1 2 9 0

UNIVERSIDADE Đ
COIMBRA

João Bernardo do Nascimento Domingues

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR SECURITY AND PRIVACY IN CLOUD-NATIVE ENVIRONMENTS

Dissertation in the context of the Master in Data Science and Engineering, advised by Professor Marco Simões and Engineer Luís Cordeiro and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2023

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Ð
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

João Bernardo do Nascimento Domingues

Artificial Intelligence and Machine Learning for security and privacy in Cloud-Native environments

Dissertation in the context of the Master in Data Science and Engineering, advised by Prof. Marco Simões and by Eng. Luís Cordeiro and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

João Bernardo do Nascimento Domingues

Inteligência Artificial e Machine Learning para segurança e privacidade em ambientes Cloud-Native

Dissertação no âmbito do Mestrado em Engenharia e Ciência de Dados, orientada pelo Professor Doutor Marco Simões e pelo Engenheiro Luís Cordeiro apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho 2023

Acknowledgements

I would like to thank Prof. Marco Simões for the availability, guidance and support throughout the period of dissertation as well as to the OneSource advisors Eng. Luis Cordeiro, Eng. Luís Rosa and Eng. Pedro Tomás for the constant monitoring and advice through the whole internship.

A word of appreciation to my family and girlfriend who supported me during the most stressful times and throughout my academic career.

A final note to my friends and colleagues who stood by my side and helped me through all the highs and lows, thank you for all the patience and support.

Abstract

The usage of cloud-native applications has been growing in the last few years, despite this they still have some challenges, with security being one of them. In order to diminish this problem, Artificial Intelligence-based security solutions have been recently proposed and taken more into consideration since it shows notable results in identifying and responding to threats released in a network. Network anomaly detection being the process of monitoring network data and detecting abnormal events that may occur, it is the main technique we used in our work

This work has the goal of researching, designing and building an AI model for network anomaly detection as part of the development of a Holistic Security and Privacy Framework in the context of the CHARITY and 5G-EPICENTRE EU-funded research projects. This framework is intended to automate, detect and mitigate anomalies, such as cyber-attacks, in cloud-based environments. In this report, we provide a review of several research topics, such as Cloud-based environments, network security and the state of the art of Machine Learning for Network Anomaly Detection, where we present and discuss various different Machine Learning, and Deep Learning approaches.

Using the Supervised Learning algorithm, Random Forest, as a baseline model that achieved a good performance in detecting attacks, we compared the performance of the same algorithm with a Conventional and Convolutional Autoencoder in detecting unknown attacks and reached the conclusion that the Random Forest had a worse performance, showing that an Unsupervised Learning approach, achieving an F1-Score in the order of 80%, was better than a Supervised Learning approach as the latter one cannot identify well enough attacks that were unknown to the model, achieving an F1-Score of 27%.

The Conventional Autoencoder presented an F1-Score value above 69% in all different types of data except for the Email, which the performance was bad compared to the other types of data. The Convolutional Autoencoder presented and F1-Score value above 59% in all of the models trained for each type of data. Comparing the Conventional Autoencoder with the Convolutional, we could conclude that both algorithms have a very similar performance overall, achieving similar AUC scores, as well as the classification time. Since the objective of the framework was to periodically train the model we were using with new data, in order for it to keep improving, the training time of the model was a really important aspect, and in this case the Convolutional Autoencoder took much more time than the Conventional Autoencoder.

Keywords

Machine Learning (ML), Network Anomaly Detection, Privacy, Autoencoders, Unsupervised Learning.

Resumo

A utilização de aplicações Cloud-native tem vindo a crescer nos últimos anos, ainda assim estes têm os seus problemas, um deles a segurança. De maneira a diminuir este problema, foram recentemente propostas e tidas mais em consideração soluções baseadas em Inteligência Artificial para segurança, visto que mostram resultados notáveis na identificação e resposta a ameaças presentes numa rede.

Este trabalho tem o objetivo de pesquisar, conceber e construir um modelo de IA para a detecção de anomalias de rede como parte do desenvolvimento de uma Framework de Segurança e Privacidade Holística no contexto dos projectos de investigação financiados pela UE, CHARITY e 5G-EPICENTRE. Esta framework destina-se a automatizar, detectar e mitigar anomalias, tais como ciberataques, em ambientes Cloud-Native. Neste relatório, fornecemos ainda uma análise de vários tópicos de investigação, tais como ambientes Cloud-native, segurança de redes e o estado da arte de Machine Learning para deteção de anomalias em redes, onde apresentamos e discutimos várias abordagens baseadas em Machine Learning e Deep Learning.

Utilizando o algoritmo de Supervised Learning, Random Forest, como modelo de referência, que obteve um bom desempenho na deteção de ataques, comparamos o desempenho deste algoritmo com um Conventional Autoencoder e um Convolutional Autoencoder e chegámos assim à conclusão que o Random Forest demonstrou um desempenho pior em comparação com os Autoencoders, o que mostra que as abordagens Unsupervised Learning, que atingiram um F1-Score na ordem dos 80%, foi melhor do que uma abordagem Supervised Learning, uma vez que esta última não conseguiu identificar adequadamente ataques desconhecidos ao modelo, alcançando um F1-Score de 27%.

O Conventional Autoencoder apresentou um F1-Score acima de 69% em todos os diferentes tipos de dados, exceto no Email, cuja performance foi má em comparação com os outros tipos de dados. O Convolutional Autoencoder apresentou um F1-Score acima de 59% em todos os models treinados para cada tipo de dados. Ao comparar o Conventional Autoencoder com o Convolutional Autoencoder, podemos concluir que ambos os algoritmos têm um desempenho muito semelhante, conseguindo valores de AUC semelhantes, assim como tempos de classificação. Sendo que o objetivo da framework era treinar periodicamente o modelo com novos dados, para que ele continuasse a melhorar, o tempo de treino do modelo era um aspeto realmente importante, e, nesse caso, o Convolutional Autoencoder levou muito mais tempo do que o Conventional Autoencoder.

Palavras-Chave

Machine Learning(ML), Deteção de anomalias em redes, Segurança, Autoencoders, Unsupervised Learning.

Contents

Acronyms

5G-EPICENTRE	ExPerimentation Infrastructure hosting Cloud-nativE Netapps for public proTection and disaster REliefing.
AI	Artificial Intelligence.
ANN	Artificial Neural Networks.
AUC	Area Under the Curve.
CHARITY	Cloud for Holographic and Augmented RealITY.
DEI	Department of Informatics Engineering.
DoS	Denial of Service.
DT	Decision Tree.
FAR	False Alarm Rate.
FCTUC	Faculty of Sciences and Technology of the University of Coimbra.
FN	False Negative.
FNR	False Negative Rate.
FP	False Positive.
GAN	Generative Adversarial Networks.
GMM	Gaussian Mixture Model.
GRU	Gated Recurrent Unit.
IAT	Inter-arrival Time.
IDS	Intrusion Detection Systems.
IF	Isolation Forest.
KNN	K-Nearest Neighbours.
LDA	Linear Discriminant Analysis.
LR	Logistic Regression.
LSTM	Long-Short Term Memory.

MECD Master in Data Science and Engineering.

MIC Maximal Information Coefficient.

ML Machine Learning.

NAD Network Anomaly Detection.

NB Naive Bayes.

NetApps Network Applications.

NLP Natural Language Processing.

NN Neural Networks.

OCSVM One Class Support Vector Machine.

OPA Open Policy Agent.

PCA Principal Component Analysis.

PSH Push.

RF Random Forest.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

ROC Receiver Operating Characteristic.

SHAP Shappley Additive Explanations.

SMEs small and medium enterprises.

SSC Sub-Space Clustering.

TN True Negative.

TP True Positive.

URG Urgent-Pointer.

XR Extended Reality.

List of Figures

List of Tables

Chapter 1

Introduction

In the context of the thesis "Artificial Intelligence and Machine Learning for security and privacy in Cloud-Native environments" and the dissertation of Master in Data Science and Engineering (MECD), by the Department of Informatics Engineering (DEI), Faculty of Sciences and Technology of the University of Coimbra (FCTUC) in the school year 2022/2023.

This work was developed at OneSource [1], with the objective of analysing and developing approaches in the area of Machine Learning with the purpose of improving the security and privacy of Cloud-Native environments.

1.1 Background

Maintaining the privacy and security when it comes to online communication is one of the most critical aspects in the context of professional and day-to-day life, so it is important that all applications used have a good and reliable security framework. Artificial Intelligence (AI) has been used, in recent years, as a tool in the assurance of the security of these applications, by detecting possible threats as well as giving feedback on the causes and possible solutions to these attack attempts. The research that will be conducted in this report will be integrated in two projects, the ExPerimentation Infrastructure hosting Cloud-nativE Netapps for public proTection and disaster RElief (5G-EPICENTRE) project [2] and the Cloud for Holographic and Augmented RealITY (CHARITY) project [3].

Due to the increasing number of interactions in Cloud-Native environments and the growing capacity of existing breaches and attacks in these environments, it is urgent to have a reliable tool to assist with the problem, deciding if an instance is normal activity or an anomaly or an attempt of an attack to the network, so we will address in our work network anomaly detection. ML has a crucial role in this problem, create a model capable of automating and improving the process of attacks detection, due to its capability of processing big loads of data and provide precise real-time alerts.

5G-EPICENTRE is an EU-funded project which its objective is to create a platform that aims to lower the difficulty of small and medium enterprises (SMEs) in entering 5G markets, enabling them to build and test their solutions without high costs. The platform will be as an open source repository for PPDR 5G Network Applications (NetApps) which will provide open access to 5G networks resources where enterpises can have access to those resources in a simple and cost effective way [4].

CHARITY is another project funded by the European Union which the main goal is to create a novel framework capable of integrating the benefits of network continuum autonomous orchestration of cloud, edge, and network resources in order to handle simultaneously low and high latency infrastructures. This open-source infrastructure is being designed to be able to deal and answer, in an effective way, to the needs of emerging applications such as holographic events, virtual reality training, and mixed reality entertainment. The framework relies on various technologies related to cloud in order to offer a complete environment capable of delivering NextGen applications [5].

With the continuous growth of 5G technologies and the use of cloud-based technologies, there is a need to carefully adapt in terms of security so that there will not be breaches, that is the task that OneSource is responsible for, be in charge of the security aspects of the NetApps. The company is tasked with developing a framework focused on the detecting and handling network attacks and intrusion attempts that will then be integrated in the overall projects that are CHARITY and 5G-EPICENTRE.

As the main technologies are cloud-based it is of maximum importance to provide safe and secure cloud end-to-end communications and service delivery. The CHARITY project has a specific area where the security and privacy of the framework is the focal point, security and privacy-aware orchestration. In this context, the work being developed by OneSource, and the one that we will detail in my report will be integrated in this project with the finality of securing a safe end-to-end communication.

Under the context of the problem and these two projects, OneSource is developing a framework, with the intent of improving the security and privacy of Cloud-Native environments in order to find a reliable solution for network anomaly detection.

1.2 Motivation

The fast growing development of cloud-based environments has created a vast surface for cybercriminals to potentially deploy more destructive cyber-attacks. As a result of this, it has been noted an exponential increase in cyber-attacks. Many of these attacks have been fulfilling their malicious purpose effectively because the attacks created have been taking novel and innovative techniques.

According to a survey highlighted in [6], the human attack surface was to reach 6 billion people by 2022 and Cyber-crime damage costs were expected to hit 6 trillion dollars annually by 2021. In order to decrease this numbers, AI has been regarded as an important tool in Cybersecurity, it is expected to grow 23.6% from 2020 to 2027.

The cloud-native approach seeks projecting, build and execute virtual vital functions in the Cloud model, where applications are developed with aid of tools which will maximize their benefits. This benefits include bigger development agility, integration and installation which are built with continuous integration, containers and orchestrators, with Kubernetes being the focal technology of this approach. Nowadays large distributed applications require an intelligent and automated management, because there is even more necessity of taking into account applications and infrastructure security and privacy. One of the most important methods to keep the security of an application or infrastructure is network anomaly detection, which its focal point is to detect out of the ordinary instances in a network, such as cyber attacks.

Machine learning suits the problem of anomaly detection because there is a need of handling large amounts of data, which ML performs well on, in comparison with other methods, it has good efficiency too when it comes to dealing with unstructured data, information that it is not arranged in any specific way and is able to process most types of data. Finally ML-based anomaly detection is the fact that it works well in real time scenarios, which is an aspect of great importance in the research we are conducting.

The main motivation of this study is to tackle the attempts of environment intrusion in order to increase the privacy and security of systems as preserving privacy of cloud-based interactions is key, taking into consideration the direction and growth this technologies have been taking in the past years. To do that we evaluated ML models focused on classifying network intrusions and attacks and integrated one of them in the framework that is being developed.

1.3 Problem Statement

The main problem, to be solved in our work, is how can machine learning make a trustworthy classification of the intrusion attempts on these cloud-based environments? So that the ML approach could be considered a feasible one it had to classify correctly the biggest number of instances possible, either normal activity or anomalies so that whenever there was an attempt of an attack to the cloud environment, it could be taken down. Another important, but more specific, problem is how could we detect accurately unknown attacks, that may be zero-day attacks which are new attacks unidentified and unknown to the detection system, or common attacks but the model was never exposed to them. The classification model used, had to be able to detect new anomalies never before seen by it. One final important problem is how could we make the classification fast enough so that it was viable with real-time data?

The main challenges we found in our work were:

- **Correctly Classify Unknown Attacks:** One of the most important problems is classifying attacks unknown to the model, it may become difficult to catch these intrusions, since the algorithm never had contact with them it does not have a pattern to guide itself by and may have a pattern similar to instances considered as normal, making the classification of these attacks tricky.

- **Imbalanced Datasets:** In this type of problem, network anomaly detection, the datasets usually are very imbalanced, due to the fact that intrusions are not as common as normal activity.

- **Feature Extraction:** A common problem in every ML based tasks is Feature Extraction, in our study it is as well. Since the datasets we will use have lots of different features, this may be an issue, as the model created can overfit and the computational and time complexity may be too high.

- **Performance Evaluation:** In order to evaluate the models that we will use, the correct selection of metrics that we want to maximize and minimize is key.

1.4 Objectives

The main objectives of this work were the following:

1. Analyse the existing studies on the topic of network anomaly detection and select a few candidate approaches to assess and evaluate.

2. Integrate at least one of the assessed approaches developed by us in the project framework.

3. Evaluate the performance of the approach, comparing it with existing state of the art algorithms with the help of the datasets for network intrusion detection and suggest improvements.

1.5 Document outline

The document was divided into seven main chapters. The structure of the remaining ones is the following.

In Chapter 2 we provide a literature review of the state of the art of network intrusion detection, we go through multiple existing machine learning approaches and the pipeline since the data gathering to the reliability and accuracy of each model.

In Chapter 3 we present the projects in which our work will be integrated, the candidate approaches implemented in the our study, as well as the datasets that were used.

In Chapter 4 we present the preprocessing made in the datasets, the implementation of each model as well as the tests performed in each one.

Chapter 5 presents the results achieved by each model and a deep analysis and discussion of thoses results in order for us to select the best approach to use in our platform.

In Chapter 6 we present a brief introduction of how our models will be integrated in the platform and how will the handling of the attacks will be made.

Finally chapter 7 presents the conclusions we took from our work as well as brief summary of the results we achieved.

Chapter 2

State of the Art

This chapter focuses on the state of the art of the Machine Learning concepts, approaches and techniques used in network anomaly detection. Section 2.1 addresses, in a general way the context of network security, and gives an overview on how this work integrates itself in the security part of a cloud-based environment. Section 2.2 addresses the two main types of approaches in Network Anomaly Detection, signature-based and anomaly-based.

The section 2.4 focuses on the most common techniques for the data preprocessing and section 2.3 in the algorithms used in different anomaly detection approaches, analysing in detail Random Forest (RF), Artificial Neural Networks (ANN), Isolation Forest (IF), Clustering techniques, Neural Networks (NN), such as Conventional Autoencoders and Generative Adversarial Networks (GAN). Section 2.5 addresses the evaluation metrics used to check the reliability of the strategies and the final section, 2.6, presents a brief summary of the chapter.

2.1 Network Security and the role of AI

Taking into consideration the objective of the thesis and as this work will be integrated in the security of a framework, it is important to have a good understanding of what is a cloud environment and the security context in networks. In this section, we first introduce what is a cloud-based environment (section 2.1.1), after that, the most common attacks (section 2.1.2) and finally the role of AI in the security of this environments (section 2.1.3).

2.1.1 Cloud-based Environments

A cloud-based environment is a type of computing environment that relies on shared resources, software and information that are accessed via the internet instead of being stored locally on a computer or server. The work we will develop will be based, as mentioned before, in security in cloud-based environments, so we will present a brief explanation of each one of the technologies we are going to use, to have an overall knowledge so that is easier to understand what can be done after identifying a threat in this systems.

Microservice applications and orchestrations

Kubernetes is an open source platform for managing containers, capable of configuring, maintaining and automating them. As it containerises applications, it facilitates the distribution of applications in scalable microservices [7].

In Kubernetes, containerized applications are grouped into units called pods. Pods can be deployed onto nodes, which are physical or virtual machines that are used to run the applications. Kubernetes provides an API based on declarative object configuration, for managing the state of the applications, and it adjusts, automatically, the actual state of this applications to match the desired state. This procedure makes sure the applications are always running as planned, even in the situation of an hardware or software failure occurs.

Service Mesh

Service mesh is an infrastructure layer for handling communications between microservices in a distributed application. It provides the possibility of adding features as service discovery, fault tolerance, traffic management and security, without the need of focusing on these aspects when building your code [8].

There are several tools to deploy a service mesh architecture. One of the most common ones is Istio [8]. Istio is an open source service mesh that runs over Kubernetes and focuses on achieving an efficient way to secure connect and monitor services.

This work is important in this infrastructure as it has the goal of improving the security of communications by detecting any possible threat that can compromise the communications.

Policy Enforcement

Policy enforcement is the process of ensuring that a set of rules or policies are followed by a system. In a service mesh architecture, this set of policies is used in order to control the communications inside and outside a cluster. This policies can be defined prior to the deployment but also during run-time, this allows services to create, delete and apply policies while the service is running which is something important for the security of it since these must keep monitoring the state of the network and take action when an attacks is detected.

Open Policy Agent (OPA) [9] is an open source, general-purpose policy engine that unifies policy enforcement across environments, such as Cloud-based ones and is used for handling policies in Service Mesh architectures. This engine provides a high-level declarative language that let's you specify policies as code and simple API's to enforce policies in microservices, Kubernetes and more. OPA can be used, as well, to supervise authorization, admission and other policies in Cloud-based environments. This process will apply the policies needed to handle a threat in the possibility of existing one that is detected.

2.1.2 Common Attacks

In this work are considering two types of attacks: active and passive. A passive attack is when an intruder intercepts data being communicated through the network without the intent of interfering and an active one is when an intruder executes commands with the finality of disrupting the normal functioning of the network [10].

Next we will detail a brief explanation of the active type of attacks that exist [10]:

- **Spoofing**: When something miss-presents its identity, so that it can have access to information it should not have access to.
- **Modification**: When something performs some kind of modification in the routine route, so that the message goes through a long way route.
- **Wormhole**: This attack is also called the tunnelling attack. The attacker receives a packet at one point in the network and replays it into the network from that point on, making the attacker be in control of the routes shared in the network.
- **Fabrication**: Here, a malicious node generates the false routing message. This means it generate the incorrect information about the route between devices.
- **Denial of Service (DoS)**: A malicious attempt to deny access to shared network resources or service, by busying the network.
- **Sinkhole**: An attack that prevents the base station from receiving the complete and correct information. A node tries to deviate the data to it from a neighbouring node.
- **Sybil**: This attack creates multiple copies of malicious nodes, in order to increase the number of malicious nodes so that the attacks are easier to happen.

As we did for the active attacks, we will present a list with a brief description of passive attacks [10]:

- **Traffic Analysis**: In this attack, the attacker tries to find the communication path between the sender and the receiver.
- **Eavesdropping**: The attacker finds out some confidential information from the communication between the sender and the receiver.
- **Monitoring**: In the following attack, it has access to secret information, but can only read it, not modify it.

2.1.3 AI in Network Security

According to [6], the three key AI applications for cybersecurity are:

1. **Network Vulnerability Surveillance and Threat Detection**: AI can provide a faster detection and identification of cyber-threats. It can perform a real-time monitoring of networks by scanning data with the finality of recognizing unauthorized communication attempts, connections and abnormal or malicious credential use, unusual data movement as well as other attack attempts.

   Threat hunting using AI tools can cover cloud, data center, enterprise networks and IoT devices and provide automatic updating and threat investigation of defense framework layers as well as diagnostic and forensic analysis for cybersecurity, answering to the question "What happened?".

2. **Incident Diagnosis and Response**: Incident diagnosis can answer to the question "Why and how it happened?". AI has tools capable of examining past datasets to find the root causes of the incidents by finding changes and anomaly indicators in the network activities. If the analysis discovers a vulnerability, predictive analysis can provide insights on consequences of such exposure.

   After the causes of the abnormality are identified, prescriptive analytics can be used to respond to the incident in an effective way based on recomendations to contain and eliminate the causes of the abnormality.

3. **Cyber Threat Intelligence Reports**: AI solutions have been deployed with the objective of supporting cyber threat analysts and address the problem of information overload. This information can be compiled and summarized, as well as be a report fully written with the help of Natural Language Processing (NLP). This reports provide the indicators and early warnings to improve the monitoring of anomalies inside of a network and detect more rapidly and efficiently cyber attacks.

   In our work we will focus on the first point, network vulnerability and threat detection. The main goal will be to develop a framework capable of monitoring networks in real-time, detect any possible threat, handle it fast and take it down before it spreads and does any damage.

2.2 Intrusion Detection Systems

Intrusion Detection Systems (IDS) are automated defense and security systems which it's main task is monitoring network traffic with the objective of detecting and analyzing hostile activities within a network or a host, as well as issuing an alert. This systems actually do not detect attacks but are capable of identifying evidence of intrusions, either during or after the occurrence [11].

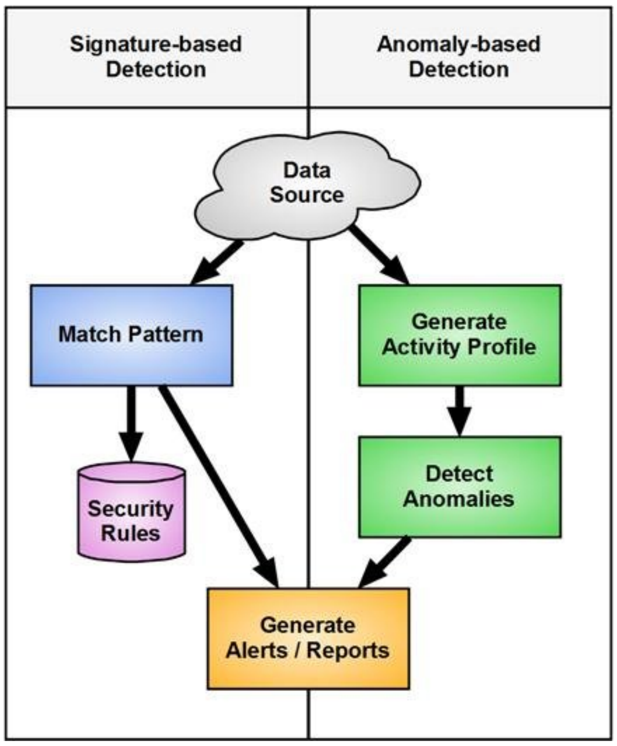

Figure 2.1: Intrusion Detection Systems - Overview [13]

In our research, the approaches we present, are based, mainly, in the most two, most common, types of IDS techniques, signature-based and anomaly-based (Figure 2.1).

Signature-based Detector, also known as misuse-based, sets on the focal point that the anomalies are already well known [12]. The detector tries to find patterns or signatures of already studied anomalies in the provided dataset. In order to use this approach, a database consisting of well known attacks, is needed and must be updated every time a new attack is discovered. This methodology has really good results in the detection of familiar attacks, although if there is a new anomaly, zero-day attack, the approach lacks information, making it unreliable in our study.

Anomaly-based Detector approach learns the patterns of "normal" activity, in order to detect an anomaly every time a given observation deviates from the the expected behaviour [12]. The idea of this method is focus on normal activity in order to create an activity profile of what characteristics normal data has and whenever the IDS catches an instance that does not follow the profile created it is classified as an anomaly. It usually performs well identifying new threats, zero-day attacks, which is why we consider it to be the best detector for our research, but it has a setback, causes quite a few False Positives.

Our work focuses on the Anomaly-based Detector because the method of detecting anomalies is more flexible to different types of threats and it overcomes the issue of not being able to detect anomalies that have not been recognized, previously, by the detector.

2.3 Network Anomaly Detection using Machine Learning Approaches

Network Anomaly Detection is a technique for identification of unusual or suspicious behaviours and events, in computer networks, that deviate from already known and expected patterns. This technique involves monitoring network traffic and other network-related data with the finality of detecting security threats or breaches and performance issues.

Machine learning has a broad area of research, with numerous possible applications, such as medicine, email filtering, speech recognition and computer vision, just to name a few. AI models are used as a way of assisting people in everyday tasks, business projects and it is a fast growing area in the recent years. Network anomaly detection is a field of study that has been getting more importance in the last few years due to the fact that cloud-based environments have been getting more popular in enterprises, and as a consequence of this the number of cyber attacks to this environments has been growing too and getting more sophisticated. ML plays a key role in the combat to this growth, by being used in order to improve the methods of Network Anomaly Detection (NAD) that exist, to the extent of detecting all possible attacks to an environment so it can be as safe as possible.

The pipeline for building a machine learning model for NAD, normally are the following ones:

- Data preparation: This step focuses on collecting and preprocessing the data gathered that will be used to train the model. Preprocessing may include cleaning the data, normalizing or scaling the features and handling missing or incomplete values.

- Feature selection or reduction: This involves selecting the most relevant features in the data that will be used in the model or reducing the existing features of the data. This step is important because the usage of too many features may increase the complexity of the model and make it difficult to interpret and train, while having too few features may not provide enough information for the model to perform well.

- Model selection: This step involves choosing the machine learning algorithm that will be used to train the model. These algorithms will be specified in this chapter.

- Model training: This step focuses on using the selected algorithm to train the model on the prepared data. The model will learn to recognize patterns in the data that are indicative of normal behavior and to identify observations that deviate from these patterns as anomalies.

- Model evaluation: This focuses on evaluating the performance of the trained model to determine its overall performance in identifying anomalies.

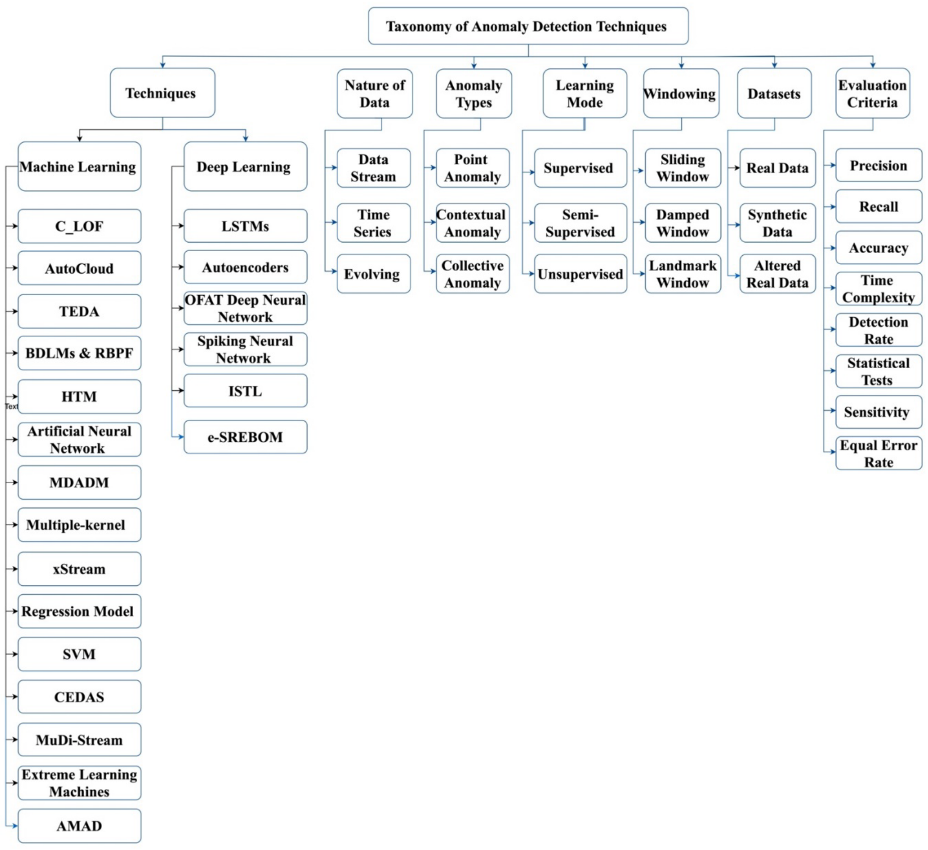

Figure 2.2: Taxonomy anomaly detection techniques [14]

Network Anomaly Detection has been increasing in the last few years. Figure 2.2 represents a high level taxonomy of the most common techniques in NAD.

Machine Learning can be divided into four types of algorithms: supervised, unsupervised, semi-supervised and Reinforcement Learning (RL). Supervised learning algorithms focal point is learning a mapping between input and output based on labeled input-output pairs, mainly used for classification or regression problems. Unsupervised learning focuses on discovering patterns based on unlabeled datasets, most common in clustering problems. Semi-supervised combines the two approaches talked previously and Reinforcement Learning perceives and interprets the environment that surrounds him, attributing rewards to desired actions and punishments to undesired ones to train the model. Another recent type of ML that has been discussed and being implemented in this research is Federated Learning, it focuses on several clients training it's own model based on it's local data and in each iteration each client communicates it's updates to a central server, where all the client-side models are aggregated to compute a new global model [15]. The different techniques that exist will be detailed further on this section.

Depending on what type of anomaly detection we are focusing on, there are 3 different types of nature of data: data stream, time series and evolving. There are, as well, three different kinds of anomaly types: point anomaly, contextual anomaly and collective anomaly. Point anomaly refers to a point in a data flow that differs significantly from the pattern. Contextual anomaly is characterized

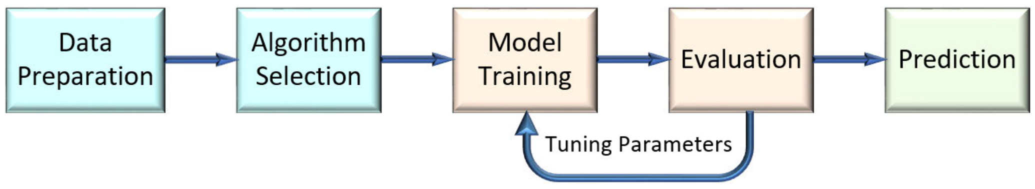

Figure 2.3: Supervised ML [16]

by being usual in a certain context but in a different context is not normal and so considered an anomaly. Collective anomaly refers to the frequent occurrence of a series of continuous anomalies, in a time period. The windowing techniques can be helpful in this type of research for processing of data, there exist 3 different types: sliding window, damped window and landmark window.

To evaluate the performance of models we may use 3 different kinds of datasets: real data, synthetic data and altered real data. Real data, which is the most common type used in the researches we found, is a dataset which has real operational data collected from real life environments. Synthetic data contains artificially generated data with the use of programming. Altered real data comes from a real data dateset but has suffered transformations. There are numerous possible evaluation criterias, that we will detail further in the state of the art in 2.5.

The remainder of this section, mainly, centres on the multiple machine learning approaches, gathered from the research papers, used for NAD. We will dive deeper in unsupervised and reinforcement learning techniques, based on prior investigation and tests made, these types of algorithms are the more suited for our study.

2.3.1 Supervised Learning Algorithms

Supervised learning, creates a set of distinction rules to predict the classification results while having their data labeled. This type of algorithms performs well when we are trying to achieve high accuracy, but can be slow compared to other algorithms and when a new kind of attack appears it has to be trained to identify it. In this chapter we will present and analyze some approaches using supervised learning.

2.3.1.1 Random Forest (RF)

Random Forest is a classification supervised algorithm. It consists of multiple predictors, individual decision trees, that operate as an ensemble. The idea is when building a decision tree, it searches for the best feature among a random subset of features to perform a split. Each decision tree outputs a classification and the one with most votes becomes the model's prediction.

In [17] the authors focus on a uniform detection system based on a Random Forest classifier to detect attacks as DoS, Probe, U2R and R2L. It only uses the 10 best classified features, of the NSL-KDD and KDDCUP99 datasets, selected with the help of a feature classifier, not specified. Other classifiers (K-Nearest Neighbours (KNN), Naive Bayes (NB), Decision Tree (DT), Random Forest (RF), Logistic Regression (LR)) were used as a way of comparison to check the accuracy of the proposed method. It achieved an accuracy of 99.9% in the KDDCUP99 dataset and 98.1% in the NSL-KDD, both superior to the other classifiers, affirming the approach as the better one.

The authors in [18] propose a system based on Random Forest algorithm for anomaly detection in IoT devices to be used in smart cities. To evaluate this approach it is used the UNSW-NB15 dataset, and the metrics accuracy and false positive rate. The approach firstly uses the RF model to classify the instances as normal or attack and the maximum value for training is used for training the Extra Tree. Finally performed feature selection using the ExtraTreesClassifier. The model presented good results 99,34% in accuracy and 0,02% FPR.

In [19], the authors revisited the Random Forest algorithm, as it already had been intensively studied, they performed a random forest classification with 10 different values for number of decision trees (10, 20, 40, 50, 80, 100, 200, 400, 500 and 800), using majority voting and no feature selection or reduction and compared, the best of all the random forests created, 800 in their case, with various state of the art algorithms.

It used 3 datasets NSL-KDD and UNSW-NB15 as well as the metrics accuracy and False Alarm Rate (FAR) to judge the reliability of algorithm. The results were really promising with accuracy of 99,57% and FAR of 0.34% for the NSL-KDD dataset and accuracy of 95.5% and FAR of 7.2% in the UNSW-NB15 dataset, outperforming all of the algorithms used as comparison.

In order to evaluate and compare the performance of the approaches it was used the dataset NSL-KDD in [19] and [17], taking into consideration the accuracy results achieved in each one, both had notable results with the first one being slightly better. [18] and [19] used the same dataset, with the first one being better since it reached 99,4% of accuracy and the second one 95,5%

2.3.1.2 Supervised Deep Learning

Deep Learning is a ML subset that tries to simulate the behaviour of the human brain, it basically is a neural network with three or more layers that learns from large amounts of data. In recent years, this type of algorithms has been getting more attention in the anomaly detection study, it has lots of upsides comparing with other approaches talked in this report. It shows great results when learning representations of complex data such as high-dimensional data.

ANN, RNN and CNN are the most common supervised DL types, the first two have been studied previously and presented good results when it comes to NAD and have shown room for improvement, on the other hand CNN have not been thoroughly studied in this type of problem and by knowing the specifications of the algorithm we came to the conclusion that it would not benefit our work since it is an algorithm that better suits problems that revolve around images and

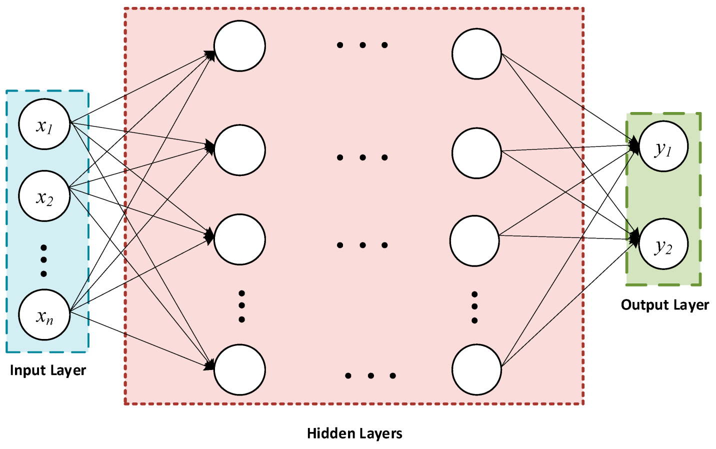

Figure 2.4: General Artificial Neural Network Architecture [20]

videos instead of data flows and so we did not studied it.

2.3.1.3 Artificial Neural Networks (ANN)

The paper, [20] proposes an anomaly detection mechanism based on a supervised deep neural network and using mutual information. The neural network, if the feature set is only numerical has 4 hidden layers with a ‘ReLU’ activation function and the output layer, using a ’sigmoid’ function as activation, only possesses 2 outputs, one for benign traffic and another for anomaly traffic, if the feature set has both numerical and categorical features the it only is furnished with 2 hidden layers and 2 outputs as the previous one. This methodology and the others used for comparison, 0.01 learning rate, Adam optimizer and binary crossentropy as the Loss function.

The dataset used to evaluate the performance of the DL methodology is the IoT-Botnet 2020, with attacks such as Denial of Service, Distributed Denial of Service , Reconnaissance and information theft attacks. For feature selection it experiments with the 80, 32, 16 and 8 best numerical features and 5 best categorical-numerical features which are calculated using mutual information. The metrics used to conduct the performance assessment showed a much better performance than the state of the art and common algorithms, achieving the following results Accuracy, 99,01%, Precision, 99,30%, Recall, 98,02%, F1-Score, 98,64%, False Alarm Rate, 3.91%, True Negative Rate, 96.08%, and False Negative Rate, 0.04%.

Another approach using ANN is [21], where it is proposed a multiclass classification network composed of 4 hidden layers to classify the data sets NSL-KDD and KDDCUP99. The structure was a input layer with 41 neurons, 4 hidden layers with "ReLU" as activation function, 1 fully connected layer in the output layer and 5 neurons in the output, using a softmax activation function. The results were extremely satisfactory as it achieved accuracy above 98% in all the types of attacks except U2R that the authors claimed it was because of shortage of records.

The article [22], proposes a simple type of Artificial Neural Network, Feed Forward Neural Network, adopting a Multilayer Perceptron, with the finality of achieving a high scalable framework for real-time classification, capable to deal with the problem of scalability. The number of hidden layers is selected with the help of hyper parameter selection method, testing 5 different numbers of hidden layers and selecting the one with best performance, with that being 5 hidden layers using “ReLU” activation function, it also used 1 neuron in the output layer. The testing segment used multiple datasets, KDDCUP99, NSL-KDD, where both had 41 neurons in the input layer, UNSW-NB15 with 43 neurons, WSN-DS with 17 neurons and CICIDS2017 with 77 neurons. All the datasets were tested with a binary classification, with only 1 neuron and sigmoid activation funcion, as it outputs 0 or 1, as well as multiclass classification, in which the number of neurons used was defined by the number of attacks each data set had and softmax activation function. In terms of results, the authors pointed that the proposed framework showed superiority in comparison with the other ML algorithms.

For the activation function, in the hidden layers it was always used the "ReLU", as it helps reduce the issues of vanishing and error gradient, it is faster than other non-linear activation functions and facilitates training the MLP model with a large number of hidden layers. For the output layer, sigmoid activation function is best in binary classification because it outputs 0 or 1 depending on if it is classified as an anomaly or not, and if it is a multi class classifier, where it distinguishes the type of attack, usually it is used the softmax activation function. Both are valid options depending on what the authors want to achieve.

2.3.1.4 Recurrent Neural Network (RNN)

Recurrent Neural Network is a type of Supervised Deep Learning where the output of the previous step is fed as input to the current step. RNN have a memory which saves the previous outputs already calculated and uses the same parameters for each input as it performs the same task on all inputs and hidden layers, reducing the complexity of the parameters.

In the paper, [23], it proposed three multi-class classification approaches based on RNN, one using Long-Short Term Memory (LSTM), Bidirecitonal LSTM (BiLSTM) and Gated Recurrent Unit (GRU) [24]. LSTM is a sub-type of RNN allows to decide whether to retain previous information or discard it, being capable of never losing previous information that might be relevant. BiLSTM is and extension of the conventional LSTM that enhances the performance of the model by learning in both directions simultaneously, it considers forward and backward activation to calculate the output. The models were tested using NSL-KDD, BoT-IoT, IoT-NI, IoT-23 and MQTT datasets and all of the models achieved accuracies above 98% and recall above 90% in all datasets.

In [25], the authors propose an IDS approach based on RNN, studying it’s performance in binary and multiclass classification, using different numbers of neurons and different learning rates in order to check each ones impact on the performance of the model. The algorithm has a sigmoid activation function in the hidden layers and a SoftMax activation function in the output layer. The proposed approach is evaluated using the well-known dataset NSL-KDD and compared with known algorithms as J48, ANN, RF and SVM. In terms of results the model

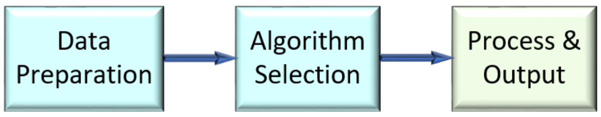

Figure 2.5: Unsupervised ML [16]

performed well with an accuracy of 83.28% with 80 hidden nodes and 0.1 learning rate for the binary classification and 81.29% with 80 hidden nodes and 0.5 learning rate for multiclass classification. Comparing with the other algorithms used in the research the RNN had the best performance in Accuracy of all in both types of classification and had a low FPR in all kinds of attacks, all below 2.2%, which makes this algorithm an interesting approach with promising results.

We can see that the first approach has better results, in the NSL-KDD dataset, when it comes to accuracy than the second one, and this may have to do with the fact that [25] uses 3 approaches that are an upgrade of the conventional RNN which in our case proves that they have benefits in our research.

2.3.2 Unsupervised Algorithms

Unsupervised learning tries to discover hidden patterns based only on the input, unlabelled data, without he interference of the human. This sort of algorithms tends to be faster than the other ones, although the accuracy can be lower but it can identify zero day attacks without the need of being trained again and usually perform well in unbalanced datasets. This chapter describes some approaches based on this type of algorithms.

2.3.2.1 Isolation Forest (IF)

Isolation Forest is a unsupervised approach, first introduced in 2009, [26] as a solution to the high dimensionality problem with large number of irrelevant attributes, with linear time complexity and low memory requirement.The main idea is that it uses multiple decision trees and focuses on the outliers rather than the normal points. It sets on the premise that the outliers, on average, are closer to the root of the tree compared to normal points, so the algorithm creates multiple decision trees and randomly selects some feature value and performs feature value splits. Finally when all the trees are created it calculates the score of each point,the closer the score of the point is to 1 the more probable it is to be an outlier. The score is calculated with the following formula:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{2.1}$$

In [27], the author tests the robustness of using the IF algorithm in anomaly detection. It used the NSL-KDD dataset and the main metric used to classify the model was the AUC score. The results were very satisfatory as it performed well for all the kinds of attacks, AUC of 98,3%, but for lesser known attacks a decrease in accuracy was observed, as for the false positive and false negative rates, they were both very small too which is something important to point out, since it's one of the main drawbacks of unsupervised learning.

The next paper, [28], proposes an approach based on Isolation Forest with a twist, since this algorithm is time-consuming and cannot adapt to anomaly detection on large-scale traffic data, the authors proposed an algorithm, called SPIF, that computes the construction of multiple isolation trees simultaneously and the calculation of the anomaly score as well, since each tree is independent and has no influence on the others.

Using the UNSW-NB15 dataset and the AUC and Accuracy metrics as well as the running time of the algorithm to evaluate the performance of the algorithm, the results were satisfatory with an Accuracy of 87% and AUC of 89%, comparing it with the normal IF algorithm it performed better in all the metrics, specially in the running time, taking 2 minutes, the proposed approach, while the IF took 14,5 min.

2.3.2.2 Clustering (based)

Clustering based algorithms, an unsupervised method, are one of the most commonly used algorithms in network anomaly detection. It receives an unlabeled input dataset, as it is an unsupervised approach, and the goal is to group the various intances based on certain similarities, clusters, with the finality of encountering patterns and also, in our study, outliers.

In [29], the authors propose a new framework for real-time anomaly detection, called SSWLOFCC, which consists on combining big data technologies, for performance improvement, not only on a computational level but time consuming, with composite clustering. The algorithms used are first Local Outlier Factor, in order to detect the outliers and then Agglomerative Clustering.

The efficiency of the approach was tested using 3 different datasets, DARPA, MACCDC, and DEFCON21. The performance of the framework was then tested using the accuracy, execution time and memory consumption. It concludes that on the accuracy level the algorithm has very positive results and on time and memory consumption it gains a lot.

The study [30], proposes an unsupervised clustering-based method, combines Sub-Space Clustering (SSC) and One Class Support Vector Machine (OCSVM). In order to evaluate the performance of the proposed method, since one of the algorithms is a binary classifier, the dataset was split into multiple ones, each one being composed of the normal activity and a specific type of attack and one more mixed with normal activity and a mixture of the multiple attacks. To preprocessing the dataset it used F-test for feature Selection and normalized each feature by removing the mean and scaling to unit variance.

Uses the public dataset NSL-KDD and metrics Recall and False Positive Rate to evaluate the trustworthiness of the model. The method was also compared to K-means, DBSCAN and SSC-EA algorithms. Comparing with this algorithms the algorithm behaved good with Recall superior to 84% except R2L attacks but it still had a better recall than the other algorithms, and False Positive Rate always below 10%, which it is not great but is a point to improve on.

In [31], the authors propose an unsupervised clustering approach with 3 steps, first create the clusters maintaining only the core points and the features that best represent them, then each cluster is represented by a Gaussian Mixture Model (GMM), after that the clusters are updated based on the GMM using the Kullback-Liebler distance as measure. The testing was performed only with normal data acquired from multiple datasets such as KDDCUP99, NSL-KDD and Darpa98. The metrics used were the ROC curve based on the detection rate and the false alarm rate, being very close to 1, compared to other algorithms tested such as OCSVM, SPAI, LOF it performed much better than all of them, the false alarme rate decreased 15% to 5% in multiple datasets and the detection rate increased from 5% to 10%.

The study [32], also uses clustering as base algorithm for the classification, it is divided into two parts, feature selection and density peak-based clustering. In the first part the least relevant features are removed, based on the calculation of the Maximal Information Coefficient (MIC), which outputs the relationship of two features for continuous features and the relevancy, that measures the features symmetric uncertainty, for discrete ones. After that it uses the density peak-based clustering, each center of the cluster have the higher densities and the neighbours have lower ones, the cluster that they belong to depend on the density of its neighbours. In terms of results it performed well, the accuracy was always high, above 92%, using the KDDCup99 dataset.

2.3.2.3 Unsupervised Deep Learning

In this section we will talk about two of the most familiar unsupervised deep learning algorithms, Autoencoders and Generative Adversarial Networks. Autoencoders is the one most studied, in NAD, between these two and has proven to be very useful, while GAN has had some more attention in the last few years without any breakthroughs, but has showed promising results and can fit, the problem we are dealing with, well.

In [33], it is presented a solution for IoT anomalies detection set on an unsupervised deep learning technique. The authors tested their approach using an unbalanced dataset and a balanced one, which was obtained from the first unbalanced dataset, extracting a similar amount of normal traffic as abnormal. The approach splits the dataset into Train-Validation-Test and focuses on a Autoencoder that is trained only with the normal activity with the finality of it learning the normal pattern with minimum error, so that a out of the ordinary instance exceeds a certain threshold. It uses a ReLU activation function for the hidden layers and a sigmoid for the output layer. The predictor takes into consideration the Mean Squared Error of the reconstruction in the Autoencoder and makes a binary classification. The dataset used was Bot-IoT and it add threats such as probing, DoS and information theft. .

The results were extremely promising, it was tested with a threshold of one standard deviation above the mean loss and after it was increased to three. The results were better when the threshold was three standard deviations, achieving lower FP and low FN although it was higher when the dataset was unbalanced, but 0 when the dataset was balanced. Recall always above 96% is something to pay attention at. The approach was also compared to GMM with LFO, being the approach presented, always lower in FP and higher in Recall, proving that it is better.

In this paper, [34], it was used another autoencoder unsupervised learning that focuses only on DDOS attacks in order to demonstrate the predictability of TCP traffic can be exploited to detect this type of attacks in real-time, using a small set of features. It uses PCA for feature reduction and only benign data for the training phase and ReLU activation function. The results are satisfactory with precision, recall and F1-score, all above 90%.

The following paper, [35], also uses a similar approach as the ones before based on autoencoders, more specifically a conventional autoencoder and a convolutional autoencoder. The approach uses Autoencoders as a dimensionality reduction and is interesting because instead of using simply a conventional autoencoder to perform the classification, it adds an autoencoder with a convolutional layer in the encoding part and a deconvolutional layer in the decoding part with the objective of reducing training time. Taking the NSL-KDD dataset into account and testing with different types of data, TCP, UDP, ICMP and a mixture of all of them, the method performed well in the tests with AUC constantly above 95%, being always superior to the conventional Autoencoder.

In [36], the authors propose a different unsupervised deep learning approach, based on a Generative Adversarial Network, introduced in [37], is an unsupervised algorithm divided into two parts, a generator and a discriminator, the first one generates fake data with the objective of deceiving the second one and the discriminator has the task of distinguishing between the real and fake data, both are Neural Networks and work as a competition in the training phase, each one improves the other one.

The paper makes use of two different types of GAN, AnoGAN [38], which works as a standard GAN, focused on likelihood estimation with sampling and ALAD [39], which sets in a somewhat different approach, adversarially learned inference.

To demonstrate the performance of the algorithms, 3 datasets were used, UNSW-NB15, CICIDS2017 and Stratosphere IPS as well as 4 different algorithms for performance comparison, Deep Structured Energy-Based Models, taking into account the reconstruction error and the energy error, Deep Autoencoding Gaussian Mixture Model and Autoencoder. As for the results, they were really promising for the ALAD, it achieved above 90% in all the datasets for the AUROC and above 75% for the AUPRC, reaching higher performance compared with all the other algorithms except Autoencoder but by a small margin, although AnoGAN performed poorly compared with the other algorithms.

Same as the previous article, the following one, [40], focuses on a GAN approach, this time a more simple one, with a standard GAN algorithm. The paper has two parts, one to try and use the algorithm to invade the IDS, GAN-based attack methodology and another to make the IDS more robust, GAN-based defense methodology, according to the finality of our research we will focus only on the second part. Here the model uses multiple classification algorithms and compares the results in each of them with a simple adversarial training and GAN-based adversarial training with the use of KDD99 dataset. The results were notoriously better than the simple adversarial training achieving accuracies above the 79% mark, the results were not perfect but very satisfactory with space for improvement.

2.3.3 Hybrid Approaches

Hybrid approach is a combination of simple algorithms, both supervised or unsupervised, with the intent of complementing and augmenting each other. This type of approaches can be helpful in our study because some algorithms alone can perform badly in a certain problem but combined with other algorithms, it will perform much better and attain good results.

The study, [41], proposes a semi-supervised detection framework based on an Unsupervised Deep learning algorithm, Autoencoder, and a clustering algorithm, K-means and mean-shift. The framework consists in, first, applying an autoencoder and classify abnormal samples, the samples with more reconstruction error are considered anomalies, and second use either K-means or mean-shift with the finality of clustering the reconstruction error to validate the the results of the autoencoder.

It used the KDD99 dataset and the training phase was performed using 3 different number of samples, 1300, 800 and 400, to perform the validation of the framework, and the main metrics used to evaluate the model were the accuracy and F1 score. The approach had really notable results, showing the approach is robust, with accuracy always above 98.5% and F1 score above 98.4%, being the best overall Autoencoder with K-means, 99%. One interesting conclusion is that the accuracy decreased when the number of samples was greater.

In [42], the authors propose 3 unsupervised learning, it guarantees the detection of anomalies regardless of their prior knowledge, approaches with the finality of maximizes the accuracy even if the recall is sometimes penalized based on Deep Autoencoders using Isolation Forest, one is Deep Autoencoding with Gaussian Mixture Model and Extended Isolation Forest, Deep autoencoder with Extended Isolation Forest and Memory Augmented Deep Autoencoder with Extended Isolation Forest. In order to evaluate the models, it was taken the KDD99, NSL-KDD and CIC-IDS2017 datasets into consideration, using only normal data for the training and the features used were selected using an explainable AI methodology called Shappley Additive Explanations (SHAP), it is simply estimates the importance of each feature in the prediction process, if the feature makes the prediction more reliable than its SHAP value will be greater and vice versa. The metrics used is a weighted average of each Recall, Precision, F1-Score and Accuracy.

The main idea of the algorithms is, first it is calculated the anomaly score of the instance after the reconstruction part of the Autoencoder and check if the score is superior to a certain threshold, if it is the it passes to the extended IF algorithm to calculate the final anomaly score with the main purpose of checking if there is any false positives and increase the accuracy. If the score after the Autoencoder is less than the threshold the instance is considered normal.

In terms of results for the KDD99 the accuracy was always above 92%, NSL-KDD above 90% and CIC-IDS2017 above 81%.

An hybrid two-level classifier is proposed in [43], the algorithm, as it conveys in the name, has two classifiers, the level one performs real-time classification in incoming traffic data and in the case it cannot classify with high probability, the classification is delayed until the data flow terminates. After the data stream stops, the second level classifier performs the classification with a higher accuracy. In the first level it is needed a rapid classifier algorithm, without much regard to the level of accuracy, which they chose DT and RF and the second level, it is needed a high accuracy classifier, also opting by a DT and a RF.

The datasets used were UNSW-NB15 and CICIDS2017 achieving accuracy above 95.8% in the first one and 99.8% in the second one, being regarded as better than all the ones compared with, HAST-I, DT and RF.

2.3.4 Reinforcement Learning Algorithms

Reinforcement learning, as pointed before, is a machine learning method based on a agent which is inserted in an environment and is fed inputs from it. The agent will perform actions that will be rewarded accordingly to how correct they are and punished corresponding to how incorrect they are. With this learning method, after training, the agent will try to understand the changes on the environment in order to act in the way that will maximize its reward. The purpose of the ML algorithm is to find the optimal policy that will maximize the reward.

The following study, [44], proposes a binary classification Deep-Reinforcement Learning approach, it is based on using a Deep Neural Network as the classifier and after the classification, the RL algorithm gives or takes points depending on the classification made, if it was correct receives one if it was incorrect takes 1.

The evaluation of the model was done using NSL-KDD and UNSW-NB15 datasets and the tests were done modifying the number of hidden layers of the DNN and different number of testing iterations performing better with a high number. Using the metrics F1-score as main performance metric, the model performed well achieving a score of over 96% in both datasets, with it being the best compared to the other algorithms already known such as SVM, Random Forest, MLP and Autoencoder.

In the article [45], the authors propose an IDS based in deep reinforcement learning with the finality of maintaining the balance between accuracy and False Positive Rate. The method is evaluated using the datasets NSL-KDD, UNSW-NB15 and AWID, the classifiers Random Forest, Adaboost, Gaussian Naive Bayes, KNN and QDA. The method used was as follows, first using 5 different classifiers, checked the performance of each one in each dataset in terms of accuracy and false positive rate. After this selected which one of the classifiers performed better in accuracy and in false positive rate and integrated them in the model with a combination of three or four classifiers, two or three best performers in accuracy and one in false positive rate in order to have a good performance in accuracy as well as in false positive rate.

The model has two parts, one is the composed of many agents that will make their classification from the feature extraction to the classification itself and the other is the IDS central which will give the real classification to the agent so that it can create the reward vector in order to improve.

The IDS performed well achieving accuracy above 81% in all the datasets, not particularly high but with good results with space to evolve and FPR below 3.3%, significantly lower compared to all the other models.

In the following paper, [46], it is presented a reinforcement-learning based approach focused on finding the best anomaly score threshold. The model called ADRL, uses prior domain knowledge to label known anomalies, it uses, both, entropy, for the categorical features, and autoencoders, for the numerical features, to obtain anomaly scores. After that is when the reinforcement learning appears, the method uses it to achieve the best threshold, until it becomes stable, in order to have the most accurate model possible.

The study uses multiple datasets, such as IDS2012, IDS2017, IDS2018 and DDoS2019 and the metrics ROC-AUC and PR-AUC curves to analyze the performance of the model. The results were really promising with ROC and PR above 97% in all the cases, having better results in almost all of the situations comparing with other state of the art algorithms.

2.4 Feature Selection and Reduction

Feature selection and reduction (Figure 2.6), usually, are used in large datasets, with a great amount of features. The methods are used with the purpose of reducing the number of features in a dataset, to lower the computational power needed and the time it takes to train the models.

Feature selection consists of selecting a set of features from an original dataset, without any kind of transformation. There are three different types: filter-based, wrapper-based and embedded-based. Filter-based selects a subset of features independent of the classifier that will be used, wrapper-based trains the algorithm iteratively using subset of features with the best performance and embedded-based is a part of the algorithm used as classifier.

A very common feature selection technique used is the Information Gain, [47], [48], where the features with the most gain are selected, based on a threshold.

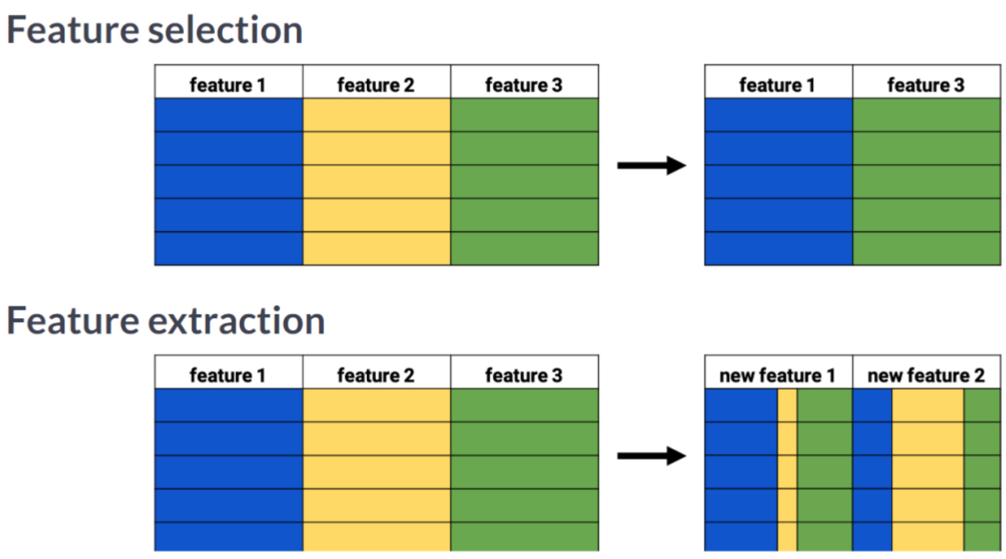

Figure 2.6: Feature Extraction and Feature Selection [52]

Some other ones are Chi-Squared [47], that determines the independence of two features, ReliefF [47], that determines the influence of features in determinating the target class and Pearson Correlation [49], which computes the statistical correlation between two features. In addition there are some methods which use Decision Trees as a process of feature selection, [50].

Feature reduction focuses on transforming the feature set of the original dataset in a new set with less number of features, performing a dimensionality reduction.

In [51], it is tested the performance of three feature reduction methods, Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and Autoencoder, in anomaly detection. PCA is one of the most commonly used processes for feature reduction, it consists in using an orthogonal transformation to convert a set of correlated features into a set of uncorrelated ones, taking into consideration the largest variance. LDA is used to project the features in a lower dimension space (use less features to separate them). Autoencoders we will adress it further in the study.

2.5 Evaluation Metrics

Evaluation metrics are an important part in our study since it is with their aid that we will evaluate the performance of the approaches. All the metrics have as base a confusion matrix.

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

Table 2.1: Confusion Matrix

The evaluation metrics that will be used to evaluate the performance of the models are Accuracy, F1 Score, False Negative Rate (FNR) and Recall, Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC).

The first four metrics take into account the number of True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN), while the last two metrics take only into account the number of TP and FP. In addition we will still take the FP and FN in consideration to analyse the performance of the models.

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- $Recall = \frac{TP}{TP+FN}$
- $Precision = \frac{TP}{TP+FP}$
- $F1Score = \frac{TP}{TP+0.5(FP+FN)}$
- $FNR = \frac{FN}{TP+FN}$

The ROC curve shows in the x-axis 1 - Specificity, with this metric being the FP divided by the sum FP and TN while the y-axis shows the Recall. The AUC shows a value of between 0 and 1 that represents the area under the ROC curve.

Our study focuses, as pointed previously, on obtaining a reliable classification of the data instances, whether they are normal or anomalies, so the metrics used to evaluate the classifier need to focus on the number of well classified instances as well as the misclassified ones.

Accuracy is the more common metric, to check the general classification, it sees if all the instances were classified correctly, in order to know if the classification is solid, it needs to be as high as possible, it also is the very biased and at times it must be avoided, specially in unbalanced datasets such as the ones we will be working with. Recall will tell us if the instances classified as anomalies correspond to all the instances that are actually anomalies, this metric is expected to be the highest possible, as our classifier is supposed to detect all the attempts of intrusion. Precision is also an important metric as it tells us, from all the instances we classified as attacks, how many of them are actually attacks, which is important because classifying normal activity as an anomaly is not as bad as the opposite but it will affect the user experience and may be costly in computational use to take down. False Negative Rate is the opposite of the Recall, it tells us how many anomalies we classified as normal activity, this one has to be as low as possible, otherwise our classifier is not trustworthy. The ROC curve, shows the capability of the model to make the separation between TP and FP and its AUC shows the performance of the model in an overall and unbiased way which the metrics cannot show since it depends on the testing dataset used, so we will make the comparison between algorithms based on these metrics.

Other metric that will be taken into consideration is the training time of the algorithms and the time it takes to classify, because of the relevant importance of the time complexity for our problem.

The metrics we will optimize are the F1-Score, as it is an harmonic mean between Precision and Recall, it shows how precise (how many instances it correctly classifies) and robust (how many instances did it miss) the model is, so we want to maximize it, the false negative rate with the intent of getting the minimum number of instances classified as not an anomaly when it actually is one and the final one the training and classification time of the algorithm, since we want the approach to perform well in real-time classification it has to have a low time complexity.

2.6 Summary

This chapter gives a wide view of how machine learning is used for network anomaly detection and make a comparison of multiple approaches. To construct this chapter, we researched various papers which proposed approaches for the problem of network intrusion detection and selected the most relevant ones, summarised them and organised them, taking into account what kind of ML algorithm it used.

It was noticed that the most common approaches in the recent years tend to use an unsupervised approach instead of supervised and DL approaches are getting more and more attention from the community presenting really notable results. We could also notice that the hybrid and reinforcement learning approaches mainly used DL algorithms and usually got good results so we believe DL may be best combined with other algorithms. Following this approaches, it was discussed in meetings that the route that made more sense to follow was prioritizing unsupervised learning approaches and DL approaches since they show lots of good results and still lack investigation, so we will focus, the remainder of our research, in this two types of algorithms.

An interesting method used in several approaches is the use of only normal data in the training of the dataset which we think makes a lot of sense since we want to learn the normal behaviour of the network so we will test this method in our future work.

The most frequently used datasets were KDDCUP99 and NSL-KDD. KDDCUP99 and NSL-KDD are very similar as the second one is an upgrade of the first one, as KDDCUP99 was considered a really good dataset to evaluate the models but had some problems that had to be fixed so it could be even better, such as the high number of redundant records and in many occasions, random parts of the train set are used as test sets which makes the performance of the algorithm not viable, so the NSL-KDD came in order to overcome these problems. Another dataset that is very common in NAD and has proven very complete is CIC-IDS2017. This dataset is the newest of the three presented here and as so is more up to date with existing attacks and has more traffic diversity. Since NSL-KDD is an improved KDDCUP99 it can be considered out of date, even though it is one of the most used datasets. After learning the specifications of these datasets and analysing the approaches that used feature selection in the state of the art approaches we concluded that feature selection or reduction will play an important part in our work because the datasets usually have lots of different features and this may be an issue, as the model created can overfit and the computational and time complexities may be too high, so in order to tackle this we will need to shorten the number of features with the help of this techniques.

The comparison of algorithms is difficult since the features used in each data set are very diverse and the approaches do not always use the same algorithms for evaluation performance so we focused our decision on the algorithms that showed more promising results and could help optimize the metrics we would use. The metrics which we will take more into consideration are the F1-Score, Recall, FNR and AUC.

Chapter 3

Proposed Approach

This chapter focuses on the projects where our work was integrated, the algorithms that were used to create the models and the dataset selected to train and validate our approach. In 3.1 we present a little more context on the projects where this work was integrated, presenting the objectives and the final product it is supposed to achieve, in 3.3 we introduce in a deeper level the datasets that were used to train and evaluate our approaches and in 3.2 it is presented the algorithms that were implemented and how they were used for the classification problem.

3.1 Projects

3.1.1 5G-EPICENTRE

5G-EPICENTRE, as introduced before, is a European project funded by the European Union, that aims to create a NetApps, to facilitate the management and deployment of existing 5G solutions. The platform will house and provide open access to 5G network resources where companies may find solutions already implemented and ready to deploy that take advantage of the 5G enhancement on communications.

The objectives of the project are the following ones:

- Build an end-to-end 5G experimentation platform, built to the needs of the public safety and emergency response market players.
- Pilot 5G systems in PPDR-based trials, with the objective of demonstrating 5G-EPICENTRE onboarded apps as important accompaniment to communications to public safety mission critical communications technologies.
- Develop a '5G Experiments as a Service' model, that will facilitate developers and SMEs to experiment with PPDR applications in easily repeatable and shareable environments.

- Facilitate the automation, continuous deployment and multi-access edge computing supported by containerized network functions, as well as to reduce the time it takes to create a service and the time to market for 5G solutions
- Take advantage of Artificial Intelligence for achieving cognitive experiment coordination and lifecycle management including application awareness and insightful analytics provided by ML.
- Implement impact-driven dissemination standardisation and exploitation.

The platform will be deployed on a shared cloud infrastructure and will accommodate microservices orchestration tools. The project seeks to configure secure network policies to deal with the increased attacks and, in order to achieve that, it plans to impose per-program restricted access profiles at container-level to reinforce isolated execution, while employing a service mesh concept to more efficiently address the security issue.

The advancements in AI have allowed it to become increasingly useful in IT infrastructure management, monitoring, scaling and security. It enables automation of application-level traffic routing, make accurate predictions on resource demand, monitor continuously the condition of applications, identify anomalies and patterns as well as analyse with ease and quickness huge amounts of data in anticipation of future events. Therefore AI can have a role in 5G infrastructures management, deployed in support of critical functions, like network slicing and application awareness in order to achieve 'cognitive network management'.

The proposed security framework is composed of three key elements, a policy engine, a security engine and an AI engine. The policy engine centralizes the configuration of the policies at the network and container levels, the security engine contains the protection to the underlying host OS, where the containers run, access control and authentication mechanisms and the AI engine will be used to aid in security and policy enforcement, in helping to identify anomalies based on the observability of the traffic of the network as well as support the enforcement of the response policies [53].

3.1.2 CHARITY

CHARITY, as introduced in section 1.1, is a European project funded by the European Union, that is set to create a framework, that relies on different enablers and technologies related to cloud and edge, capable of supporting next generation Extended Reality (XR) applications such as holographic events, virtual reality training and mixed reality entertainment.

The objectives of the project are:

- Automate the orchestrating network, compute and storage of resources in hybrid edge/cloud infrastructures.

- Provide holistic support for the orchestration of advanced media solutions.
- End-to-end management of applications lifecycle.
- Develop highly interactive and collaborative services and applications.
- Augment project visibility, outreach and business sustainability.

This work will integrate the security and privacy-aware part of the cloud infrastructure orchestration, by taking advantage of the approach we create for NAD to identify possible network threats and find a way to shut them down in order to keep the cloud infrastructure secure.

3.2 Proposed Approach Algorithms

The developed approach was integrated in a framework, which its objective is to detect network anomalies in real-time scenarios. Our approach starts by using Mobitrust, an application developed by OneSource that is constituted by various microservices for PPDR scenarios. It was used as Use Case in this project, as a reference to collect traffic to test the framework.

We started by idealizing possible scenarios, including traffic without any type of threat as well as traffic with attacks in order to assess if our framework is capable of distinguishing these attacks from the normal traffic.

After that we began collecting dataset scenarios with the usage of Mobitrust. The datasets we collected to train our models were free of threats, it only presented normal traffic, since our approach is based in Unsupervised Learning (train the dataset only with normal data and test it with attacks to see if it can detect them).

After that we developed, trained and validated our models. The development approaches followed were focused in Unsupervised Learning approaches and Unsupervised Deep Learning approaches. Supervised Learning achieves an high accuracy but are slower in the classification task, an aspect not favourable to us, since rapid classification is an important aspect that we want to achieve and supervised learning in NAD has already been studied in great depth, so we thought that there was not much we could add to the research already made. Unsupervised Learning has been getting more importance in NAD, since the algorithms are faster in the classification part and detect effectively attacks unknown to the network without the need of being trained again, an aspect that makes them a lot more interesting to study. Deep Learning approaches have been proving to be a great method to follow in NAD based on the great results it presents and its ease in dealing with high-dimensional data.

For the training and validation of our dataset, the algorithms we used in our approaches were Conventional Autoencoder and Convolutional Autoencoder, both of them Unsupervised Learning and Deep Learning algorithms. In order to provide a deeper understanding of the approaches we developed, we provide a description of the algorithms we used and the architectures we built and its pipeline.

The first algorithm we tested was the Random Forest. This algorithm was used as a baseline model. It was the algorithm that presented best results in previous studies conducted for this work and we used the results from it as comparison for the other algorithms tested.

The next algorithm we implemented, tested and analyzed was the Conventional Autoencoder. This decision was made taking into consideration the results this algorithm showed in the state of the art approaches, and by according with the team that is responsible for this project that it may be an algorithm that would perform well in detecting with accuracy the threats in real-time scenarios.

The final algorithm we implemented, tested and analyzed was the Convolutional Autoencoder. The choice of this algorithm was based on the fact that network attacks are always a set of multiple anomalies in a row, so a the thought process was that if we give a window of instances to the model, it would detect an attack more easily if a window has multiple anomalous instances than just detecting isolated instances as anomalous.

Finally, after training our models and validating them, we selected the ones that performed best and began the integration in the framework, where we trained these models and tested them by analyzing their performance in classifying real-time traffic and assess if they were ready to be used in real scenarios.

3.2.1 Conventional Autoencoder

The Conventional Autoencoder is not an algorithm of classification but of compression and decompression. With this in mind and our problem being a classification problem, the more usual approach to perform a classification using a Conventional Autoencoder was by assessing the quality of the instances reconstruction.

The steps to the classification with this algorithm follows the next architecture, as shown in Figure 3.1, each input instance goes through the encoder and next goes through the decoder, following an architecture similar to the one of an ANN. Following a formula previously accorded it is calculated the error of the instance decoded based on the original instance. Taking into consideration a threshold previously agreed upon, we check if the error is higher or lower than that threshold. If it is lower the instance is considered as normal, if it is higher then it is considered as an anomaly. By learning to replicate the most salient features in the training data, the model is taught how to precisely reproduce the most frequently observed characteristics. When facing anomalies, the model should worsen its reconstruction performance as it has not learnt how to reconstruct them.

Since it does not exist an already implemented Python function in a library, we needed to implement our own Autoencoder, taking advantage of the TensorFlow library.

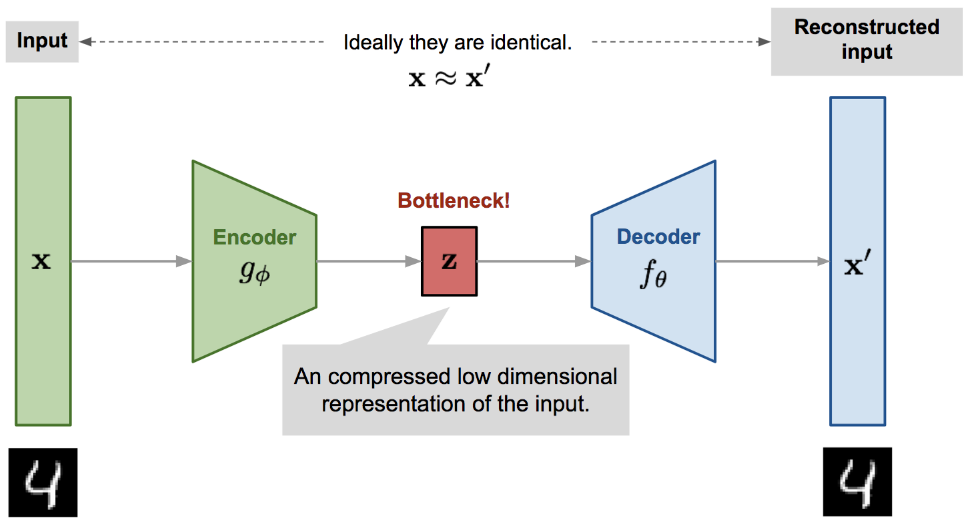
Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Figure 3.1: Autoencoder Architecture [54]

In the different tests that were performed in this algorithm we assessed the results with different parameters values, such as the learning rate, the number of hidden layers, the activation functions, the optimizer and the threshold that determines if an instance is normal or anomalous.

3.2.2 Convolutional Autoencoder

The decision to test the Convolutional Autoencoder was taken based on the fact that it was noticed in the dataset, that when a cyber attack appears it does not create only one flow, it creates multiple consecutive flows, so we used this type of Autoencoder to keep track of the order of the instances. The objective is that this type of algorithm would reconstruct a window of a certain number of instances so it could learn patterns of combinations of instances.

The implementation of the Convolutional Autoencoder is very similar to the one done with the Conventional Autoencoder, with its architecture presented in Figure 3.2. In the case of this algorithm the input is composed of a window of flows instead of a single flow, each windowed input goes through an encoder and then through a decoder same as in the Conventional Autoencoder.

The reduction of dimensions is performed by a Convolutional layer which hash a kernel window that goes through the input and shrinks the dimension according to the window. Following a formula previously accorded it is calculated the error of the instance decoded based on the original instance. Taking into consideration a threshold previously agreed upon, we check if the error is higher or lower than that threshold. If it is lower the instance is considered as normal, if it is higher then it is considered as an anomaly.

As there is not an implemented Python function for this algorithm, we implemented it on our own.

In the different tests that were performed in this algorithm we assessed the results

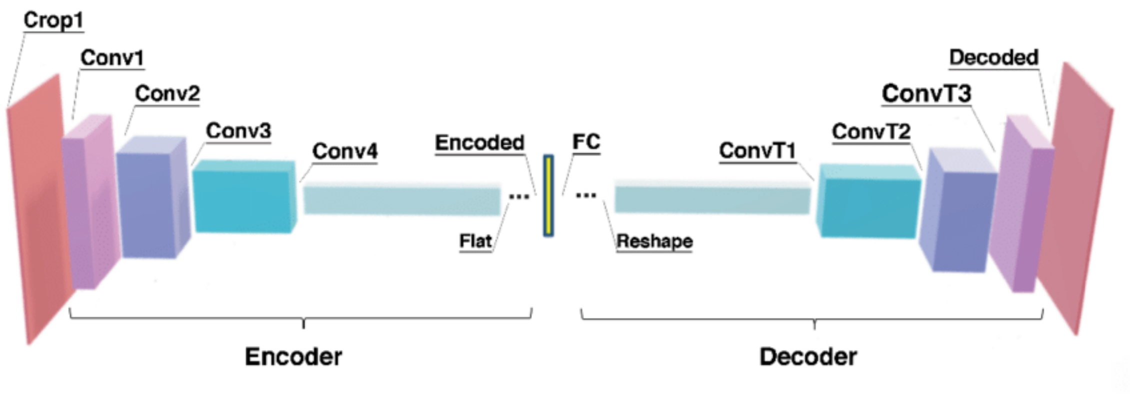

Figure 3.2: Convolutional Autoencoder Architecture [55]

with different parameters values, such as the learning rate, the kernel sizes, the number of filters, the activation functions, the optimizer and the threshold that determines if an instance is normal or anomalous.

3.3 Dataset

The dataset used to evaluate the performance of our approaches was the CIC-IDS2017 [56]. We chose this dataset first because it was the one that had, previously, been used by the people that started this research,before me, at OneSource, so they already had done an assessment of the best datasets they could use and decided this one was the best. Besides that point, this dataset was a really recent one, compared to the ones usually used, it contained latest threats and features, which were not addressed by older datasets and fulfilled the criteria of real-world attacks [57].

The CIC-IDS2017 dataset is composed of eight different files one for each day of the week (excluding weekends), two for Thursday and three for Friday, each one with different types of attacks, composed of data gathered from the time span of the working hours of these days. The whole dataset is composed of 3119345 instances, 81 features and contains 15 class labels (1 normal and 14 attack labels), it is constituted by network traffic captured over a period of 3 weeks. The dataset has 1338 NaN and 4126 infinte values. Thursday afternoon and Friday data are really fit for binary classification since they only have bening data and 1 kind of attack, the rest of the files, on the other hand are best for multiclass classification since they are composed of benign data and various other attacks.

Table 3.1 illustrates the distribution of instances by class.

As for the features, all of them are numerical, except the label column, which is categorical. Before we talk about the feature set it is important to detail some aspects. Forward packets refer to packets that are being sent from one device to another along the path to its destination, while backward packets refer to packets that are being sent in the opposite direction, usually as a response to a forward packet, it acknowledges the receipt of forward packets.

Class Labels	Number of instances
Benign	2359087
DoS Hulk	231072
PortScan	158930
DDoS	41835
DoS GoldenEye	10293
FTP-Patator	7938
SSH-Patator	5897
DoS slowloris	5796
Dos Slowhttptest	5499
Bot	1966
Web Attack - Brute Force	1507
Web Attack - XSS	652
Infiltration	36
Web Attack - SQL Injection	21
Heartbleed	11

Table 3.1: Number of instances by class of CIC-IDS2017 dataset

Flow Inter-arrival Time (IAT) is the measure of the time elapsed between the arrival of consecutive packets of a flow in a network. Push (PSH) Flag used to indicate that the receiving device should pass the data to the application immediately upon receipt, rather than buffering it. Its values are either 0 or 1. Urgent-Pointer (URG) Flag used to indicate that the data in the packet has urgent priority and should be passed to the application as soon as possible. Its values are either 0 or 1.

A bulk transfer, also known as a bulk data transfer, is a method of transferring large amounts of data between devices or systems. A Subflow is a logical or physical connection within a larger network flow that allows for the transfer of data between two devices.

The features that the dataset presents are the following(some may be in the same point because they are similar):

- ***Destination Port:*** TCP or UDP port that receives the data.
- ***Flow Duration:*** Amount of time that a connection remains open between two devices.
- ***Total Forward Packets*** and ***Total Backward Packets***: Number of packets that are being sent in one direction or the other in a network connection.
- ***Total Length of Forward Packets*** and ***Total Length of Backward Packets***: Amount of data each packet contains.
- ***Forward Packet Length Max, Min, Mean*** and ***Std***: Minimum and maximum average and standard deviation of the size of the forward packets of data that are being transmitted.
- ***Backward Packet Length Max, Min, Mean*** and ***Std***: Minimum and maximum average and standard deviation of the size of the backwards packets of data that are being transmitted.

- ***Flow Bytes/s***: Known as flow rate, it is the measure of the amount of data that is being transferred over a network in a second.
- ***Flow Packets/s***: Known as packet rate, it is measure of the number of packets that are being transmitted over a network in a second.
- ***Flow IAT Max, Min, Mean*** and ***Std***: Maximum, minimum, average time of the flow inter-arrival time of all packets and also the standard deviation.
- ***Forward IAT Total, Max, Min, Mean*** and ***Std***: Total, maximum, minimum, average time of the flow inter-arrival time of forward packets and also the standard deviation.
- ***Backward IAT Total, Max, Min, Mean, Std***: Total, maximum, minimum, average time of the flow inter-arrival time of backward packets and also the standard deviation.
- ***Forward PSH Flags*** and ***Backward PSH Flags***: Indicates the number of PSH flags there are in the forward and in the backward packets.
- ***Forward URG Flags*** and ***Backward URG Flags***: Indicates the number of URG flags there are in the forward and in the backward packets.
- ***Forward Header Length*** and ***Backward Header Length***: Indicates the size of the header in bytes.
- ***Forward Packets/s*** and ***Backward Packets/s***: Is a measure of the number of packets that are being transmitted over a network in a second.
- ***Packet Length Min, Max, Mean, Std*** and ***Variance***: Indicates the minimum and maximum length, the mean and the standard deviation of the length of the packets.
- ***FIN, SYN, RST, PSH, ACK, URG, CWE, ECE Flag Count***: Indicates the number of flags there were, each one is a different feature and a different count.
- ***Down/Up Ratio***: Indicates the ratio of data being downloaded (received) to data being uploaded (sent) in a network.
- ***Average Packet Size***: Indicates the Average size of the packets.
- ***Average Forward Segment Size*** and ***Backward Segment Size***: Indicates the Average size of the segments.
- ***Forward Header Length***: Indicates the Length of the Forward header.
- ***Forward Average Bytes/Bulk*** and ***Backward Average Bytes/Bulk***: Indicates the measure of the amount of data that is being transmitted in a single bulk transfer in forward or backwards packets. It is typically used to measure the performance of data transfer operations
- ***Forward Average Packets/Bulk*** and ***Backward Average Packets/Bulk***: Indicates the average of the measure of the number of packets that are being transmitted in a single bulk transfer of the forward or backwards packets. It is typically used to measure the efficiency of data transfer operations

- ***Forward Average Bulk Rate*** and ***Backward Average Bulk Rate***: Indicates average of the rate at which bulk transfers are occurring on a network either in forward or backwards packets.
- ***Subflow Forward Packets*** and ***Subflow Backward Packets***: refers to the number of packets that are being sent from one device to another along the path to its destination within a specific subflow.
- ***Subflow Forward Bytes*** and ***Subflow Backward Bytes***: refers to the number of bytes that are being sent from one device to another along the path to its destination within a specific subflow.
- ***Active Mean, Std, Max*** and ***Min***: Mean, standard deviation, maximum and minimum time a network connection was active.
- ***Idle Mean, Std, Max*** and ***Min***: Mean, standard deviation, maximum and minimum time a network connection was idle.

3.4 Summary

This chapter gives a deeper view of the projects where our work was integrated in and in what capacity it added something to them, as well as, what algorithms we used in our approaches, how they were used as classifiers and how our approach was trained and validated. We also presented the dataset we used to evaluate and train the models.

The algorithms we addressed, Conventional Autoencoder, Convolutional Autoencoder, all are based on Unsupervised Learning. This decision was made based on the fact that this type of algorithms is able to detect threats unknown to the system and are faster in the classification. Both algorithms that were used are based on Deep Learning, another approach that has potential, taking into consideration the good performances it produces in classification and how they can deal with high-dimensional data with quite ease compared with other algorithms.

Chapter 4

Implementation

The following chapter presents the techonologies and implementation processes used to implement the algorithms used, the preprocessing done in each dataset and the tests performed for each algorithm before we settled on the best one to analyze more in depth.

The first section, 4.1 presents the preprocessing done in each dataset as well as a simple exploratory data analysis, the following section, 4.2 presents, how each algorithm was implemented as well as the various architectures tested for each algorithm. In 4.3, we presents the division made on the datasets and how was the process of training and validating the models.

4.1 Datasets Preprocessing

4.1.1 CIC-IDS2017

The dataset used, as mentioned before, was the CIC-IDS2017, composed of 81 features. The dataset was divided into multiple files for each day of the week, so we started by joining them all and making a simple preprocessing of the entire dataset. Since it was decided that we would be performing a binary classification, the label column was altered so the benign instances had the value 0 and all the instances considered anomalies had the value 1.

After that we looked for NaN values and infinite values, so that we could eliminate those rows, as they would be prejudicial to our models. The preprocessing of the dataset was finished by eliminating all the features that were constant, meaning they had the same value in the entire dataset, as they would not be beneficial in any way to our work, since they are redundant. Finally the dataset ended up with 70 features.

In the Figure 4.1 it is presented the values of each feature for a normal instance and an anomalous instance. As we can see the values of the errors of each feature in the normal flow have a maximum of less than 2, while in the anomalous flow the maximum is higher than 17.5.

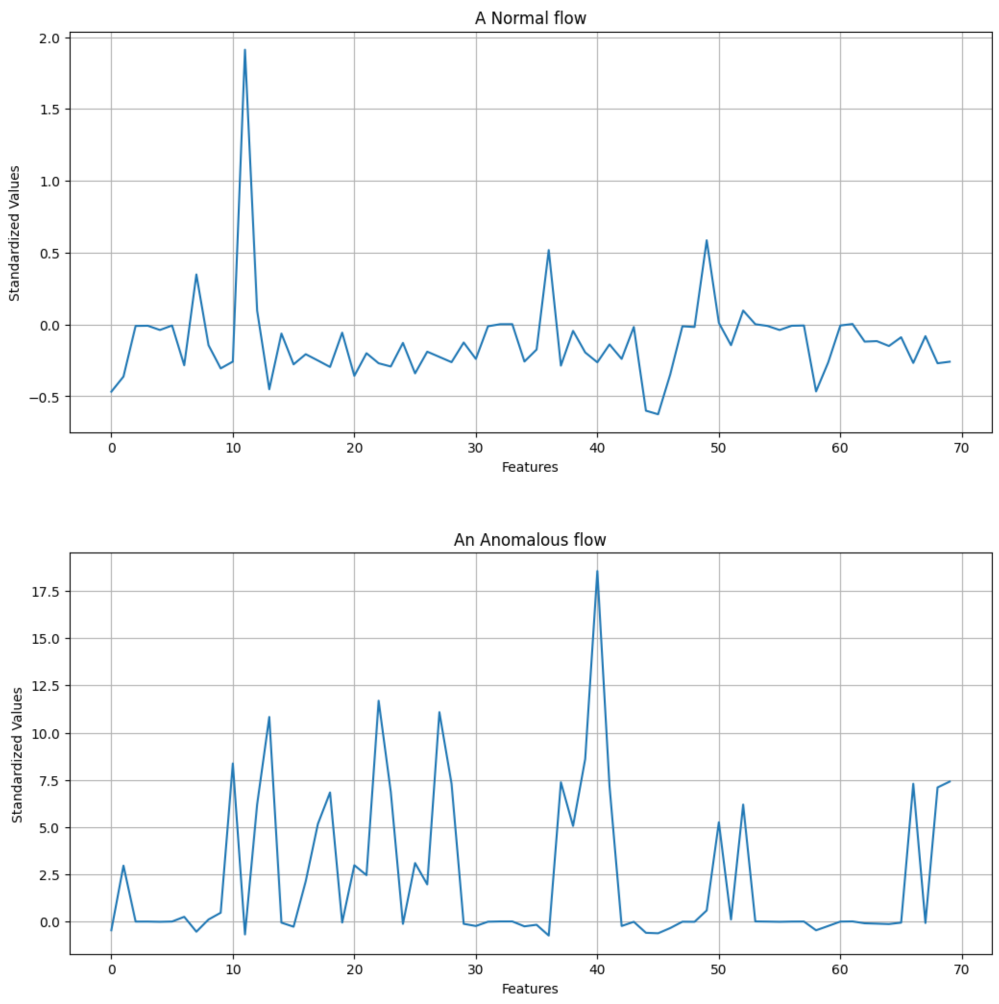

Figure 4.1: Example of a Normal and Abnormal flow values per feature

After dealing with the preprocessing of the dataset, the next step was to divide it into a training, a validation and a testing datasets. The training and validation datasets, both, had only normal data and the testing set normal as well as anomalous instances.

As the main focus of this work was to create a model capable of detecting anomalies in real time scenarios and it was not possible to assume a range of values with certainty, instead of normalizing the data as most of the implementations presented in the state of the art, the data was standardized using the Standard Scaler, which subtracts the values of each instance (x) by the mean of the data (μ) and divides it by its standard deviation (σ).

$$z = \frac{x - \mu}{\sigma} \tag{4.1}$$

For this models, it was decided that performing any kind of feature reduction would not make as much sense, since Autoencoders already perform a dimensionality reduction and it would not increase the performance of the model.

4.1.1.1 Windowed Datasets

One of the algorithms tested were the Convolutional Autoencoders. This type of algorithm required a different kind of dataset, the datasets we presented previously had 2 dimensions, (samples x features), while the ones that a Convolutional Autoencoder received were 3 dimensional, (samples x time x features). In order for us to be able to use this algorithm we converted the original dataset as a windowed one, achieving the 3 dimensions.

At first, the initial idea was to create a windowed dataset for each day of the week, and use the Monday dataset as training, since it was constituted of only normal data, and the following ones as testing. To reach a conclusion if it was possible to make this separation or not, a simple analysis was performed. We wanted to check if the number of samples in the Monday dataset was big enough and if its features values were similar to the features values of the other datasets.

In order for us to perform this analysis we checked the length of each dataset with the purpose of assessing if the Monday dataset was big enough, Figure 4.2.

We can conclude that the Monday dataset was the largest one with exception for the Wednesday one, but this one was mainly composed of anomalous data.

It was also used the Man-Whitney test to compare the statisitical similarity of the features of the Monday dataset with the remaining, Figure 4.3. It was possible to notice that from the 80 existing features only 13 have a pvalue higher than 0.05, [*Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, Bwd Packets/s, RST Flag Count, CWE Flag Count, ECE Flag Count, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate*]. So we could not affirm that the Monday dataset could be used as training because the Monday dataset compared to the other ones did not have a high similarity

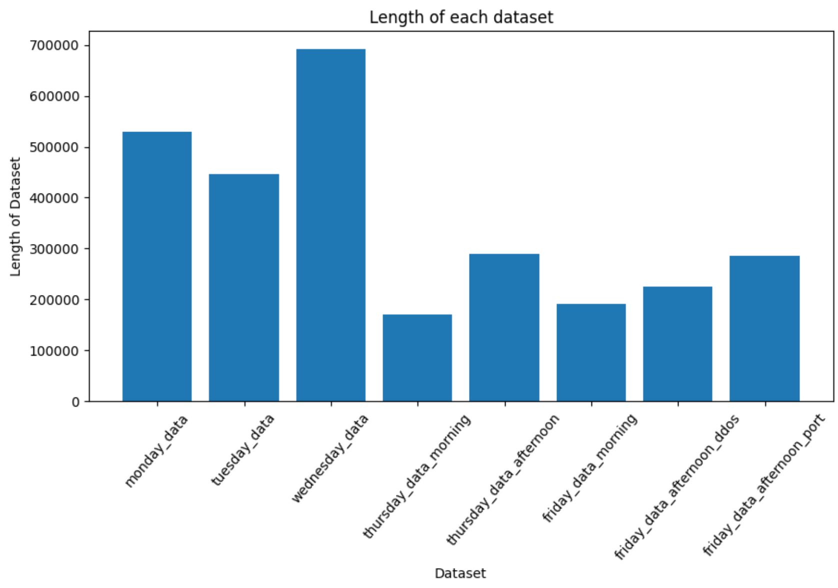

Figure 4.2: Datasets Length

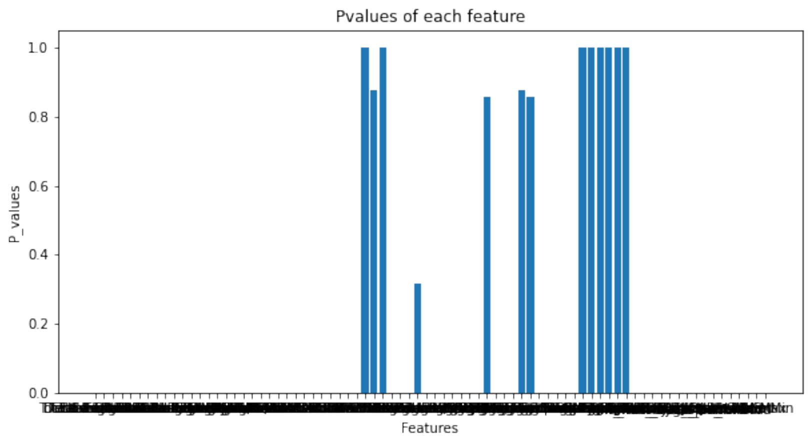

Figure 4.3: Features pvalues

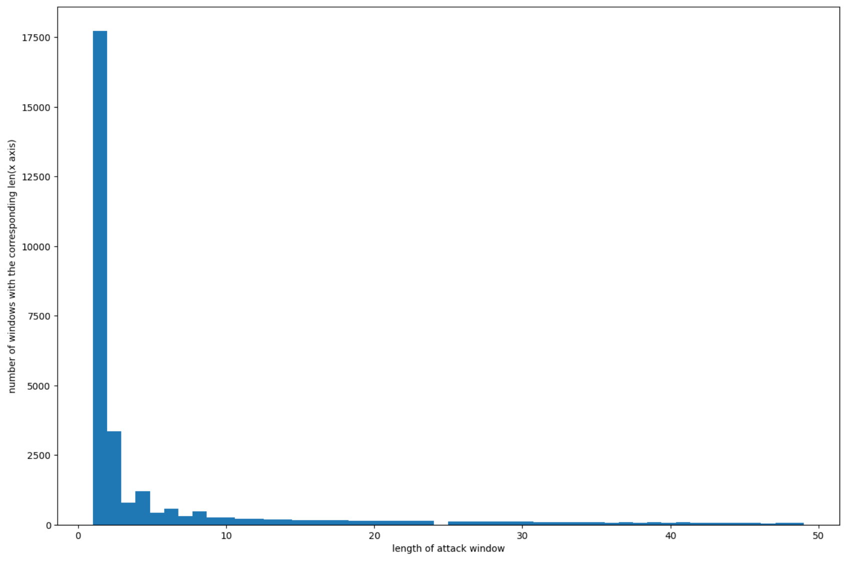

Figure 4.4: Number of windows with the different lengths

between features which was what we were trying to prove in order to use the dataset as training set.

The next step was to assess the best value for the window size. After careful consideration, it was decided that the best strategy to find this length was to find the mean and the standard deviation of the length of the windows composed, only, of anomalous flows.

So our strategy was, first to check the number of anomalous flows in a row there were and save those as window lengths. After having all of the possible lenghts we checked the number of windows with each length, presented in Figure 4.4.

It was possible to notice that most of the attacks in the datasets were composed of only one anomalous flow, a conclusion that we were not expecting. We wanted the minimum possible number for the length of the windows because the majority of the sizes were low and if the size of the window was too high the anomalous flows inside of the window could be overshadowed.

So we selected the value as the sum between the mean and the standard deviation of the lengths of the windows, 2 and 2 respectively, so the selected value for the size of the window was fixed on 4 but we wanted to test with different sizes to see how it influenced the performance of the models, so we chose a bigger size as well, 8.

All the datasets were concatenated and create a windowed dataset. The windows with at least one attack were considered anomalous windows.

Finally it was additionally checked how many windows there were considered as normal and anomalous, for length 4 (Figure 4.5), as well as length 8 (Figure 4.6).

We could see that in both situations there was more or less the same number of normal and abnormal windows, but when the length was 8 there were more abnormal windows contrarily to when the length was 4.

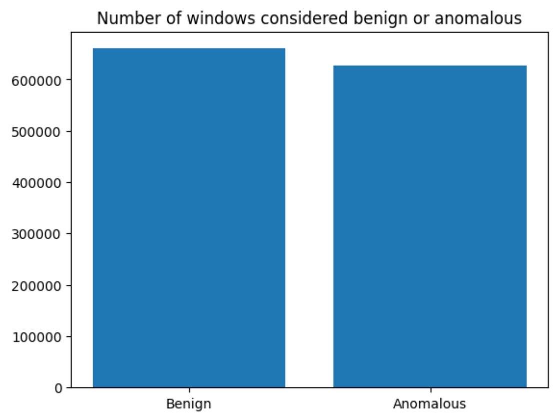

Figure 4.5: Number of normal and abnormal windows with length 4

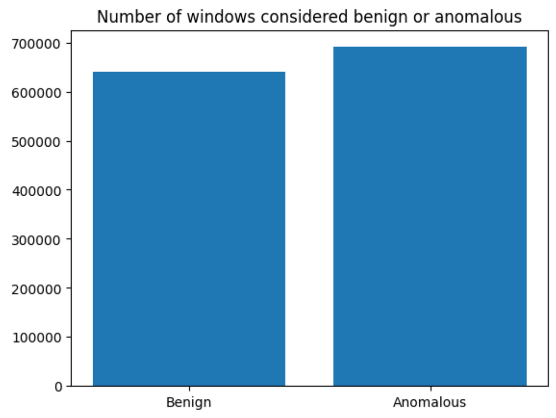

Figure 4.6: Number of normal and abnormal windows with length 8

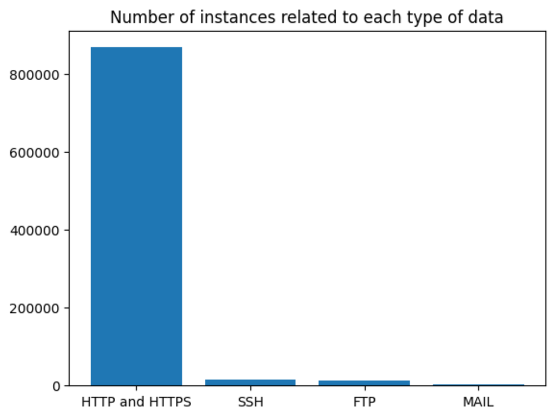

Figure 4.7: Number of instances for each type of traffic

4.1.1.2 Different types of traffic datasets

Based on a different analysis we intended on performing, create a model for each different type of traffic, it was necessary to implement a different preprocessing method for the datasets.

In terms of cleansing and standardization, it was the exact same methods as before. What had to change was the type of data inside of the datasets. The CIC-IDS2017 dataset has 5 different, known, types of traffic, which can be distinguished by checking the feature *Destination Port*. Those kinds of traffic and its corresponding *Destination Port* are:

- HTTP: 80
- HTTPS: 443
- SSH: 22
- FTP: 21
- Mail: 110, 995, 143, 993, 25, 26, 465

The datasets creation was performed by searching for these destination ports and creating subsets of the dataset for each type of traffic, HTTP and HTTPS were joined as one, since the type of traffic was very similar.

4.1.2 Custom Datasets

The custom datasets used were extracted from the platform Mobitrust. These datasets were similar to the CIC-IDS2017 since this dataset served as influence to create the datasets this platform provides.

The datasets were extracted from different applications, since each application had different types of traffic with different values in each feature. We had four different datasets each with different types of traffic:

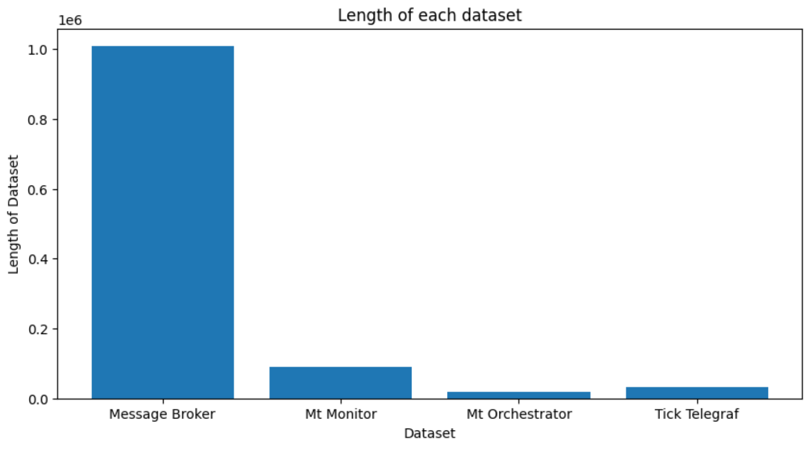

Figure 4.8: Length of each custom dataset

- ***Message Broker:*** Composed of messages exchanged through multiple devices and uses TCP protocol.
- ***Mt Monitor:*** Composed of Video and Sound and uses both TCP as well as UDP protocols.
- ***Mt Orchestrator:*** Composed of communications performed between different components and uses TCP protocol.
- ***Tick Telegraf:*** Composed of Time Series and uses TCP protocol.

The data present in each dataset was extracted from multiple simulations of different applications. All the datasets were composed of 80 features initially, also none of them had attacks.

In the custom datasets, the preprocessement that was done, was exactly the same as the one done in the CIC-IDS2017. First we turned the Label column from a string to a integer column with 0 for normal instances. After that the Nan and the infinite values were identified and the rows in which those values were present, were eliminated. Since it was defined earlier that the best approach was to create a model for each type traffic data, we skipped the concatenation of all of the datasets and trained a model for each application and gathered the results it presented.

After this we eliminated the features that were constant, the values were always the same, exactly the same way as we did in the previous dataset.

The next step was to standardize the data using a Standard Scaler the same way as it was done in dataset from the State of the Art and it was decided as well that it would not be done any kind of preprocessing of the dataset, since Autoencoders already perform a reduction of dimensionality and it would not benefit our results in any way.

In Figure 4.8 it is presented the lengths of each custom dataset provided by the platform Mobitrust. As we could see the Message Broker had considerably more data than the rest of the datasets and this was an important aspect for the training of the data, low data could lead to bad training of the models.

4.2 Models Architectures

4.2.1 Conventional Autoencoder

Autoencoders are a ML algorithm based on Unsupervised Deep Learning. This type of algorithms, as it was shown, usually show some really good results in identifying anomalies and still have some space for improvement. Another aspect is the fact that it shows a fast classification of instances compared to other algorithms, something that is really important in our study as it will be used in real time scenarios.

The problem using an Autoencoder was divided into two parts, the first one being the training of the model in order for it to learn how to reconstruct the normal instances with the lowest possible error and the second part finding the best threshold value, for the reconstruction errors, to make the separation between the normal instances and the anomalous ones.

The implementation of the Autoencoder started by performing a simple grid search in order to understand and select the best hyperparameters to train our model. The selected hyperparameters were:

- Number of layers: [7 (dimensions of each layer 70, 35, 18, [7, 5, 3], 18, 35, 70), 9 (dimensions of each layer 70, 50, 30, 10, [7, 5, 3], 10, 30, 50, 70), 11 (dimensions of each layer 70, 55, 40, 25, 10, [7, 5, 3], 10, 25, 40, 55, 70)]
- Hidden layers activation function: [*ReLU*, *eLU*, *LeakyReLU*]
- Optimizers: [*Adam*, *Nadam*]
- Bottleneck layer dimensions: [3, 5, 7]
- Learning Rate: [0.001, 0.0001, 0.00001]

The values for the number of layers chosen were these ones because we wanted to test how different number of layers as well as different decays in layers dimensions would impact the reconstruction learning step. The activation functions and the optimizers were selected taking into account a previous investigation on the matter and by some preliminary tests made and these were the ones that presented bests results in learning how to reconstruct the instances.

The bottleneck layer dimensions were selected taking small values in order to be more difficult for the anomalous instances to be reconstructed, meaning if the bottleneck layer was higher the anomalous instances would probably be easier to reconstruct without being trained and with the finality of understanding how different values would influence the performance of the Autoencoder.

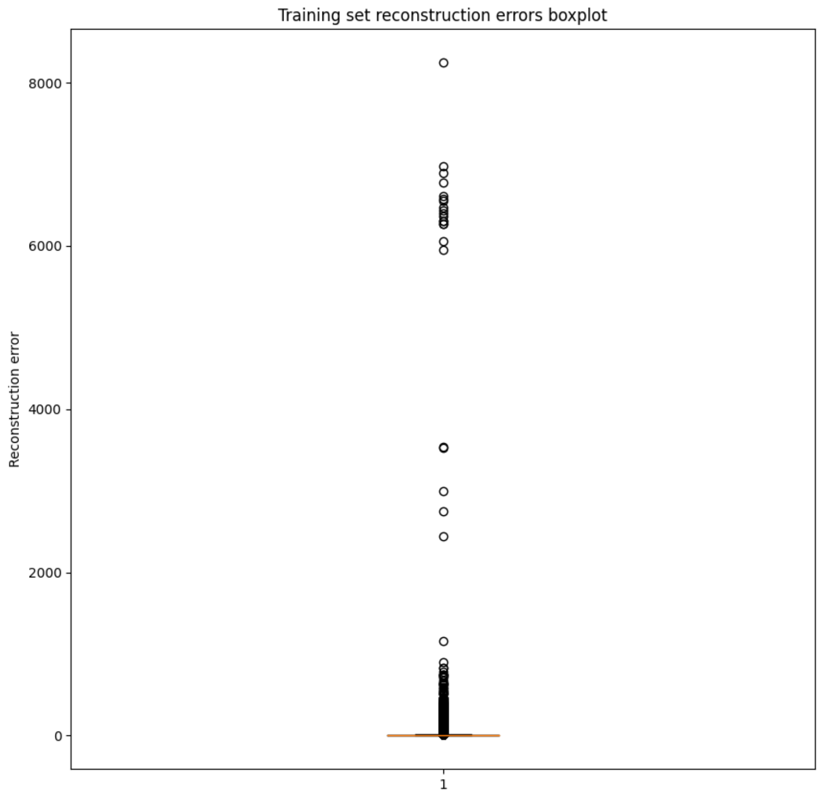

Figure 4.9: Reconstruction Errors

The activation function of the output layer was fixed as *Linear* because of the fact that we used standardization instead of normalization, so the output data ranged between 2 values different from 0 and 1, positive or negative, so the activation function that made more sense in this situation since it could output such values was the linear one because of the type of function it was.

The best hyperparameters combination was selected based on the validation loss, the lowest ones were selected to check the performance in classifying the instances.

As for the second part, the classification, what was meant to do was, reach the optimal threshold value so it would be possible to make a separation between the normal and the anomalous instances based on that value. In order to find that threshold value it was gathered all the reconstruction errors of the training and validation sets and calculated the Mean Squared Error of the original instance and the reconstructed instance, after this it was calculated the mean and the standard deviation of the reconstruction errors.

We reached the conclusion that there was a large number of normal instances that presented a great reconstruction error, some higher than 1000, as we can see in Figure 4.9. Considering this instances as outliers, they were eliminated, as they increased the mean and the standard deviation, which gave a bad threshold.

The detection of this outliers was performed using a Interquartile Range based outlier detection. First it was calculated the upper bound which was the 75% percentile summed to the 1.5 times the interquartile range, which is the difference

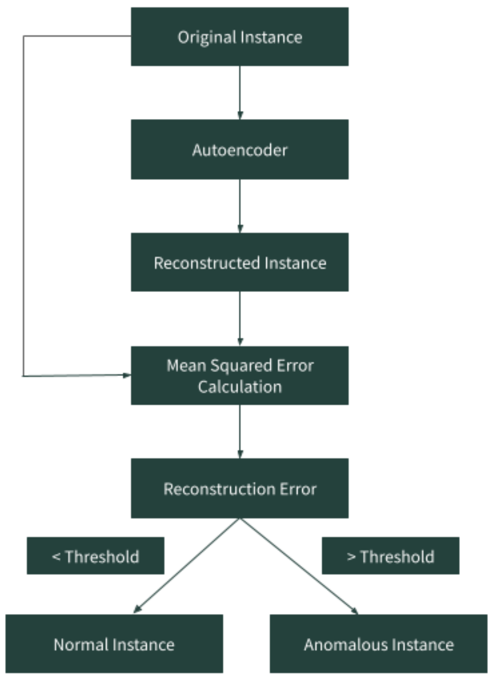

Figure 4.10: Conventional Autoencoder Approach Process

between the 75% percentile and the 25% percentile of the data. After that all the errors that were above that upper bound were eliminated. After this preprocessing the threshold produced a value much more acceptable.

So we could achieve the threshold that gave us the best performance possible in classifying, we used the F1-Score metric to select which threshold had the best performance. The metric selected to check the best results was F1-Score because, the most important features for our problem are Recall and Precision and the metric we selected was an harmonic mean between the two latter. The threshold was calculated by summing the mean (μ) and the standard deviation (σ) multiplied by a certain value, x, which was the only parameter we changed.

$$threshold = \mu + (x * \sigma) \tag{4.2}$$

The calculation of the error of each instance was performed by calculating the Mean Squared Error of the original instance with the reconstructed one, as it is presented in the next formula, 4.3

$$error = \frac{\sum_{i=1}^{n}(x_i - y_i)^2}{n} \tag{4.3}$$

The formula calculates the error of every feature in the instance and outputs its mean, i is the feature, x is the original instance, y is the reconstructed instance and n is the number of features.

The overall process of the approach is presented in the Figure 4.10.

4.2.2 Convolutional Autoencoder

Convolutional Autoencoders are similar to Conventional Autoencoders but have some differences. This type of algorithm is based on Unsupervised Deep Learning, as well, and merges the paradigm of the conventional Autoencoders and the Convolutional Neural Networks.

The anomaly detection using Convolutional Autoencoders was still divided into two parts as well, the first one being the reconstruction of the instances, in this case the windows, with the minimum error and only after that the classification of the windows.

The implementation of this algorithm started by performing a hyperparameter tuning with the help of the framework Optuna, which executes a Bayesian optimization, skipping some trials that are not promising, saving us time. The selected hyperparameters were:

- Optimizers: [*Adam, Nadam*]
- Hidden layers activation function: [*ReLU, eLU, leakyReLU*]
- Learning Rate: [0.001, 0.0005, 0.0001]
- Number of Filters: [1, 4, 8]
- Kernel size (Features): [4, 12, 20]
- Kernel size (Flows): [2, 4, 8]

One important hyperparameter we decided to not test more values was the stride. It was decided that this hyperparameter would have its value fixated at 2 in order for the dimensionality reduction to be significant from layer to layer. If it was 1 the reduction of dimensionality would be slow and take much more time to train and to make the predictions, because it would need more layers to reach a low bottleneck dimension, while 3 would be too rapid and most of the times reach the bottleneck dimension in the second layer which was not optimal. Finally it was concluded that 2 was the best value to use as stride.

The activation functions and the optimizers were selected by the same way they were selected for the Conventional Autoencoders, conducting some preliminary tests and reading some examples from previous papers.

The values selected for the number of filters were selected based on the knowledge that we would want a small number of filters since our datasets were not that complex, so we decided to go with smaller number for the number of filters.

The dimensionality reduction calculations are performed taking into consideration the kernel size. The values selected had to be checked before using them because of the math, as some sizes would give errors, most of the times negative dimensions. The kernel size corresponding to the features is horizontal while the one corresponding to the flows is vertical. The values that were selected, are justified by our need of catching a low number of blocks as well as a high number of blocks.

The training was performed with 100 epochs and a batch size of 32, early stop with patience of 10 and the loss was calculated using the Mean Squared Error.

After the training of the model we passed to the classification part. The prediction of the values in this algorithm gave an error for each flow inside of the window, which was not what we wanted, so in order for us to get the error of the entire window, we calculated the mean of the error inside of each window in order for it to give the reconstruction error of the window. The equations used to calculate the threshold and the errors were the same ones that were used in the Conventional Autoencoder.

The same way as it was done in the Conventional Autoencoder it first had to be found an optimal threshold to use as the value that would separate the windows considered normal from the anomalous ones. To find that value we proceeded to use the trained model to reconstruct the instances present in the training set and gather all these reconstruction errors. After that it was calculated the mean and the standard deviation of the errors.

Since there is a quite large number of normal instances with great errors, which were considered outliers, these errors were eliminated, as they increased quite a bit the mean and greatly the standard deviation, which gave a bad threshold.

The detection of this outliers was executed the same way as in the Conventional Autoencoder. Contrarily to what was seen in the Conventional Autoencoder, the best value to multiply so we could get to the optimal threshold always had the best results when it was 1 so we fixated this value.

The overall process of the approach is presented in the Figure 4.11.

4.3 Training and Validation of the models

This section provides an overview on what were the techniques used to implement the algorithms and its architectures as well as the tests conducted for each Machine Learning algorithm.

The programming language used during the implementation process was *Python*, more precisely, Python 3.9.2. We also used Python's library scikitlearn as well as the frameworks Tensorflow, Pandas and Optuna.

After completing the datasets preprocessing and the division in windowed dataset or division in subsets, explained in detail in the previous section 4.1. The following step was to divide the dataset into a training set, a validation set and a testing set, for training and validation purposes.

The division of the datasets and the training that was performed are shown in Figure 4.12, the entire datasets were divided into a training set which was com-

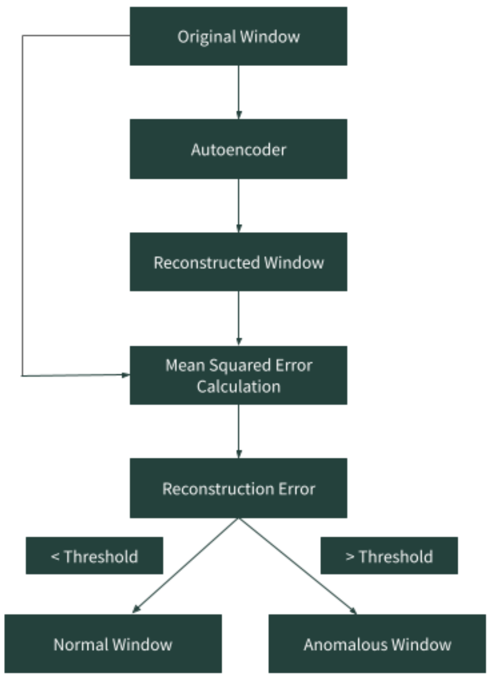

Figure 4.11: Convolutional Autoencoder Approach Process

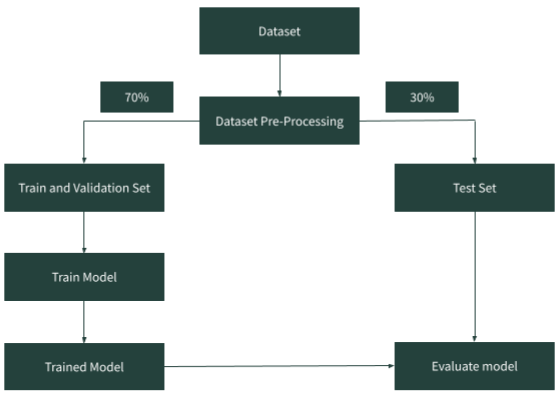

Figure 4.12: Training and Validation of the models

posed of only normal data, so we got only the normal data and divided it 70% and 30% with these 30% being data that would be used in the testing set. The training set would also be divided in 70% for training and 30% for a validation set, as we were building Deep Learning models there was a need to have a validation set to keep validating the training of our model in every iteration. Finally the 30% of only normal data that would be used for the testing set would be concatenated with all the anomalous data, to give us the testing set that would be used to check the performance of the model.

The training of the models was performed with 300 epochs and a batch size of 64, early stop with patience of 20 and the loss was calculated using Mean Squared Error, for the Conventional Autoencoder. The batch size was quite high for the training to be quicker and because of some preliminary tests we concluded that it made not difference on training with other values of batch size, the patience was the value that seemed best for us to see if the model had stopped learning or not. While for the Convolutional Autoencoder it was performed with 100 epochs and a batch size of 32, early stop with patience of 10 and the loss was calculated using Mean Squared Error. The number of epochs and patience was lower because the model needs less epochs to stop learning, the batch size was lower as well because of constraints in the Virtual Machine resources, we needed the model to consume less RAM so the Machine would not crash in training.

4.4 Summary

This chapter covered two important aspects: the pre-processing of the the datasets and the implementation of the candidate models. As part of the datasets pre-processing, a description of each step performed in the pre-processing of the datasets, both CIC-IDS2017 and custom datasets, as well as how the windowed datasets were created and how it was decided on the best size for the windows.

Afterwards, it was described the split performed in the datasets and which ones were used for the training and which ones used for the validation of the models.

Finally, it was presented the implementation of all the algorithms tested, Conventional Autoencoder and Convolutional Autoencoder. Additionally to the implementation it was also presented all the tests performed in order for us to get to the best parameters combination to achieve the best model possible, as well as a description on why we selected the values for each model's hyperparameters.

Chapter 5

Results and Discussion

The following chapter presents the results attained from the different approaches implemented and a deeper discussion and comparison of the various algorithms and models tested.

The chapter is divided in 5 sections. In the first one, 5.1, we present the results of the Random Forest as a baseline model, in order for us to justify the usage of Unsupervised Learning. In section 5.2 and 5.3 we present the performance results of the Conventional Autoencoder and Convolutional Autoencoder in various datasets.

Finally in 5.4 it is performed a deeper analysis of the performance results of each algorithm as well as a comparison between them with the intent of selecting the best one.

5.1 Random Forest

First it is presented a baseline model used in previous studies for this work, [58]. This model was used as base for comparison with the work we developed, mainly to show that Unsupervised Learning was better than Supervised Learning in this work in particular. All the parameters used in the algorithm were the default ones, as it was the same way as it was done in the previous studies.

As we could see in Table 5.3 and Table 5.4 the model presented a very good performance, almost perfect, something that had been already concluded in the previous studies made.

Metric	**Score**
Precision	1.0
Recall	1.0
F1-Score	1.0
FNR	0.00

Table 5.1: Random Forest Classification results

	Actually Positive	**Actually Negative**
Predicted Positive	13%	0%
Predicted Negative	0%	87%

Table 5.2: Random Forest Confusion Matrix

Metric	**Score**
Precision	1.0
Recall	0.15
F1-Score	0.27
FNR	0.85

Table 5.3: Random Forest (DoS) Classification results

The main problem with using Supervised Learning approaches was the fact that it could not detect new anomalies, that were unknown to the model, known as zero day attacks. Having this aspect in mind we trained a model using the entire dataset except the *DoS* attacks, to check how well the model could detect this type of attacks without being trained to do so.

As it was possible to notice by the results presented in Table 5.3, the results were worse than before. The test set, used for analysis of the model's performance was composed of DoS attacks only and as we could see the False Negative Rate was extremely high, the model was only capable of detecting 15% of attacks. The Precision was 100% because as the data was only attacks all the instances classified as attacks were actually attacks so it had to give 100%, but this metric was not important for this analysis.

We can conclude by this analysis that, when the model was fed new attacks that it was not trained to identify it would not be able to classify this attacks correctly. This constituted a problem to our solution as new attacks are constantly being discovered.

5.2 Conventional Autoencoder

This section presents the training and prediction results achieved with the Conventional Autoencoder. The results that are presented are correspondent to the entire dataset and to subsets of the dataset corresponding to the different types of known traffic, in this subsets it was also evaluated the performance of the model with a parameterization using the entire dataset and a parameterization using only the subset in question.

	Actually Positive	**Actually Negative**
Predicted Positive	15%	0%
Predicted Negative	85%	0%

Table 5.4: Random Forest (DoS) Confusion Matrix

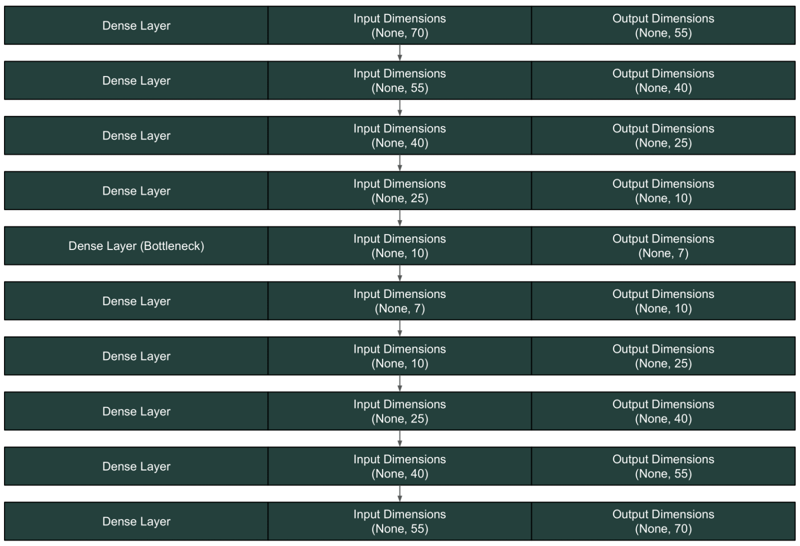
Dense Layer | Input Dimensions (None, 70) | Output Dimensions (None, 55)
Dense Layer | Input Dimensions (None, 55) | Output Dimensions (None, 40)
Dense Layer | Input Dimensions (None, 40) | Output Dimensions (None, 25)
Dense Layer | Input Dimensions (None, 25) | Output Dimensions (None, 10)
Dense Layer (Bottleneck) | Input Dimensions (None, 10) | Output Dimensions (None, 7)
Dense Layer | Input Dimensions (None, 7) | Output Dimensions (None, 10)
Dense Layer | Input Dimensions (None, 10) | Output Dimensions (None, 25)
Dense Layer | Input Dimensions (None, 25) | Output Dimensions (None, 40)
Dense Layer | Input Dimensions (None, 40) | Output Dimensions (None, 55)
Dense Layer | Input Dimensions (None, 55) | Output Dimensions (None, 70)

Figure 5.1: Conventional Autoencoder Architecture

5.2.1 Entire Dataset

5.2.1.1 CIC-IDS2017 Dataset

The first tests performed were done using the CIC-IDS2017 dataset to check which models had a better performance to transpose them to the datasets provided by the Mobitrust platform.

First we performed an hyperparameter tuning to conclude which parameters combination was the best one. The most relevant results of the hyperparameter tuning are presented in Table 5.5.

Number of layers	Hidden layers activation functions	Optimizer	Bottleneck layer dimension	Learning rate	Validation loss
11	elu	nadam	7	0.0001	0.0182
11	leakyrelu	adam	7	0.0001	0.0184
11	elu	adam	7	0.0001	0.0187
9	leakyrelu	nadam	7	0.0001	0.0196
9	elu	nadam	7	0.0001	0.0201
9	elu	adam	7	0.0001	0.0207
11	elu	adam	5	0.0001	0.0210
11	leakyrelu	nadam	7	0.0001	0.0217

Table 5.5: Autoencoder parameter tuning best results

As we could conclude by analyzing the data from the hyperparameter tuning, the models with more hidden layers produced less error, which made sense, as the encoding and decoding was more gradual, the dimensions of the bottleneck layer was usually best when it was higher, 7, because the encoding was not so high and was simpler for the model to decode having a bigger dimension to begin with. The learning rate was the middle value of the ones we tested.

The best ones were the combinations of 11 layers, activation function eLU, optimizer Nadam as well as Adam, bottleneck layer dimension 7 and learning rate 0.0001, with its architecture presented in Figure 5.1. Its training performance is presented in Figure 5.2.

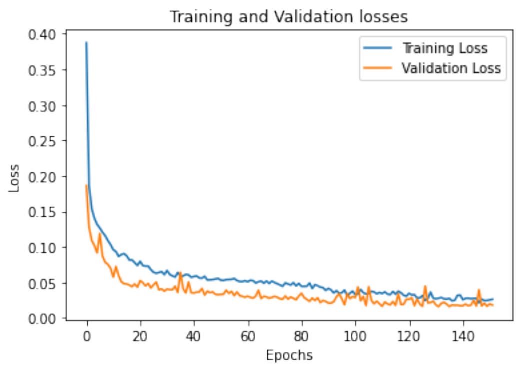

Figure 5.2: Train and Validation losses in the training of the model

x	**F1-Score**
1	0.649
10	0.714
20	0.741
30	0.755
40	0.759
50	0.753

Table 5.6: F1-Score based on the value **x**

In order for us to find the best threshold we needed to find the value, **x** that maximized the F1-Score, as it was specified in section 4.2.1. The values that the standard deviation was multiplied by, **x** were [1, 10, 20, 30, 40, 50] and the F1-Scores of each one are the ones presented in Table 5.6 As we could see the value that maximized the F1-Score was 40 so we used this one to calculate the threshold.

By combining the best model in terms of reconstruction and the best threshold attained previously we used the test set to analyze the performance of the model in reconstructing the instances and in classifying them. In terms of reconstruction it is presented two instances with the input values of each feature and the reconstruction values of each feature as well as the error in each one, Figure 5.3.

The classification results it achieved were the ones presented in Table 5.7 and Table 5.8

The model was not achieving as good results as we were expecting, the Recall was expected to be higher as well as the precision. The reasons found for such type of results were the fact that there was a quite large number of normal instances with values in some features that were way different from the expected values in

Metric	**Score**
Precision	0.92
Recall	0.64
F1-Score	0.76
FNR	0.36

Table 5.7: Classification results

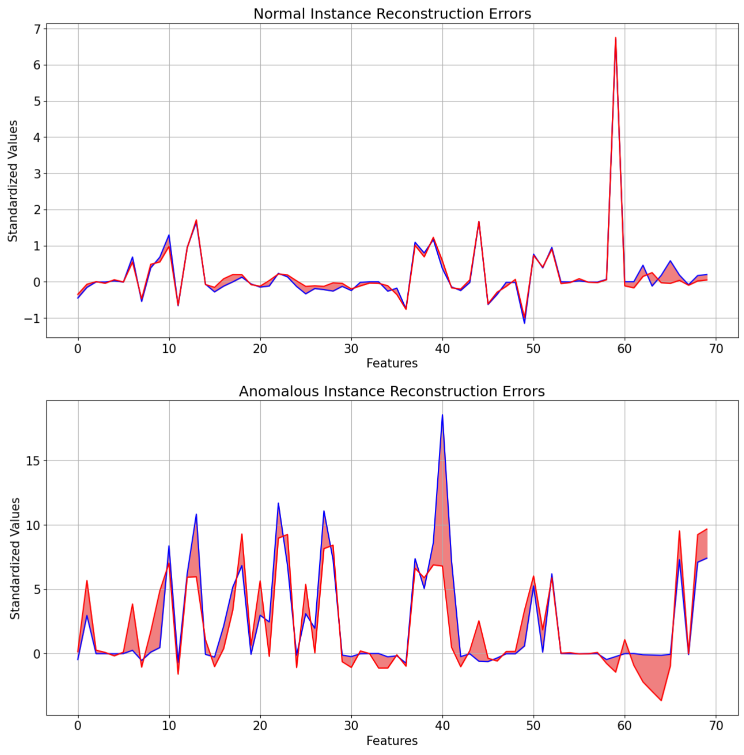

Figure 5.3: Normal and Anomalous Flows Feature Errors

	Actually Positive	**Actually Negative**
Predicted Positive	21%	2%
Predicted Negative	12%	65%

Table 5.8: Classification Confusion Matrix

those features, this made the Autoencoder perform a bad reconstruction in these instances that would later be considered anomalies, even though they were not. Another reason found was the fact that there was a high number of anomalous instances which had values that were similar or even equal to the ones of the normal instances, this made the Autoencoder perform a good reconstruction of the instances which made the reconstruction error low and these instances were considered False Negatives by the classifier.

Even though the results were not the best ones we were expecting, using, for comparison, a paper described in the State of the Art where it was used also used a Conventional Autoencoder, [42], we could see that our model outperformed theirs, as it presented a Precision, Recall and F1-Score of 0.6774 each.

5.2.2 Different traffic Datasets

5.2.2.1 CIC-IDS2017

The next section presents the results of each model implemented for each type of known traffic present in the CIC-IDS2017 dataset. The results shown are from the general parameterization and from a parameterization made for the subset in question.

HTTP and HTTPS

First it was performed an hyperparameter tuning using only the subset, using the same values used in section 4.2.1 and the best result was 11 layers, activation function eLU, the bottleneck layer dimensions equals 7, the optimizer is Adam and its learning rate equals to 0.001.

As we could see by looking at the best hyperparameters, they were not very different from the ones we achieved from the parameterization of the entire dataset, this may be because of the fact that most of the data that belongs to the HTTP and HTTPS was way larger than the other types so the general parameterization had more HTTP and HTTPS data than the other ones.

The training of the model was performed with batch size of 64, early stop with patience of 10 and the loss was calculated using Mean Squared Error, training presented in Figure 5.4.

The number of epochs to train the model was much lower than before, previously achieving the 140 epochs and now only needing 40 epochs to reach the early stoppage, this happened because of the fact that the dataset was smaller.

In terms of the classification, in order for us to get the best threshold we used the same strategy as before being the best value to multiply 5 and the results the model presented are reflected in Table 5.9

It was possible to notice that the model had an overall performance quite good

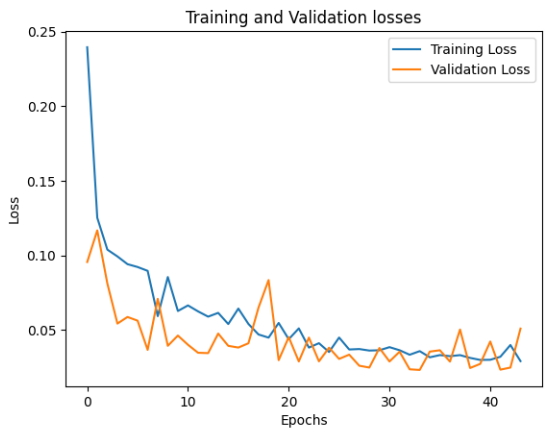

Figure 5.4: Training and Validation losses throughout the training of the HTTP and HTTPS model

Metric	**Score**
Precision	0.91
Recall	0.76
F1-Score	0.83
FNR	0.24

Table 5.9: HTTP and HTTPS specific parameterization classification results

with the Precision being 91%, which is high, and a relatively low false negative rate, 24%, although it could be lower.

After that we used the parameterization we consider general and trained the model with the same number of epochs, batch size and patience in the Early Stopping as before.

The model achieved the performance presented in Table 5.10.

It was possible to notice that the performance of the model was very good, the Precision was slightly lower than the one in Table 5.9, but the FNR was lower as well which was a positive aspect and, finally, the F1-Score was equal in both of them so the model was considered fit to use.

SSH

Metric	**Score**
Precision	0.81
Recall	0.85
F1-Score	0.83
FNR	0.15

Table 5.10: HTTP and HTTPS general parameterization classification results

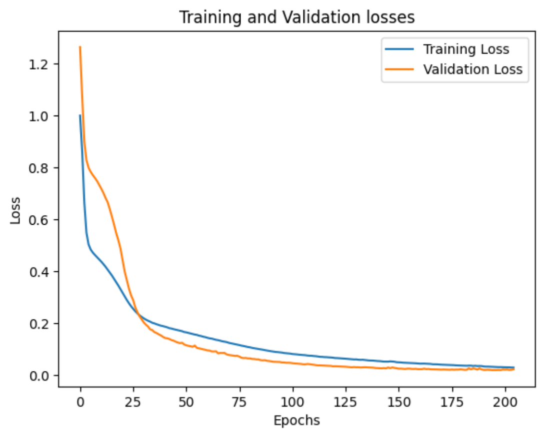

Figure 5.5: Training and Validation losses throughout the training of the SSH model

Metric	Score
Precision	0.99
Recall	0.53
F1-Score	0.69
FNR	0.47

Table 5.11: SSH specific parameterization classification results

The combination of parameters that achieved the best validation loss was 9 layers, activation function *eLU*, bottleneck layer dimensions 5, optimizer Adam and its learning rate was 0.0001.

In this hyperparameter optimization we could see that the parameters used were slightly different which shows how the change of the dataset had influence in the best combination of parameters to use to achieve the best trained model possible.

The training of the model was presented in Figure 5.5, it was possible to notice that the number of epochs that it took to converge to the best validation loss was much higher than the one needed in HTTP/HTTPS but the value achieved was almost the same.

As for the classification, the threshold used, the value multiplied was 50, a much higher number than in the previous situations, and the results of the model are presented in the Table 5.11.

Looking at the results we could see that the model had a very similar performance to the one with a specific parameterization, which made us conclude that it did not have a good performance. 47% False Negative Rate was a bad value for the metric, although the precision was almost perfect, 99%, with the FNR being so high the model was not possible to apply.

Taking into consideration the general parameterization, the model was trained exactly the same way as before and the performance it presented is displayed in Table 5.12

The model, in terms of Precision, had a really good result, which meant in all the

Metric	Score
Precision	0.95
Recall	0.53
F1-Score	0.68
FNR	0.46

Table 5.12: SSH general parameterization classification results

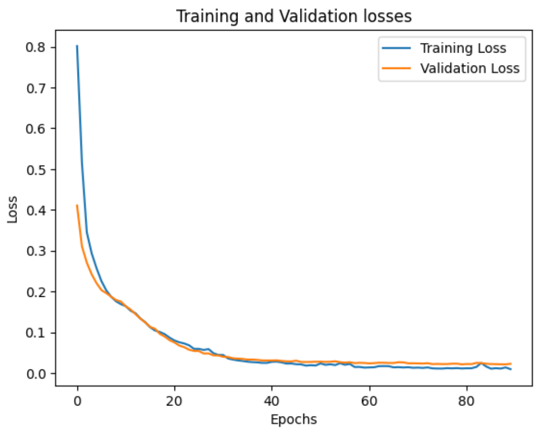

Figure 5.6: Training and Validation losses throughout the training of the FTP model

flows it considered an anomaly 95% of them were correct, but the FNR was too high, almost 50%.

FTP

For FTP the parameters used were not very different from the previous ones, 9 layers, activation function *eLU*, bottleneck layer dimensions 7, optimizer Nadam and learning rate of 0.001.

The training of the model is presented in the Figure 5.6, we could notice that the validation loss was always quite low comparing to the training loss and the lowest value was similar to the one achieved in the previous models.

The value multiplied, x, to get the threshold was 2, and the results of the model are presented in Table 5.13

This model as we could conclude had a really good performance with high values

Metric	Score
Precision	0.97
Recall	0.99
F1-Score	0.98
FNR	0.00

Table 5.13: FTP specific parameterization classification results

Metric	Score
Precision	0.96
Recall	0.99
F1-Score	0.98
FNR	0.00

Table 5.14: FTP general parameterization classification results

Metric	Score
Precision	0.84
Recall	0.99
F1-Score	0.91
FNR	0.01

Table 5.15: MAIL specific parameterization classification results

in all the relevant metrics, but the results had to be taken with some care since the dataset was small and may have had a low variance in data.

Focusing on the general parameterization, the results are presented in Table 5.14

The model had a performance quite similar to the previous one with a specific parameterization, achieving optimal results in all of the metrics, still the dataset was quite low in terms of variance and data so it could have had a different outcome when using more data.

MAIL

The hyperparameter tuning for the Mail subset returned the following parameters, 9 layer, activation function *eLU*, bottleneck layer dimensions 7, optimizer Nadam and learning rate 0.001. The parameters, as we could see were always quite similar in all of the subsets.

The results of the model were presented in Table 5.15, and it was possible to conclude that the model had a good performance, capable of achieving a relatively high Precision as well as a low FNR.

Taking into consideration the general parameterization, the results the model presented were the ones in Table 5.16.

The values of each metric were below satisfactory which shows that the parame-

Metric	Score
Precision	0.08
Recall	0.02
F1-Score	0.03
FNR	0.98

Table 5.16: MAIL general parameterization classification results

terization used was not the best, making this model impossible to use.

5.3 Convolutional Autoencoder

The following section shows the performance results the models using the Convolutional Autoencoder algorithm. The results that are presented are correspondent to the entire dataset and to subsets of the dataset corresponding to the different types of known traffic, in this subsets it was also evaluated the performance of the model with a parameterization using the entire dataset and a parameterization using only the subset in question. It also presents the results of two different window sizes, 4 and 8.

5.3.1 Entire Dataset

5.3.1.1 CIC-IDS2017

Window Length 4 Starting with the hyperparameter tuning using a window length of value 4, the combination of parameters that showed a better were the following ones:

- Hidden Layers activation function: eLU
- Number of filters: 8
- Kernel: 4
- Kernel X: 2
- Learning Rate: 0.0005
- Optimizer: Nadam

The training of the model achieved a validation loss value of 0.070. It was possible to notice that the learning rate that was needed was still quite low and the activation function still needed to be *eLU*. The number of filters was high, and the dimensions of the kernel were 4 features per kernel window, a low value, this meant that the model, using lower kernel sizes, its reduction of dimensions was more gradual and more specific.

The classification results are presented in Table 5.17 as the confusion matrix in Table 5.18.

It was possible to conclude that the results the model achieved were not the best, far from it. The Precision was very low and the FNR was too high for it to be considered a good model to use.

Metric	Score
Precision	0.68
Recall	0.51
F1-Score	0.58
FNR	0.49

Table 5.17: Convolutional Autoencoder (window length 4) Classification results

	Actually Positive	Actually Negative
Predicted Positive	25%	11%
Predicted Negative	24%	40%

Table 5.18: Convolutional Autoencoder (window length 4) Confusion Matrix

Window Length 8 Starting with the hyperparameter tuning using a window length of value 8, the combination of parameters that showed a better performance were the following ones:

- Hidden Layers activation function: LeakyReLU
- Number of filters: 8
- Kernel: 4
- Kernel X: 4
- Learning Rate: 0.001
- Optimizer: Nadam

The training of the model achieved a validation loss value of 0.018. As we could see the hyperparameters used were all very similar especially the number of filters and the kernel sizes.

The values of each metric are shown in Table 5.19 and the confusion matrix in Table 5.20.

The results using a length of 8 were slightly worse than the ones achieved using 4 as window length, it showed a high FNR, above 60% although the Precision was the same.

In summary we could conclude that the best model to use was the one achieved with the window length of 4. It's architecture is presented in Figure 5.7, and as

Metric	Score
Precision	0.64
Recall	0.37
F1-Score	0.47
FNR	0.63

Table 5.19: Convolutional Autoencoder (window length 8) Classification results

	Actually Positive	Actually Negative
Predicted Positive	19%	11%
Predicted Negative	33%	37%

Table 5.20: Convolutional Autoencoder (window length 8) Confusion Matrix

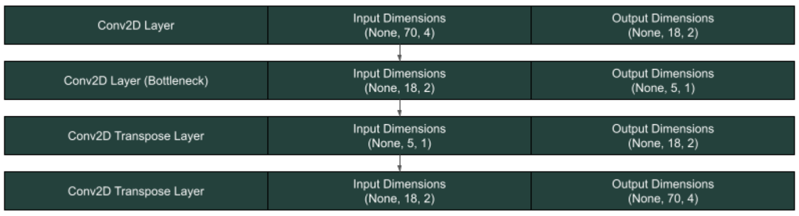
Conv2D Layer
Input Dimensions (None, 70, 4)
Output Dimensions (None, 18, 2)
Conv2D Layer (Bottleneck)
Input Dimensions (None, 18, 2)
Output Dimensions (None, 5, 1)
Conv2D Transpose Layer
Input Dimensions (None, 5, 1)
Output Dimensions (None, 18, 2)
Conv2D Transpose Layer
Input Dimensions (None, 18, 2)
Output Dimensions (None, 70, 4)

Figure 5.7: Convolutional Autoencoder Architecture

we could see the architecture was much smaller than the one presented in Figure 5.1 since the reduction of the dimensions was much faster.

5.3.2 Different traffic Datasets

5.3.2.1 CIC-IDS2017

The following section reveals the performance of the Convolutional Autoencoder for the different types of known traffic contained in the CIC-IDS2017 dataset. Taking the results of the different window lengths into consideration, we only used a window length of 4 and the original parameterization instead of trying a parameterization for each dataset.

HTTP and HTTPS

Focusing, first, on the HTTP and HTTPS subset, the Table 5.21 presents the performance of the model.

The results of the model were satisfactory, it had a Precision with a relatively high value, 84% although the False Negative Rate was too high at 38% a value that we already gotten way lower.

Metric	Score
Precision	0.84
Recall	0.62
F1-Score	0.71
FNR	0.38

Table 5.21: HTTP and HTTPS Convolutional Autoencoder Classification results

Metric	Score
Precision	0.89
Recall	0.90
F1-Score	0.90
FNR	0.10

Table 5.22: SSH Convolutional Autoencoder Classification results

Metric	Score
Precision	0.95
Recall	0.84
F1-Score	0.90
FNR	0.16

Table 5.23: FTP Convolutional Autoencoder Classification results

SSH

Looking at the SSH subset, we could observe, in the Table 5.22, the metrics results.

The following model, as the results in the table show, had a very good performance with a very high Precision and a relatively low FNR which made the model quite good.

FTP

The results the FTP set achieved are presented in the following Table 5.23.

The model, as we could notice, achieved very satisfactory results, a high Precision as well as a low FNR.

MAIL

Finally, focusing on the Mail subset, the metrics scores are presented in the Table 5.24.

Metric	Score
Precision	0.67
Recall	0.52
F1-Score	0.59
FNR	0.48

Table 5.24: MAIL Convolutional Autoencoder Classification results

Metric	Score
Precision	1.0
Recall	0.67
F1-Score	0.80
FNR	0.33

Table 5.25: Conventional Autoencoder (DoS) Classification results

The performance of the model was way worse than we were expecting, achieving a very low Precision score and an abruptly high False Negative Rate, this may be caused by the fact that the data is quite different from each type o mail protocol.

5.4 Discussion

This section presents a deeper analysis and discussion of the results each model achieved as well as a comparison between models in order for us to select the best model to integrate in our platform. We divided the analysis, first, by algorithms and then a comparison of the best results of each algorithm.

5.4.1 Conventional Autoencoder

First we compared the Conventional Autoencoder with the Random Forest, in order for us to show why we intended to use this algorithm over the one that was previously considered the best. In Table 5.3 it is shown the results of the metrics using the Random Forest trained without instances with the *DoS* attack type and tested in identifying only *DoS* attacks.

In Table 5.25 we show the results of the Conventional Autoencoder in detecting only the *DoS* attacks.

As we could see the performance of the Autoencoder in detecting attacks that were unknown to the model was much better than the one achieved by the Random Forest, which showed that this Unsupervised Learning algorithm was better for our solution.

The performance of the Conventional Autoencoder, overall, had an undesirable behaviour, in terms of precision it presented some good results, achieving high values, but the False Negative Rate was the metric that surprised us a little bit since it had quite high values in most situations.

We alredy displayed the values of the metrics, so we present the ROC curves of each model, as well as its AUC.

In the Figure 5.8 is displayed the ROC as well as the AUC value of the model used for the entire dataset.

We can conclude that the model has a relatively good performance achieving 0.087 AUC.

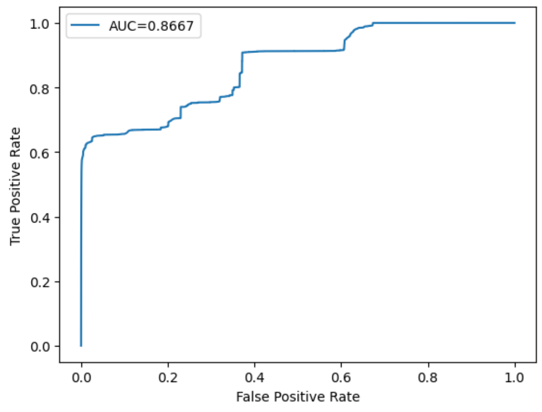

Figure 5.8: ROC curve and AUC for the General Conventional Autoencoder

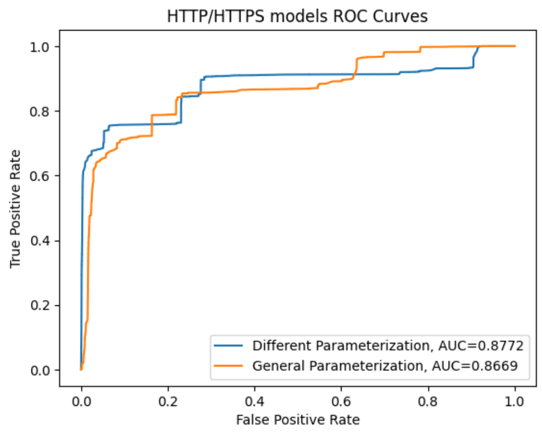

Figure 5.9: HTTP/HTTPS specific vs general parameterization

Now focusing on the models for each type of traffic, we presented the ROC curves and the correspondig AUC of each type of traffic.

First in the AUC of the HTTP and HTTPS subset, Figure 5.9, we could see that both had a high value and its ROC curves showed quite a good performance in all the possible predictive situations, the good preformance of the model could be corroborated by the metrics shown previously where the model achieved an F1-Score of 83% in both parameterizations which was a high value.

Looking at the models used in the SSH subset, Figure 5.10, it was possible to conclude that one model had a way better performance than the other. The model that used a parameterization specific to the dataset had a quite good AUC value, 0.85, while the one that used the parameterization used for the entire dataset lacked a lot in performance, achieving only 0.53 AUC and in the ROC curve the True Positives Rate stayed constant in almost all of the time, while the False Positive Rate increased. This behaviour made sense since the parameterization was an important aspect of the models implementation and a parameterization that was not specific to a dataset may have presented a bad performance.

Considering the models implemented for the FTP subset, as we could notice by looking at the Figure 5.11, both models had an almost perfect performance, achieving almost a perfect True Positive Rate right in the beginning and both had

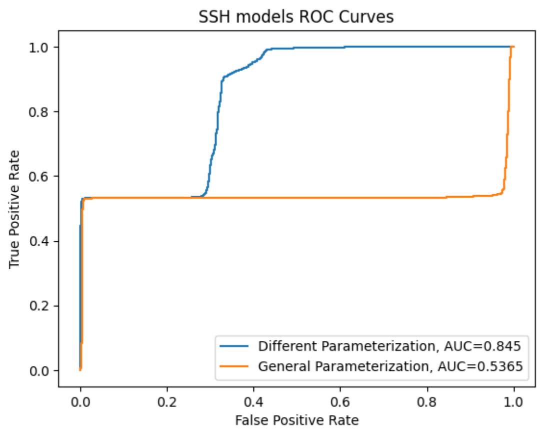

Figure 5.10: SSH specific vs general parameterization

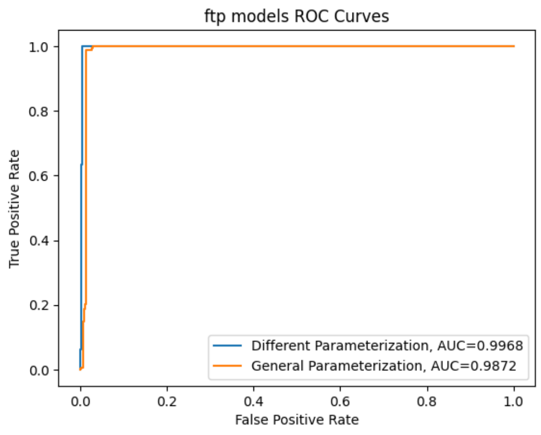

Figure 5.11: FTP specific vs general parameterization

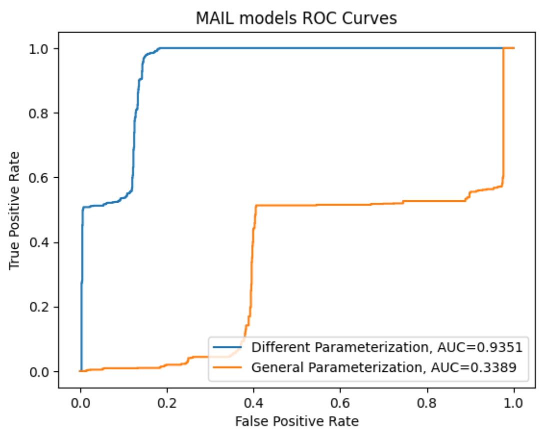

Figure 5.12: MAIL specific vs general parameterization

an AUC above 0.98. This showed that both models in this situation were fit to implement and that both parameterizations were able to achieve a good performance.

The final models, corresponding to the MAIL models had very different behaviours and performances. The model that used a parameterization made for the subset had a really good performance, achieving an AUC of 0.94, while the one that used the parameterization performed with the entire dataset had a bad performance, achieving an AUC of 0.34.

The most correct way to develop our study was by always performing an hyperparameter tuning whenever we had a different dataset, since it could modify a lot the performance of the model. The reason why we checked the performance of the model using a general parameterization was because of the final objective of our study. The model that would be implemented had to be scalable, since it would be used in multiple machines at the same time with different datasets and be able to classify instances of different types of traffic and different types of protocol, so the development of a hyperparameter tuning for each model depending on the dataset it used was not possible because of the scalability.

Although as we could see in the performance of all the models, there were times where the general parameterization had bad results and showed really bad performances, which may show that it may not have been the best approach to focus on, since a parameterization made for each dataset showed that the model would always have a somewhat good performance and achieve good results in the classification of instances.

As we could see, by comparing the results, these were not very different and not the best results as well. This may have been because the datasets were quite small so the training of the model was not the best and lacked some diversity and samples, as we were using Deep Learning, the models needed a lot of samples and way more variation in order for them to learn to reconstruct the instances.

Taking the ROC curves and the AUC of the each model, we could see that HTTP/HTTPS and FTP had similar results comparing to when we used the original parameterization and the parameterization for each type of data, although in the SSH and

Metric	Score
Precision	1.0
Recall	0.72
F1-Score	0.84
FNR	0.27

Table 5.26: Convolutional Autoencoder (DoS) Classification results

MAIL traffics, the AUC and ROC curve had significantly worse performances, when using the original parameterization, than when using the parameterization for each type of traffic.

This may be because the parameterization was quite different from the original one so the results tended to be a little bit different. Since we wanted to use these models in a real time situation, the parameterization for each model may have been difficult, so we wanted to use a more general one that could perform well in all of the models, but we could not assume that this was a good approach based on the results presented. In the HTTP/HTTPS the results were very similar because the parameters used were almost the same as the ones used in the original one.

5.4.2 Convolutional Autoencoder

First we made the comparison between the Convolutional Autoencoder with the Random Forest in terms of detecting unknown attacks to the model. In the Table 5.3 it was shown the results of the metrics using the Random Forest trained without instances with the *DoS* attack type and tested in identifying only *DoS* attacks.

In Table 5.26 we showed the results of the Conventional Autoencoder in detecting only the *DoS* attacks. The windows that had at least one instance as *DoS* were considered to be this type of attack.

As we could see the performance of the Convolutional Autoencoder in detecting attacks that were unknown to the model were much better than the one achieved by the Random Forest, which showed that this Unsupervised Learning algorithm was better for our solution.

The next analysis we intended to perform was assessing if the number of instances a window had to have to be considered an anomaly had any relevance in the performance of the model. To make this assessment we analyzed the performance of a Convolutional Autoencoder, using a window of 4, and considered a window anomalous when the number of anomalous instances it had was first 1, then 2, 3 and 4 and assessed the results of the performance metrics to conclude if it had any influence. The results are presented in Figure 5.13.

Looking at the graphs we could conclude that by considering a larger number of anomalous instances inside a window for it to be considered anomalous, the performance of the model decreased, the Precision decreased quite a lot while the

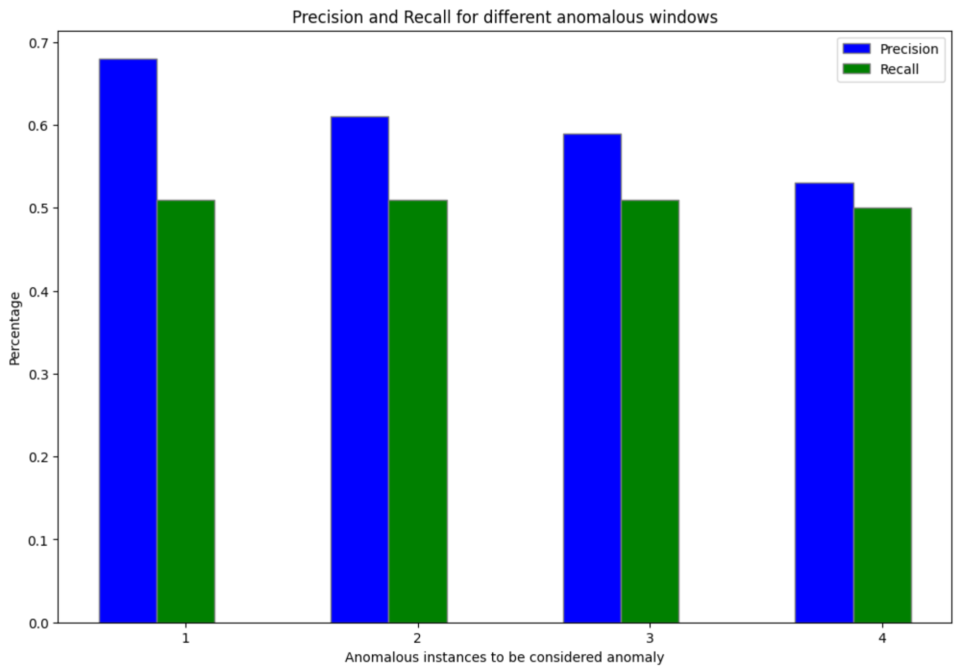

Figure 5.13: Convolutional Autoencoder's Precision and Recall for different anomalous windows

Recall stayed practically the same in all, this could be explained by the fact that the dataset would have less and less anomalous windows, and because of that the model would detect more instances as false that were, in fact, true but would compensate in the instances that were actually false. So it made more sense to consider that whenever a window had at least 1 anomalous instance it would be considered an anomaly.

The performance of the Convolutional Autoencoder, overall, was less satisfying than the one achieved by the Conventional Autoencoder. As specified earlier in the document, the idea behind testing this algorithm was the fact that the attacks normally appear as multiple anomalous flows in a row, so by classifying windows of flows, instead of individual flows, would be beneficial.

We could see, by analysing the ROC curves and the AUC, as well as the values of the Precision and FNR of the Convolutional Autoencoder of the entire dataset both window length 4 as well as 8 that the performance was very similar achieving AUC values of 0.61 and 0.59 respectively.

As we could conclude by looking at the graphs the performances of the models using Convolutional Autoencoders, Figure 5.14, this model was not close to being the one we had envisioned. We tested using 2 different window lengths, 4 and 8, even though the more usual value for the number of anomalous flows inside of a window was 1 and not 2 or more as we expected. This already had downgraded our expectations for the usage of the algorithm.

Comparing these two algorithms with the Conventional Autoencoder we could see that the two Convolutional presented a worse performance.

The results using a length of 4 were slightly better than the ones achieved us-

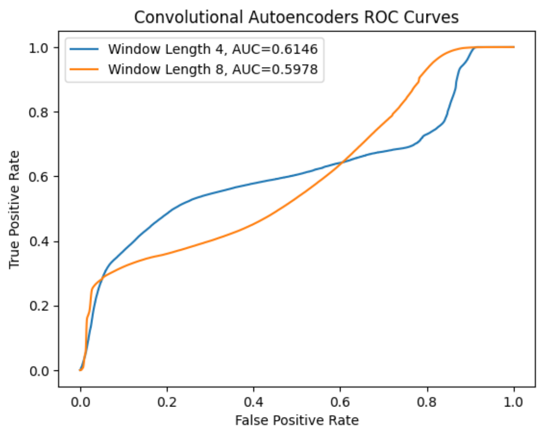

Figure 5.14: ROC curves and AUC for Convolutinal Autoencoders using the entire dataset

ing 8 as window length, although it showed a very high percentage of FN and a very low percentage in precision and the AUC was really low as well. This behaviour may be explained by the fact that since the attacks mainly produce only 1 anomalous flow instead of multiple ones the model was not capable of making this association and as it could reconstruct most of the flow inside of the window, the error would be low, as most of the times, the windows would have only 1 anomalous flow.

The bigger the window got the worst the performance would be as we can see by analyzing the performance of the Convolutional Autoencoder using a window length of 8.

Focusing on the performance for each type of traffic, Figure 5.15. As we could see by analyzing the figure, the AUC value of the SSH and FTP were relatively high and their curves showed that the model had a pretty good performance, being able to make the separation between True Positives and False Positives quite well, the HTTP and Mail were a bit lower achieving a value that was quite low which was proven by the values it showed in the metrics Precision and False Negative Rate.

The models shown in this picture were all implemented using the hyperparameter tuning performed in the entire dataset and were all using a window length of 4, so we could affirm that the performance was quite better than the one showed in the Conventional Autoencoder, where the AUC presented values of 0.87, 0.54, 0.99 and 0.34. This showed that overall the Convolutional Autoencoder for each type of traffic would have had a better performance if we used a more general parameterization, which was what we were aiming for, because of the scalability of our solution, as it was mentioned before.

5.4.3 Classification Time

Now we focused on another important aspect of our analysis, the time it takes each algorithm to classify each instance, presented in Table 5.27.

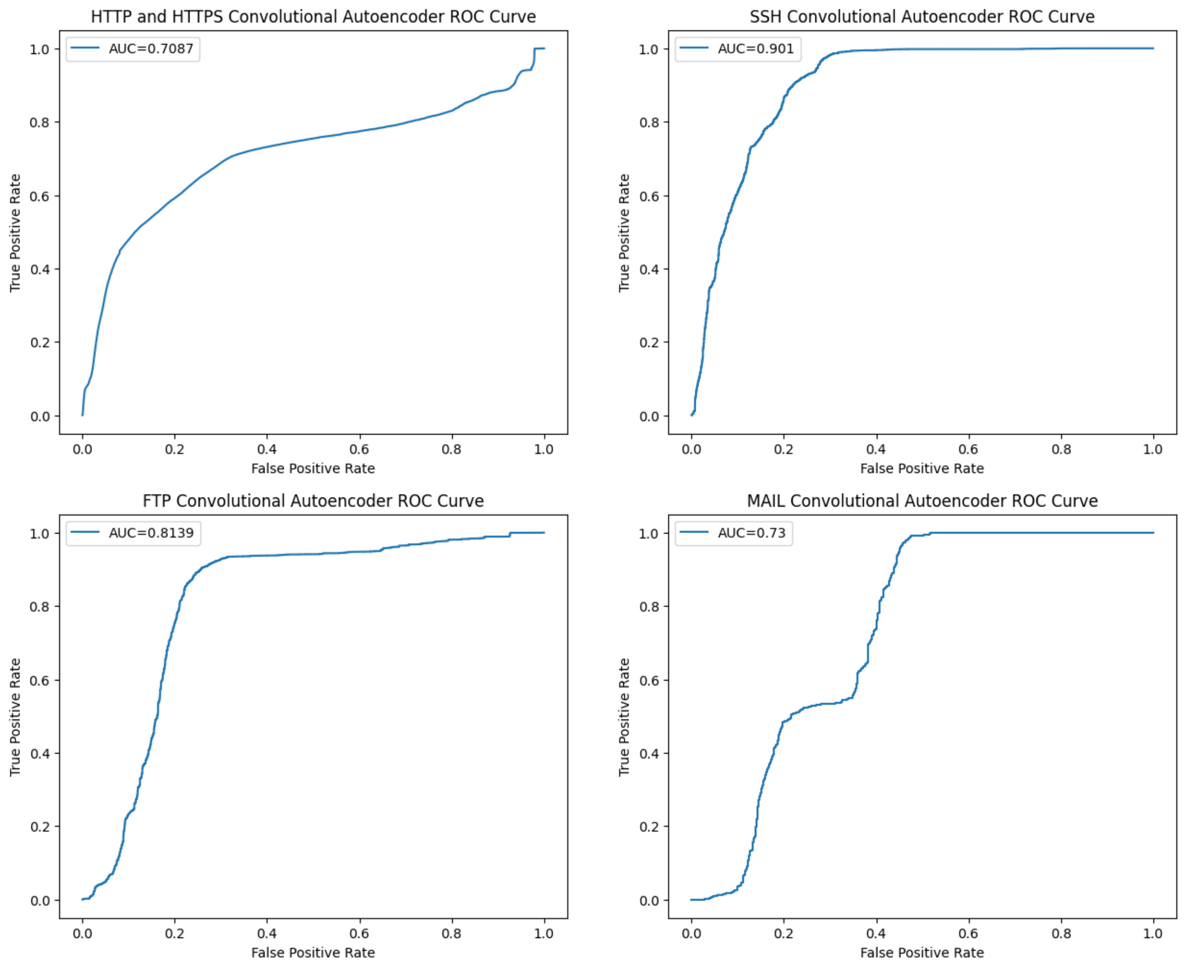

Figure 5.15: ROC curves and AUC Convolutinal Autoencoder different traffic

Algorithm	Time(milliseconds)
Random Forest	0.12
Conventional Autoencoder	1.02
Convolutional Autoencoder (window length 4)	1.08
Convolutional Autoencoder (window length 8)	1.09

Table 5.27: Time each algorithm takes to classify one instance

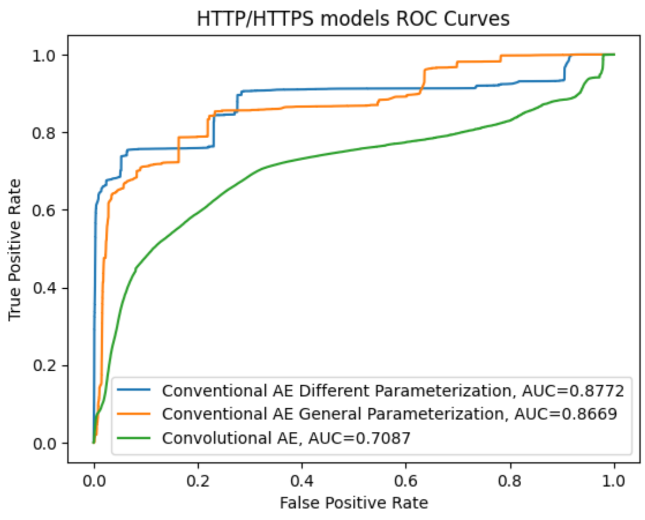

Figure 5.16: ROC curves and AUC of Conventional and Convolutional Autoencoders for HTTP and HTTPS

The machine where we ran our tests was a Virtual Machine running a Ubuntu OS, with 96 GB of RAM, a CPU with 12 cores, without dedicated GPU.

It was possible to notice that both kinds of Autoencoders took, almost, the same amount of time to classify each instance, the difference was almost none so we could conclude that this aspect would not help much in the decision of which model to choose. Comparing them to the Random Forest time we could notice that it took 10 times less the amount of time to classify each instance which was a lot of time and made quite a difference in a real time situation, so we agreed that in terms of time the Random Forest algorithm was best.

5.4.4 Comparison of Results

This section had the objective of making a more direct comparison between the performances of the models.

Both in Figure 5.16 as well as in Figure 5.18, it was possible to observe that both models that referred to the Conventional Autoencoder had a better AUC than the one that referred to the Convolutional. The AUC value in the first situation was quite high in all the situations while in the second one the value was almost 1 in the Conventional Autoencoders but a little bit lower for the Convolutional Autoencoder.

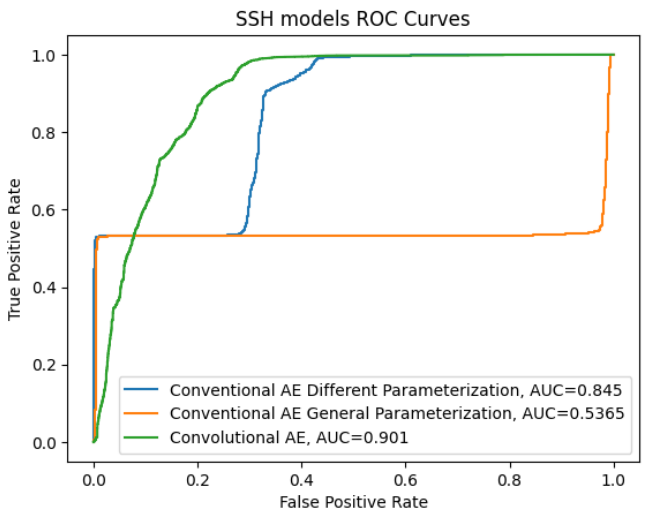

Figure 5.17: ROC curves and AUC of Conventional and Convolutional Autoencoders for SSH

In Figure 5.17, on the other hand, it was possible to observe that the model referring to the Convolutional Autoencoder had a quite higher AUC than the ones achieved by the Conventional AE models, with the AUC achieved by the Conventional Auteoncoder using the general parameterization being quite low.

Finally in Figure 5.19, the AUC of the Conventional Autoencoder with a parameterization made specifically for the dataset had a good AUC value, better than the one achieved by the other ones. The Convolutional Autoencoder on the other hand had a way better performance than the Conventional Autoencoder with the general parameterization.

We could conclude that in almost all situations the Conventional Autoencoder with a parameterization performed for the dataset in question had the best performance, achieving values of the order of 0.9 which was very positive, this showed that the Conventional Autoencoder was best in most situations. Although the Convolutional still had a better performance than the Conventional in the general parameterization we assumed that the cost of time was not worth for us to implement the Convolutional.

Finally we concluded that the Conventional Autoencoder and the Convolutional Autoencoder, both, presented a quite similar performance overall, in mutliple situations, sometimes one overtaking the other. With this in mind we had to turn to other aspects to compare the two. It was defined that the best approach was to train a model for each type of data, because the training would be much better and directed and the parameterization that would be used needed to be a more general one because of the scalability of the models as we mentioned before. With this in mind, we concluded that the Conventional Autoencoder was be the best algorithm to test in our platform as the training of the Conventional Autoencoder was much faster, each epoch of the Conventional Autoencoder took 30 seconds to train while in the Convolutional Autoencoder it took 150 seconds, as the models would be trained in time fixed intervals.

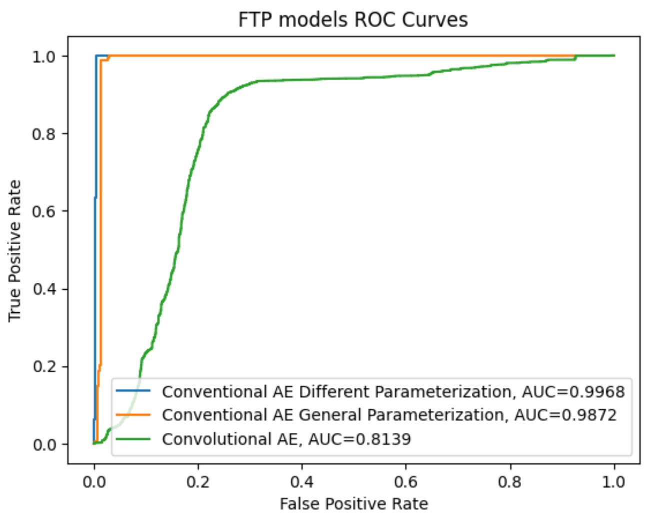

Figure 5.18: ROC curves and AUC of Conventional and Convolutional Autoencoders for FTP

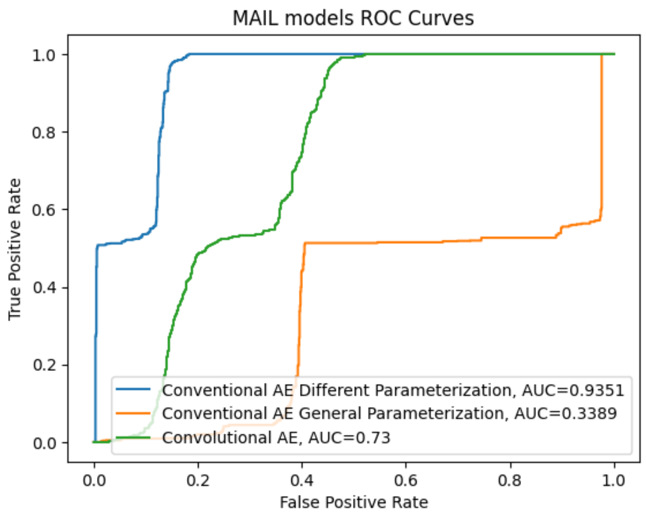

Figure 5.19: ROC curves and AUC of Conventional and Convolutional Autoencoders for MAIL

5.5 Summary

This chapter presented the results of each algorithm tested as well as a deep analysis on the results each model tested achieved in order for us to make an informed and factual based choice on which model was best to integrate in our final platform.

The analysis performed in this chapter allowed us to conclude that Supervised Learning is far from being the best technique since it cannot classify unknown anomalous instances correctly, which Unsupervised Learning is able to. The best window size for the Convolutional Autoencoder is the lower value, 4, with the justification being that if a window of 4 has only one anomalous instance it would have more influence in the reconstruction error than in a window of 8.

Both Conventional and Convolutional Autoencoders presented a quite similar performance overall, with the prediction time being almost exactly the same and the AUC as well as the performance metrics, being quite similar throughout all the tests performed. The one that seemed more suitable for our problem was the Conventional Autoencoder since it presented really good results with space to improve and since we would train the model in real time, we needed the training time to be as low as possible and the Convolutional Autoencoder took too much time to train.

Chapter 6

Integration

In the following chapter it is presented the algorithm implemented in the platform as well as how the implementation was performed.

The chapter is divided into two parts, the first one, 6.1, where we present how the algorithm will be trained depending on the type of data, as well as the performance results it achieved. The second and last section, 6.2, we present how the integration in the platform was achieved and how the framework will work.

6.1 Models Training

This section presents an overview of the models we tested for the platform and their integration in the platform that is being built.

We trained one model for each type of application that the platform was aware of, Message Broker, Mt Monitor, Mt Orchestrator and Tick Telegraf.

Because of time constraints, there was not enough time to perform hyperparameter tuning on the following datasets, therefore it was used the parameterization that had been being used throughout all the experiments, the best one achieved in section 5.2.1.1.

Since the datasets took more time than expected to be made available it was not possible to provide attacks for the datasets so we had to train the models and analyze the validation losses that they achieved and compare it to the ones achieved by the models trained with the CIC-IDS2017 dataset.

The only difference there was from the CIC-IDS2017 dataset to the custom datasets was that the dimensions of each layer was different since the number of constant features was larger in the custom datasets. We kept the changing factor the same (+15 and -15) but the dimensions were 50, 35, 20, 5.

Message Broker

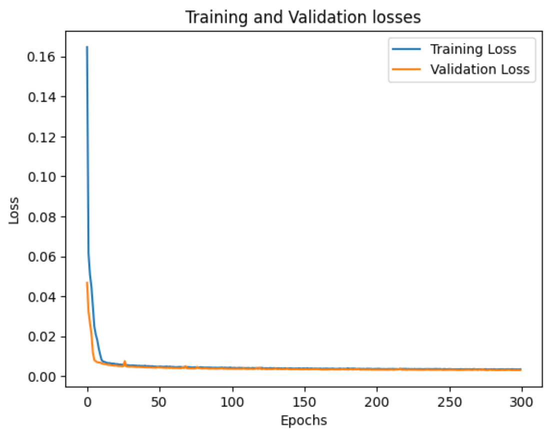

Figure 6.1: Training and Validation losses throughout the training of the Message Broker model

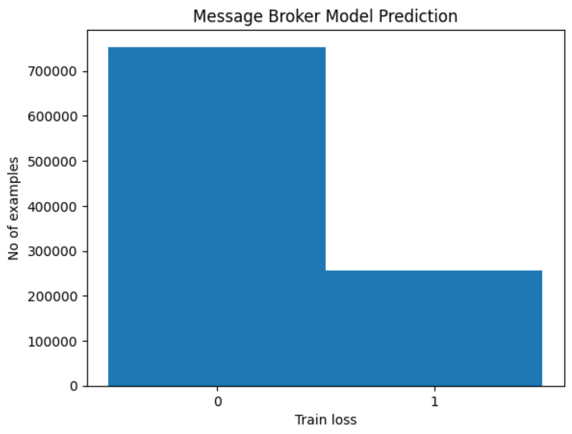

Figure 6.2: Classification Results with Message Broker model

In the Figure 6.1, we can see the training and validation loss of the model, we can notice that both losses stagnate in a very low value, almost 0, in the first 50 epochs.

Analyzing the Figure 6.2, we could see how many instances the model classified correctly. As it was already mentioned we were only using normal data, so the data used to make this prediction was all normal, corresponding to the label 0. Looking at the Figure we could conclude that the model was classifying most of the instances as normal, which may mean that it will have a somewhat good performance, although it classifies quite a lot of instances as anomalous.

Mt Monitor

In the Figure 6.3, it was possible to notice that the training loss takes a little more time to converge to a point, while the validation loss was quicker, in 25 epochs.

Focusing on the Figure 6.4, we could conclude that the model classified the in-

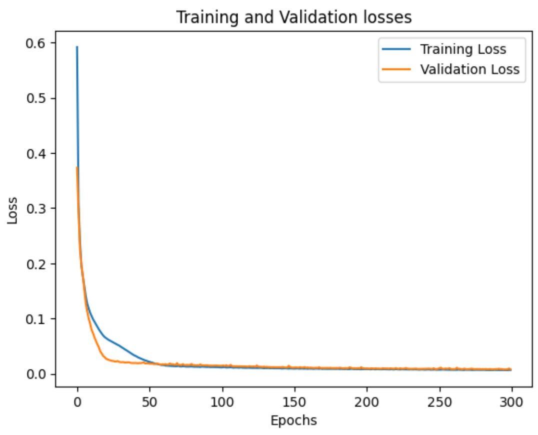

Figure 6.3: Training and Validation losses throughout the training of the Mt Monitor model

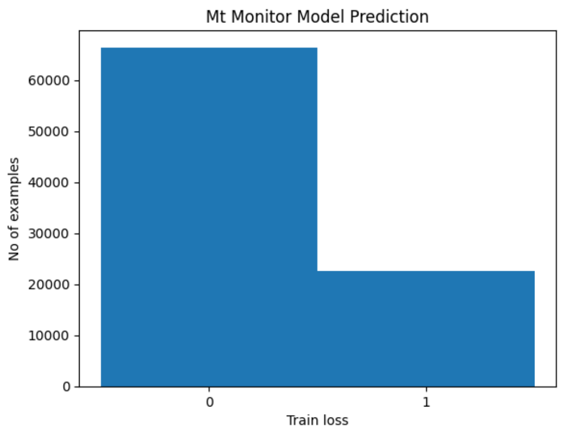

Figure 6.4: Classification Results with Mt Monitor model

stances quite well, classifying more than 60000 instances correctly, but still classified a large number of instances as attacks which was not ideal. It was possible to find a possible explanation for this fact and it was because of the low number of instances the dataset had, less than 100 thousand, so the training may not be the best.

Mt Orchestrator

In Figure 6.5, it was possible to notice that the training loss and validation loss took much more time than the rest of the models to converge to a value, almost 200 epochs, even though that loss was still low.

The Figure 6.6 showed that the model was able to correctly predict more than 12000 normal instances correctly but more than 4000 incorrectly, the ratio was still good , and the fact that the dataset was small, having only 16000 instances.

Tick Telegraf

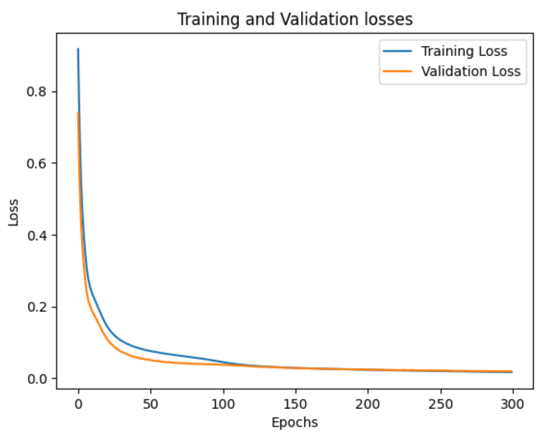

Figure 6.5: Training and Validation losses throughout the training of the Mt Orchestrator model

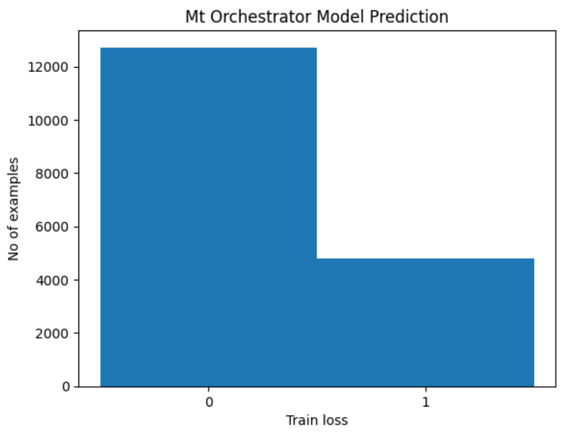

Figure 6.6: Classification Results with Mt Orchestrator model

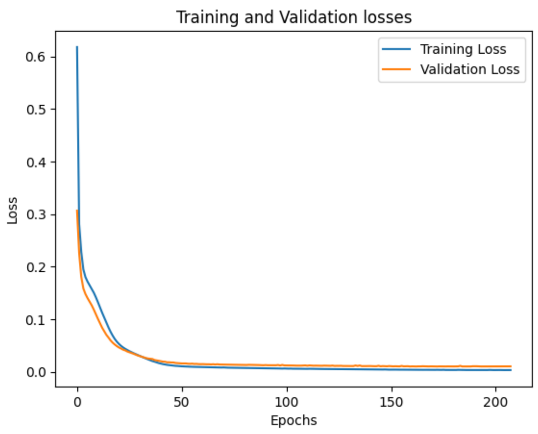

Figure 6.7: Training and Validation losses throughout the training of the Tick Telegraf model

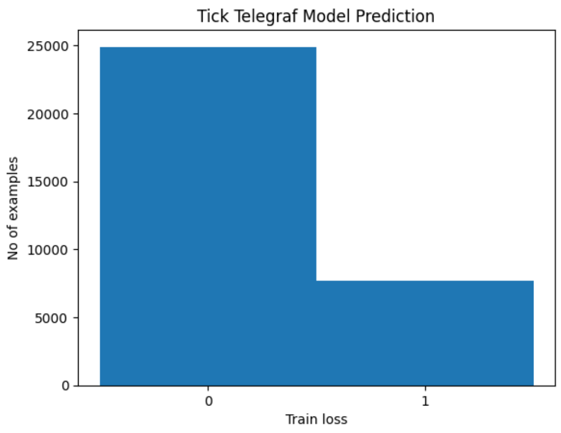

Figure 6.8: Classification Results with Tick Telegraf model

The Figure 6.7 presented the losses of the model throughout the training process, showing that the model reached the lowest value in the epoch 50, quite fast.

In Figure 6.8 we could see that the model was able to correctly predict almost 25000 normal instances correctly but more than 7000 incorrectly, one more time this can be explained by the fact that the dataset was quite small.

6.2 Integration

The normal flow of the platform will be the following one, first the information is collected by a Collector and stores that information locally, this process is executed in intervals of 2 minutes. When the information is collected it is performed a data preprocessing same as it was done in the training and evaluation of the models, remove outliers and infinites and standardization of the data.

Following this procedure, each instance of the data is classified the same way we classify the instances previously, we alreafy have a predefined threshold and as we calculate the reconstruction error of each instance, if it is below the threshold it is considered normal, if it is above a certain threshold it is considered an anomaly.

The platform is divided into multiple sidecars that each one will perform its tasks such as collecting traffic and performing the classification of the traffic flows.

The platform is centered in the idea of Federated Learning, described earlier and in the paper [15]. The idea behind it is that the platform is divided into microservices each one collects its own data and stores it internally and have an assigned model which performs the classification of the data. The microservices are divided by application, for example, the microservices that are using musical data from a certain application only communicate with the microservices that are using the same data from the same application.

The instances that are considered normal of each microservice are saved in a new "dataset" so that they can be used for training and keep improving the performance of our model. Every 8 hours it is performed a new training round using the dataset collected by the microservices. After the training is completed the new model is passed on to each microservice so that it can perform the classification now using a supposedly better and improved model than the one being used before.

In terms of what is done with the instances that are considered anomalies, the plan idea is, each instance has an identifier IP, whenever an instance is considered anomalous, the IP is added to a *gray list*. This graylist will store all the IPs of the anomalous instances and will be emptied every 5 minutes. If in those 5 minutes there is an IP which has 10 or more entries inside the *gray list* it will be considered an attempt of attack to the network and that IP will be blocked.

6.3 Summary

This chapter presented the training of the models that will be used in the platform that is being built, presenting how the training was done as well as how the model is behaving initially, trying to classify only normal data. There were four different models created one for each type of data and was presented how many normal instances it classified correctly.

The analysis of this models was difficult to perform since we did not have anomalous instances to classify so we focused on the normal instances and how well they were classified. Overall the models could detect more than 70% of the normal instances, not the best ratio but a good one to start since the training datasets were considerably small.

It also presented an overview of how the platform will work, since the extraction of data to the classification of the instances and its handling.

Chapter 7

Conclusion and Future work

Network Anomaly Detection based on AI techniques has been getting more attention from the community in recent years. Namely, Deep Learning and Unsupervised Learning approaches have been achieving promising results and overall improving the accuracy of detecting anomalies while reducing the computation and time complexity of it. Nevertheless, there are still some open challenges and room for improvement.

This work started with a review of the projects this work integrates. Then, a literature review of the role of AI in network security and a review of the state of the art AI and ML for Network Anomaly Detection, including a discussion of the candidate algorithms to further explore in this research work and the rationale that makes them more suitable for our problem. Later, we detailed the proposed approaches, steps and methods we planned on implementing and what kind of tests we were going to perform as well as how we would evaluate our approaches.

We leveraged the information about AI models for this purpose, concluding that, Deep Learning and Unsupervised Learning algorithms were the ones that presented better performances overall and were able to detect unknown attacks. On the other hand, it was also observed that they require more investigation when applicable to the problem of NAD.

The finality of our study was to get a solid model, that could be integrated in the platform being developed by OneSource. In terms of preprocessing it was quite simple.

Comparing Supervised with Unsupervised learning it was possible to conclude that Unsupervised Learning is much more useful in our study since it could detect unknown instances with much more precision than the approaches using Supervised Learning although this kinds of approaches usually performed really well in instances that they already known.

Focusing on the results we could conclude that the Conventional Autoencoder and the Convolutional Autoencoder had similar results, although the first one had a slight leverage over the second one. In the Convolutional Autoencoder, the bigger the size of the window the worse the performance of the model.

As both models presented a very similar performance, the selection of the approach to integrate on the platform came down to certain details, such as the training time, since the framework will have recurrent trainings, in intervals of 8 hours, the training of the models is quite important so we needed a training time that was not too long, and the training time of the Conventional Autoencoder was 5 times lower.

The objectives set for this study were all achieved. The first one, analyse the existing studies was done in the State of the Art, and selected the Autoencoders as the best algorithms to explore and assess its performance. Next one, integrate at least one of the approaches was successful, as we implemented a Conventional Autoencoder into the framework and it is currently working and is able to detect anomalies in real time. The last one, evaluating the performance of the candidate approaches and comparing it to the ones presented in the State of the Art was semi achieved as we evaluated the performance of the Conventional and Convolutional Autoencoders and showed that the Unsupervised Learning approach was better than the Supervised approach in detecting attacks unknown to the network, an important aspect in our work, but a direct comparison with more algorithms presented in the State of the Art was not concluded, but the main objectives of our work were achieved successfully.

Finally for future work, it would be beneficial for the framework to be able to increase its scalability with the objective of performing an hyperparameter tuning for each model that is created, because as we concluded in our study, the Conventional Autoencoder performs very good in detecting anomalous instances when it has a specific parameterization. If this could be achieved it would increase the precision of the model and it would be more reliable.

References

[1] "Onesource." https://onesource.pt/. Accessed: 2022-11-06.

[2] "5g-epicentre." https://www.5gepicentre.eu/. Accessed: 2022-11-06.

[3] "Charity." https://www.charity-project.eu/en. Accessed: 2022-11-06.

[4] K. C. Apostolakis, G. Margetis, C. Stephanidis, J.-M. Duquerrois, L. Drouglazet, A. Lallet, S. Delmas, L. Cordeiro, A. Gomes, M. Amor, A. D. Zayas, C. Verikoukis, K. Ramantas, and I. Markopoulos, "Cloud-native 5g infrastructure and network applications (netapps) for public protection and disaster relief: The 5g-epicentre project," in *2021 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit)*, pp. 235–240, 2021.

[5] A. Makris, A. Boudi, M. Coppola, L. Cordeiro, M. Corsini, P. Dazzi, F. D. Andilla, Y. González Rozas, M. Kamarianakis, M. Pateraki, T. L. Pham, A. Protopsaltis, A. Raman, A. Romussi, L. Rosa, E. Spatafora, T. Taleb, T. Theodoropoulos, K. Tserpes, E. Zschau, and U. Herzog, "Cloud for holography and augmented reality," in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, pp. 118–126, 2021.

[6] "Three key artificial intelligence applications for cybersecurity by chuck brooks and dr. frederic lemieux." https://www.forbes.com/sites/chuckbrooks/2021/09/24/three-key-artificial-intelligence-applications-for-cybersecurity/?sh=114ffc657b7e. Accessed: 2022-12-21.

[7] "Kubernetes." https://kubernetes.io/docs/concepts/overview/. Accessed: 2022-12-21.

[8] "Istio." https://istio.io/latest/about/service-mesh/. Accessed: 2022-12-21.

[9] "Opa." https://www.openpolicyagent.org/docs/latest/. Accessed: 2022-12-21.

[10] M. V. Pawar and J. Anuradha, "Network security and types of attacks in network," *Procedia Computer Science*, vol. 48, pp. 503–506, 2015.

[11] G. Junior, J. Rodrigues, L. Carvalho, J. Al-Muhtadi, and M. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, 03 2019.

[12] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[13] R. Fekolkin, "Intrusion detection and prevention systems: Overview of snort and suricata," 01 2015.

[14] R. Al-amri, R. K. Murugesan, M. Man, A. F. Abdulateef, M. A. Al-Sharafi, and A. A. Alkahtani, "A review of machine learning and deep learning techniques for anomaly detection in iot data," *Applied Sciences*, vol. 11, no. 12, 2021.

[15] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[16] S. Wang, J. F. Balarezo, S. Kandeepan, A. Al-Hourani, K. G. Chavez, and B. Rubinstein, "Machine learning in network anomaly detection: A survey," *IEEE Access*, vol. 9, pp. 152379–152396, 2021.

[17] D. Rani and N. C. Kaushal, "Supervised machine learning based network intrusion detection system for internet of things," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–7, 2020.

[18] I. Alrashdi, A. Alqazzaz, E. Aloufi, R. Alharthi, M. Zohdy, and H. Ming, "Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0305–0310, 2019.

[19] R. Primartha and B. A. Tama, "Anomaly detection using random forest: A performance revisited," in *2017 International conference on data and software engineering (ICoDSE)*, pp. 1–6, IEEE, 2017.

[20] Z. Ahmad, A. Shahid Khan, K. Nisar, I. Haider, R. Hassan, M. R. Haque, S. Tarmizi, and J. J. P. C. Rodrigues, "Anomaly detection using deep neural network for iot architecture," *Applied Sciences*, vol. 11, no. 15, 2021.

[21] Y. Jia, M. Wang, and Y. Wang, "Network intrusion detection algorithm based on deep neural network," *IET Information Security*, vol. 13, no. 1, pp. 48–53, 2019.

[22] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *Ieee Access*, vol. 7, pp. 41525–41550, 2019.

[23] I. Ullah and Q. H. Mahmoud, "Design and development of rnn anomaly detection model for iot networks," *IEEE Access*, vol. 10, pp. 62722–62750, 2022.

[24] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using

rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[25] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access*, vol. 5, pp. 21954–21961, 2017.

[26] F. T. Liu, K. Ting, and Z.-H. Zhou, "Isolation forest," pp. 413 – 422, 01 2009.

[27] A. Vikram and Mohana, "Anomaly detection in network traffic using unsupervised machine learning approach," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pp. 476–479, 2020.

[28] X. Tao, Y. Peng, F. Zhao, P. Zhao, and Y. Wang, "A parallel algorithm for network traffic anomaly detection based on isolation forest," *International Journal of Distributed Sensor Networks*, vol. 14, no. 11, p. 1550147718814471, 2018.

[29] R. A. Ariyaluran Habeeb, F. Nasaruddin, A. Gani, M. A. Amanullah, I. Hashem, E. Ahmed, and M. Imran, "Clustering-based real-time anomaly detection—a breakthrough in big data technologies," *Transactions on Emerging Telecommunications Technologies*, vol. 33, p. e3647, 06 2019.

[30] G. Pu, L. Wang, J. Shen, and F. Dong, "A hybrid unsupervised clustering-based anomaly detection method," *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 146–153, 2021.

[31] E. Bigdeli, M. Mohammadi, B. Raahemi, and S. Matwin, "Incremental anomaly detection using two-layer cluster-based structure," *Information Sciences*, vol. 429, pp. 315–331, 2018.

[32] D. He, S. Chan, X. Ni, and M. Guizani, "Software-defined-networking-enabled traffic anomaly detection and mitigation," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1890–1898, 2017.

[33] I. Apostol, M. Preda, C. Nila, and I. Bica, "Iot botnet anomaly detection using unsupervised deep learning," *Electronics*, vol. 10, no. 16, 2021.

[34] R. Bhatia, S. Benno, J. Esteban, T. Lakshman, and J. Grogan, "Unsupervised machine learning for network-centric anomaly detection in iot," pp. 42–48, 12 2019.

[35] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, "Autoencoder-based network anomaly detection," in *2018 Wireless Telecommunications Symposium (WTS)*, pp. 1–5, 2018.

[36] T. Truong-Huu, N. Dheenadhayalan, P. Kundu, V. Ramnath, J. Liao, S. Teo, and S. P. Kadiyala, "An empirical study on unsupervised network anomaly detection using generative adversarial networks," 10 2020.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Advances in Neural Information Processing Systems*, vol. 3, 06 2014.

[38] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *International conference on information processing in medical imaging*, pp. 146–157, Springer, 2017.

[39] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially learned inference," *arXiv preprint arXiv:1606.00704*, 2016.

[40] M. Usama, M. Asim, S. Latif, J. Qadir, and Ala-Al-Fuqaha, "Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems," in *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*, pp. 78–83, 2019.

[41] A. Dawoud, S. Shahristani, and C. Raun, "Deep learning for network anomalies detection," in *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)*, pp. 149–153, 2018.

[42] F. Carrera, V. Dentamaro, S. Galantucci, A. Iannacone, D. Impedovo, and G. Pirlo, "Combining unsupervised approaches for near real-time network traffic anomaly detection," *Applied Sciences*, vol. 12, no. 3, 2022.

[43] W. Seo and W. Pak, "Real-time network intrusion prevention system based on hybrid machine learning," *IEEE Access*, vol. PP, pp. 1–1, 03 2021.

[44] H. G. Gülmez and P. Angın, "A study on the efficacy of deep reinforcement learning for intrusion detection," 2021.

[45] K. Sethi, S. Edupuganti, R. Kumar, P. Bera, and Y. Madhav, "A context-aware robust intrusion detection system: a reinforcement learning-based approach," *International Journal of Information Security*, vol. 19, 12 2020.

[46] Z. Wang, Y. Wang, H. Xu, and Y. Wang, "Effective anomaly detection based on reinforcement learning in network traffic data," in *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 299–306, 2021.

[47] L. Hakim, R. Fatma, and Novriandi, "Influence analysis of feature selection to network intrusion detection system performance using nsl-kdd dataset," in *2019 International Conference on Computer Science, Information Technology, and Electrical Engineering (ICOMITEE)*, pp. 217–220, 2019.

[48] S. Aljawarneh, M. Aldwairi, and M. B. Yassein, "Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model," *Journal of Computational Science*, vol. 25, pp. 152–160, 2018.

[49] M. A. Siddiqi and W. Pak, "Optimizing filter-based feature selection method flow for intrusion detection system," *Electronics*, vol. 9, no. 12, 2020.

[50] M. A. Umar and C. Zhanfang, "Effects of feature selection and normalization on network intrusion detection," 2020.

[51] M. Sarhan, S. Layeghy, N. Moustafa, M. Gallagher, and M. Portmann, "Feature extraction for machine learning-based intrusion detection in iot networks," *Digital Communications and Networks*, 2022.

[52] "Chan's jupyter." https://goodboychan.github.io/python/datacamp/machine_learning/2020/07/09/01-Feature-extraction.html. Accessed: 2022-11-22.

[53] J. Henriques, L. Rosa, A. Gomes, L. Cordeiro, K. C. Apostolakis, G. Margetis, C. Stephanidis, M.-A. R. Anastasi, C. Skoufis, A. Siokis, and K. Ramantas, "The 5g-epicentre approach for decreasing attack surface on cross-testbeds cloud-native 5g scenarios," in *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pp. 7–12, 2021.

[54] "From autoencoder to beta-vae." https://lilianweng.github.io/posts/2018-08-12-vae/. Accessed: 2022-12-27.

[55] D. Snover, C. W. Johnson, M. J. Bianco, and P. Gerstoft, "Deep clustering to identify sources of urban seismic noise in long beach, california," *Seismological Society of America*, vol. 92, no. 2A, pp. 1011–1022, 2021.

[56] "Cic-ids2017 dataset." https://www.unb.ca/cic/datasets/ids-2017.html. Accessed: 2022-12-21.

[57] R. Panigrahi and S. Borah, "A detailed analysis of cicids2017 dataset for designing intrusion detection systems," *International Journal of Engineering Technology*, vol. 7, pp. 479–482, 01 2018.

[58] P. R. B. N. Tomás, "Using machine learning(ml) for anomaly detection over traffci present in service mesh architectures," 2022.