and use this experience to an end. Specifically, depending on the use case of the applied ML algorithm, using mathematical tools like functions, models and transformations, the ML systems produce a mathematical model related to the input data. This model is then used to solve a targeted problem. These problems can be categorized into three separate cases as described in the next section.

4.2.2 Machine Learning Categories

Depending on the problem, the problem parameters, the way the given data are obtained but also on the nature of the said data, ML problems can be separated into three major categories. Based on these categories, a different and unique implementation of the ML theory is applied to solve the given problem. These categories mainly describe the relation that the model will have with the input data and are called i) Supervised Learning, ii) Semi-supervised Learning and finally iii) Unsupervised Learning.

- **Supervised Learning:** In Supervised Learning a mathematical model is produced that correlates the possible input with a series of labeled outcomes. It is established as the most common ML category based on its practicality in modern data-oriented problems. This category encapsulates the practices of data classification and regression

- **Semi-supervised Learning:** Semi-supervised Learning is a byproduct of Supervised Learning. It is used to correlate the problem's input with a number of predefined outputs, but in this case there may be a number of possible outcomes that are not previously defined in the known outputs

- **Unsupervised Learning:** This type of learning is oriented in recognizing and grouping input data that are not labeled. It is used in problems that have the outcome is not categorized and as such there is a need for clustering the problem's variables into groups to produce a solution

This work delves into two of the defined categories. AD is a Semi-supervised Learning problem since it is tasked in recognizing abnormal samples in a dataset while the normal samples have been defined and are used to produce the given model. AC is a Supervised Learning problem since it corresponds to a given input with a given output.

4.3 Deep Learning & Intrusion Detection

Lately, and with the advancements in the DL technology, a big interest has been observed in using the DL methodologies in the field of Intrusion Detection. The goal of using DL in Intrusion Detection is to overcome the challenges of designing and developing an efficient Network Intrusion Detection System (NIDS) [21], but also to tackle the main problems that ML applications in the field bring, with the versatile nature of Deep Learning Systems [22]. Specifically, in ID the branch of Deep Learning is utilized to overcome the need to discretized and in sequence specify the features on which the learning will occur. Utilizing its complex and powerful mathematical basis, the DL implementation helps in avoiding the clustering of features but rather uses their full extent to learn the best possible correlations of the given data so that it may recognize and in extent classify intrusions correctly. To that end, modern processes have been created and common DL methodologies are used to supplement the support of Intrusion Detection in modern IDSs. Some of these methodologies are thoroughly used as a base to construct the DL models discussed in this thesis.

4.4 Artificial Neural Networks

Artificial Neural Networks (ANN) are mathematical-based, connection-oriented systems that try to imitate the workings of biological neurons [23]. They work by building a numeric adaptation of a given input learning from experience, trying to imitate the process of natural knowledge acquisition. By implementing a set of interconnected layers of neurons or units, each containing given weights on which the new knowledge is aggregated on the previous one, they learn the relation of the given input by the means of mathematical processes specifically created to help correlate the relationships of data. Assisting in this process, every network contains a number of hyperparameters, defines as core node configurations that have a big impact in a Neural Network's aspect. Examples of hyperparameters are the learning rate of a network, the neuron bias and others.

4.4.1 Architecture

4.4.1.1 Neuron

A neuron is a unit mathematical function that takes one or multiple inputs, multiplies them with a set of weights and outputs their aggregation through a given non-linear activation function. In every aggregation there is also provided a bias value for each neuron, that helps with the accumulation of the data. The simplest and most fundamental form of a neural network is a single Perceptron network, created by Frank Rosenblatt in 1958, being a one neuron network able to learn. Figure 4.2 depicts the structure of an artificial neuron, while figure 4.1 shows a biological neuron.

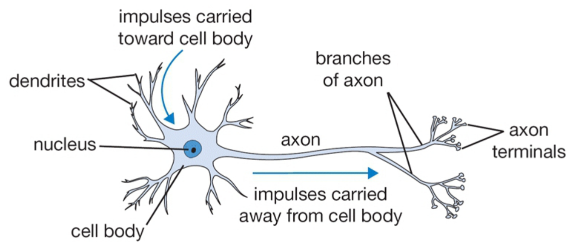

impulses carried toward cell body
branches of axon
dendrites
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Figure 4.1 Biological Neuron [24]

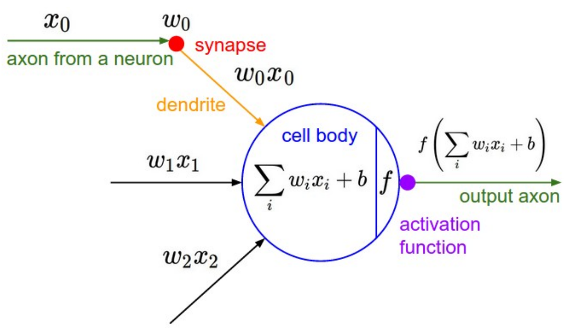


Figure 4.2 Artificial Neuron [24]

4.4.1.2 Multi-layer Networks

Even though Rosenblatt's single perceptron network is a functional example of a system that successfully accumulates data it is not applicable in real world deep learning problems, except in the case of binary classification of small problems (Support Vector Machine - SVM algorithm). Modern neural networks are composed of multiple layers of interconnected neurons, called hidden layers. By increasing the number of connected neurons an ANN is able to accumulate a big amount of data and achieves its intended purpose with good results. As such, ANNs are considered acyclic graph structures. Figure 4.3 shows a simple 3-layered network containing the input layer, 2 hidden connected layers and the output layer. There are different kinds of layers, the most common of which are the fully connected or Dense layers. In a network there is no need for all of the layers to be connected with each other or have the same number of neurons. Also, an ANN can have multiple inputs and multiple outputs. The structure of the network depends on the use case and the architectural axes that it follows.

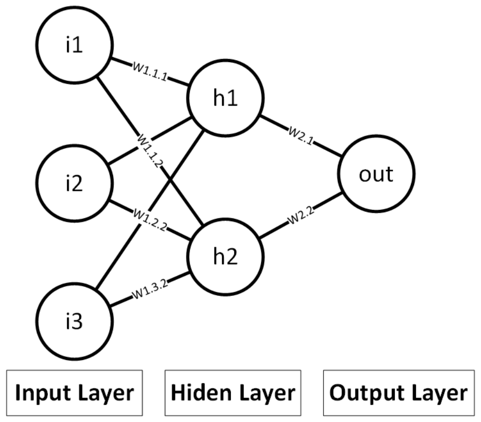


Figure 4.3 3-layered Perceptron

In the various frameworks that support deep learning there is a variety of predefined layers that are designed to have a distinct effect on the training process of a network. These layers emerge from the research done in the field of applied mathematics and artificial intelligence and can be found in the corresponding literature. This work utilized the said layers as a means to achieve a better outcome.

4.4.1.3 Activation Functions

Activation functions are mathematical tools, also called non-linearities, that affect the output of a neuron. For any given input, after the computation of the weighted sum in a neuron, an activation function is the deciding factor based on which the neuron "fire" or activates, or not. Since given that a computation in a neuron can have variable results in a wide range between $-\infty$ to ∞, it needs a way to know if the output is in a certain range to let it pass. In all, an activation function work on the output of the neuron to help decide the activation of the specific neuron. Below is a list of activation functions used in this work.

- Sigmoid
- Tanh
- Softmax

The usage of these functions is explained in the following sections.

4.4.1.4 Loss Functions

A Loss function can be described as the computed error between two values under comparison. In ML and DL, the loss function measures the deviation of the predicted values of a structure from the real given data, through a mathematical relation. Neural Networks use these functions to help optimize the learning process and produce a better output, but also to monitor the learning curve of a given network. Examples of Loss functions also used in this work are the ReLU and LeakyReLU functions.

4.4.1.5 Forward Propagation

The Forward Propagation or Forward Pass is the process through which a Neural Network accumulates the inputted data in a single epoch or iteration and produces and output. The process dictated that the input is pipelined through the different layers of the network, each layer accepting its input, processing and passing the output to the next, as they are connected. During the Forward Pass, the data flow in the forward direction but not in the reverse, as it would produce a cyclic loop. After the training of the network this process is used to produce the outcome of the network. Figure 4.4 shows the Forward Propagation process.

4.4.1.6 Back Propagation

Back Propagation is the process through which the fine-tuning of the network is realized. By feeding the computed gradient of the functions used in the Forward pass in a chain rule, the weights are updated in a way that produces a better loss score. This process ensures the better learning of the network. The Back Propagation phase can be seen in figure 4.5.

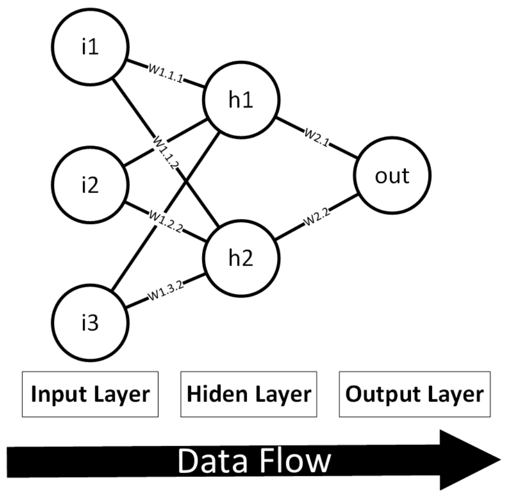


Figure 4.4 Forward Propagation

4.4.1.7 Optimizers

Bach Propagation is an essential process for the optimization of the learning process of a Neural Network and its generalization. This means that a method to procure this optimization has to be developed and establish. Due to the fact, though, that the range of problems that Deep Learning tries to solve is only constrained by the accompanying data, there is not one global, catholic optimization method. For each different problem and data, a different optimization method should be applied. Thus, Optimizers offer distinct optimization methods, developed to adhere to a variety of problems. There are a lot of Optimizers, some examples of which are a) Stochastic Gradient Descent [25], b) Adam [26], c) RMSProp [27], d)Adadelta [28] and more. This work relies heavily on optimizers to train the developed networks in an optimal way.

4.4.2 Training & Testing

As mentioned before, to prepare an Artificial Neural Network and optimize it to produce the desired results, it has to go through an accumulative process to learn the input data. This process is call Training and it helps the network to learn from experience and generalize the information it is fed. The training process is an essential part of Deep Learning. It is

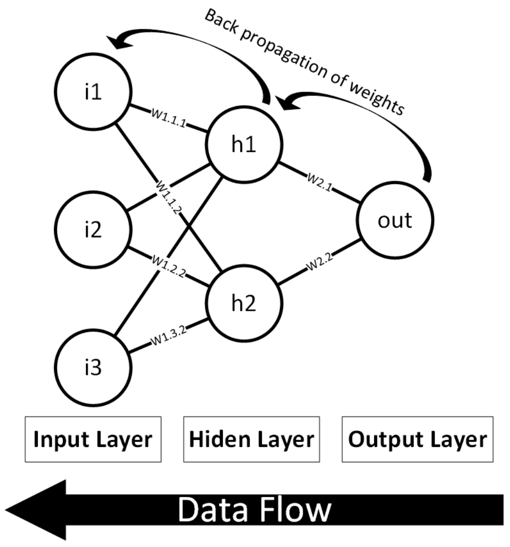


Figure 4.5 Backward Propagation

comprised of a certain number of N iterations called epochs. In each epoch the network takes an input of a portion of the data called a Batch that consists of K samples of the input, usually selected randomly. Both the number of epochs and the batch sized are chosen through an experimental process to find the optimal solution for the specific network. There are some predefined methods for selecting these quantities, i.e. the number constitutes a power of 2 and increments as such (2^4, 2^8, etc) but it still depends on the variables of the problem. The training process encapsulates both the Forward Pass and Back Propagation steps in each iteration. Through the monitoring of the loss scores, computed in every epoch and by visualizing them, the learning process can be supervised and audited. an example of a Training's loss history is given in figure 4.6.

Figure 4.6 depicts the exponential decrease in the loss between the data predicted by the Neural Network and the real data while the epochs increase.

The Testing phase usually occurs either between training sessions or in the end of the training of a Neural Network. In this stage the network is tested in a set of data that was not part of the training, to evaluate the overall performance of the network against the given problem. Since the data were not part of the training procedure, the output of the testing

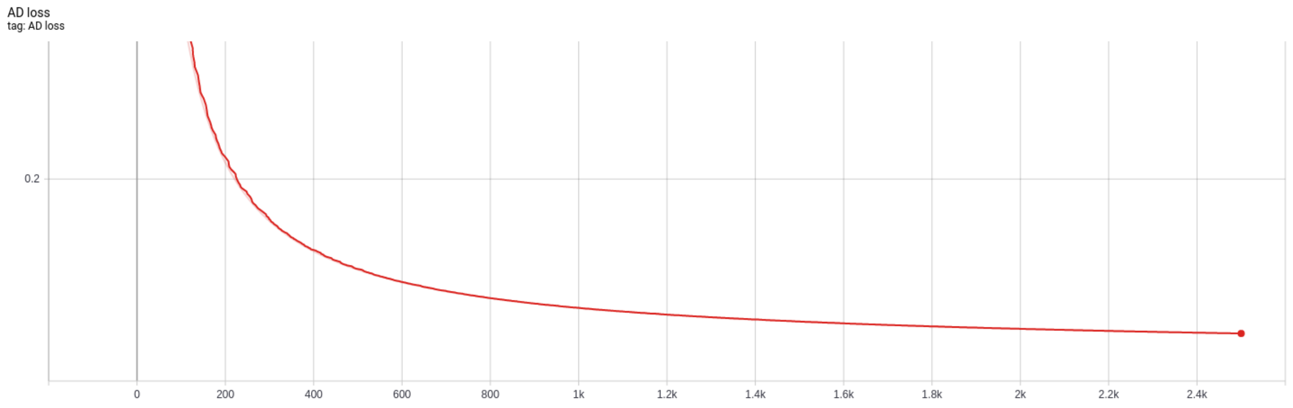


Figure 4.6 Anomaly Detection Neural Network - Loss History

gives a representative picture of the network's capability to produce the desired results. Depending on the problem category, i.e. Classification, which is a Supervised Learning problem, a distinct testing method is followed, accompanied by the corresponding data. During the Testing phase, a set of tools are used to measure the performance of the Neural Network. These tools are called Metrics and are explained in the following section.

4.4.3 Metrics

In order to evaluate a Neural Network, a series of quantitative metrics are utilized to quantify and compare the produced results. These metrics are commonly mathematical tools that measure the performance of a given network by comparing information around its intended purpose. These tools give meaning to the Accuracy, Precision, Sensitivity, Specificity and other aspects of the network and help to recognize penitential flaws, diagnose and, in extend, optimize that network. The most basic metrics that are also used in this work are explained in the following sub-sections.

4.4.3.1 TP/TN - FP/FN

One of the most fundamental and important metrics, when it comes to Deep Learning, and Machine Learning in general, is the TP/TN - FP/FN quantification. The True Positive

(TP), True Negative (TN), False Positive (FP) and False Negative (FN) represent the four basic elements that the performance of a system like the Neural Network that tries to predict the meaning of the input information, can be described in. They are leveraged to present the number of rightly or wrongly predicted (and labeled) samples that the network outputs. A confusion matrix of size $[N \text{ x } M|$ can be abstracted to these four basic elements. Furthermore, a lot of more complex metrics use these four elements to describe more complicated and representative measures.

4.4.3.2 Accuracy

Accuracy or Classification Accuracy is the quantitative metric describing the ratio of correctly predicted samples to the total number of input samples [29]. The generalized expression for calculating the Accuracy score is given by (4.1).

$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{4.1}$$

For classification problems the formula is (4.2).

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.2}$$

4.4.3.3 Precision

Precision describes the ratio between the correctly classified sample against the number of classified samples in that specific category [29]. The formula is depicted in equation (4.3).

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

4.4.3.4 Recall

Recall or Sensitivity is the metric that defines the ration between the correctly classified positive sample against all the samples that are in actuality positive [29]. (4.4) shows the expression through which we can calculate the Recall score.

$$Recall = \frac{TP}{TP + FN} \tag{4.4}$$

4.4.3.5 F1-Score

the F1-Score describes the weighted average of the precision and recall of the produced results. This measure indicated how many samples were correctly predicted but also how accurate it is, i.e. the amount of missed samples. Its range is confined in $[0, 1]$. This metric is one of the primary indicators that the network is robust and produces good results. Its formula is described in (4.5).

$$F1\text{-}Score = 2\frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{4.5}$$

4.4.3.6 FID Score

The Fréchet Inception Distance (FID score)[30] [31], is a high-level quantitative metric that reveals the similarity between two datasets of images. It is a useful tool utilized to compare the output of a specific category of Neural Networks called Generative Adversarial Networks (GANs). Since in this work the GAN architecture is used to produce the described networks, the FID score will be used to measure the performance of the developed Neural Networks. The FID score, equation (4.6), is derived by adding the sum squared difference of the means of the real data vector μ_r and the predicted data vector μ_p and adding the double of the squared root of the covariance matrices of the aforementioned vectors Σ_r and Σ_p, respectively.

$$FID = ||\mu_r - \mu_p||^2 + tr(\Sigma_r + \Sigma_p - 2\sqrt{(\Sigma_r \cdot \Sigma_p)}) \tag{4.6}$$

4.5 Federated Learning

As the demand for more sophisticated Machine Learning algorithms became a demand but also the size of the data began to expand in size, arose the need for a system that can handle big models that require monumental amounts of data across the cloud. Federated Learning is a stochastic distributed learning and privacy-preserving method that undertakes the distribution, orchestration, learning and aggregation of Deep Learning model across a big corpus of devices or edge nodes in the cloud [32] [33]. It works by stochastically disbursing a central Deep Learning model over a given corpus of devices. When received by the edge nodes, the model is trained locally on-device with the data collected by the given device and then, either the model or a targeted model update, is collected by the central system. Consequently, a technique called Federated Averaging [34], which is defined as the aggregation of the edge-computed weights with the central model, takes place. Figure, 4.7 presents the Federated Learning flow architecture as defined by the Google Research team.

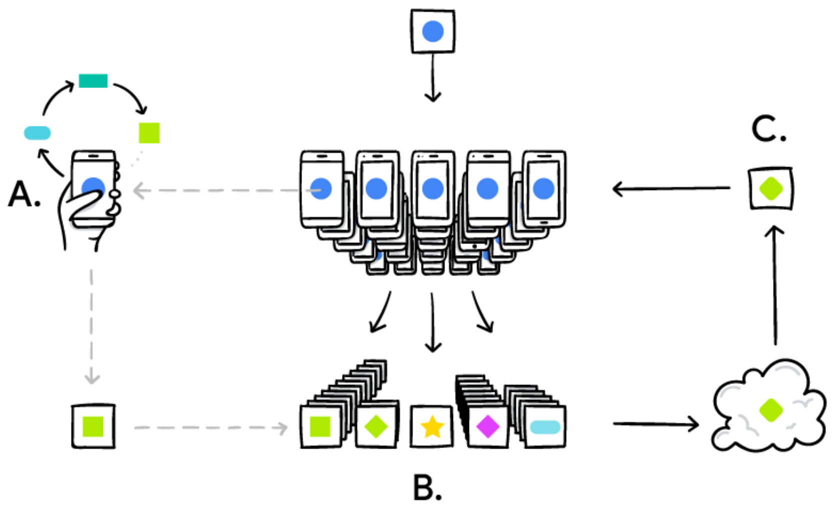


Figure 4.7 Federated Learning Flow [34]

Except the decentralized learning, Federated Learning offers yet another powerful advantage. By distributing and training the models locally on the edge devices there is no longer the need to transfer the datasets to the central system. Since data privacy has been a major concern from when the internet was engineered and put into commercial use, this implementation actively comforts the concerns for interception by just transferring the models to be trained. Moreover, a wide range of implementations have been proposed for securing the Deep Learning model transit and handling to further fortify the process. Finally, without the need for large data communication, the possible ramifications of network bottleneck and slow data transfer are avoided. These characteristics make the decentralized Federated Learning method a powerful tool for Machine and Deep Learning cloud interconnection.

4.6 Tools and Frameworks

To realize the proposed work, a certain set of tools must be leveraged. There is a big variety of tools available for Deep Learning as well as Data Manipulation and Visualization, each with its own perspective to the implementation. This work relies heavily on such frameworks to facilitate the base of the concept and the realization of the work described in each part of this thesis. The following list describes the main Machine and Deep Learning specific and supporting tools used in this work.

- **Tensorflow:** Tensorflow is an open-source programming symbolic math framework containing a range of libraries for dala flow and differentiable programming tasks. It is logged under the *Apache License 2.0* License. The Framework is Python friendly, the core programming language primarily used in this work. It also supports GPU acceleration for heavy computations across clusters with CUDA support. It is used for Machine Learning in the field of Artificial Neural Networks.

TensorFlow

Figure 4.8 Tensorflow Library [35]

- **Keras:** Keras is an open-source API that focuses on Deep Learning. It is registered under the *MIT* License. It is able to run on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. It offers a large number of tools and building elements to research, construct and test Artificial Neural Networks and is designed to be modular, user friendly, easy to integrate and extend. Most of the modern Deep Learning applications in Python are using Keras with a Tensorflow backend to establish their designs.

K Keras

Figure 4.9 Keras API [36]

- **Scikit-learn:** Scikit-learn is a free Machine Learning library, featuring a plethora of tools for designing, implementing and testing ML applications. It supports a number of programming languages, namely, Python, Cython, C, C++ and is registered under the *New BSD License* License. It offers a variety of ready to deploy Machine Learning algorithms for Classification, Regression, Clustering and others. It also supports tools for measuring the performance of these algorithms like the metrics described in the previous sub-section.

- **Pytorch:** Pytorch is an open-source programming library, extending the Torch library. Developed by Facebook's AI Research lab, cointains multiple tools for the development

Figure 4.10 Scikit-learn Library [37]

of Artificial Intelligence related applications like natural laguage processing, computer reading and more. It is licensed under the *BSD* License. Moreover, it is developed for using with Python but also supports GPU acceleration for heavy computations with CUDA and can be used with the C++ programming language.

Figure 4.11 Pytorch Library [38]

- **Numpy:** Numpy is a Python mathematical library oriented in data handling in multi-dimensional arrays and matrices and supports an array of high-level mathematical functions for operating on these data . It is defined under the *BSD* Licence.

Figure 4.12 NumPy Library [39]

- **Pandas:** Pandas is a high-level data manipulation library, commonly used in Data Science. It offers a modular, high performance and easy to use data handling and data analysis tools. In Machine Learning it is used to analyze, transform and format the data fed into the ML and DL algorithms.

Figure 4.13 Pandas Library [40]

- **Matplotlib:** Matplotlib is an extension of NumPy used in the Python programming language. It is a data visualization library, primarily utilized for plotting mathematical data and provides a collection of tools and methods for formatting the said visualizations.

Figure 4.14 Matplotlib Library [41]

- **PySyft:** PySyft is a library that extents the Tensorflow, Pytorch and Keras frameworks. It is intended for remote management of Machine and Deep Learnign applications, on-device prediction, privcy preserving and other communication and security functions.It's under the *Apache License 2.0* license and can be used to develop Federated Learning Environments.

Figure 4.15 PySyft Library [42]

Chapter Five

ADA-CLA-GAN Design & Implementation

5.1 Design

In this section is collocated the basic design principles of the ADA-GAN and the CLA-GAN proposed architectures. They are analyzed and their configuration is described in depth.

5.1.1 Basis

5.1.1.1 Literature Review

In this part, work in Deep Learning Intrusion Detection is outlined.

In [43], a Deep Learning Intrusion Detection System, named RNN-IDS, is researched for use in commercial networks. The authors with the use of Recurring Neural Networks (RNNs) implement an IDS system for binary and multiclass classification. They evaluate the proposed network and try to optimize the hyperparameters that compose it. Moreover, they compare the proposed network with a number of known Machine and Deep Learning algorithms and produce their results.

In [44] the authors develop a Deep Neural Network-based IDS system that is trained to detect computer network anomalies. The network is trained to be able to detect and classify novel and zero-day cyber-attacks. The developed model is trained using the KDDCup-99 dataset and is evaluated by testing it on a number of publicly available anomalous network datasets in order to evaluate its performance and further optimize it. The authors extend their work by proposing a scalable multi-DNN IDS, called Scale-Hybrid-IDS-AlertNet (SHIA), for real-time operation.

The work conducted in [45] produces an IDS system for in-vehicle networks, using the Generative Adversarial Network (GAN) architecture. The produced system called "GIDS" is able to detect novel unseen attacks in the network. By leveraging the utility of two discriminator module, for known and unknown attack classification respectively, it can reach high accuracy percentages in the novel attack detection.

The authors in [46] present an anomaly-based intrusion detection system that relies on two Deep Neural Networks. The featured networks, namely, Levenberg-Marquardt and Error BackPropagation, leverage the use of data extracted by a window-based feature extraction process training, testing and evaluation. The data are composed of both normal and abnormal traffic. The produced models are also evaluated on publicly available network intrusion datasets. The resulting performance evaluation shows a perfect detection score for the given attacks.

5.1.1.2 The GAN Architecture

A Generative Adversarial Network (GAN) architecture [47, 48] relies on two sub-neural networks, the Generator G and the Discriminator D. The Generator takes an input of random noise and generates data similar to the real data. On the other hand, the Discriminator inputs a sample of data and tries to classify them as real or fake. GAN's aim is to push both sub-networks, that rival each other, to train to a point that the Generator can produce data that the Discriminator can't distinguish from the real ones. Equation (5.1) below presents the relation of G and D.

$$\underset{G}{min}\underset{D}{max}V(G,D) = \underset{G}{min}\underset{D}{max}E_{x\sim p_{data}}[log(D(x))] + E_{z\sim p_z}[log(1-D(G(z)))] \tag{5.1}$$

G accumulates noise z from space Z mapping it to the space X from which D inputs x. $p_{data}(x)$ and $p_z(z)$ denote the probabilistic distribution of Spaces X and Z respectively.

5.1.1.3 The Autoencoder Architecture

Autoencoders are networks that learn to mimic the input data by a process of compressing them and consequently inflating them in a multilayer pipeline. Specifically, the network consists of two sub-networks, the Encoder and the Decoder. The Encoder network compresses the input data of space X to a manifold F. In contrast, the Decoder module inflates the data of manifold F to a sample P, where $P \sim X$. The aim of the Autoencoder architecture is to help the network, through the training process, produce samples p that are similar to the given real data r. After the training process, the network inputs unseen data and produces similar to the training. Equation (5.2) presents the data pipeline of the Autoencoder architecture.

$$r, p : \underset{r,p}{argmin} \left\| X - (p \circ r)X \right\|^2 , \; r : X \rightarrow F, \;\; p : F \rightarrow P \tag{5.2}$$

5.1.1.4 The AnoGan Architecture

In this work, the aforementioned deep neural network architectures are merged to produce a unified neural network architecture used for anomaly detection and anomaly classification. This is achieved by encapsulating the Autoencoder architecture into the structure of the GAN network. GAN's Generator takes the form of the Decoder while the Discriminator takes the structure of the Encoder module. In this schema, the Generator takes an input of a noise sample $N \times M$, where N is the number of noise points in a sample and M is the number of input samples. The Generator then inflates those samples to produce samples that mimic the desired data, like the normal GAN architecture. The Discriminator compresses the output that the Generator produced into a single point, which is the validity label of the sample. This is used to discriminate between real and fake samples. An intermediate model is exported after the training from the Discriminator module. This model is the part of the Discriminator from the input up to a latent layer before the output sequence of the network and is used for the anomaly detection process. Specifically, it is used to reduce the input dimension of the intake into a specified latent space. Two samples pass through the intermediate model, a real data sample and a generated sample. At this point, the Generator has learned to generate close to real data that mimic the normal samples. To calculate the anomaly score for the real sample, the Adversarial Loss of the two samples is taken. The Adversarial Loss is the difference between the generated and the real sample. Since the generator has learned to produce normal samples, the greater the Adversarial loss the bigger the probability of the real sample being abnormal. The equation below describes the Adversarial Loss as

$$AdvL(x) = \|d_r - d_p\| \tag{5.3}$$

where $AdvL(x)$ is the adversarial loss score of the function, d_r and d_p is the prediction of the latent model on the real and generated sample respectively. To extend this methodology to include anomaly classification also, a second novel implementation of the first proposed system is devised. The Anomaly Classification system presented utilizes a lightweight GAN-Autoencoder architecture that effectively classifies the intake data into separate anomalies. Both the proposed anomaly detection and the anomaly classification architectures are described below.

5.1.2 ADA-GAN Design and Specifications

In this instance of the aforementioned joined architecture, the network works as an anomaly detector. It is trained on a set of normal samples and as so can distinguish abnormal outlier samples in a dataset containing both normal and abnormal samples. The layout of the network can be separated into three components, the input layer, the Generator and the Discriminator module. Figure 5.1 presents the ADA-GAN deep neural network structure.

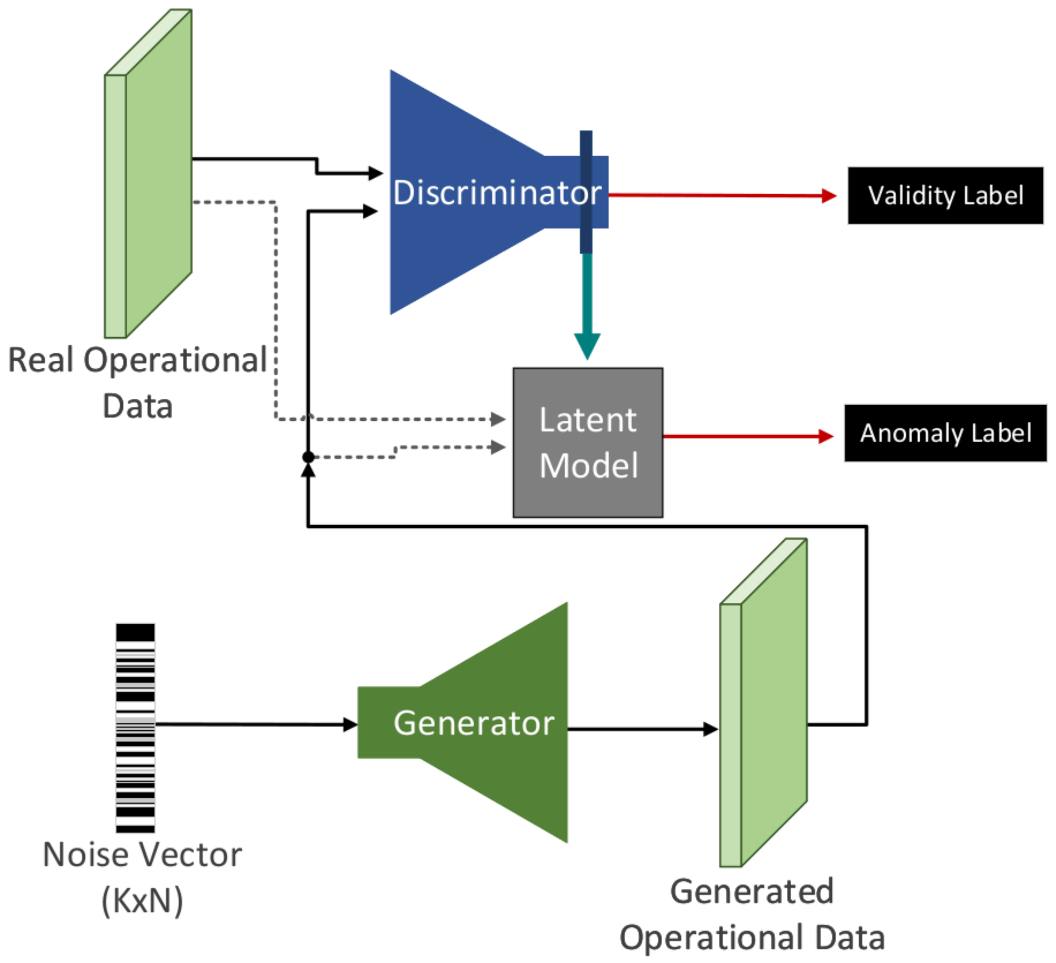


Figure 5.1 ADA-GAN Architecture

5.1.2.1 Input Layer

The input layer represents the input of the proposed deep neural network. The layer takes as input a noise vector of size N. The random noise vector is generated using a uniform distribution with a minimum value of 0 and a maximum of 1 from a uniform distribution with mean μ and standard deviation σ. Figure 5.2 shows the noise distribution of the noise vector.

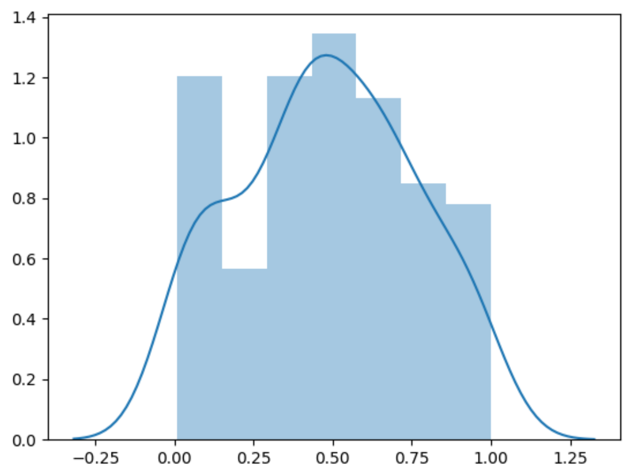


Figure 5.2 Uniform Noise Distribution

5.1.2.2 Generator-Decoder Module

In this proposed implementation, the Generator is in charge of inflating a random noise input vector of size $z = 10$ to a size M, where M is the number of features, while the generated data mimic the real ones. Since the use case for this network is anomaly detection, the Generator is trained on producing normal samples. The Generator's structure consists of thirteen layers, an input layer, an output $Tanh$ layer and a sequence of $Dense$, $ReLU$, $LeakyReLU$, $Batch\ Normalization$ and $Dropout$ layers in between.

$$tanh(x) = 2s(2x) - 1, tanh \rightarrow [-1, 1] \tag{5.4}$$

where equation (5.4) describes the $Tanh$ function. $tanh(x)$ is the output of the $tanh$ function, $s(x)$ is a Sigmoid function (5.6) and x is the input vector.

An explanatory depiction of the Generator's structure is shown in Figure 5.3. This module is compiled with the Binary Cross-Entropy function (5.5) and the RMSprop optimizer with a learning rate parameter of $lr = 0.0002$. The Binary Cross-Entropy function is defined as follows. N is the number of samples given, y is the label, in this case, a random sample. $p(y_i)$ is the probability of the sample being a match to the label sample when $1 - p(y_i)$ presents the inverse of that probability. Finally, H represents the result of the Binary Cross-Entropy loss at a given point.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{5.5}$$

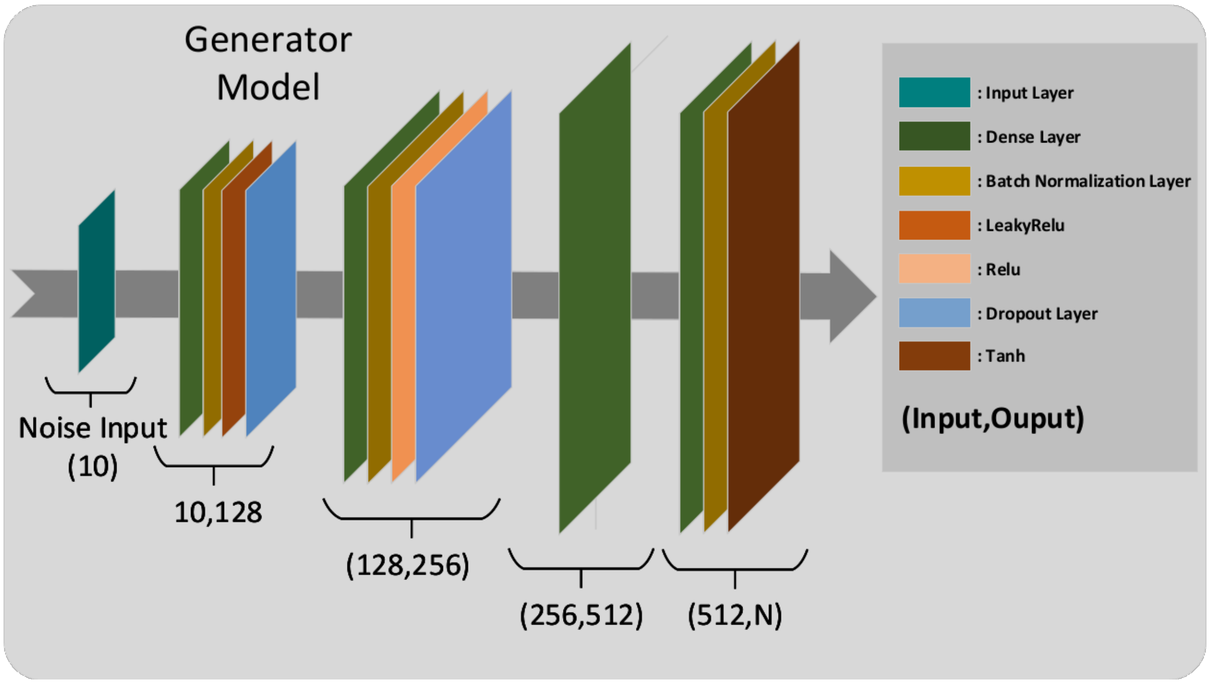


Figure 5.3 Generator-Decoder Structure

5.1.2.3 Discriminator-Encoder Module

The Discriminator module takes an input vector of M features, which is a vector representing a data instance sample. It then compresses the data through a multi-layer pipeline to a single point representing the validity layer, i.e. the classification of the sample being real or fake. The Discriminator is trained alongside the Generator, but receiving both real and generated samples, each with a ground truth label. The ground truth labels for the samples given as input to the Discriminator are a label of $tl \rightarrow 1$ for the Generator's output and $fl \rightarrow 0$ for the real sample, so to be trained to classify them properly. In the training process Discriminator's training ability is deactivated when the Generator is training.

From this module, the intermediate latent model is extracted later, as described above. This module is also compiled with the Binary Cross-Entropy function (5.5) and the RMSprop optimizer with a learning rate parameter of 0.0002. The thirteen layers consisting of the Discriminator are an input layer, an output Sigmoid layer (5.6) and a sequence of Dense, ReLU, Leaky ReLU, Batch Normalization and Dropout layers in between. Figure 5.4 presents the Discriminator's architecture.

$$s(x) = \frac{1}{1+e^{-x}}, s \to [0,1] \tag{5.6}$$

Equation (5.6) describes the sigmoid function. $s(x)$ is the output of the sigmoid function and x is the input vector.

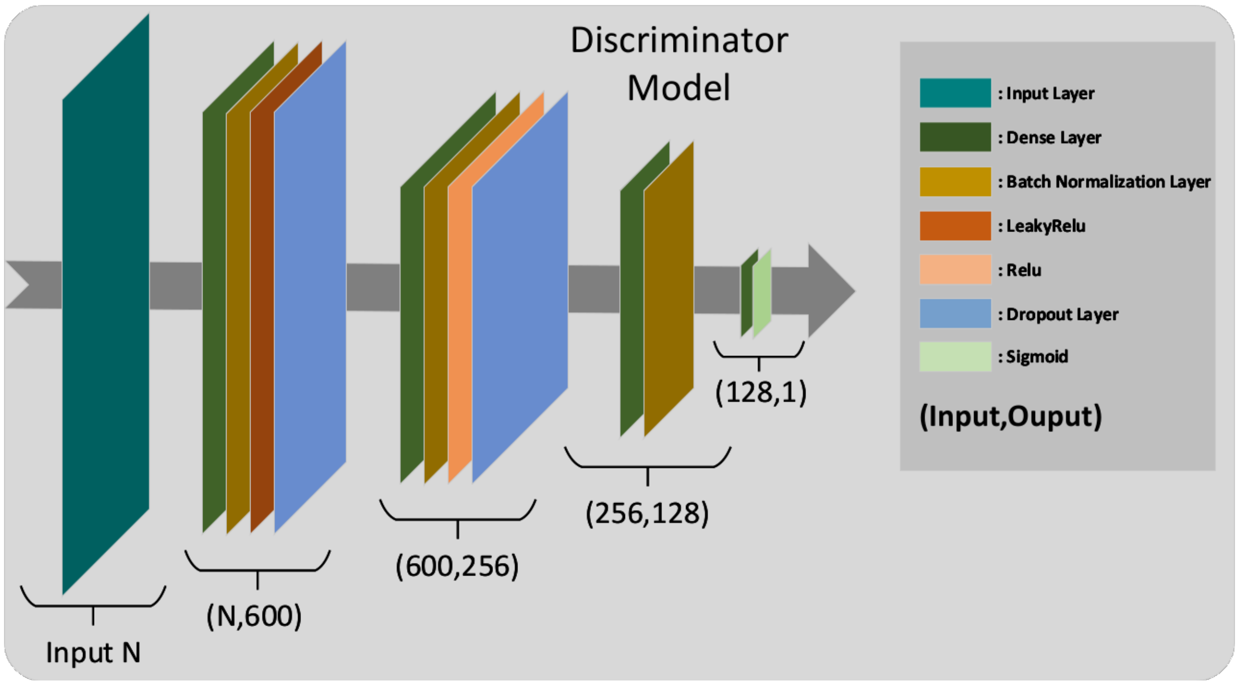


Figure 5.4 Discriminator-Encoder Structure

5.1.3 CLA-GAN Design and Specifications

The second proposed architecture is a derivation of the ADA-GAN architecture described previously. This implementation combines both the process of anomaly detection and anomaly classification into a single deep neural network structure. Specifically, it produces three separate ground-truth label points, one for the validity of the sample, one for the anomaly approximation and lastly one describing the anomaly class of the sample. This architecture can also be discretized into three different sub-modules, the Input layer, the Generator-Decoder module and the Discriminator-Encoder module. The layout of this proposed deep neural network architecture is depicted in figure 5.5. The main difference with the ADA-GAN architecture is that this network is less specific, more lightweight and is designed to handle multiclass data with fewer features, whereas the ADA-GAN is designed to handle one class and data with a large number of features.

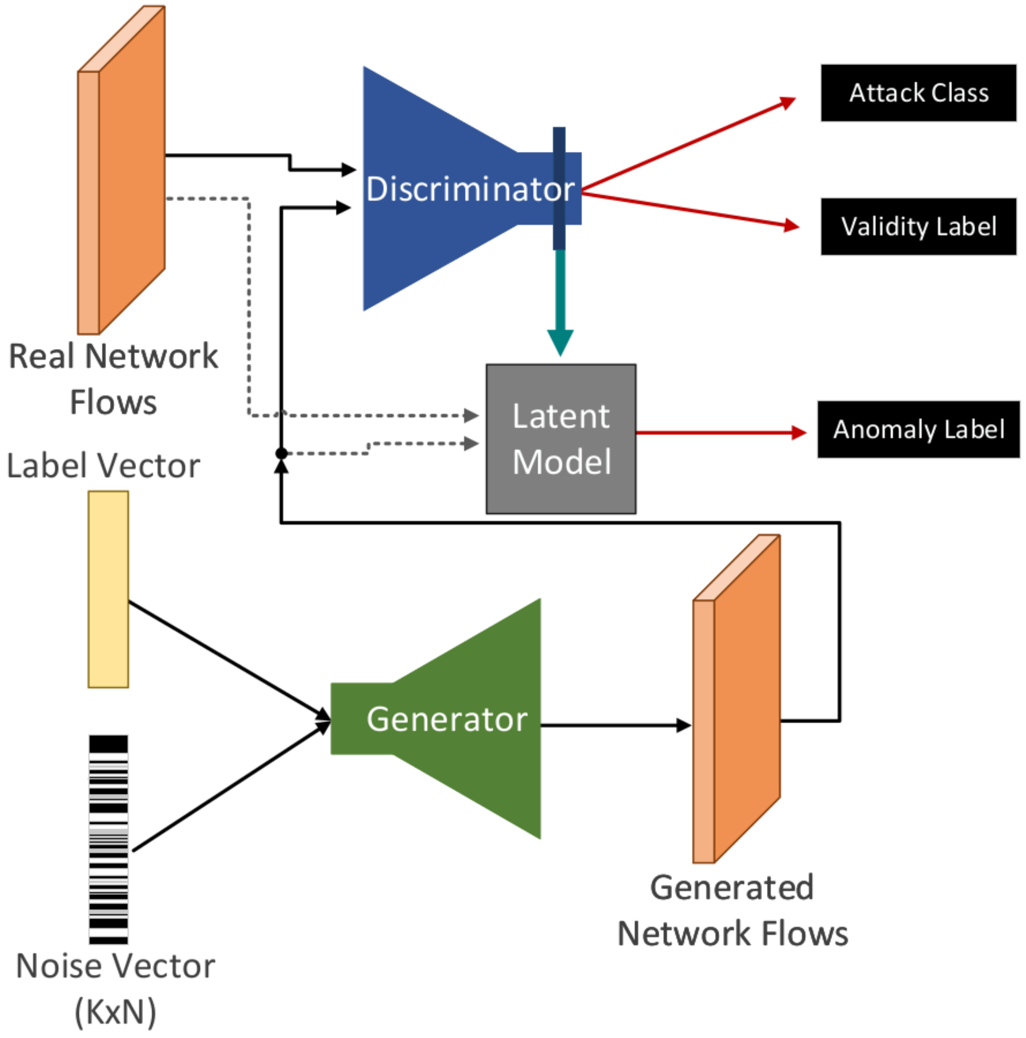


Figure 5.5 CLA-GAN Architecture

5.1.3.1 Input Layer

The input layer takes a noise vector input of size N and a vector containing the class representation of the sample. The elements of the random noise vector follows a normal distribution with mean μ and standard deviation σ, 0 and of 1 respectively. In figure 5.6 the distribution of the random noise input vector can be seen. The label vector with a dimension of [1 x C], where C is the number of classes that exist in the given dataset, is a zero vector with a 1 in the position of the class. The class of the sample is represented by c_p which is derived by the following formula.

$$c_p = argmax(V_{label}) \tag{5.7}$$

where V_{label} is the label vector.

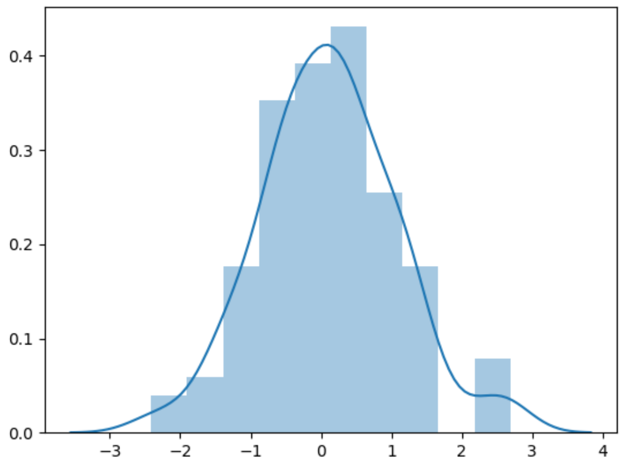


Figure 5.6 Normal Noise Distribution

5.1.3.2 Generator-Decoder Module

The Generator module in the anomaly classification case is a modified version of the Generator module used in the ADA-GAN architecture. The module is a Decoder adapted to the Generator structure. In this instance, the Generator inputs the two vectors explained in the Input Layer and concatenates them so to pass through the Generator's structure. Its structure consists of 9 layers, an input layer, an output *Relu* layer and a sequence of *Dense* and *ReLU* layers in between. The Generator's structure is shown in Figure 5.7. This module is compiled with the Categorical Cross-Entropy function (5.8) and the Adadelta optimizer [28]. During training, the Generator learns tho reproduce the data representing every class in the dataset, represented by a given label vector. This means that it produces a sample of a certain class that is inputted as a label vector. The output of this module is a vector of size M, where M is the number of features of the sample.

$$L_{cc}(r,p) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(r_{ij} * log(p_{ij})) \tag{5.8}$$

The above equation denotes the Categorical Cross-Entropy loss function, used to compile the Generator. $L_{cc}(y,p)$ is the Categorical Cross-Entropy output, r is the real sample and p is the generated sample.

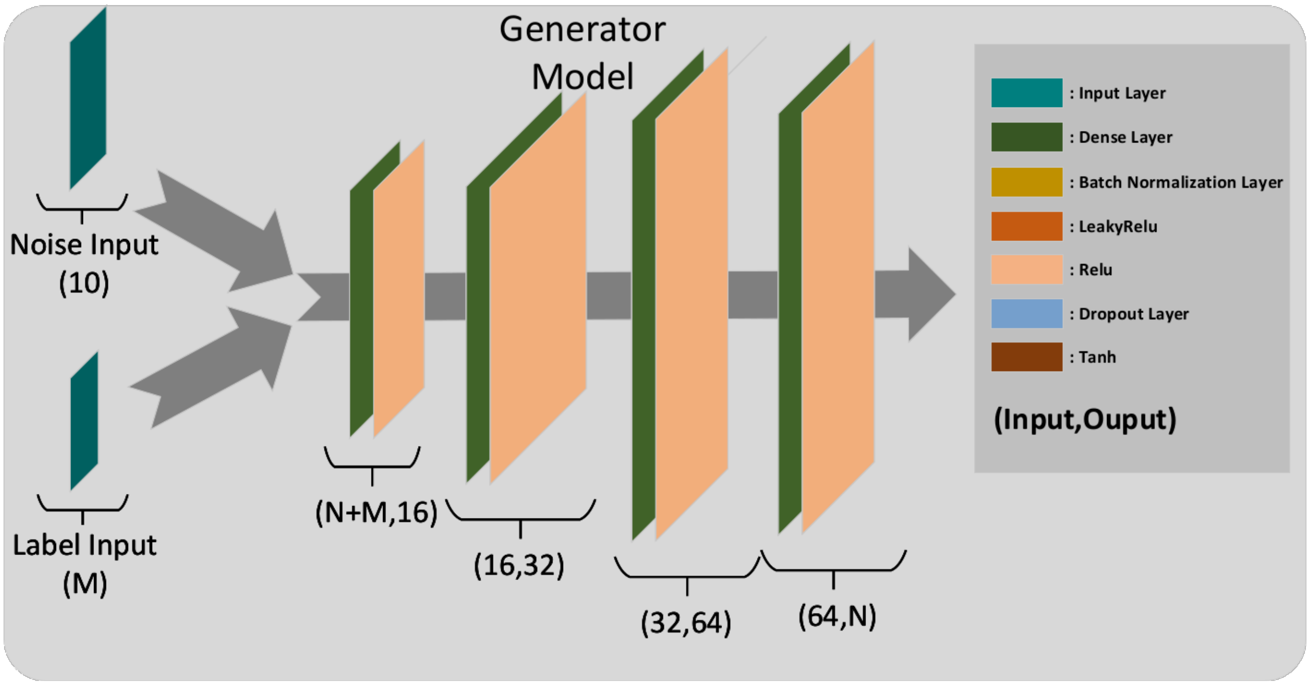


Figure 5.7 Generator-Decoder Structure

5.1.3.3 Discriminator-Encoder Module

The Discriminator module takes an input vector of M features, which represents a data sample. Since this proposed architecture produces not only the validity approximation but also the anomaly classification of the inputted sample, the output of the Discriminator module is divided into two parts. The first part is the validity label of the given sample, which highlights the sample as real or fake. The second part is a label vector, as described above, that denotes the processed classification of the sample to the separate classes given in the dataset. This vector of size C contains the numbers predicted by the Discriminator in the range of $[0, 1]$, using the Softmax activation function.

The class of the sample is considered the position of the highest value in that vector, as described in (5.7). As in the proposed ADA-GAN architecture, the Discriminator is trained alongside the Generator, but receiving both real and generated samples, each with a ground truth label and a label vector. The ground truth labels for the samples given as input to the Discriminator are a label of $tl \to 1$ for the Generator's output and $fl \to 0$ for the real sample. In the case of the label vectors, for the real sample the corresponding label vector is given as input to the Discriminator, while for the fake or predicted sample, a vector with a random label is given. As before, the Discriminator's training ability is deactivated when the Generator is training. The Discriminator module is compiled with the Binary Cross-Entropy (5.5) for the validity and the Categorical Cross-Entropy (5.8) for the classification part and the Adadelta optimizer [28].

5.2 Implementation

In this section, the proposed architectures are implemented. Below are presented the workings of the created networks, the training and testing phases of those networks and the way they were developed.

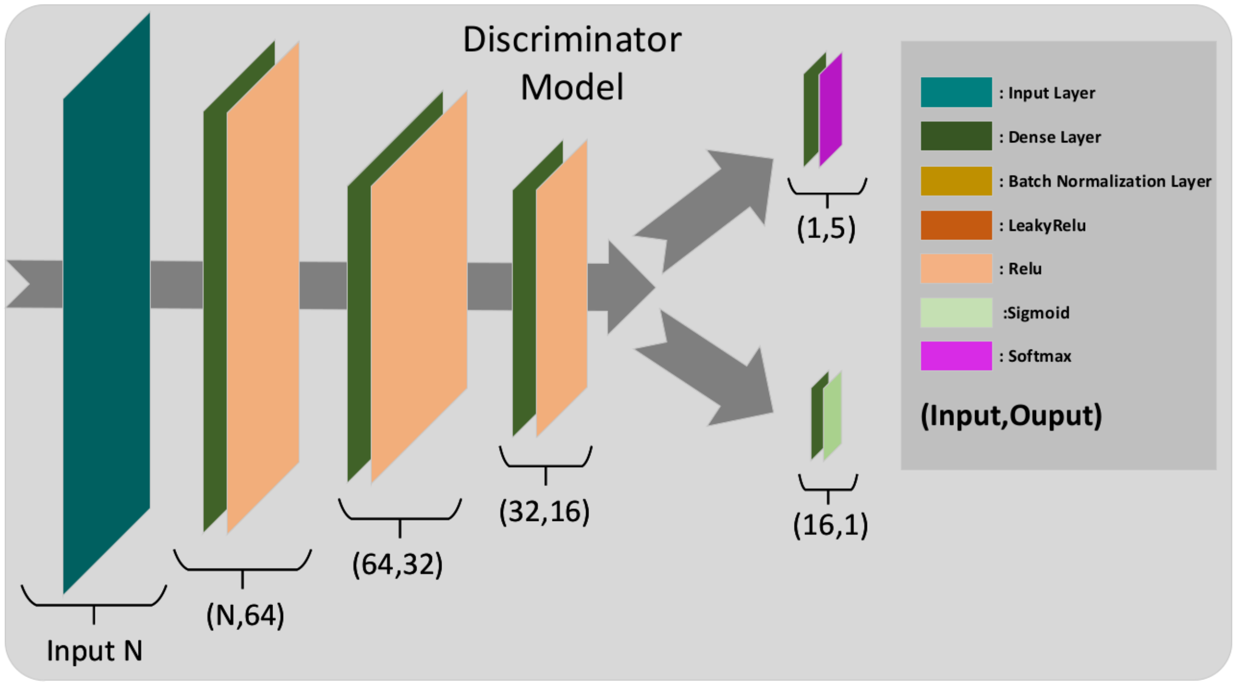


Figure 5.8 Discriminator-Encoder Structure

5.2.1 Development Environment

Both described Neural Networks were developed under the same system specifications. Some fundamental tools were used to implement the design and run the architecture, that are outlined below.

5.2.1.1 Language

In order to develop the described architectures and the system that handles and feeds data to the Neural Networks but also handles the results, an adequate programming language supporting Deep Learning and the corresponding tools, mentioned in section 4.6, should be used. The Python (5.9) programming language offers the aforementioned functionalities and thus was selected to realize this work. Specifically, the Python 3 (3.6-1.8) is utilized to script the Neural Networks and the accompanying systems. Python is a high-level scripting language that supports a wide variety of scientific tools used for Deep Learning development. It also offers the advantage of system interconnection and cross-platform support, leaving space for the proposed work to be tested and evaluated in different systems.

Figure 5.9 Python Language [49]

5.2.1.2 Test Bed

Training and Testing the produced ADA-GAN and CLA-GAN architectures a testbed is needed. Both networks were built and ran on the Ubuntu Operating System. Ubuntu Linux provides a versatile and modular system that helps in developing and debugging the created application. The system utilizes an Intel(R) Core(TM) i7-8550U CPU - 1.80GHz with an 8GB Ram memory. This work was tested on both the Ubuntu 18.04.4 LTS Bionic Beaver and the Ubuntu 20.04 LTS Focal Fossa releases.

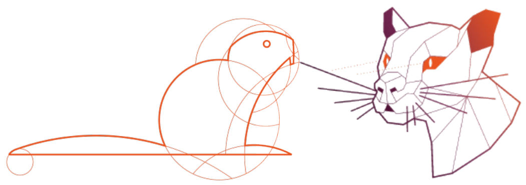


Figure 5.10 Ubuntu OS

5.2.2 ADA-GAN

5.2.2.1 Class Organization

To offer a function of modality and extensibility this architecture was implemented as a class structure. Firstly the core ADA-GAN class contains two sub-functions encapsulating the structure of the two fundamental components of a GAN architecture, the Generator and the Discriminator. When an ADA-GAN object is created, the module initializes the two sub-networks but it doesn't compile them until the fitting process is started. This is done so the module creating an ADA-GAN object can have the ability to change the network's dependencies and their hyperparameters before the training process. As shown in figure 5.11 the module also consists of some core and some helper functions. The function $fit()$ is a wrapper of the training function that is responsible for initializing the new Neural Network for the first time. This includes defining the loss functions and optimizers for the two sub-networks and, after the training phase, evaluating the point and returning the threshold with which the anomaly detection should be conducted. The $evaluation()$ function evaluates the performance of the trained network and either produces metrics related to its performance on a given dataset or returns the aforementioned threshold. All this with the help the $metricsAnalysis()$ functions that take over, analyzing either in high-level or in deep inspection the produced metrics, depending on the mode with which it was called with. The $Ladv()$ function calculates and returns the Adversarial Loss of the inputted data, as described in (5.3). The $Ladv()$ function depends on the $feature_extractor()$ function that, as the name suggests, extracts the latent model with the Adversarial Loss will be computed with.

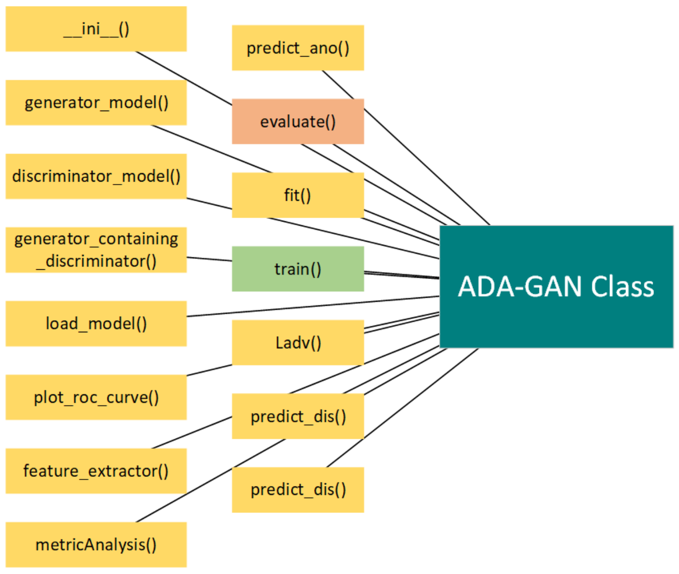
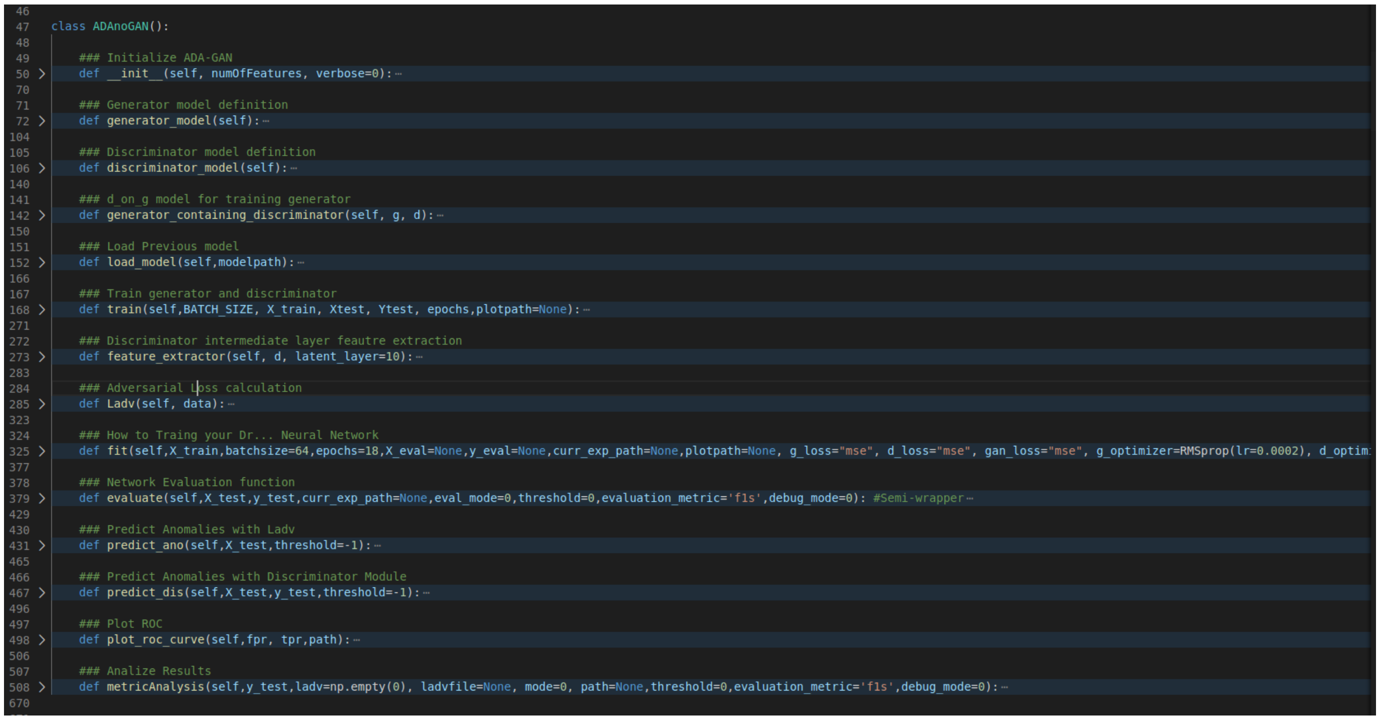


Figure 5.11 ADA-GAN Class



```python
class ADAnoGAN():

    ### Initialize ADA-GAN
    def __init__(self, numOfFeatures, verbose=0):⋯

    ### Generator model definition
    def generator_model(self):⋯

    ### Discriminator model definition
    def discriminator_model(self):⋯

    ### d_on_g model for training generator
    def generator_containing_discriminator(self, g, d):⋯

    ### Load Previous model
    def load_model(self,modelpath):⋯

    ### Train generator and discriminator
    def train(self,BATCH_SIZE, X_train, Xtest, Ytest, epochs,plotpath=None):⋯

    ### Discriminator intermediate layer feautre extraction
    def feature_extractor(self, d, latent_layer=10):⋯

    ### Adversarial Loss calculation
    def Ladv(self, data):⋯

    ### How to Traing your Dr... Neural Network
    def fit(self,X_train,batchsize=64,epochs=18,X_eval=None,y_eval=None,curr_exp_path=None,plotpath=None, g_loss="mse", d_loss="mse", gan_loss="mse", g_optimizer=RMSprop(lr=0.0002), d_optim

    ### Network Evaluation function
    def evaluate(self,X_test,y_test,curr_exp_path=None,eval_mode=0,threshold=0,evaluation_metric='f1s',debug_mode=0): #Semi-wrapper⋯

    ### Predict Anomalies with Ladv
    def predict_ano(self,X_test,threshold=-1):⋯

    ### Predict Anomalies with Discriminator Module
    def predict_dis(self,X_test,y_test,threshold=-1):⋯

    ### Plot ROC
    def plot_roc_curve(self,fpr, tpr,path):⋯

    ### Analize Results
    def metricAnalysis(self,y_test,ladv=np.empty(0), ladvfile=None, mode=0, path=None,threshold=0,evaluation_metric='f1s',debug_mode=0):⋯
```

Figure 5.12 ADA-GAN Class Python

Finally, there are two predicting functions, the $predict_ano()$ and the $predict_dis()$ the former returning the anomaly score of each sample based on the Adversarial Loss, while the latter used the role of the Discriminator module to predict irregular samples.

5.2.2.2 Modulation

The reason behind the modular composition of the ADA-GAN network is that it is designed to be mounted on a separate independent system and be ready-to-deploy without imposing on the host system's architecture, as depicted in figure 5.13.

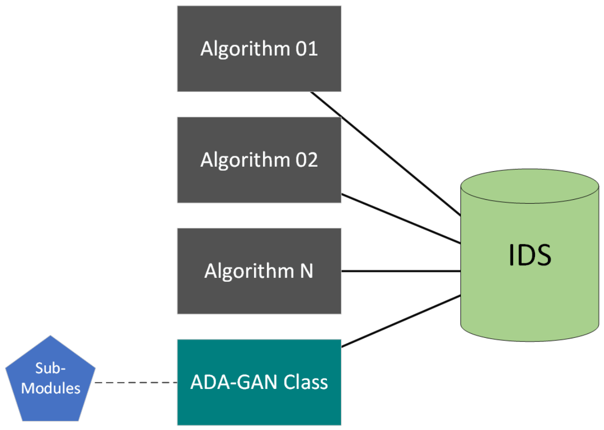


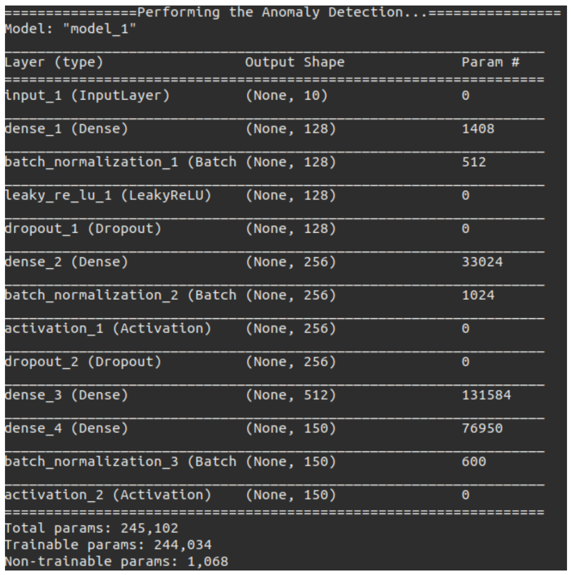
Figure 5.13 ADA-GAN Class Mounting

5.2.2.3 Data Preprocessing

Data preprocessing is one of the most important steps of Machine and Deep Learning. Often, the incoming data are in their raw and unprocessed form. A system like ADA-GAN is not developed with the purpose of handling any kind of input, even though it can handle any given dataset in the correct form. As such a correct method to normalize and clean the input data is needed. Since the preprocessing is algorithm and data specific, a custom implementation was realized for this network. The Operational Data that was used to train and test the ADA-GAN architecture against anomalous behaviour, in its raw form it represents field device voltage sensor readings. This is a time-dependent dataset that is described by time series. Since an impending attack is possibly going to have slow results, projected as long term voltage fluctuations, the data were combined in a sliding window of 30 samples per window. Also, to help the detector network accumulate the data better, a normalization of the data in the range of $(0, 1)$. The null and infinite values were also dropped from the dataset.

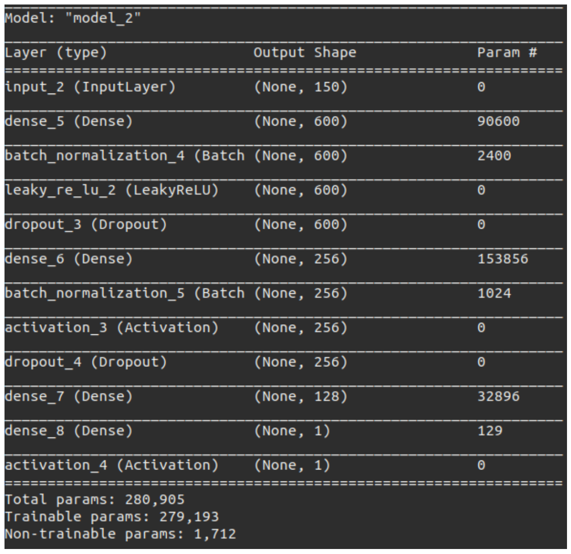
5.2.2.4 ADA-GAN Initialization

For the network to be effective in its intended purpose, Anomaly Detection, it has to go through a correctly laid out training procedure, so to have accumulated the corresponding



```
===============Performing the Anomaly Detection...===============
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 10)                0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1408
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_4 (Dense)              (None, 150)               76950
_________________________________________________________________
batch_normalization_3 (Batch (None, 150)               600
_________________________________________________________________
activation_2 (Activation)    (None, 150)               0
=================================================================
Total params: 245,102
Trainable params: 244,034
Non-trainable params: 1,068
```

Figure 5.14 Generator Initialization



```
_________________________________________________________________
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 150)               0
_________________________________________________________________
dense_5 (Dense)              (None, 600)               90600
_________________________________________________________________
batch_normalization_4 (Batch (None, 600)               2400
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 600)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 600)               0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               153856
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
activation_3 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 129
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 280,905
Trainable params: 279,193
Non-trainable params: 1,712
```

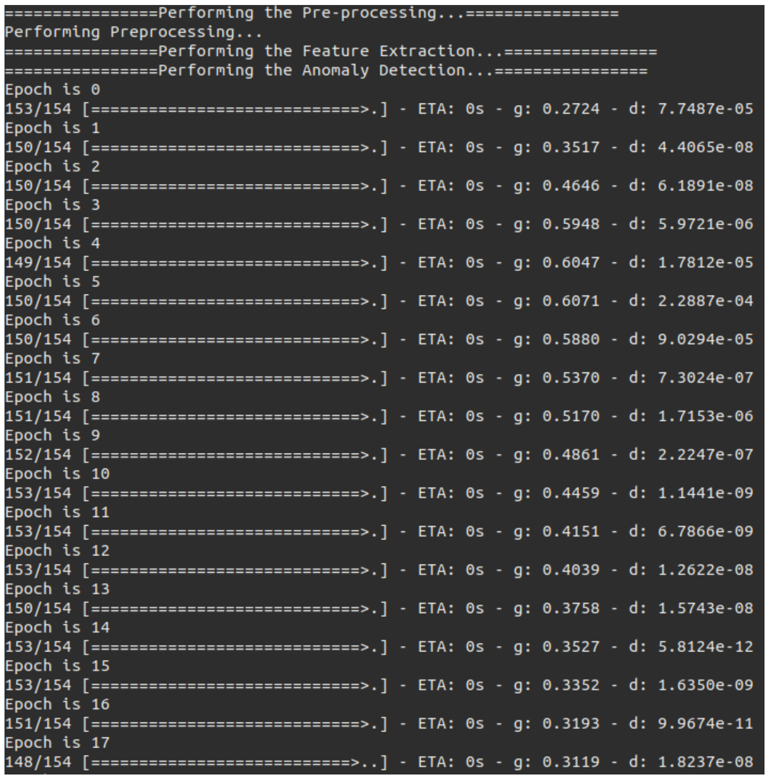
Figure 5.15 Discriminator Initialization

experience, accordingly. When an ADA-GAN object is initialized, depending on the given verbosity level, it informs about its given characteristics, like, the Generator and Discriminator structures and other useful information. At the same time, the come parameters and modules are defined, figure 5.14 and 5.15. As mentioned, even though the two sub-networks are initialized, but not compiled with the correct parameters. This occurs just before the Training process begins for practical reasons.

5.2.2.5 ADA-GAN Training

The ADA-GAN Training is the process that helps the network accumulate the input data and reach its intended purpose. The training takes place in a number of iterations, each session given a sample batch with a predefined batch size. Both the training epochs and batch size are found through experimenting and are described in the Results, chapter 6. Just before the training is begun, the ADA-GAN object compiles its networks with the given dependencies and hyperparameters. This is realized in this step, because the training process can take place multiple times, either from scratch or from a pre-trained ADA-GAN network. This gives the parent of the network the ability to handle the network's core characteristics on demand.

The training step also needs two sets of data, the Training Set and its accompanying Ground-truth labels to train the network properly. Finally, the training function's wrapper, the $fit()$ function, acts as a conduit for centralizing the production of the important features of the ADA-GAN network. When completed it returns the two trained sub-networks, it extracts the latent model, needed for the Anomaly Detection, returns the needed threshold with which the samples are classified and lastly it evaluates the produced network's performance. A snapshot of the Training process can be seen in 5.16.

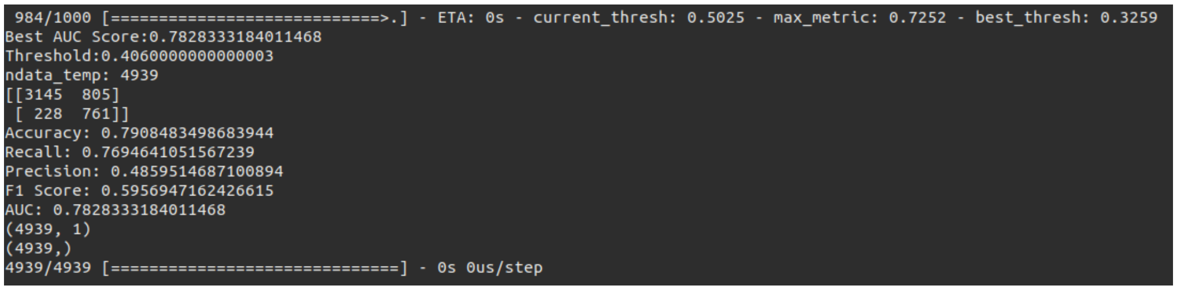


```
================Performing the Pre-processing...================
Performing Preprocessing...
================Performing the Feature Extraction...================
================Performing the Anomaly Detection...================
Epoch is 0
153/154 [===========================>.] - ETA: 0s - g: 0.2724 - d: 7.7487e-05
Epoch is 1
150/154 [===========================>.] - ETA: 0s - g: 0.3517 - d: 4.4065e-08
Epoch is 2
150/154 [===========================>.] - ETA: 0s - g: 0.4646 - d: 6.1891e-08
Epoch is 3
150/154 [===========================>.] - ETA: 0s - g: 0.5948 - d: 5.9721e-06
Epoch is 4
149/154 [===========================>.] - ETA: 0s - g: 0.6047 - d: 1.7812e-05
Epoch is 5
150/154 [===========================>.] - ETA: 0s - g: 0.6071 - d: 2.2887e-04
Epoch is 6
150/154 [===========================>.] - ETA: 0s - g: 0.5880 - d: 9.0294e-05
Epoch is 7
151/154 [===========================>.] - ETA: 0s - g: 0.5370 - d: 7.3024e-07
Epoch is 8
151/154 [===========================>.] - ETA: 0s - g: 0.5170 - d: 1.7153e-06
Epoch is 9
152/154 [===========================>.] - ETA: 0s - g: 0.4861 - d: 2.2247e-07
Epoch is 10
153/154 [===========================>.] - ETA: 0s - g: 0.4459 - d: 1.1441e-09
Epoch is 11
153/154 [===========================>.] - ETA: 0s - g: 0.4151 - d: 6.7866e-09
Epoch is 12
153/154 [===========================>.] - ETA: 0s - g: 0.4039 - d: 1.2622e-08
Epoch is 13
150/154 [===========================>.] - ETA: 0s - g: 0.3758 - d: 1.5743e-08
Epoch is 14
153/154 [===========================>.] - ETA: 0s - g: 0.3527 - d: 5.8124e-12
Epoch is 15
153/154 [===========================>.] - ETA: 0s - g: 0.3352 - d: 1.6350e-09
Epoch is 16
151/154 [===========================>.] - ETA: 0s - g: 0.3193 - d: 9.9674e-11
Epoch is 17
148/154 [==========================>..] - ETA: 0s - g: 0.3119 - d: 1.8237e-08
```

Figure 5.16 ADA-GAN Training

5.2.2.6 ADA-GAN Testing

The ADA-GAN Testing is a straightforward operation. It is realized at the end of the Training phase or independently on demand by the $evaluation()$ function. It takes in two sets of data, the Testing Set and its corresponding labels. This step informs about the network's performance and evaluates its ability to function properly and fulfill its intended goal. Figure 5.17 shows the results of a high-level evaluation.



```
 984/1000 [============================>.] - ETA: 0s - current_thresh: 0.5025 - max_metric: 0.7252 - best_thresh: 0.3259
Best AUC Score:0.7828333184011468
Threshold:0.4060000000000003
ndata_temp: 4939
[[3145  805]
 [ 228  761]]
Accuracy: 0.7908483498683944
Recall: 0.7694641051567239
Precision: 0.4859514687100894
F1 Score: 0.5956947162426615
AUC: 0.7828333184011468
(4939, 1)
(4939,)
4939/4939 [==============================] - 0s 0us/step
```

Figure 5.17 ADA-GAN Testing

5.2.2.7 Produced Model

When the ADA-GAN module has finished the Training and Testing and is ready for application there are a number of ways for utilizing it. Since the GAN architecture consists of two sub-networks, it makes the handling of the whole structure dependant on the use case. If only one of the two models is needed, the ADA-GAN module can save them as separate binary models in a $.h5$ configuration, which is a product of the HDF5 extension for big data persistence. As it can do the same with the joint network. Finally, the whole ADA-GAN object can be serialized, so as to keep and transfer its ADA-GAN characteristics.

5.2.3 CLA-GAN

As described in the Design and Specification section, both the ADA-GAN and CLA-GAN structures share the same Deep Learning architecture. The GAN architecture offers a powerful function by utilizing the agency of two competing Neural Network structures to optimize the generalization of the produced model. As such, the CLA-GAN network is based on this principle to successfully classify incoming data.

5.2.3.1 Class Organization

The class design of the CLA module follows the same organization as the ADA-GAN module. It's developed to be modal and extensible and to be able to mount on host systems with the same suppleness. The core CLA-GAN class contains two sub-networks, the Generator and the Discriminator. Figure 5.18 reveals the class structure of the encapsulated modules. Due to its more complex nature, except the $fit()$, $metricsAnalysis()$, $Ladv()$, $evaluation()$ functions that were explained in the ADA-GAN module implementation, it also contains further functionalities oriented in helping with its classification property. Specifically, since the main purpose of the CLA-GAN network is to classify correctly the input data, it needs tools to configure and evaluate the accumulating data. The network's input is a batch of samples accompanied from their respective ground-truth label which is a vector of the class identity, described by (5.7). Since the labels in the input dataset, usually, represent the class in a numeric or nominal form, the $vectorizeLabels()$ function takes over the conversion of the labels to the correct form. Also, since the CLA-GAN module also offers an Anomaly Detection method to further detect abnormal samples, there was devised a method, $Mul2BinLabels()$, for converting the multiclass labels to binary for the anomaly analysis. To evaluate the classification results the $classificationMetricAnalysis()$ function is designed specifically for classification metrics analysis. Lastly, the remaining functions are training specific and need not be mentioned.

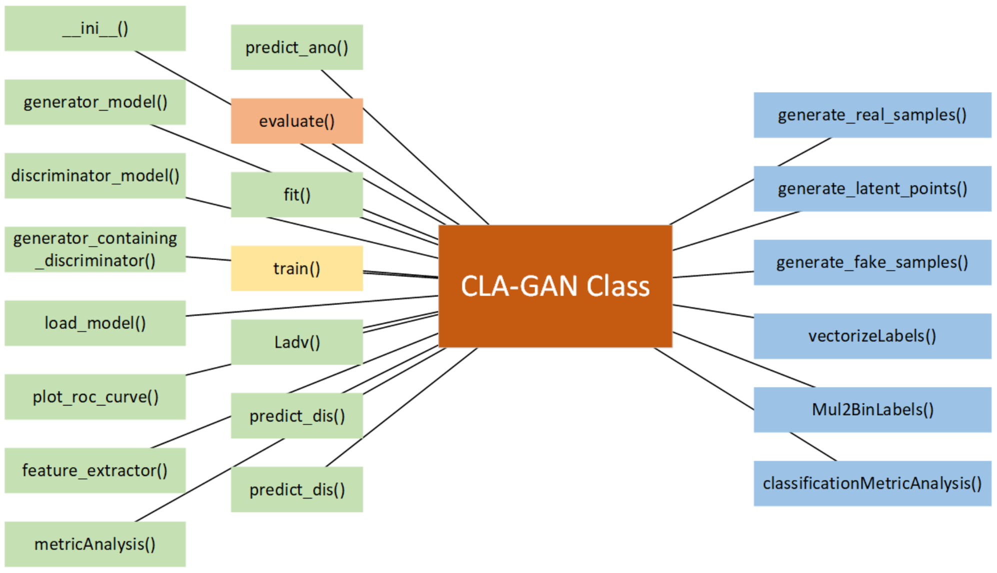
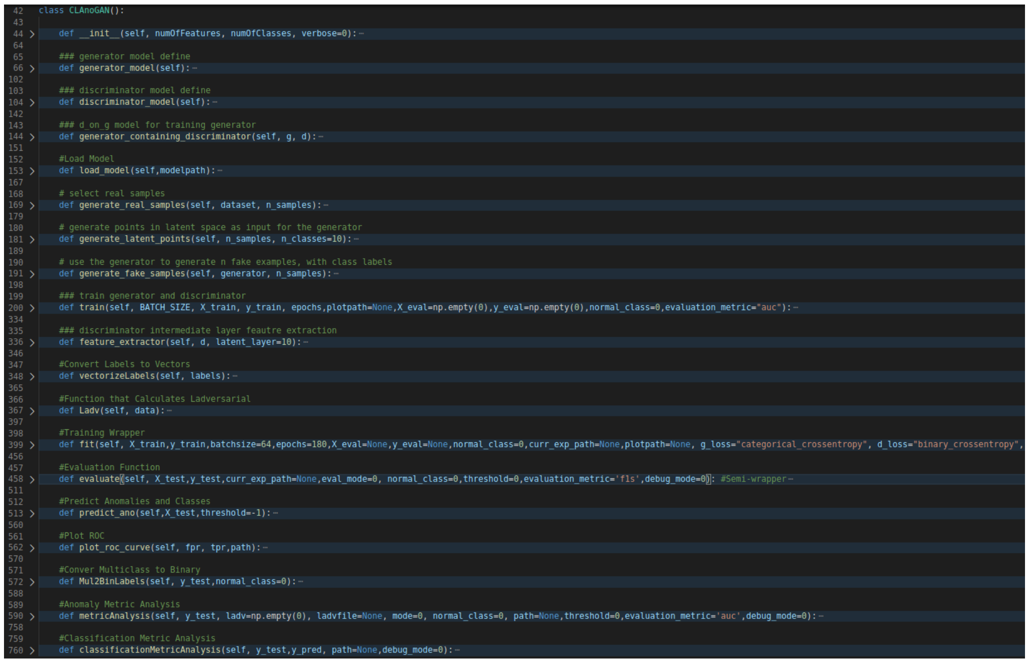


Figure 5.18 CLA-GAN Class



```
class CLAnoGAN():

    def __init__(self, numOfFeatures, numOfClasses, verbose=0):⋯

    ### generator model define
    def generator_model(self):⋯

    ### discriminator model define
    def discriminator_model(self):⋯

    ### d_on_g model for training generator
    def generator_containing_discriminator(self, g, d):⋯

    #Load Model
    def load_model(self,modelpath):⋯

    # select real samples
    def generate_real_samples(self, dataset, n_samples):⋯

    # generate points in latent space as input for the generator
    def generate_latent_points(self, n_samples, n_classes=10):⋯

    # use the generator to generate n fake examples, with class labels
    def generate_fake_samples(self, generator, n_samples):⋯

    ### train generator and discriminator
    def train(self, BATCH_SIZE, X_train, y_train, epochs,plotpath=None,X_eval=np.empty(0),y_eval=np.empty(0),normal_class=0,evaluation_metric="auc"):⋯

    ### discriminator intermediate layer feautre extraction
    def feature_extractor(self, d, latent_layer=10):⋯

    #Convert Labels to Vectors
    def vectorizeLabels(self, labels):⋯

    #Function that Calculates Ladversarial
    def Ladv(self, data):⋯

    #Training Wrapper
    def fit(self, X_train,y_train,batchsize=64,epochs=180,X_eval=None,y_eval=None,normal_class=0,curr_exp_path=None,plotpath=None, g_loss="categorical_crossentropy", d_loss="binary_crossentropy",

    #Evaluation Function
    def evaluate(self, X_test,y_test,curr_exp_path=None,eval_mode=0, normal_class=0,threshold=0,evaluation_metric='f1s',debug_mode=0): #Semi-wrapper⋯

    #Predict Anomalies and Classes
    def predict_ano(self,X_test,threshold=-1):⋯

    #Plot ROC
    def plot_roc_curve(self, fpr, tpr,path):⋯

    #Conver Multiclass to Binary
    def Mul2BinLabels(self, y_test,normal_class=0):⋯

    #Anomaly Metric Analysis
    def metricAnalysis(self, y_test, ladv=np.empty(0), ladvfile=None, mode=0, normal_class=0, path=None,threshold=0,evaluation_metric='auc',debug_mode=0):⋯

    #Classification Metric Analysis
    def classificationMetricAnalysis(self, y_test,y_pred, path=None,debug_mode=0):⋯
```

Figure 5.19 CLA-GAN Class Python

5.2.3.2 Data Preprocessing

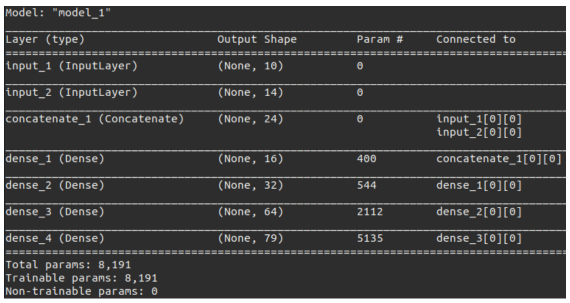
In this instance, the input data are primarily network flows. The preprocessing firstly drops the nominal features of the dataset, like, timestamps, flow directions and so on, and proceeds to fill or drop the null and infinite values. After that, The data are normalized in the range of $(0, 1)$ using a Min-Max Scaler. The ground-truth labels need also be preprocessed. A function in the host system converts the labels from nominal to numerical, e.g., $"Normal" \rightarrow 0$. The rest of the data processing takes place inside the CLA-GAN module, like vectorizing the numerical labels with the $vectorizeLabels()$ function when needed for computational reasons.

5.2.3.3 CLA-GAN Initialization

When the CLA-GAN module is initialized, depending on the given verbosity level, it produces debug information about its given characteristics, like, the Generator and Discriminator structures, figures 5.20 and 5.21. Like the ADA-GAN module, the core parameters of CLA-GAN are set just before training.

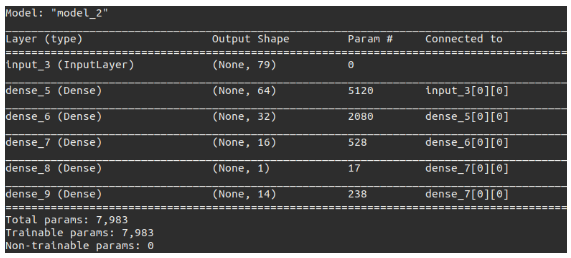
5.2.3.4 CLA-GAN Training

Before Training, when the $fit()$ wrapper is called by the host system, the core parameters and dependencies of the CLA-GAN object are set. During Training, the network accumulates the input data and trains the sub-networks for classifying the input correctly, but it also trains



```
Model: "model_1"
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
input_1 (InputLayer)            (None, 10)           0
____________________________________________________________________________________________________
input_2 (InputLayer)            (None, 14)           0
____________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 24)           0           input_1[0][0]
                                                                 input_2[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                 (None, 16)           400         concatenate_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                 (None, 32)           544         dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                 (None, 64)           2112        dense_2[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                 (None, 79)           5135        dense_3[0][0]
====================================================================================================
Total params: 8,191
Trainable params: 8,191
Non-trainable params: 0
```

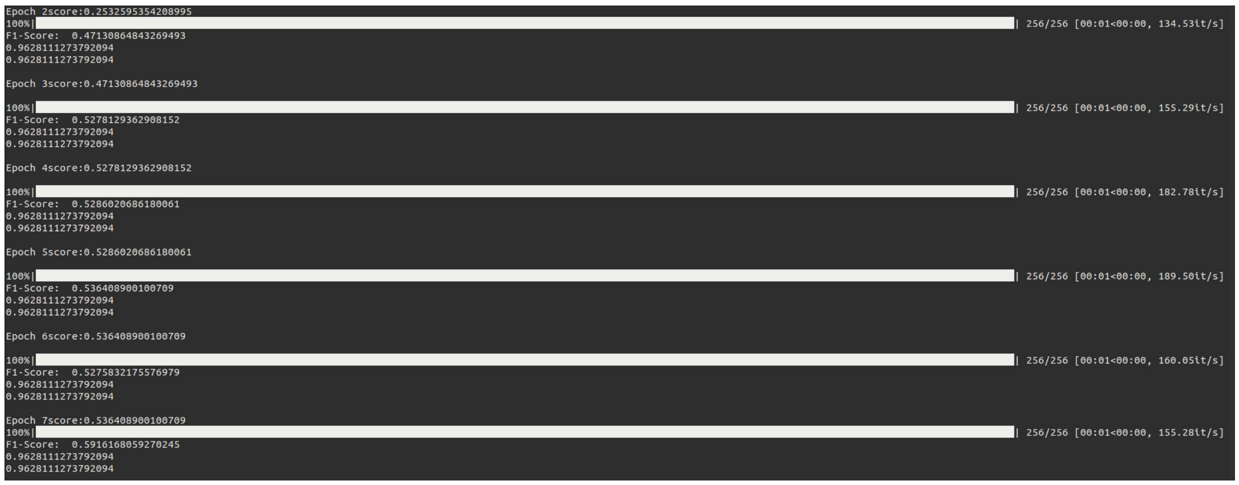
Figure 5.20 Generator Initialization



```
Model: "model_2"
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
input_3 (InputLayer)            (None, 79)           0
____________________________________________________________________________________________________
dense_5 (Dense)                 (None, 64)           5120        input_3[0][0]
____________________________________________________________________________________________________
dense_6 (Dense)                 (None, 32)           2080        dense_5[0][0]
____________________________________________________________________________________________________
dense_7 (Dense)                 (None, 16)           528         dense_6[0][0]
____________________________________________________________________________________________________
dense_8 (Dense)                 (None, 1)            17          dense_7[0][0]
____________________________________________________________________________________________________
dense_9 (Dense)                 (None, 14)           238         dense_7[0][0]
====================================================================================================
Total params: 7,983
Trainable params: 7,983
Non-trainable params: 0
```

Figure 5.21 Discriminator Initialization

the Discriminator to detect anomalies. The training is completed in a number of epochs, each session given a sample batch with a predefined batch size. Both the training epochs and batch size are found through experimenting and are described in the Results, chapter 6. In 5.22 a CLA-GAN instance is depicted.



```
Epoch 2score:0.2532595354208995
100%|██████████████████████████████| 256/256 [00:01<00:00, 134.53it/s]
F1-Score:  0.47130864843269493
0.9628111273792094
0.9628111273792094

Epoch 3score:0.47130864843269493

100%|██████████████████████████████| 256/256 [00:01<00:00, 155.29it/s]
F1-Score:  0.5278129362908152
0.9628111273792094
0.9628111273792094

Epoch 4score:0.5278129362908152

100%|██████████████████████████████| 256/256 [00:01<00:00, 182.78it/s]
F1-Score:  0.5286020686180061
0.9628111273792094
0.9628111273792094

Epoch 5score:0.5286020686180061

100%|██████████████████████████████| 256/256 [00:01<00:00, 189.50it/s]
F1-Score:  0.536408900100709
0.9628111273792094
0.9628111273792094

Epoch 6score:0.536408900100709

100%|██████████████████████████████| 256/256 [00:01<00:00, 160.05it/s]
F1-Score:  0.5275832175576979
0.9628111273792094
0.9628111273792094

Epoch 7score:0.536408900100709
100%|██████████████████████████████| 256/256 [00:01<00:00, 155.28it/s]
F1-Score:  0.5916168059270245
0.9628111273792094
0.9628111273792094
```

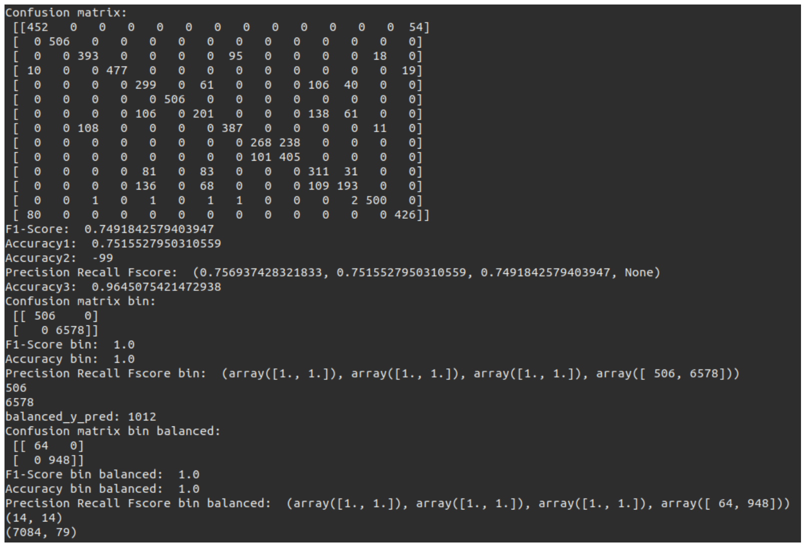
Figure 5.22 CLA-GAN Training

5.2.3.5 CLA-GAN Testing

The Testing phase utilizes the evaluation function and, by providing a Testing dataset, it evaluates the performance of the produced network with unseen data. By calling the $metricsAnalysis()$ and $classificationMetricAnalysis()$ methods and depending on the evaluation mode, a high-level, figure 5.19, or deep metrics inspection can be performed on the Neural Network.

5.2.3.6 Produced Model

When the CLA-GAN module has passed the Trining and the evaluation phases and is ready for deployment, depending on its use-case it can be handled accordingly. Since CLA-GAN also follows the GAN architecture, it consists of two sub-networks. If only one of the two models is needed, the CLA-GAN module can save them as separate binary models in a $.h5$



```
Confusion matrix:
 [[452   0   0   0   0   0   0   0   0   0   0   0   0  54]
 [  0 506   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0 393   0   0   0   0  95   0   0   0   0  18   0]
 [ 10   0   0 477   0   0   0   0   0   0   0   0   0  19]
 [  0   0   0   0 299   0  61   0   0   0 106  40   0   0]
 [  0   0   0   0   0 506   0   0   0   0   0   0   0   0]
 [  0   0   0   0 106   0 201   0   0   0 138  61   0   0]
 [  0   0 108   0   0   0   0 387   0   0   0   0  11   0]
 [  0   0   0   0   0   0   0   0 268 238   0   0   0   0]
 [  0   0   0   0   0   0   0   0 101 405   0   0   0   0]
 [  0   0   0   0  81   0  83   0   0   0 311  31   0   0]
 [  0   0   0   0 136   0  68   0   0   0 109 193   0   0]
 [  0   0   1   0   1   0   1   1   0   0   0   2 500   0]
 [ 80   0   0   0   0   0   0   0   0   0   0   0   0 426]]
F1-Score:  0.7491842579403947
Accuracy1:  0.7515527950310559
Accuracy2:  -99
Precision Recall Fscore:  (0.756937428321833, 0.7515527950310559, 0.7491842579403947, None)
Accuracy3:  0.9645075421472938
Confusion matrix bin:
 [[ 506    0]
 [   0 6578]]
F1-Score bin:  1.0
Accuracy bin:  1.0
Precision Recall Fscore bin:  (array([1., 1.]), array([1., 1.]), array([1., 1.]), array([ 506, 6578]))
506
6578
balanced_y_pred: 1012
Confusion matrix bin balanced:
 [[ 64   0]
 [  0 948]]
F1-Score bin balanced:  1.0
Accuracy bin balanced:  1.0
Precision Recall Fscore bin balanced:  (array([1., 1.]), array([1., 1.]), array([1., 1.]), array([ 64, 948]))
(14, 14)
(7084, 79)
```

Figure 5.23 CLA-GAN High-level Evaluation

configuration. The same can be done with the joint network. Finally, like the ADA-GAN object, the CLA-GAN can be serialized and be transported as a class object.

Chapter Six

ADA-CLA-GAN Experimental Results

6.1 Evaluation

The evaluation of a Neural Network is the process through which the performance of a given network is scrutinized in order to prove the viability of its function. In this work, the proposed networks are analyzed and compared against other Machine Learning and Deep Learning algorithms of their respective field of use. Bellow follows the description of the data used to develop and train the Deep Neural Networks in this work.

6.1.1 Dataset

In order to train and evaluate the two deep neural network architectures, namely, a) the ADA-GAN and b) the CLA-GAN, two different datasets were leveraged. These data represent key information about the internal processes of critical infrastructures and serve as a fundamental conduit for anomaly detection and anomaly classification research. Though the described neural networks can be used with a plethora of datasets, as described in their corresponding implementation, both for anomaly detection and anomaly classification, data that originate from critical infrastructures in Industrial Networks were chosen. Specifically, of the two datasets, the first describes power utility information of power distribution installations and the second network flow data from those industrial infrastructures.

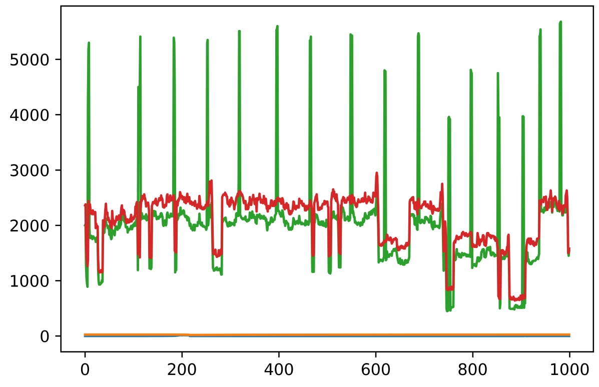


Figure 6.1 Normal

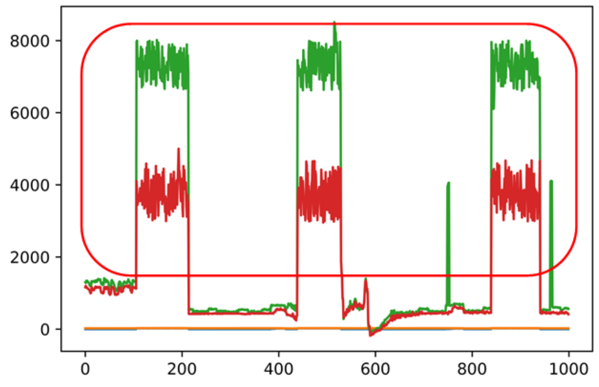


Figure 6.2 Abnormal

Figure 6.3 PDP03 Readings

6.1.1.1 Anomaly Detection Data

For the ADA-GAN network that conforms with the Anomaly Detection function, the Operational dataset was used. This data category contains power usage information from four power distribution facilities. They are identified as, i) Power Distribution Plant 01 (PDP01), ii) Power Distribution Plant 02 (PDP02), iii) Power Distribution Plant 03 (PDP03) and iv) Power Distribution Plant 04 (PDP04), respectively. The values represent real-time voltage readings from the sensors and field devices throughout each facility's network. In every dataset, the number of features and their corresponding values as well as the meaning of those readings, vary depending on each independent facility. The data are composed of normal and abnormal power readings, the former describing the normal function of each facility and the latter, the effort of a malicious entity altering those readings by exploiting the operation of the facilities' devices. Normal power readings of those facilities can be seen in figures 6.1 and 6.4, while abnormal power reading are depicted in figures 6.2 and 6.5. Each set is separated in the training data, which contains normal power readings and the testing-evaluation data that contain both normal and abnormal readings.

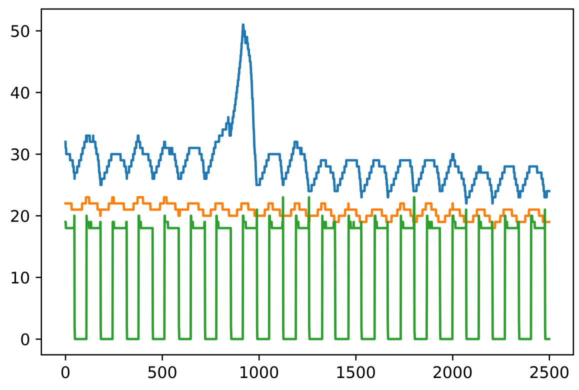


Figure 6.4 Normal

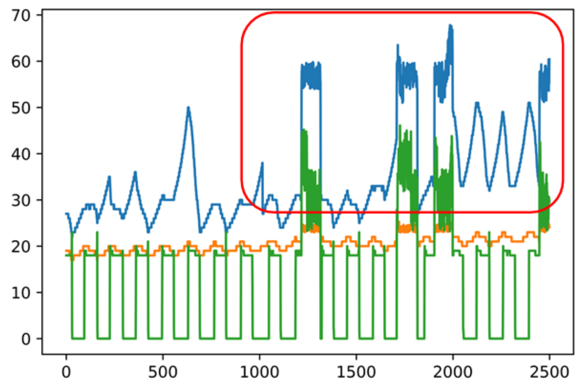


Figure 6.5 Abnormal

Figure 6.6 PDP04 Readings

6.1.1.2 Anomaly Classification Data

For the CLA-GAN that adheres to the Anomaly Classification case, the latter data category was used. The data describe Modbus network flows circulating in each of the mentioned facilities' computer networks. Each set is separated into a training and testing-evaluation set. Both sets contain normal and abnormal network flow samples, accompanied by a ground-truth label describing the attack category of the sample. The captured flows contain a specific number of features that do not change in the different datasets. Table 6.1 lists the attack classes used in the datasets as well as their enumeration and vectorization.

These data are preprocessed as described in section 5.2.3.2. The nominal features are dropped and the rest of the features are normalized in a given range. In this work, the normalization range is [0,1]. For the purpose of feeding the data into the proposed classifier, the attack classes are enumerated and then vectorized, as described in the implementation section. An example of the attack vectors is seen in table 6.1.

No.	Attack Class	Attack Vector
0	*Normal*	*[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]*
1	*modbus/dos/writeSingleCoils*	*[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]*
2	*modbus/dos/writeSingleRegister*	*[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]*
3	*modbus/function/readCoils*	*[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]*
4	*modbus/function/readCoils (DoS)*	*[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]*
5	*modbus/function/readDiscreteInput*	*[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]*
6	*modbus/function/readDiscreteInputs (DoS)*	*[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]*
7	*modbus/function/readHoldingRegister*	*[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0]*
8	*modbus/function/readHoldingRegister (DoS)*	*[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0]*
9	*modbus/function/readInputRegister*	*[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]*
10	*modbus/function/readInputRegister (DoS)*	*[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]*
11	*modbus/function/writeSingleCoils*	*[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]*
12	*modbus/function/writeSingleRegister*	*[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]*
13	*modbus/scanner/getfunc*	*[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]*
14	*modbus/scanner/uid*	*[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]*

Table 6.1 Attack Classes

6.1.2 Metrics Comparison

To Evaluate the proposed work, in their respective cases, a number of tools were used. Specifically, a series of quantitative metrics were utilized to quantify and compare the produced results on the problems of anomaly detection and anomaly classification. Of the metrics used in problems of machine learning and deep learning, in this work four representative ones were chosen to quantify the measured results, namely, a) Accuracy (ACC), b) True Positive Rate (TPR), c) False Positive Rate (FPR), and d) F1 Score (F1). From the metrics used, the ACC score describes the fraction of correctly classified samples, the TPR describes the number of detected or classified anomalies in both problems, the FPR describes the wrongly classified samples and lastly, the F1 describes the weighted average of the precision and recall of the produced results.

To establish the function of the two designed and implemented Deep Neural Network architectures and validating the results they produce, the outcome of the evaluation process

Model	ACC	TPR	FPR	F1
ABOD	0.2966	1.0000	0.8678	0.3501
Iforest	0.7692	0.9763	0.2792	0.6158
PCA	0.8594	0.9763	0.1679	0.7246
LOF	0.5703	1.0000	0.5301	0.4686
MCD	0.7293	0.9921	0.3321	0.5813
Iforestv2	0.7493	0.9789	0.3044	0.5966
LOFv2	0.1894	1.0000	1.0000	0.3185
ADA-GAN	**0.9432**	**0.9079**	**0.0486**	**0.8582**

Table 6.2 PDP01 Anomaly Detection Results

Model	ACC	TPR	FPR	F1
ABOD	0.2966	1.0000	0.8678	0.3501
Iforest	0.7692	0.9763	0.2792	0.6158
PCA	0.8594	0.9763	0.1679	0.7246
LOF	0.5703	1.0000	0.5301	0.4686
MCD	0.7293	0.9921	0.3321	0.5813
Iforestv2	0.7493	0.9789	0.3044	0.5966
LOFv2	0.1894	1.0000	1.0000	0.3185
ADA-GAN	**0.9432**	**0.9079**	**0.0486**	**0.8582**

Table 6.3 PDP02 Anomaly Detection Results

each network is consequently compared against known machine learning and deep learning algorithms.

6.1.2.1 ADA-GAN Comparison

In the case of Anomaly detection, the algorithms used to compare with the ADA-GAN network are i) Angle-Based Outlier Detection (ABOD) [50] [51], ii) Isolation Forest (Iforest) [52] [53], iii) Principal Component Analysis (PCA) [54], iv) Iforestv2, v) Minimum Covariance Determinant (MCD) [55] [56], vi) Local Outlier Factor (LOF) [57], vii) LOFv2.

In the tables 6.2-6.5, the Anomaly Detection results are depicted. In respect to the best detector, the proposed ADA-GAN Anomaly Detector, in most cases, produces a higher score on the detection of abnormal samples with a lower FPR rate. Specifically, it produces a higher Accuracy, True Positive Rate and F1 score with a lower False Positive rate in all cases, tables 6.2, 6.3, 6.4, except one where the LOF algorithm has a better performance, as can be seen in table 6.5.

Model	ACC	TPR	FPR	F1
ABOD	0.8391	0.9959	0.2003	0.7135
Iforest	0.8502	0.9510	0.1751	0.7186
PCA	0.8474	0.9612	0.1813	0.7169
LOF	**0.8732**	**0.9939**	**0.1572**	**0.7592**
MCD	0.8223	0.9918	0.2203	0.6918
Iforestv2	0.6976	0.9714	0.3713	0.5636
LOFv2	0.6422	0.9959	0.4468	0.5281
ADA-GAN	**0.9432**	**0.9079**	**0.0486**	**0.8582**

Table 6.4 PDP03 Anomaly Detection Results

Model	ACC	TPR	FPR	F1
ABOD	0.5813	0.9939	0.5220	0.4874
Iforest	0.7169	0.9484	0.3410	0.5730
PCA	0.7455	0.9788	0.3129	0.6063
LOF	0.5791	0.9970	0.5256	0.4868
MCD	0.7338	0.2103	0.1352	0.2403
Iforestv2	0.5319	0.9778	0.5797	0.4555
LOFv2	0.5256	0.9970	0.5924	0.4570
ADA-GAN	**0.8836**	**0.8716**	**0.1134**	**0.7499**

Table 6.5 PDP04 Anomaly Detection Results

A comparison between the proposed ADA-GAN architecture can also be observed in figures 6.7 and 6.8, where the produced metrics have been visualized and compared per metric category. In the former, the accuracy of the four datasets can be seen, with respect to the utilized models. The ADA-GAN architecture has a visually higher score than the rest of the algorithms. The same can be seen in the case of the figure 6.8, where the F1-Score is depicted.

Figures 6.9 and 6.10, also, reveal the TPR and FPR of the compared algorithms. When it comes to the TRP metric, ADA-GAN shows a relatively low score. This is due to the fact that the network has a low Sensitivity, or rather that is generalization is higher than the rest of the algorithms. In the case of the FPR, it shows the lowest score. This means that it is

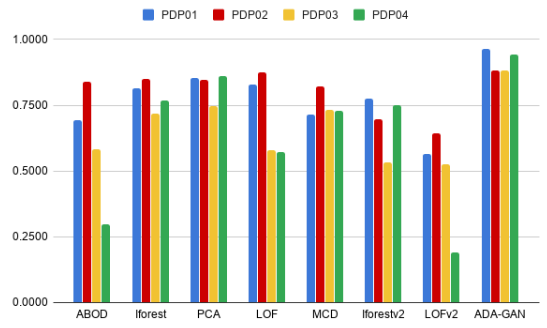


Figure 6.7 ADA-GAN Accuracy Comparison

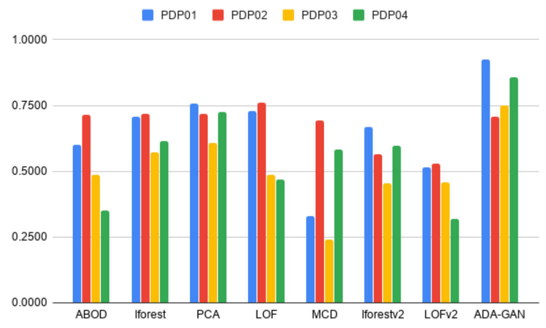


Figure 6.8 ADA-GAN F1-Score Comparison

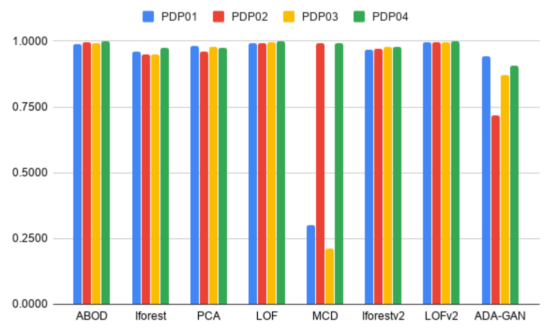


Figure 6.9 ADA-GAN TPR Comparison

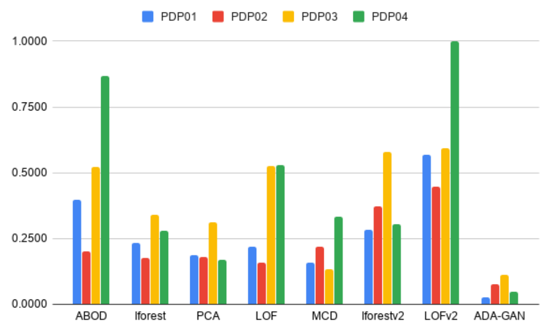


Figure 6.10 ADA-GAN FPR Comparison

more difficult for the network to falsely classify abnormal data as normal than, in correlation with the TPR, to classify normal data as abnormal. In the case of Intrusion Detection this is an acceptable contrast.

A statistical analysis of the quantitative metrics was performed. Table 6.6 lists the mean, variance and standard deviation of the ADA-GAN produced metrics. It can be seen that the ADA-GAN system keeps a high, or low, score depending on the metric, while on the same time keeping a stable output range. Specifically, the standard deviation reveals only a maximum of about 5% deviation of the metrics on the four different datasets.

6.1.2.2 CLA-GAN Comparison

For the Anomaly Classification use case the algorithms used for the comparison are i) Logistic Regression [58], ii) Linear Discriminant Analysis (LDA) [59], iii) Decision Tree Classifier

	Mean	**Variance**	**Standard Deviaton**
ACC	0.9181	0.001791534875	0.0423
TPR	0.8594	0.009922230038	0.0996
FPR	0.0670	0.001368770842	0.0370
F1	0.8104	0.009937333983	0.0997

Table 6.6 ADA-GAN Metric Statistics

Model	ACC	TPR	FPR	F1
Logistic Regression	0.9433	0.6030	0.0305	0.6030
LDA	0.9435	0.6046	0.0304	0.6046
Decision Tree Classifier	0.9642	0.7493	0.0193	0.7493
Gaussian NB	0.9283	0.4979	0.0386	0.4979
SVM RBF	0.9181	0.4266	0.0441	0.4266
SVM Linear	0.9219	0.4533	0.0421	0.4533
Random Forest	0.9477	0.6337	0.0282	0.6337
MLP	0.9387	0.5707	0.0330	0.5707
AdaBoost	0.8878	0.2143	0.0604	0.2143
Quadratic Discriminant Analysis	0.9420	0.5939	0.0312	0.5939
Dense DNN ReLU	0.9456	0.6191	0.0293	0.6191
Dense DNN Tanh	0.9456	0.6194	0.0293	0.6194
CLA-GAN	**0.9668**	**0.7679**	**0.0179**	**0.7679**

Table 6.7 PDP01 Anomaly Classification Results

Model	ACC	TPR	FPR	F1
Logistic Regression	0.9450	0.6148	0.0296	0.6148
LDA	0.9441	0.6086	0.0301	0.6086
Decision Tree Classifier	0.9642	0.7496	0.0193	0.7496
Gaussian NB	0.9381	0.5665	0.0333	0.5665
SVM RBF	0.9313	0.5189	0.0370	0.5189
SVM Linear	0.9301	0.5110	0.0376	0.5110
Random Forest	0.9473	0.6314	0.0284	0.6314
MLP	0.9406	0.5843	0.0320	0.5843
AdaBoost	0.9183	0.4281	0.0440	0.4281
Quadratic Discriminant Analysis	0.9447	0.6131	0.0298	0.6131
Dense DNN ReLU	0.9456	0.6191	0.0293	0.6191
Dense DNN Tanh	0.9444	0.6111	0.0299	0.6111
CLA-GAN	**0.9656**	**0.7592**	**0.0185**	**0.7592**

Table 6.8 PDP02 Anomaly Classification Results

[60], iv) Gaussian Naive Bayes (Gaussian NB) [61], v) Radial Basis Function Support Vector Machine (SVM RBF) [62], vi) Linear Support Vector Machine (SVM Linear) [62], vii) Random Forest [63], viii) Multilayer Perceptron (MLP) [64], ix) Adaptive Boosting (AdaBoost) [65], x) Quadratic Discriminant Analysis [66], xi) Dense DNN ReLU [67] [68] and xii) Dense DNN Tanh [67].

In the tables 6.7, 6.8, 6.9, and 6.10, in the Anomaly Classification problem, the proposed CLA-GAN classifier archives the highest score between all of the datasets. It surpasses the best classifier by 0.1-0.4% in Accuracy and up to 0.9-3% in F1 score, while keeping a high TP and low FP rate. Tables 6.7, 6.8, 6.9, 6.10 show the classification metric comparison.

In the case of the CLA-GAN architecture, not much can be said about the accuracy comparison of the algorithms. All of the utilized models show a small variance in the accuracy score as see in figure 6.11, with the CLA-GAN network having the highest accuracy score between the tested algorithms. In figure 6.12, though, an increased F1-Score for the CLA-GAN in all of the tested datasets, can be observed. This proves that CLA-GAN is more robust than the commonly used classification algorithms when used with Modbus network flow data.

Model	ACC	TPR	FPR	F1
Logistic Regression	0.9452	0.5889	0.0294	0.5889
LDA	0.9372	0.5290	0.0336	0.5290
Decision Tree Classifier	0.9608	0.7061	0.0210	0.7061
Gaussian NB	0.9417	0.5631	0.0312	0.5631
SVM RBF	0.9314	0.4855	0.0368	0.4855
SVM Linear	0.9326	0.4946	0.0361	0.4946
Random Forest	0.9452	0.5889	0.0294	0.5889
MLP	0.9413	0.5596	0.0315	0.5596
AdaBoost	0.9111	0.3333	0.0476	0.3333
Quadratic Discriminant Analysis	0.9372	0.5289	0.0336	0.5289
Dense DNN ReLU	0.9438	0.5781	0.0301	0.5781
Dense DNN Tanh	0.9403	0.5526	0.0320	0.5526
CLA-GAN	**0.9641**	**0.7307**	**0.0192**	**0.7307**

Table 6.9 PDP03 Anomaly Classification Results

Model	ACC	TPR	FPR	F1
Logistic Regression	0.9463	0.5972	0.0288	0.5972
LDA	0.9397	0.5480	0.0323	0.5480
Decision Tree Classifier	0.9605	0.7034	0.0212	0.7034
Gaussian NB	0.9421	0.5658	0.0310	0.5658
SVM RBF	0.9292	0.4686	0.0380	0.4686
SVM Linear	0.9336	0.5023	0.0356	0.5023
Random Forest	0.9497	0.6230	0.0269	0.6230
MLP	0.9417	0.5627	0.0312	0.5627
AdaBoost	0.9111	0.3333	0.0476	0.3333
Quadratic Discriminant Analysis	0.9372	0.5291	0.0336	0.5291
Dense DNN ReLU	0.9488	0.6163	0.0274	0.6163
Dense DNN Tanh	0.9402	0.5516	0.0320	0.5516
CLA-GAN	**0.9647**	**0.7350**	**0.0189**	**0.7350**

Table 6.10 PDP04 Anomaly Classification Results

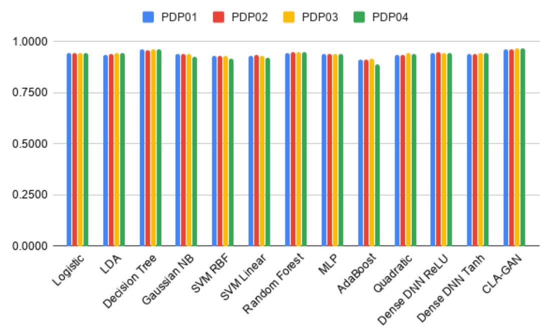


Figure 6.11 CLA-GAN Accuracy Comparison

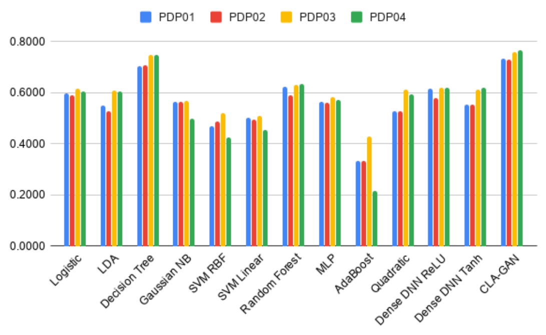


Figure 6.12 CLA-GAN F1-Score Comparison

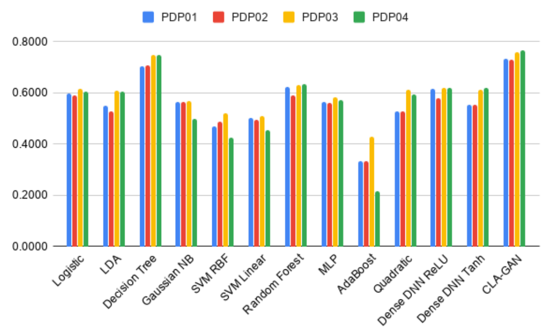


Figure 6.13 CLA-GAN TPR Comparison

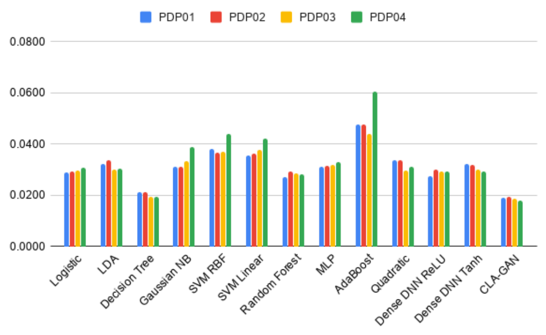


Figure 6.14 CLA-GAN FPR Comparison

Additionally, both the TPR and FPR scores of the CLA-GAN's architecture, in figures 6.13 and 6.14, are higher and lower respectively. As so, the CLA-GAN has a lower possibility of wrongly classifying a sample in the wrong class. Specifically, in the case of the TPR, it possesses a higher rate among the tested methods, that gives the CLA-GAN network the ability correctly classify each sample to its own category. Since the TPR does not fluctuate between the different datasets provided, it reveals a certain stability in this ability. In table 6.11 are presented the mean, variance and standard deviation. Like in the case of ADA-GAN, this network also shows good and consistent results regarding the produced metrics. In this instance, it is important to point out the variance of the network's metrics. As mentioned before, the data have a consistent number of features that are not unique to the specific dataset studied. Due to this, the produced metrics, and output of the network, are more robust and versed that the ADA-GAN network. As so, the variance for the CLA-GAN metrics is minimal.

From the produced results it is clear that both the ADA-GAN detector and the CLA-GAN classifier constitute adequate tools to tackle a variety of anomaly detection and anomaly classification problems around their respective use cases.

	Mean	**Variance**	**Standard Deviaton**
ACC	0.9653	0.000001447175008	0.0012
TPR	0.7482	0.000330270653	0.0182
FPR	0.0186	0.0000003578668212	0.0006
F1	0.7482	0.000330270653	0.0182

Table 6.11 CLA-GAN Metric Statistics

6.2 Experimental Configuration

Through experimentation during the different simulations run on the networks, the value of certain parameters and hyperparameters was obtained. To monitor the results of the Neural Networks a Tensorboard system was initialized. Tensorboard system is able to project in real-time certain plots encapsulated in the networks' training process.

6.2.1 ADA-GAN Configuration

6.2.1.1 Number of Epochs

The ADA-GAN network was set to run continuously for the duration of 2500 epochs on the four different Operational Data datasets. In contrast with the network flow data described bellow, the ADA-GAN network on the Operational Data reached quickly a saturation effect after only 160 epochs. The peak of its accumulation was reached at around 130-157 epochs, depending on the dataset.

6.2.1.2 Batch Size

The batch size of the ADA-GAN network is dependant on the number of features on the Operational Data. A number of 32, 64 and 128 were measured to have a steady effect on the Training process on the different datasets. Unfortunately, since this category of data possesses huge differences in size because of the different configuration of each Industrial facility, a general number of epochs cannot be extracted from these results.

6.2.2 CLA-GAN Configuration

6.2.2.1 Number of Epochs

The CLA-GAN network was set to also run continuously for the duration of 2500 epochs over the four different Modbus network flow datasets. Though almost all of the metrics described up to now were measured and plotted, the key metric that the number of epochs were decided on was the F1-Score. Figure 6.15 depicts the F1-Score curve throughout the epochs on the PDP04 dataset. 10 Training sessions on all four datasets were realized to obtain an overview of the performance.

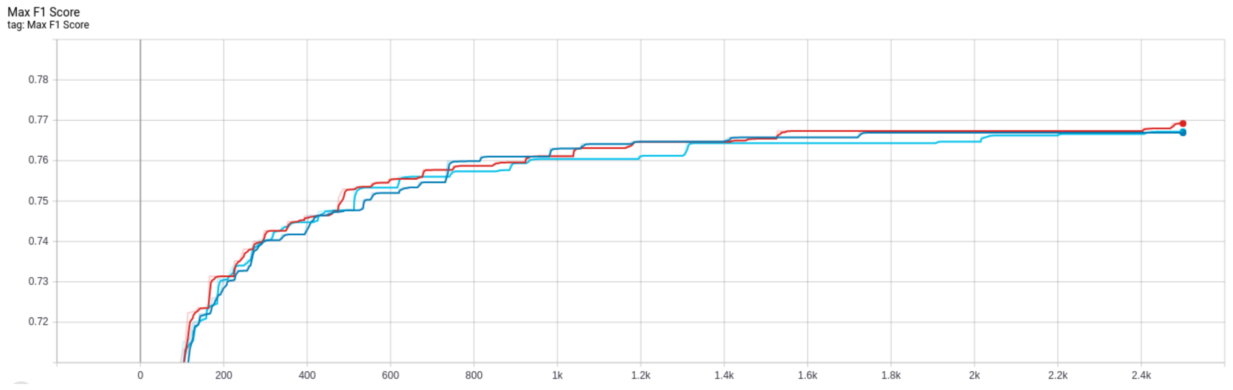


Figure 6.15 Anomaly Detection F1-Score Curve - PDP04

As can be seen, after about 1700 to 1800 iterations the networks begin to saturate with minimal progress. So it was decided that the optimal number of epochs for the ADA-GAN network on the Operational Data is in the range of $e\epsilon[1700, 1800]$.

6.2.2.2 Batch Size

After a number of simulations with different configurations, it was decided that a number of 256 is the optimal batch size for this network architecture. Usually, the batch size is a number in the power of 2, hence 256. Any batch size under 256 underfits the network, while any over 256 severally over-trains the network to the point of data memorization.

Chapter Seven

Federated Learning Extension - Design & Implementation

7.1 Design

In this section, the Federated Learning design of this work is described. The architectures that were developed in chapter 5: Federated Learning Extension - Design & Implementation, present only a centralized solution to be conducted in the heart of a security system. It doesn't have the ability to train in a distributed environment. By implementing the mentioned architectures with the Federated Learning model, the distributed learning of the classifications is possible with the added privacy preservation fortification that this process offers.

7.1.1 Basis

The Federated Learning methodology consists of a number of specialized systems that allow the interconnection, communication, organization and implementation of its intended purpose. In this section, two important modules are described. These modules, namely, the Orchestrator and the Worker, implemented in the Federated Learning Extension of this work are responsible for completing the aforementioned functions and constitute the main pillars of operation in this work.

7.1.1.1 Orchestrator

An Orchestrator is the module responsible for planning the training of the central model to the edge nodes and disperses either the model, or in the simulation of this work, both the data and the model to the distributed devices [69]. This system can be the central system of a Federated architecture or it can be an intermediate that works under it. In bigger systems there can be numerous Orchestrators, each responsible for different tasks.

In the Federated Learning part of this work, one Orchestrator was realized for each architecture, the ADA-GAN and the CLA-GAN. The module takes over pre-training the model before the Federated Training phase, distributing the respective datasets to the edge nodes, imposing the remote training process and finally, collecting the models and aggregating them through the Federated Averaging operation. Moreover, an evaluation function has been added to the Orchestrator to audit the performance of the aggregated models. The process of the Orchestrator as well as the data flow can be seen in figure 7.1.

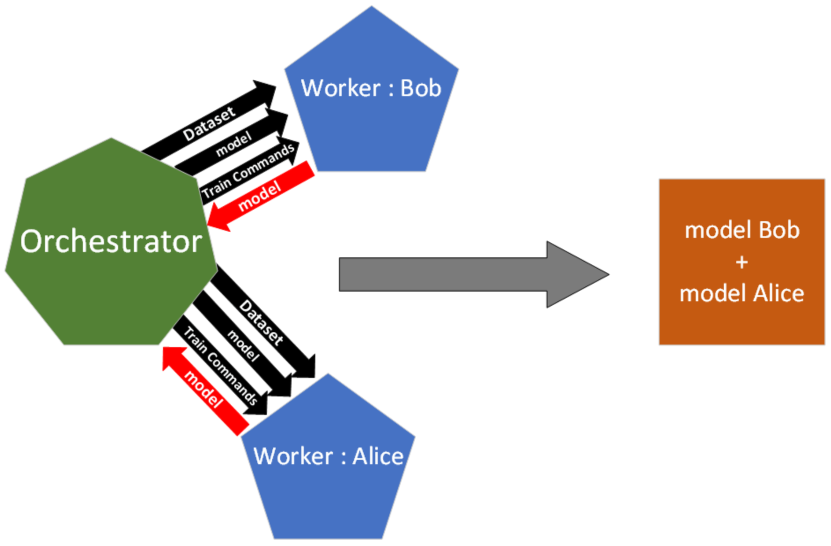


Figure 7.1 Federated Flow

7.1.1.2 Worker

A remote client or worker is the edge node of a Federated Learning cluster. It's the device on which the central system will send the model for training and will then collect to aggregate over the rest of the model from other edge nodes. A worker can be any device that has the capability to collect data and perform the computations required to train the given model.

In this work, as will be explained in the layout of the experiment testbed, virtual workers were used. Virtual workers are a part of the framework used to realize the Federated Learning extension of this work and have the same functionality as a normal worker, though they are spawn by the Orchestrator as a means to test the developing application. Since virtual workers are non-corporeal, meaning they do not own their own code base, they share a part of the main application, specifically designed to serve this purpose.

In this implementation, the remote workers first receive a portion of the dataset to be used in the training via a Federation dispersal and then receive a pre-trained model. The model was shortly trained on the centralized system and then sent to the workers for specification training. After the training process on the remote workers, the models are sent back to the Orchestrator which aggregates the model to its own, to produce a generalized model.

7.1.2 Federated Testbed

The Testbed of the Federated Learning operation is composed of a main Orchestrator, conducting the correct function for each of the use-cases, ADA-GAN and CLA-GAN, and two remote workers. The workers were given the names *bob* and *alice*, in respect to the involved APIs. The network topology of the testbed and the dataflow is depicted in figure 7.2.

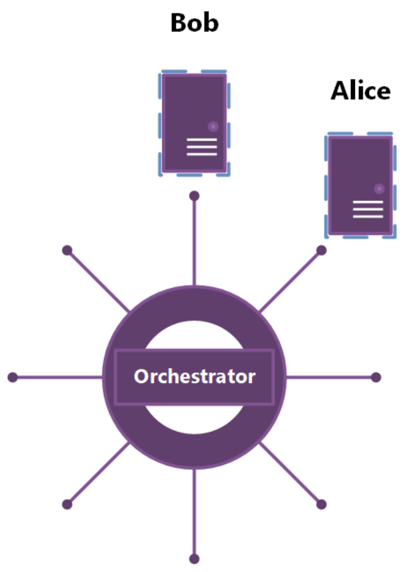


Figure 7.2 Federated Topology

7.1.3 Literature Review

Federated Learning is a relatively new and unexplored field of theoretical and applied distributed learning. Because of this fact, the literature targeting specific regions of computer science is all but limited. To conduct the Federated Extension of this work, a portion of the available literature was reviewed. This section summarizes the main related work to support the proposed architecture.

The authors in [70], propose a novel methodology for Federated Learning implementation for remote learning. They audit this methodology through an empirical performance evaluation of four common Deep Learning algorithms, deducing that through the Federated Averaging process, robust and high quality models can be extracted.

In [71], the authors develop a novel autonomous system able to detect compromised IoT devices. By utilizing device-specific profiles, acquired autonomously form the network, and by leveraging the Federated Training method to create a unified model, the system is successful in recognizing known and new, zero-day attacks, achieving a high accuracy score.

In [72] an Anomaly Detection system utilizing Deep Neural Autoencoders is presented. The system accumulated attack indicators, learning to identify possible attacks. At the same time, the proposed system tries to optimize network utilization for use with the Federated Learning methodology.

7.2 Implementation

In this section, the Implementation of the Federated Learning extension of this work is described.

7.2.1 Federated Learning Convention

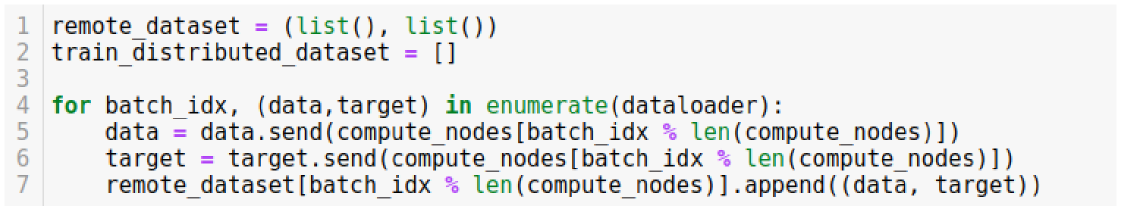
The defined network architectures that were outlined in the previous chapters, namely, the ADA-GAN and the CLA-GAN structures, were developed and tested using the Keras and Tensorflow frameworks that were analyzed in the Tools & Frameworks section. Even though these two tools constitute powerful tools for Deep Learning applications and have a wide range of applicability, they offer limited support for Federated Learning development, as of yet, that is only oriented in simulating the process. Because of this, to extend the proposed architectures to envelop the Federated Learning utility, the PyTorch and Syft frameworks were leveraged. To correlate with the functional APIs of both these tools, the produced Deep Neural Networks were re-written in using the PyTorch APIs. A lot of problems arose when following the transference procedure so certain conventions were made in order to present the following work as a proof of concept. These conventions are that a) the performance of the produced networks using the PyTorch framework is lower than the fully researched TensorFlow one, b)even though the datasets used are of the Anomaly Detection and Anomaly Classification use-cases, they are not necessarily the same and c) this extension is used to simulate a Federated Environment and is treated as a real one.

7.2.2 Pre-Federation Training

Before the Federated disbursal of the model to the corresponding virtual workers the both of the ADA-GAN and CLA-GAN networks are trained locally at the Orchestrator [34]. The process of training both of the networks is the same as implemented in section 5.2.2.5 and 5.2.3.4 respectively.

7.2.3 Dataset Federation

In a real Federated Environment, the models would be sent from the Orchestrator to the edge nodes and would train with the data collected by each device separately. In the simulation tested in this work, because the topology is virtual, there are no collected data from the virtual devices. To properly train the networks a dataset is sent, prior to the training process, to each virtual node to simulate a normal Federated Learning flow. Before sending, the data are sampled and separated into two different sets each for one of the machines so as to not train on the exact same data. This has no effect in the training process, though because the data come from the same source there is a certain homogeneity in the produced model and result. Figure 7.3 shows the use of the PyTorch API to Federate the dataset to each worker.



```
remote_dataset = (list(), list())
train_distributed_dataset = []

for batch_idx, (data,target) in enumerate(dataloader):
    data = data.send(compute_nodes[batch_idx % len(compute_nodes)])
    target = target.send(compute_nodes[batch_idx % len(compute_nodes)])
    remote_dataset[batch_idx % len(compute_nodes)].append((data, target))
```

Figure 7.3 Dataset Federation - Pytorch API

7.2.4 Federated Model Training

The process of the Federated Training takes place after the model has firstly trained on the central machine. The operation starts by Federating the models to the according edge

	Accuracy	Recall	Precision	F1 Score	AUC
Pre-Feterated	0.9131	0.8057	1.0000	0.8924	0.9028
Federated	0.9185	0.8236	1.0000	0.9033	0.9118

Table 7.1 ADA-GAN Federated Results

nodes. Then, since the data on which the models will be trained are already on the devices, it trains locally on these devices following the same training algorithm that it followed on the non-Federated Training process. Consequently, the models are collected by the central machine and go under the Federated Averaging process which aggregates the models into one main model. This procedure is repeated for a given number of Federated epochs.

7.2.5 Results

After the Federated Training process of the proposed networks, certain outcomes were observed. Firstly, the performance of the networks before the Federated operation and after remain almost the same. This is due to the fact that the virtual device corpus is too small for the Federated process to produce solid and advancing results. Another reason is that the utilized dataset is the same. An example is given in table 7.1 where the results before and after the Federated process, for the ADA-GAN networks, can be seen, respectively. Furthermore, it was also observed that due to the complex nature of the GAN's training procedure, a standard delay was introduced in the model and data communication to the remote workers. The synchronous nature of the implemented training process produces a standard training delay per model training iteration depending on the model size, training algorithm and data size. Because the testbed is virtual and realizes a simulation of the Federated Environment and also uses the same data samples as the non-Federated training, the evaluation has not produced satisfactory metric results, though the Federated procedure is successfully completed.

Chapter Eight

Conclusion & Future Work

8.1 Conclusion

In the modern digital age, data communication and transit collocate the majority of the digital processes. These data, coming in various forms, most of the time represent sensitive commercial or personal information. If this information falls into the hands of malicious third parties that try to exploit such data for malevolent purposes the results can be catastrophic. Especially in the Industrial application field, data privacy and preservation constitute one fundamental need for their correct and integral operation. To fortify their normal function, ID(P)S systems are employed to try and detect and in extent prevent the intrusion of possible attackers in their networks. They achieve this by leveraging algorithms from the field of Machine and Deep Learning, producing models trained specifically for detecting possible intrusions by classifying, predicting, or just detecting abnormal activity.

In this work two separate Deep Learning Neural Network architectures were designed, developed and tested, addressing a different part of Anomaly Detection or Anomaly Classification in the Industrial Environment operation. By using either network flow data or operational measurements from Industrial networks or equipment, they were trained to detect anomalous behaviour by learning from experience using abstract representations from previously detected intrusions. Their performance evaluation on the data of four different industrial facilities demonstrated that both network architectures are able to detect the majority of abnormalities with high evaluation metrics. Specifically, the networks were actively compared against several common and widespread Machine and Deep Learning algorithms that are often used to solve the aforementioned problems. In the case of Anomaly Detection, it was observed that the ADA-AGN architecture achieves a higher score in the produced metrics in all the cases except one, in which it is surpassed by the LOF algorithm. In the Anomaly Classification the CLA-GAN network produces higher metric scores that all of the tested algorithms. A consequent statistical analysis of the resulting metrics followed. This analysis pointed to the robustness of the proposed networks and their ability to produce consistent results throughout the different datasets given. In both cases, the networks showed a tendency to generalize that indicates the reason for their high capacitance to classify the anomalies in the correct way and produces a more robust and reliable model for the two architectures.

Furthermore, to consolidate the preservation of the privacy and anonymity of the data needed to train such Neural Networks, their architecture was extended to envelop the decentralized training method of Federated Learning. By distributing the models to a corpus of remote devices and consequently training them locally at those devices, the need for data collection for unified centralized training becomes obsolete. Through the implementation of a virtual corpus topology, it was deducted that the Federated Learning method can become a powerful tool for decentralized Deep Learning and can evolve the notion of remote network protection through the use of Artificial Intelligence. The ability to disburse the produced models to remote edge nodes for on-device training was realized and tested. Even though, due to physical topology and data variety, the produced Federated metric results had a slight or none progress in contrast to the non-Federated results, a critical insight to the Federated process of the two networks was acquired. The process was realized successfully, by distributing, training and then collecting the produced models back to the centralized system.

8.2 Future Work

The work performed in this thesis can be extended and become a stepping stone for further research in the field of Intrusion Detection using Deep Learning and Federated Learning IDSs. Some of the possible improvements that can be realized based on this work are the following:

- Further experiment with the structures of the ADA-GAN and CLA-GAN networks to possibly optimize the produced metrics and each network's generalization
- Extend the architectures of the produced Neural Networks to support more industrial communication protocols, like, the a) IEC-104, b) BackNet, c)MQTT, d) Profinet protocols or other forms of timeseries data like the Operational Data.
- Build and test the algorithms in a real Federated Learning Environment and corpus populated from a large number of remote devices
- Reinforce the Federated Learning technique with more efficient data privacy and security protocols

REFERENCES

[1] Paul Innella. *The Evolution of Intrusion Detection Systems*. Jan. 2006.

[2] router-switch.com. *What is a LAN: Concept, Features, Topologies and Setting*. URL: https://www.router-switch.com/faq/what-is-lan-concept-features-topology-setting.html (visited on 05/26/2020).

[3] router-switch.com. *What is metropolitan area network?* URL: https://www.router-switch.com/faq/metropolitan-area-network-definition.html (visited on 05/26/2020).

[4] Network Encyclopedia. *What is a Wide Area Network (WAN)?* URL: https://networkencyclopedia.com/wide-area-network-wan/ (visited on 05/26/2020).

[5] Markel Sainz Oruna, Mikel Iturbe, Iñaki Garitano, and Urko Zurutuza. "Software Defined Networking Opportunities for Intelligent Security Enhancement of Industrial Control Systems". In: (Jan. 2018), pp. 577–586. DOI: 10.1007/978-3-319-67180-2_56.

[6] Vangie Beal. *The 7 Layers of the OSI Model*. URL: https://www.webopedia.com/quick_ref/OSI_Layers.asp (visited on 05/26/2020).

[7] Peter Huitsing, Rodrigo Chandia, Mauricio Papa, and Sujeet Shenoi. "Attack taxonomies for the Modbus protocols". In: *International Journal of Critical Infrastructure Protection* 1 (Dec. 2008), pp. 37–44. DOI: 10.1016/j.ijcip.2008.08.003.

[8] B Claise, Brian Trammell, and P Aitken. "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information". In: (Sept. 2013).

[9] Brian Trammell and Elisa Boschi. "Bidirectional Flow Export Using IP Flow Information Export (IPFIX)". In: (Jan. 2008).

[10] Andreas Fink and Torsten Reiners. "Modeling and solving the short-term car rental logistics problem". In: *Transportation Research Part E: Logistics and Transportation Review* 42 (July 2006), pp. 272–292. DOI: 10.1016/j.tre.2004.10.003.

[11] Jun li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. "Real-Time P2P Traffic Identification". In: (Jan. 2008), pp. 2474–2478. DOI: 10.1109/GLOCOM.2008.ECP.475.

[12] Simon Hansman and Ray Hunt. "A taxonomy of network and computer attacks". In: *Computers Security* 24 (Feb. 2005), pp. 31–43. DOI: 10.1016/j.cose.2004.06.011.

[13] Chris Simmons, Charles Ellis, S. Shiva, Dipankar Dasgupta, and Chase Wu. “AVOIDIT: A Cyber Attack Taxonomy”. In: (Jan. 2009).

[14] CSIS. *"Significant Cyber Incidents"*. URL: https://www.csis.org/programs/technology-policy-program/significant-cyber-incidents (visited on 05/26/2020).

[15] Hung-Jen Liao, Chun-Hung Lin, Ying-Chih Lin, and Kuang-Yuan Tung. “Intrusion detection system: A comprehensive review”. In: *Journal of Network and Computer Applications* 36 (Jan. 2013), pp. 16–24. DOI: 10.1016/j.jnca.2012.09.004.

[16] Rebecca Bace and Peter Mell. *Intrusion Detection Systems*. 2001.

[17] LLC SolarWinds Worldwide. *IDS vs. IPS: What’s the Difference?* URL: https://www.dnsstuff.com/ids-vs-ips.

[18] Eric Conrad, Seth Misenar, and Joshua Feldman. *Eleventh Hour CISSP, Third Edition: Study Guide*. 3rd. Syngress Publishing, 2016. ISBN: 0128112484.

[19] Karen Scarfone and Peter Mell. *Recommendations of the National Institute of Standards and Technology*. 2007. DOI: 10.6028/NIST.SP.800-94.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[21] Ahmad Javaid, Quamar Niyaz, Weqing Sun, and Mansoor Alam. “A Deep Learning Approach for Network Intrusion Detection System”. In: *EAI Endorsed Transactions on Security and Safety* 3 (Dec. 2015). DOI: 10.4108/eai.3-12-2015.2262516.

[22] Tuan Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. “Deep Learning Approach for Network Intrusion Detection in Software Defined Networking”. In: (Oct. 2016). DOI: 10.1109/WINCOM.2016.7777224.

[23] Mohamad H. Hassoun. “Fundamentals of Artificial Neural Networks”. In: *Proceedings of the IEEE* 84.6 (1996), pp. 906–.

[24] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/neural-networks-1/.

[25] Quoc Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Ng. “On Optimization Methods for Deep Learning”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* 2011 (Jan. 2011), pp. 265–272.

[26] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[27] Olga Wichrowska, Niru Maheswaranathan, Matthew Hoffman, Sergio Gómez, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. “Learned Optimizers that Scale and Generalize”. In: (Mar. 2017).

[28] Matthew Zeiler. “ADADELTA: An adaptive learning rate method”. In: 1212 (Dec. 2012).

[29] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN: 0321321367.

[30] Shane Barratt and Rishi Sharma. “A Note on the Inception Score”. In: (Jan. 2018).

[31] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. “Self-Attention Generative Adversarial Networks”. In: (May 2018).

[32] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. McMahan, Timon Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. “Towards Federated Learning at Scale: System Design”. In: (Feb. 2019).

[33] Arjun Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. “Analyzing Federated Learning through an Adversarial Lens”. In: (Nov. 2018).

[34] Brendan McMahan and Daniel Ramage. *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. 2017. URL: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[35] *An end-to-end open source machine learning platform*. URL: https://www.tensorflow.org/.

[36] *Simple. Flexible. Powerful.* URL: https://keras.io/.

[37] *scikit-learn Machine Learning in Python*. URL: https://scikit-learn.org/stable/.

[38] *FROM RESEARCH TO PRODUCTION*. URL: https://pytorch.org/.

[39] *The fundamental package for scientific computing with Python*. URL: https://numpy.org/.

[40] *Pandas*. URL: https://pandas.pydata.org/.

[41] *Matplotlib: Python plotting — Matplotlib 3.2.2 documentation*. URL: https://matplotlib.org/.

[42] *ANSWER QUESTIONS USING DATA YOU CANNOT SEE*. URL: https://www.openmined.org/.

[43] Yin Chuan-long, Zhu Yue-fei, Fei Jin-long, and He Xin-zheng. “A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks”. In: *IEEE Access* PP (Oct. 2017), pp. 1–1. DOI: 10.1109/ACCESS.2017.2762418.

[44] R Vinayakumar, Alazab Mamoun, Kp Soman, Poornachandran Prabaharan, A. Al-Nemrat, and Venkatraman Sitalakshmi. “Deep Learning Approach for Intelligent Intrusion Detection System”. In: *IEEE Access* PP (Apr. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2895334.

[45] Eunbi Seo, Hyun Song, and Huy Kang Kim. “GIDS: GAN based Intrusion Detection System for In-Vehicle Network”. In: (July 2019).

[46] Ondrej Linda, Todd Vollmer, and Milos Manic. “Neural Network based Intrusion Detection System for critical infrastructures”. In: *Proceedings of the International Joint Conference on Neural Networks* (June 2009), pp. 1827–1834. DOI: 10.1109/IJCNN.2009.5178592.

[47] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil Bharath. “Generative Adversarial Networks: An Overview”. In: *IEEE Signal Processing Magazine* 35 (Oct. 2017). DOI: 10.1109/MSP.2017.2765202.

[48] Yongjun Hong, Uiwon Hwang, Jaeyoon Yoo, and Sungroh Yoon. “How Generative Adversarial Nets and its variants Work: An Overview of GAN”. In: *ACM Computing Surveys* 52 (Nov. 2017). DOI: 10.1145/3301282.

[49] *Python.org*. URL: https://www.python.org/.

[50] Ninh Pham and Rasmus Pagh. “A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug. 2012). DOI: 10.1145/2339530.2339669.

[51] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. “Angle-based outlier detection in high-dimensional data”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug. 2008), pp. 444–452. DOI: 10.1145/1401890.1401946.

[52] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. “Isolation Forest”. In: (Jan. 2009), pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[53] Zhiguo Ding and Minrui Fei. “An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data Using Sliding Window”. In: (Sept. 2013), pp. 12–17. DOI: 10.3182/20130902-3-CN-3020.00044.

[54] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and Liwu Chang. “A Novel Anomaly Detection Scheme Based on Principal Component Classifier”. In: *Proceedings of International Conference on Data Mining* (Jan. 2003).

[55] Peter Rousseeuw and Katrien Driessen. "A Fast Algorithm for the Minimum Covariance Determinant Estimator". In: *Technometrics* 41 (Aug. 1999), pp. 212–223. DOI: 10.1080/00401706.1999.10485670.

[56] Johanna Hardin and David Rocke. "Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator". In: *Computational Statistics Data Analysis* 44 (Jan. 2004), pp. 625–638. DOI: 10.1016/S0167-9473(02)00280-3.

[57] David Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. "Incremental Local Outlier Detection for Data Streams". In: *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2007* (Jan. 2007), pp. 504–515. DOI: 10.1109/CIDM.2007.368917.

[58] Yun Wang. "A multinomial logistic regression modeling approach for anomaly intrusion detection". In: *Computers Security* 24 (Nov. 2005), pp. 662–674. DOI: 10.1016/j.cose.2005.05.003.

[59] Kang Kim, Heung Choi, Chang Moon, and Chi-Woong Mun. "Comparison of k-nearest neighbor, quadratic discriminant and linear discriminant analysis in classification of electromyogram signals based on the wrist-motion directions". In: *Current Applied Physics - CURR APPL PHYS* 11 (May 2011), pp. 740–745. DOI: 10.1016/j.cap.2010.11.051.

[60] Muhammad Shafiq, Xiangzhan Yu, Asif Laghari, Lu Yao, Nabin Karn, and Foudil Abdessamia. "Network Traffic Classification techniques and comparative analysis using Machine Learning algorithms". In: (Oct. 2016), pp. 2451–2455. DOI: 10.1109/CompComm.2016.7925139.

[61] Nigel Williams, Sebastian Zander, and Grenville Armitage. "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification". In: *SIGCOMM Comput. Commun. Rev.* 36.5 (Oct. 2006), pp. 5–16. ISSN: 0146-4833. DOI: 10.1145/1163593.1163596. URL: https://doi.org/10.1145/1163593.1163596.

[62] Michaela Bray and Dawei Han. "Identification of support vector machines for runoff modelling". In: *Journal of Hydroinformatics* 6 (Oct. 2004), pp. 265–280. DOI: 10.2166/hydro.2004.0020.

[63] Yaping Chang, Wei Li, and Zhongming Yang. "Network Intrusion Detection Based on Random Forest and Support Vector Machine". In: (July 2017), pp. 635–638. DOI: 10.1109/CSE-EUC.2017.118.

[64] Zahra Jadidi, Vallipuram Muthukkumarasamy, Elankayer Sithirasenan, and Mansour Sheikhan. "Flow-Based Anomaly Detection Using Neural Network Optimized with GSA Algorithm". In: *Proceedings - International Conference on Distributed Computing Systems* (July 2013), pp. 76–81. DOI: 10.1109/ICDCSW.2013.40.

[65] Weiming Hu, Wei Hu, and Stephen Maybank. "AdaBoost-Based Algorithm for Network Intrusion Detection." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 38 (Apr. 2008), pp. 577–583.

[66] Sheraz Naseer, Dr. Yasir Saleem, Shehzad Khalid, Mr Khawar, Jihun Han, M Iqbal, and Kijun Han. "Enhanced Network Anomaly Detection Based on Deep Neural Networks". In: *IEEE Access* (Aug. 2018), pp. 1–1. DOI: 10.1109/ACCESS.2018.2863036.

[67] Jin Kim, Nara Shin, Seung Jo, and Sang Kim. "Method of intrusion detection using deep neural network". In: (Feb. 2017), pp. 313–316. DOI: 10.1109/BIGCOMP.2017.7881684.

[68] Abien Fred Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: (Mar. 2018).

[69] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. "Advances and Open Problems in Federated Learning". In: (2019). arXiv: 1912.04977 [cs.LG].

[70] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: (2016). arXiv: 1602.05629 [cs.LG].

[71] Thien Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad-Reza Sadeghi. "DÏoT: A Federated Self-learning Anomaly Detection System for IoT". In: (July 2019), pp. 756–767. DOI: 10.1109/ICDCS.2019.00080.

[72] Joseph Schneible and Alex Lu. "Anomaly detection on the edge". In: (Oct. 2017), pp. 678–682. DOI: 10.1109/MILCOM.2017.8170817.