VISTULA UNIVERSITY

The Faculty of Computer Engineering, Graphic Design and Architecture

Program of study Computer Science

Devon Cardoso Elias

Student's number 64238

Implementation of a neural network classifier for an intrusion detection system

The engineer's thesis

written under the supervision of

dr. Mariusz Jakubowski

Warsaw, 2025

Abstract

This project is geared at exploring the development and implementation of an anomaly-based Intrusion Detection System (IDS) specifically focused on neural networks, to classify IoT network traffic. This project attempts to accomplish this by using the IoT-23 dataset, which includes both benign and malicious traffic captures. This research leverages a deep learning model to detect various types of network intrusions. We will attempt to achieve the most suitable outcome for the project.

The end results should show the model's effectiveness in distinguishing between benign and malicious traffic, demonstrating its potential as a reliable tool for enhancing IoT security.

Acknowledgement

I extend my gratitude to my family and friends for their support during this journey. To my advisor, Mariusz Jakubowski, for his guidance throughout my project process.

Index

1.Introduction

In our day and age of ever growing and expanding technological advancements, Internet of things (IoT) has rapidly become a staple in our day-to-day operations, found in majority of our tools such as portable devices or in in industrial ecosystems, IoT devices have cemented their place in our time. But with these devices also comes their own set of adversaries and drawbacks; for the purpose of this project, we will be focusing on these device's security and how they function, mainly how they can become targets for various types of cyberattacks, nefarious social engineering and most of all a growing danger to an unsuspecting individual. Additionally, we will dive into which tools and systems are being developed and implemented to fend off these malicious attacks.

IoT devices, often with limited computational resources and a primary focus on functionality, are particularly susceptible to vulnerabilities in the form of human errors or disregards such as weak passwords, insecure implemented network protocols, and lack of update mechanisms. These factors, combined with the growing sophistication of cyberattacks, render IoT ecosystems highly vulnerable and perfect candidates for our project.

To render these devices safer and less susceptible to cyberattacks, researchers have developed an array of tools to detect, alert, and even prevent said attacks. These efforts are in response to evolving threats to IoT devices, as highlighted by the World Economic Forum (2021) in their discussion on IoT security challenges [1].

Intrusion detection systems (IDS) are one such tool, serving as an important part of IoT device's security eco-system. Gaining its emergence in the 1970s and early 1980s, the first IDS had the primary focus of detecting unauthorized access or malicious traffic that could not be detected by the already implemented firewalls. Relying on predefined malicious signatures across networks and hosts, the IDS would actively monitor for anomalies, serving as the foundation of this tool.

In the 1990s, IDS matured into a commercial technology, and by the early 2000s, systems were implemented into larger security platforms, improving their effectiveness in protecting against various attack vectors. While effective, traditional IDS systems often require constant updates to their signature databases, especially in ever changing environments like IoT networks. This has pushed ongoing research into more adaptive detection systems and over time, IDS systems have evolved from above simple signature-based detection to more sophisticated systems combining both signature and anomaly detection. Thus, this research focuses on developing a comprehensive anomaly-based IDS using machine learning techniques to enhance IoT security.

Machine learning-based IDS offers a promising solution by using algorithms to learn normal network behaviors and identifying anomalies. These systems can potentially adapt to evolving threats, handle large volumes of data, and detect complex attack patterns. By analyzing network

traffic data, machine learning models can build profiles of normal activity and flag deviations that may indicate malicious behavior.

By utilizing the *IoT-23 dataset*, which features both benign and malicious network traffic, we aim to train a deep learning model capable of accurately classifying network traffic as either normal or malicious. Techniques such as Synthetic Minority Oversampling Technique (SMOTE) were used to address class imbalance by oversampling the minority class [2]. Feature engineering was performed to encode categorical variables, and model performance was enhanced by scaling features with RobustScaler. A Stratified K-Fold cross-validation strategy was employed to ensure robust evaluation, as detailed by Scikit-learn's documentation [3]. Additionally, the model was optimized using class weighting and early stopping to avoid overfitting [4].

The project questions that this paper will attempt to answer are:

How can the proposed model be optimized for real-time detection of anomalies in high-speed network environments?

What are the trade-offs between model accuracy and computational efficiency for practical deployment?

The results obtained in this project will attempt to answer the questions above and open the possibility for further studies.

2. Background and Related works

2.1 Neural networks

In the 1940s, McCulloch and Pitts proposed a mathematical model of neural computation. This model can be considered the foundation of neural networks as we know them today. This model was very important in understanding how artificial systems can behave like neurons. In 1958, Frank Rosenblatt developed the perceptron, the first computer binary classifier, which grabbed great attention. The Perceptrons book by Marvin Minsky and Seymour Papert published in 1969 showed that the neural network could not resolve non-linear problems. Due to this limitation, the initial enthusiasm for neural networks got a hit which resulted in an "AI Winter".[10]

Development of backpropagation algorithm in the 1980s renewed interests, multi-layer networks could be trained properly using backpropagation algorithm. David Rumelhart, Geoffrey Hinton and Ronald Williams are the researchers who popularized this technique. Due to this development, the capability of neural networks to learn more complex representations was possible. After this, artificial intelligence (AI) witnessed a surge.[11]

A major development in the 1990s and 2000s was convolutional neural networks (CNN) introduced by Yann LeCun. It was used for handwritten digit recognition (LeNet). Another important development was the recurrent neural network (RNN) introduced by Sepp Hochreiter and Jürgen Schmidhuber. This network was useful for sequential data processing. By the 2010s, deep learning architectures with many layers (e.g., AlexNet) were achieving state-of-the-art results in image classification. This led the way for further deep learning innovations, such as transformers, in natural language processing. [11]

2.1.1 Types of Neural Networks

1. **Feedforward Neural Networks (FNNs):** FNNs are the simplest neural network architecture, where information flows in a single direction from input to output. These networks are foundational to many early applications of AI, including basic classification tasks[11]
2. **Convolutional Neural Networks (CNNs):** Designed specifically for spatial data like images, CNNs use convolutional layers to extract features such as edges and textures. Applications include facial recognition and medical imaging. The use of pooling layers and filters in CNNs ensures efficient feature extraction and scalability[11]

3. **Recurrent Neural Networks (RNNs):** RNNs are ideal for sequential data, utilizing feedback connections to process temporal information. Variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) address challenges like vanishing gradients, making them essential in applications such as language translation and time-series prediction.
4. **Transformer Networks:** Introduced in 2017 with the paper *Attention Is All You Need*, transformers revolutionized NLP tasks by employing self-attention mechanisms. Models

like BERT and GPT have become foundational in tasks such as text generation and question answering.[12]

5. **Generative Adversarial Networks (GANs):** GANs consist of a generator that creates data samples and a discriminator that evaluates their authenticity. Applications include realistic image generation, video synthesis, and even audio creation

2.1.2 Applications

Neural networks have revolutionized diverse fields:

- **Computer Vision:** CNNs enable applications like facial recognition, autonomous vehicles, and diagnostic imaging in healthcare
- **Natural Language Processing (NLP):** Transformers power tools such as chatbots, machine translation systems, and sentiment analysis platforms.
- **Healthcare:** Neural networks assist in disease detection, drug discovery, and personalized medicine
- **Finance:** Fraud detection, credit scoring, and algorithmic trading rely heavily on the pattern recognition capabilities of neural networks

2.2 Anomaly-Based Intrusion Detection Systems (IDS)

An anomaly-based Intrusion Detection System (AIDS) is an ever-evolving component of modern cybersecurity eco-system. Unlike signature-based IDS, which function on known patterns of malicious activity, anomaly-based IDS monitors network traffic and identifies deviations from any 'normal' behavior. These deviations, or anomalies, often indicate the presence of a security threat such as malware or unauthorized access. The recent growth of Internet of Things (IoT) devices, and their inherent security vulnerabilities, has widened the need for adaptive IDS capable of actively monitoring these network environments.

The idea of anomaly detection dates to the early 80s when researchers started exploring the possibility of detecting malicious behavior through statistical and behavioral analysis. The idea was first proposed by Dorothy E. Denning in 1986. Her work laid the foundation for modern anomaly detection systems by promoting the use of profiles and thresholds for detecting intrusions.[5]

Intrusion Detection Systems (IDS) made considerable advances in the 1990s when the use of a lot of data together with its dynamic nature became the reason behind using these data. Old ways which only focused on signatures were getting slower as new malware detected by antivirus was growing. Consequently, researchers started to incorporate machine learning approaches and data mining into IDS for better detection and decreased rate of false positives. Through these techniques, the IDS systems were able to catch new threats and modify to the ever-changing network traffic.

The systems like decision trees, support vector machines (SVMs), and neural networks are used in machine learning to develop intrusion detection systems (IDS) through training them on past

experiences so they can learn to detect attackers. These models were able to process large datasets and detect minute variations that indicate unwanted activity. We also used data mining on the traffic data, especially to pull out large amounts of data to see if we can identify some patterns and correlate events in different parts of the network. This helped in designing a fairly competent anomaly detection system that goes beyond simple rules, and has the potential to learn from and adapt to complex attack scenarios [5].

IDS systems utilize several key features to differentiate between normal and abnormal traffic. Packet size, for example, is an important metric as anomaly in packet sizes can indicate attempted intrusion or data loss. You can also monitor the type of protocol used in network communication. Rare protocols or breaches of protocol usage can be a sign of an attack. When there are sudden spikes in traffic volume, it's probably a Denial of Service (DDos) attack. Likewise, abnormal connection patterns, such as unusually long session durations or frequent reconnections, can indicate suspicious behavior like port scanning or unauthorized access attempts. These features help the IDS conclude and flag malicious activities [6].

When an anomaly is detected, an alarm or alert is generated to notify the analyst of a potential threat. The response is often determined by the level of alarm severity. Newer IDSs come with automated responses like blocking the suspected IP address and shutting down the harmful session. These systems are integrated with machine learning and statistical analysis which allows a faster and quicker response to growing threats one of the most important tools in today's cybercrime. [5].

2.3 Risks and Benefits of Anomaly-Based Intrusion Detection Systems (IDS)

2.3.1 Benefits of Anomaly-Based IDS:

Anomaly-Based IDS Advantages:

In the context of dynamic and changing network environments such as those found in Internet of Things ecosystems anomaly-based intrusion detection systems or IDS provide a number of noteworthy advantages.

While signature-based intrusion detection systems (IDS) are limited to detecting known attack patterns, anomaly-based IDS keeps an eye on network activity and alerts users to any deviations from standard operating procedures [13] . Being able to identify new attack vectors and zero-day exploits that traditional security systems haven't yet been able to catalog is important. Another major advantage of anomaly-based IDS is its adaptability, with time these systems are able to adapt to changes in network traffic patterns by learning and changing [14] . Since IoT devices are frequently used in a variety of quickly changing environments this flexibility guarantees that the IDS will continue to function as the network changes. Additionally, comprehensive coverage for various kinds of network traffic is another benefit of anomaly-based intrusion detection systems. These systems are able to identify a variety of malicious activities including those that apply non-

traditional means of circumventing established security protocols by concentrating on departures from normal behavior[15] . Anomaly-based intrusion detection systems (IDS) are especially useful in settings like Internet of Things networks where the threat landscape is complex and varied because of their wide detection capability.

2.3.2 Risks of Anomaly-Based IDS

Regardless of the effectiveness of the anomaly-based IDS, there are inherent risks that should be known and contained. One such risk is the degree of false positives that are likely to be experienced. Unfortunately, these systems do not perform a risk analysis on all activities, and flag everything as non-normal, which in turn leads to more non harmful activities being labeled as attacks for no reason leading to an excess burden on security personnel[13]. The false positive problem is exacerbated in highly dynamic networks where there are notable behavioral variations within legitimate users.

Another risk is that of adverse attack. Those who are aware of the detection parameters of an anomaly-based IDS are able to slightly change their approach as regards to the backdoors[15]. The gradual introduction of the backdoor towards the norm becomes the common goal. Known as adversarial evasion, this method of hiding attacks makes it difficult to realize the presence of the IDS being attacked until it is too late. Furthermore, the use of more assimilative machines in the anomaly-based IDS has its own risks in terms of model development as well as information disinformation. For instance, if the training data set is tampered with or an update of the model is not done, the anomaly detection system is likely to perform its duties poorly leaving the network at risk [14].

Resource consumption is another concern. Anomaly based IDS, especially those based on sophisticated machine learning algorithms can be resource demanding and can make use of heavy processing and memory power. This could pose a problem within resource restricted areas such as IoT networks since their devices may not be able to carry the necessary power and resources to run these systems effectively.

2.4 Understanding Malicious and Benign Network Traffic

In network security, the identification of malicious traffic from benign traffic is particularly important for efficient threat detection. Malicious activity includes the use of a network with the purpose of causing damage or granting unlawful access to one's network system. This can involve the dissemination of malware, carrying out Distributed Denial of Service (DDoS) attacks, exfiltration of unauthorized data, or attempting to breach the vulnerable areas of networks [16] . Such traffic is associated with malignant activities whose purpose includes bringing the services to a halt, stealing confidential information or hijacking the entire network.

While benign traffic is traffic that is able the normal functioning of the network without causing any damage. This is made up to all communications that occur freely between devices, activities that involve people and information exchange and some system functional tasks that need to be performed for proper functionality of the network [17] . Benign traffic is usually created by legitimate users and devices carrying out their duties as expected, for example sending messages over the internet, opening web pages and file transfers.

It is this identification that presents a difficulty whenever there is cause for concern regarding security within the network. Most misleading messages are crafted to look like they are harmless, and this makes it difficult for ordinary detection software to be used effectively without raising alarms. It is here that these more advanced techniques come into play, such as the use of anomaly-based IDS and even machine learning. By focusing on patterns and discovering when things go off the expected path, these systems can also do a better job at separating good and bad behavior, thus lowering the risk of false alarms and undetected attacks.

2.5 Machine Learning in Intrusion Detection

Machine learning (ML) has been the solution for pushing the frontiers in several fields of technology, including for making more efficient the performance of anomaly-based IDS. Machine Learning algorithms can be trained to tell apart good and bad behavior by using datasets of network traffic. There are many types of neural networks where it works best when the input includes labels such as normal, DDoS and C&C traffic and several other labels with various combinations of transient features.

In this concern, we are going to apply ML in the singular aspect of IDS, since like with most other tasks, there are some difficulties. One of the reasons why the performance of models on standard datasets might be poor is due of the imbalanced distribution of the datasets in which in one class Labeling type called ‘Benign’ traffic is extremely high as compared to malicious traffic. Methods like Synthetic Minority Over-sampling Technique (SMOTE) are utilized in an attempt to correct this discrepancy so that the IDS performs well irrespective of the type of network traffic. Apart from addressing these challenges through data reconstruction, an improvement of the algorithms is also very necessary.

2.6 Different Interactions of Machine Learning in IDS

One aspect of intrusion detection system (IDS) that has been significantly improved by the inclusion of machine learning (ML) is the ability to automatically evaluate massive amounts of traffic over a network in search of data patterns that are indicative of potential threats. There are different ways in which ML integrates with the IDS with each offering distinct pros and cons:

Supervised Learning in IDS: Supervised learning is the prevalent technique in the framework of ML based IDS. At this stage, the model is first exposed to the induction procedure with some data outcomes known (benign or malicious) coming from the labelled dataset. After learning this correlation, the model can now be used on new data to predict the outcome associated with a pattern of network traffic. Such learning models involve supervised learning, where classification methods such as decision tree, support vector machines (SVM), and neural networks are developed to fight various types of attacks including Phishing, Distributed Denial of Service (DDoS) attacks and so on [13]. The main feature of supervised learning is that sufficient and relevant data is provided is trained, accuracy is expected and is often achieved. On the downside, this almost-perfect model does not mean the absence of challenges. The performance on a task depends heavily on the training set which has its limitations such as content bias and may not adapt to the changing dynamics of threats. [14]

Unsupervised Learning in IDS: Unsupervised learning does not use the labeled data. Rather, it is more focused toward the underlying data itself, in the search of discovering trends and differences based on the ways in which the data is organized. In this case, clustering or K-means and anomaly detection techniques such as the Isolation Forest are very often employed. There are cases when there is lack of the available labeled data or specific new risks are being sought, for instance, in these cases unsupervised learning is also relevant [15]. In this way, anomalous behavior, which may be identified by traditional means as suspicious and which may be overlooked, would be helpful in triggering suspicion since large volumes of raw data working in this manner will always regard outlier classification as the norm. However, because of this tendency, one of the disadvantages of this approach is that there is a risk of having very high false-reporting cases [16] .

Semi-Supervised Learning in IDS: Semi supervised learning combines supervised and unsupervised learning techniques for training. Here, the model is trained on a small set of labeled data and a large set of unlabeled data. The idea of this strategy is to use the available labeled data to assist learning, as well as, learning about the rest of the data in search of some hidden structures and associations. This method is particularly useful in situations where acquiring a fully labeled dataset is impractical or costly [17]. It can enhance the model performance under these attack categories without forfeiting the benefits of supervised learning in the first instance. Thence, the risk associated with this strategy is that success is dependent on the quantity of labeled data while the rest of the unlabeled data is put to meaningful or productive uses by the model .

Reinforcement Learning in IDS: The goal in reinforcement learning is to train an agent through feedback obtained from the environment. In the case of IDS, such learning can enable the clinician to create real-time models that respond to threats as they emerge. An agent receives rewards when it accurately identifies true threats, and is penalized when it misidentifies a non-existent threat, or misses the real threat aiming at increasing its performance [19] . Such schooling works well in volatile situations with newly emerging threats, like in IoT networks. Yet, this approach and practical application will involve many details making it heavy computational wise and absolutely

an adroit balance on strategy of seeking out other ways of doing things and making use of ways that had been considered successful previously [20] .

Ensemble Learning in IDS: Ensemble learning is one of the techniques in machine language which incorporates two or more ML models with the intention of increasing the overall effectiveness of the IDS. Ensemble methods, which primarily consist of model averaging, allow mitigating the probability of misclassifying an object as belonging to a class or unjustly rejecting it from that very class. Bagging, boosting and stacking are well-studied and implemented methods that allow building an assembly of weaker models into a stronger one. This type of learning is beneficial when several models are required to cover the complexity of the task at hand. It also has its challenges. The challenges lie around the resource and knowledge requirements to administer several models at a time [21].

2.7 IoT Security Challenges

The growth of technology and techniques that help connect multiple devices in real-time has further worsened the traditional security configurations. IoT devices, which include but are not limited to smart devices and industry sensors, usually do not have enough power support abstraction, console-level programming tools and operating systems to use standard security practices. In addition, they are accessible to the internet and available in large numbers which is a magnet for cyber criminals. [22]

Another major issue existing in securing IoT networks is the incorporation of new devices which employs various kinds of protocols and communication approaches. This variation also makes it difficult to locate suspected abnormal behaviors, as what may appear to be normal for one machine may seem out of place for another machine. Moreover, the fast diffusion of threats aimed at IoT devices has made this even worse since there is a need for the IDS models to be left astray in one design. [23]

3. IoT-23 Dataset

3.1 Overview

The IoT-23 Dataset helps researchers create machine learning models for intrusion detection on Internet of Things (IoT) devices. The dataset created by Stratosphere Laboratory with the cooperation of Avast Software contains network traffic from malevolent and benign internet of things devices. Between 2018 and 2019 are a total of 23 scenarios, which include 20 malicious and 3 benign. The dataset has over 500 hours of network traffic, more than 300 million labeled flows, and captures traffic from a variety of IoT devices infected by different types of malwares, including Mirai, Torii, and Okiru. For this project, the captures used in this study are *CTU-IoT-Malware-Capture-34-1*, *CTU-IoT-Malware-Capture-48-1*, and *CTU-IoT-Malware-Capture-60-1*. An analysis will focus on these captures to understand their significance and the reasons for selecting them. [24]

The dataset is organized into separate scenarios, each of which has multiple files, including the following:

- .pcap files (the raw network capture data),
- conn.log. labeled files (created by Zeek IDS, these contain labeled network flows),
- README.md (provides metadata about the capture, including malware names and other relevant details).

Using the type of activity detected, every traffic flow is labelled as Attack, DDos, or Mirai botnet traffic, etc. These labels are important in training machine learning models to differentiate between normal and malicious behaviors in the IoT environment.

Each traffic flow is labeled based on the type of activity detected, such as **Attack**, **DDoS**, or **Mirai** botnet-related traffic. These labels are crucial for training machine learning models to differentiate between normal and malicious behavior in IoT environments.[7] [8]

To access the dataset, there are two main download options:

1. **Full download** (20 GB) includes all raw capture data, logs, and detailed metadata.
2. **Light version** (8.7 GB) offers only the essential metadata and labeled log (This was the selected version for this project)

For more details about the dataset, you can access the data repository here: IoT-23 Dataset Repository

3.2 IoT-23 Dataset Labels

The IoT-23 dataset provides detailed labels for network traffic flows to aid in the identification of malicious activities. These labels are crucial for researchers developing machine learning models for intrusion detection.

- Attack: Indicates an active attempt by an infected device to exploit vulnerabilities, such as brute force attacks or command injections.
- Benign: Denotes normal, non-malicious traffic with no suspicious activities detected.
- C&C (Command and Control): Refers to communication between an infected device and a remote C&C server, often used to issue commands or download malicious binaries.
- DDoS (Distributed Denial of Service): Marks traffic involved in a DDoS attack, characterized by a large volume of flows targeting a single IP.
- FileDownload: Identifies when a file is being downloaded to an infected device, typically from a suspicious server associated with malware distribution.
- HeartBeat: Represents periodic communication from an infected device to a C&C server, often used to maintain a connection or send status updates.
- Mirai: Labels flows exhibiting characteristics of the Mirai botnet, which is known for large-scale DDoS attacks using IoT devices.
- Okiru: Similar to Mirai, but pertains to the Okiru botnet, a less common but equally dangerous family of IoT malware.
- PartOfAHorizontalPortScan: Indicates that the traffic is part of a horizontal port scan, where multiple IP addresses are probed on the same port, often as a precursor to further attacks.
- Torii: Labels traffic associated with the Torii botnet, which, like Mirai and Okiru, targets IoT devices but with different attack patterns.

3.2.1 Key Tools and Technologies for IoT Dataset Creation

1. **Zeek IDS** (formerly known as Bro): Zeek is a powerful IDS. It processes raw network data to produce labeled log files. One such output is conn.log. labeled. This contains information about the traffic flows and the activity type.
2. **Packet Capture Tools**: Packet Capture Tools are those that allow the recording of the original network traffic in. pcap format. For example, Wireshark. This ensures that each packet sent between IoT devices and external servers is recorded for later analysis.

3. **Python**: The labeled data is processed through Scripts written in Python. This allows the client to use such data in Machine Learning models for training and extraction of features.
4. **Machine Learning Libraries**: Things like TensorFlow or Scikit-learn framework can be used to train and evaluate the model as per the labeled traffic in the dataset.
5. **Storage Solutions:** The dataset consists of metadata (README.md), raw captures (.pcap), and labelled logs (conn.log.labeled), all organized into directories specific to each scenario. The files are stored in repositories for public access.

3.3 Information in the dataset

Table 1: Dataset description

Column	Description	Data Type
Ts	Timestamp in UNIX epoch format	Int
Uid	Unique ID of Connection	Str
Id_orig.h	Originating endpoint's IP address	Str
Id_orig.p	Originating endpoint's TCP/UDP port	Int
Id_resp.h	Responding endpoint's IP address	Str
Id_resp.p	Responding endpoints TCP/UDP port	Int
Proto	Transport layer protocol of connection	Str
Service	Dynamically detected application protocol	Str
duration	Duration of time between the first seen packet and the last seen packet	Float
Orig_bytes	The number of payloads bytes the infected device sent	Int
Resp_bytes	The number of payload bytes the responder sent	Int
Conn_state	Possible connection	Str
Local_orig	If the connection is originated locally, this will be T	Bool
Local_resp	If the connection is responded locally, this will be T	Bool

Missed_bytes	Indicates the number of bytes missed in content gaps	Int
History	Records the state history of connections as a string	Str
Orig_pkts	Number of packets that the originator sent	Int
Orig_ip_bytes	Number of IP level bytes that the originator sent	Int
Resp_pkts	Number of packets that the responder sent	Int
Resp_ip_bytes	Number of IP level bytes that the responder sent	Int
Tunnel_parents	UID values for any encapsulating parent connections	Str
Label	The type of capture; it can be benign or malicious	Str
Detailed_label	If the capture is malicious, denotes what malicious counterpart it is	Str

3.3.1 Example Records from the Dataset

These records provide a snapshot of how network traffic is captured, labeled, and structured, showcasing various malicious activities such as C&C Heartbeat and Horizontal Port Scanning.

Table 2: Raw dataset snapshot

ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	resp_bytes	conn_state	label	detailed-label
1551377734.2	C9mqzS28ln5Lv41J09	192.168.1.200	52724	167.99.182.238	80	tcp	http	1.978591	149	119442	SF	Malicious	C&C-HeartBeat-FileDownload
1551377736.2	CUQLPn2ZXYFN1lNgo6	192.168.1.200	52726	167.99.182.238	80	tcp	http	2.182247	149	119442	SF	Malicious	C&C-HeartBeat-FileDownload
1551377744.2	CeGuVh3VK2bqRWuQEc	192.168.1.200	38448	1.1.1.1	23	tcp	-	3.141713	0	0	S0	Malicious	PartOfAHorizontalPortScan
1551377744.2	ClyBYW2sZ2r420lBS1	192.168.1.200	47056	2.2.2.2	23	tcp	-	3.141708	0	0	S0	Malicious	PartOfAHorizontalPortScan

1551377744.2	CWKlQsAuidgnIYxjb	192.168.1.200	33046	5.5.5.5	23	tcp	-	3.141709	0	0	S0	Malicious	PartOfAHorizontalPortScan

3.4 Dataset Selection Process

Before landing on the CTU-IoT-Malware-Capture-48-1 dataset, I first experimented with two other malware captures; CTU-IoT-Malware-Capture-34-1 and CTU-IoT-Malware-Capture-60-1. However, after running these captures for analysis I noticed that they were less suitable for the project's needs and prospects. Although both datasets proved valuable to build the model and experiment with different flows, they presented limitations in diversity and data quality in their flows which resulted in their effectiveness in capturing every aspect of network malware activity.

3.4.1 CTU-IoT-Malware-Capture-34-1:

Table 3: Capture-34-1 structure

Label	Flows
Benign	1,923
C&C	6,706
DDoS	14,394
PartOfAHorizontalPortScan	122

Staring with CTU-IoT-Malware-Capture-34-1, this capture is presented by heavily leaning towards DDoS and C&C flows, with limited Benign traffic (1,923 flows). The capture’s emphasis on DDoS at 14,394 flows over the benign traffic could cause the model to have a poor generalizability of real-world network scenarios. Additionally, the limited flows of PartOfAHorizontalPortScan activity at 122 flows limits the insight into scanning activities, further reducing the model's ability to detect various types of network activities.

3.4.2 CTU-IoT-Malware-Capture-60-1:

Table 4: Capture-60-1 structure:

Label	Flows
Benign	2,476
C&C-HeartBeat	95
DDoS	3,578,457

Similarly, CTU-IoT-Malware-Capture-60-1 proved challenging due to a larger imbalance, with 3,578,45 DDoS flows dominating the dataset and only 2,476 benign flows. It is tihs large

imbalance that further deviated from real-world network scenarios and prevent the model to accurately understand benign and malicious network. Additionally, with only 95 C&C-Heartbeat flows, the dataset lacks diverse and robust types of essential attack characteristics, that are present in real-world scenarios.

3.4.2 CTU-IoT-Malware-Capture-48-1:

Table 5: Capture-48-1 structure

Label	Flows
Benign	20,574,934
C&C	3,498
C&C-FileDownload	14
DDoS	65,803
FileDownload	1
Okiru	8,765,885
PartOfAHorizontalPortScan	37,911,674

But finally, having reached CTU-IoT-Malware-Capture-48-1, it provided a more balanced representation of network activities and anomalies. Being dominated by port scanning activity, with **37,911,674 flows** marked as ***PartOfAHorizontalPortScan,*** with ***O*kiru** malware flows also being prominent, with **8,765,885 flows**. We can also note Benign traffic being present but less significant, totalling **20,574,934** flows. Other malicious flows include 65,803 DDoS flows, highlighting an active attempt at network disruption, and 3,498 C&C flows, pointing to ongoing command and control communications. Additionally, we can note other flow types such as C&C-FileDownload with 14 flows and a single FileDownload instance, suggesting limited direct download activity from the C&C servers. This capture's greater volume and diversity enabled a more comprehensive training ground for a model aimed at detecting multiple types of network threats, this also more closely related real-world scenarios where these factors are present and prominent making it a better fit for the goals of my project.

3.5 Training data and the model selection

3.5.1 Dataset Complexity and Structure

The selected CTU-IoT-Malware-Capture-48-1 dataset is very large and complex, featuring millions of instances of both benign and malicious network traffic. Moreover, due to the dataset's vast size and high imbalance between different types of attacks and benign activity, it becomes challenging to achieve high accuracy, especially in detecting rare attack types such as C&C-FileDownload (only 14 occurrences) or single FileDownload instances. These characteristics demand a robust approach capable of detecting patterns across a wide variety of classes, handling

imbalanced data, and scaling to large datasets, thus neural networks were chosen when creating the model.

Seeing the datasets characteristics, neural networks and their capacity to model complex, and imbalanced datasets. While traditional algorithms could have been considered, but after experimenting with both versions the algorithms trade-offs finally led to the decision to prioritize neural networks.

3.5.2 Neural Networks

Neural networks are well suited to deal with the non-linear and high-dimensional nature of network traffic data in malware detection. Their hierarchical architecture allows them to learn complex patterns that improve feature extraction from the data, and is especially useful when dealing with multiple types of attacks that vary widely in frequency and characteristics. This is a design choice that improves the ability to generalize models without memorizing data, an important feature for malware datasets that contain large amounts of benign traffic and diverse attack types.

3.5.2.1 Limitations of Traditional Algorithms and Trade-Offs

1. **Random Forests**: Random Forests are often strong performers in tabular data, but data with this size and complexity have limitations. Random forests build multiple decision trees, each trained in a subset of data, and make predictions by measuring or voting on the results of these trees. However, this approach may require excessive calculation time and memory for a dataset of this size, such as CTU-IoT-Malware-Capture-48-1. Given the high dimension and volume, a random forest may also be prone to over-adaptation without careful adjustment. In addition, this method would require balancing techniques such as SMOTE to counter the severe class imbalance, which would increase computation requirements. Random forests may also struggle to capture the nuanced relationships between characteristics that neural networks can learn from their layer structures.

2. **Support Vector Machine (SVM)**: SVM is ideal for binary classification tasks and is especially useful in large-scale projects. However, when applying to large databases, they are computationally intensive and usually require significant pre-processing and adjustments. Training SVMs on millions of samples would take time and memory, and while they may achieve reasonable results, they would likely not achieve the performance of the neural network in the handling of complex patterns. In addition, multi-class imbalances in the datasets present a major challenge for SVMs, which are generally sensitive to class distribution and may require complex rebalancing techniques to achieve acceptable results.

3. **K-Nearest Neighborhoods (KNN)**: KNN classifications are examples-based learners that classify samples by finding the majority class among the closest examples in the set. Although KNN is relatively easy to implement, due to its high memory consumption and slow time to infer, it is not practical for large-scale datasets because it requires calculating the distance between each new sample and each point in the training set. With data sets such as CTU-IoT-Malware-Capture-48-1, containing tens of millions of streams, KNN would struggle to increase efficiency. Furthermore, the KNN performs badly in unbalanced data because the common classes dominate the nearest neighbors, making it difficult to correctly identify the rare attack types. This algorithm would require extensive sampling or balance techniques to be executed at a reasonable level, which further complicates its use for this task.

3.5.3 Hypothetical performance of traditional algorithms

Using a traditional algorithm on this data might give performance that is less consistent on all classes as compared to a neural network. Since the dataset has a Iot of common attacks like DDoS flows, these methods can achieve reasonable accuracy on them. But sometimes things like only downloading one file, or not connecting to the C+C for a long time, can be missed. This means the rate of missing things that matter is high. In the real world, malware detection requires detecting nuanced, rare threats and this would compromise the model's ability to do such detection.

Besides, to the traditional algorithms we might encounter significant overfitting or underfitting with heavy tuning and pre-processing without the use of neural networks. Neural networks may provide better results for this data set in the long see. They can get moderate win on some classes with careful tuning, but unlike a neural network, they may be too restrictive in their learned relationships.

The CTU-IoT-Malware-Capture-48-1 dataset is very large, complex and imbalanced so neural networks were considered best among other techniques in this study. Use SMOTE, regularization, and other techniques to reduce class imbalance. Learn complex patterns efficiently with all of that. Although they could be used, they would require a Iot of preprocessing and balancing to make the data workable. All of this means that, by themselves, they wouldn't be as accurate or efficient as the neural network. The use of neural network model is to balance the computational cost with the ability to generalize on various network attacks

4.Data Processing and methodology

Here is some sample data from the CTU-IoT-Malware-Capture-48-1 log file. This data contains the traffic sent by IoT networks. It has several fields such as the timestamp, source IP and destination IP, type of protocol, state of the connection, and labels which show whether the traffic is benign or malicious.

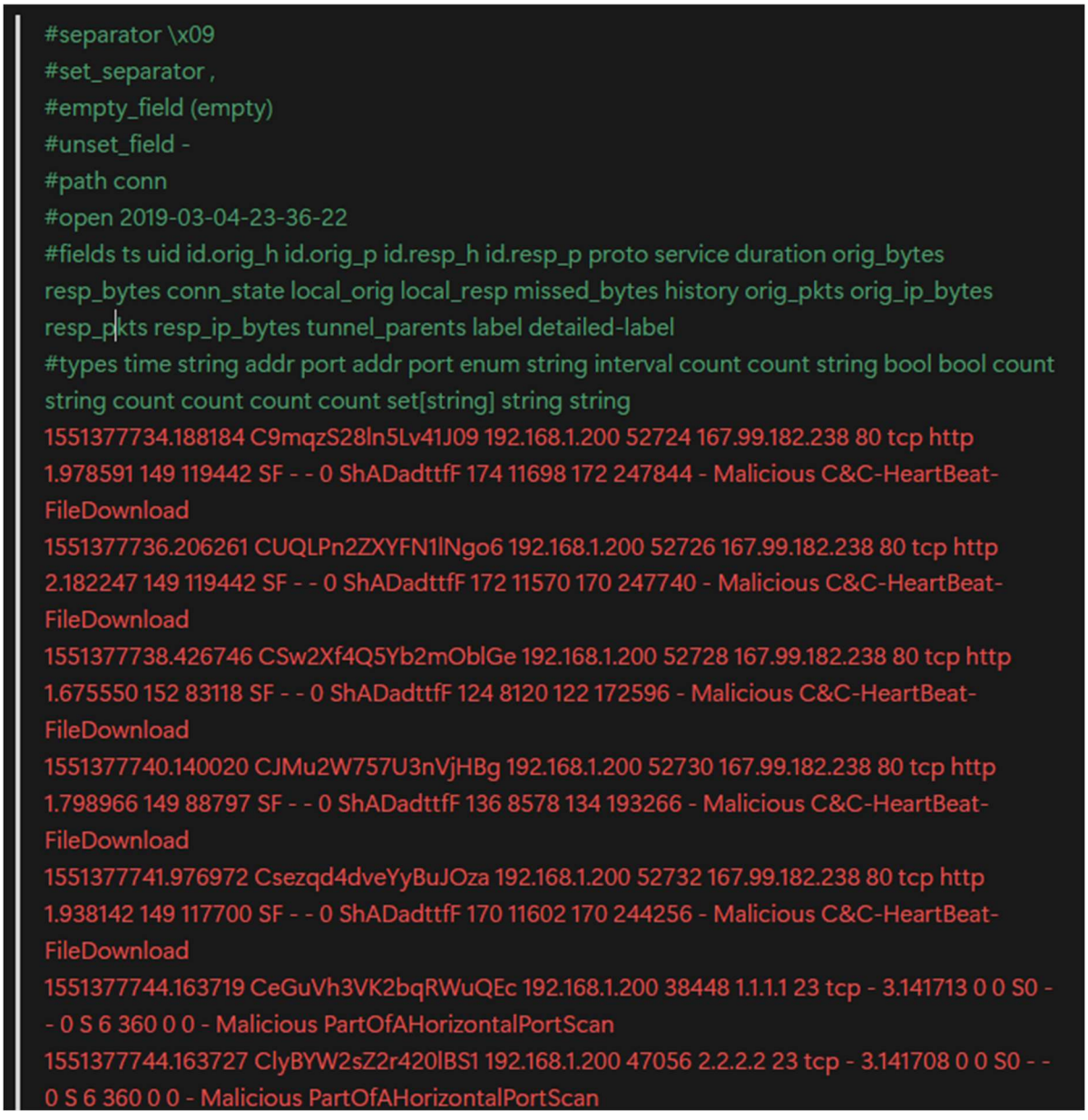

Fig. 1: Raw code data. Source: Own development.

4.1 Raw data

#separator: Specifies the delimiter used in the file (Tab \x09).

#fields: Lists the names of the columns in the dataset (e.g., ts, uid, id.orig_h, etc.).

#types: Describes the data types of each column (e.g., time, string, count).

Raw Data: Each record in the dataset is represented in raw format, separated by the delimiter (\x09 for tab). Example records contain information such as:

- **ts**: Timestamp of the event.
- **uid**: Unique identifier for the connection.
- **id.orig_h**: IP address of the originating endpoint.
- **proto**: Protocol used in the connection.
- **service**: Service detected (e.g., http).
- **label**: Indicates whether the traffic is benign or malicious (e.g., Malicious).
- **detailed-label**: Specifies further details if the traffic is malicious (e.g., C&C-HeartBeat-FileDownload).

4.2 Data Loading

To begin, I loaded my dataset in a panda DataFrame. The information from a log file specified the delimiter and many headers among the data. I indicated that the data should ignore comments, treat certain characters as missing values, and recognize the appropriate data types for each column to read the file.

```
file_path = r"C:\Users\Canela\Desktop\Project\CTU-IoT-Malware-Capture-48-1\bro\conn.log.labeled"

df = pd.read_csv(file_path, delimiter='\t', comment='#', na_values='-', header=None, dtype={'service': str})


df.columns = [
    'ts', 'uid', 'id.orig_h', 'id.orig_p', 'id.resp_h', 'id.resp_p', 'proto', 'service',
    'duration', 'orig_bytes', 'resp_bytes', 'conn_state', 'local_orig', 'local_resp',
    'missed_bytes', 'history', 'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes',
    'label'
]
```

Fig. 2: Code: Data loading. Source: Own development.

After loading the data, the next important step is data pre-processing that plays a key role in making the data ready for the machine learning models. The LabelEncoder, a component of sklearn, is utilized to transform the categorical labels into numerical values. The code drops several columns that we think are not useful for training the model like ‘uid’, ‘ts’, ‘id.orig_h’ and ‘id.resp_h’. Often the features do not carry any information or can lead to data leak, which is misleading the model to learn from the features in training. Afterwards, the "proto", "service", "conn_state", etc. are converted using one hot encoding. This technique effectively creates dummy columns for every category.

```
# Encode labels
label_encoder = LabelEncoder()
df['label_encoded'] = label_encoder.fit_transform(df['label'])

# One-hot encode categorical features
categorical_features = ['proto', 'service', 'conn_state']
df = pd.get_dummies(df, columns=categorical_features, drop_first=True)
```

Fig. 3: Code: One-Hot encoding. Source: Own development.

Pre-processing is crucial for preparing the dataset for machine learning models. Key steps included encoding categorical features, dropping unnecessary columns, and balancing the data.

- **Label Encoding**: The LabelEncoder from scikit-learn transformed categorical labels (e.g., Malicious, Benign) into numerical values.
- **One-Hot Encoding**: Categorical features such as proto, service, and conn_state were converted into dummy variables to avoid numerical bias.

4.2.1 Handling Class Imbalance

```
# Handle class imbalance using SMOTE
smote = SMOTE()
X_res, y_res = smote.fit_resample(X, y)
```

Fig. 4: Code: Class Imbalance. Source: Own development.

Class imbalance is a typical problem that occurs in the case of classification tasks. Security-related datasets, usually have a lot more benign samples than malicious ones. To solve this problem, the code applies the SMOTE algorithm, which stands for the Synthetic Minority Over-sampling Technique. This algorithm generates synthetic samples of the minority class (malicious traffic, in this case). By enhancing the training set this way, the model is likely to learn meaningful patterns from the benign and malicious classes.

- **Class Imbalance Handling**: The Synthetic Minority Oversampling Technique (SMOTE) was applied to generate synthetic samples for underrepresented classes, addressing the class imbalance issue.

4.2.2 Feature Scaling

```
scaler = RobustScaler()
X_res_scaled = scaler.fit_transform(X_res)
```

Fig. 5: Code: Feature Scaling. Source: Own development.

After ensuring that the dataset is balanced, it is now time to perform the feature scaling on the dataset using RobustScaler. RobustScaler is outlier resistant. It will scale the dataset by centering it at median and scaling it according to IQR. This step helps to ensure that each feature counts equally. This means that features with larger ranges don't have an outsized effect on the results.

The **RobustScaler** was chosen for its resistance to outliers. It scales the data by centering at the median and scaling according to the interquartile range (IQR).

4.2.3 Stratified K-Fold Cross-Validation

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for train_index, val_index in skf.split(X_res_scaled, y_res):
    X_train, X_val = X_res_scaled[train_index], X_res_scaled[val_index]
    y_train, y_val = y_res[train_index], y_res[val_index]
```

Fig. 6: Stratified K-Fold Cross-Validation. Source: Own development.

The next step is to set up a stratified K-fold cross-validation process, which is necessary to evaluate the performance of the model. Stratification ensures that each cross-validation fold maintains the proportion of classes present in the dataset, thus avoiding distortion of the predicted metrics. The code defines five partitions and divides the dataset into training and validation subsets during each iteration. This method not only avoids overfitting, but also provides a more accurate prediction of the model's performance.

4.2.4 Model Building and Training

```
model = Sequential([
    Flatten(input_shape=(X_train.shape[1],)),
    Dense(128, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.4),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.3),
    Dense(len(np.unique(y_res)), activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
| Ctrl+L to chat, Ctrl+K to generate
```

Fig. 7: Code: Model training. Source: Own development.

Key Terms:

- **Dropout**: A regularization technique where random neurons are dropped during training to prevent overfitting.
- **Softmax Activation**: A function that converts output scores into probabilities, used in multi-class classification.
- **Adam Optimizer**: An adaptive learning rate optimization algorithm.

In each layer, the code uses the Keras Sequential API to build the neural network model: Flatten layer converts the input features into a 1D array, Dense layer introduces a non-linear structure using the ReLU activation function, Dropout layer randomly sets the input units to zero to reduce overload. The output layer uses softmax activation, which is suitable for multi-class classification tasks. The model is built using Adam's optimizer and a sparse categorical cross entropy loss function. During the learning process, class weights are calculated to address imbalances in the class representation. The learning process uses call-backs for early stopping (to stop learning when the validation loss no longer improves) and TensorBoard to monitor learning progress.

4.2.5 Model Evaluation

```
val_predictions = model.predict(X_val)
y_pred_classes = np.argmax(val_predictions, axis=1)
print(confusion_matrix(y_val, y_pred_classes))
print(classification_report(y_val, y_pred_classes))
| Ctrl+L to chat, Ctrl+K to generate
```

Fig. 8: Code: Model evaluation. Source: Own development.

As models are trained on each floor, they are evaluated using validation data. Evaluation metrics include validation loss, accuracy, ROC AUC and log loss, all of which provide insight into model performance. The predictions are processed to generate class labels, which are then compared to

the actual labels to generate a confusion matrix and a classification report. The confusion matrix provides estimates of the proportion of true positives, false positives, true negatives and false negatives for different classes, while the classification report provides accuracy, recall and F1 scores for a deeper understanding of model performance.

4.3 Detailed Analytical summary

Table 6: Model performance table

Class	Precision	Recall	F1-Score
Benign	0.9997	0.9994	0.9996
Malicious Attack	0.9835	0.8430	0.9078
Malicious C&C-HeartBeat-Attack	0.8629	0.9862	0.9204
Malicious C&C-HeartBeat-FileDownload	0.9996	1.0000	0.9998
Malicious C&C-PartOfAHorizontalPortScan	0.5765	0.6837	0.6255
Malicious PartOfAHorizontalPortScan	0.6114	0.4977	0.5487

Table 7: Model performance summary table

Metric	Mean	Std Dev	Min	Max
Accuracy	0.8366	0.0017	0.8347	0.8387
ROC-AUC	0.9714	0.0002	0.9710	0.9717
Log Loss	0.2940	0.0026	0.2908	0.2972

4.3.1 Model Performance and Validation

The created deep learning modelling performed well on various measures of assessment. Through 5-fold cross-validation, the model attains a mean accuracy of 83.66% (±0.17%) and a ROC-AUC score of 0.9714 (±0.0002) indicating good stability. This means that the model is expected to show the same results across different partitions of data.

Table 8: Detailed Cross-Validation metrics table

Fold	Accuracy	ROC-AUC	Log Loss
1	0.8387	0.9717	0.2908
2	0.8358	0.9714	0.2957
3	0.8347	0.9710	0.2972
4	0.8387	0.9715	0.2910
5	0.8350	0.9715	0.2954

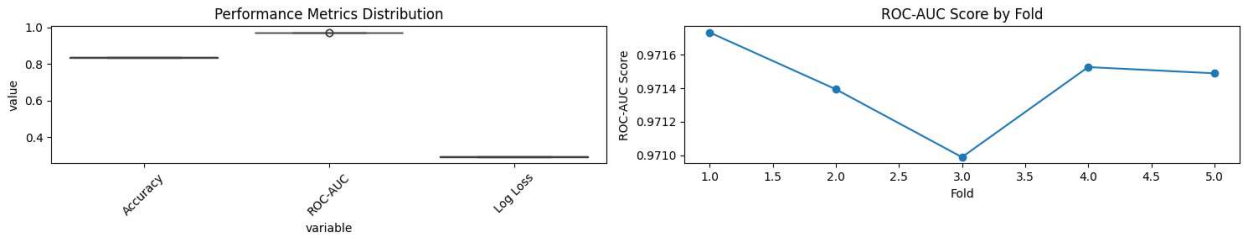

Fig. 9: Metric distribution & ROC-AUC scores. Source: Own development.

4.3.2 Class-Specific Performance

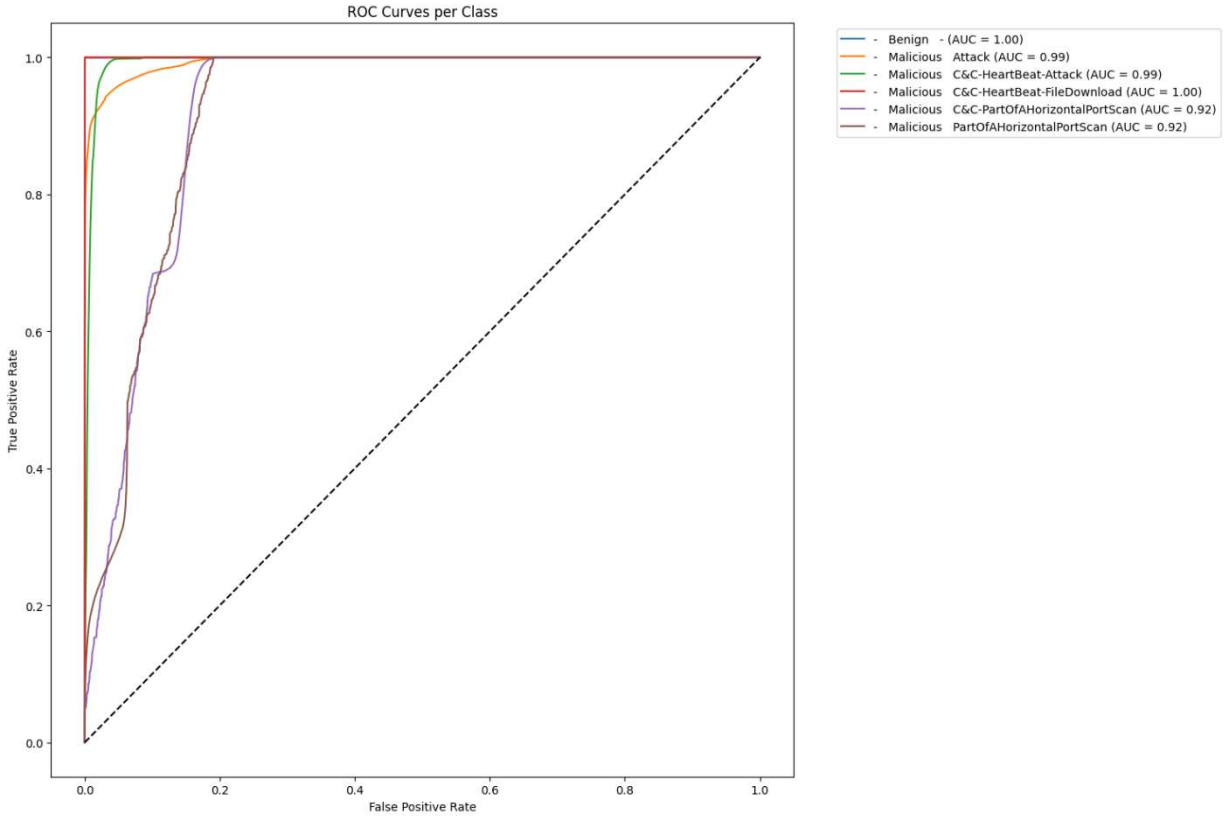

Fig. 10: ROC Curve. Source: Own development.

Analysis of class-wise performance revealed attack effectiveness varied by class. The model showed very good performance at identifying benign traffic and C&C-HeartBeat-FileDownload attacks. It achieved 100% precision and 100% recall for both. The detection of malicious attacks had a precision of 97-98% and a recall of 84-88%. Nevertheless, the model was comparatively weak in distinguishing between PartOfAHorizontalPortScan variants (-57-61% precision and -50-68% recall), which presents an opportunity for improvement.

Table 9: Error analysis table

Class	Misclassification Rate (%)
- Benign -	0.06
- Malicious Attack	15.70
- Malicious C&C-HeartBeat-Attack	1.38
- Malicious C&C-HeartBeat-FileDownload	0.00
-Malicious C&C-PartOfAHorizontalPortScan	31.63
- Malicious PartOfAHorizontalPortScan	50.23

4.3.3 Computational Efficiency and Resource Utilization

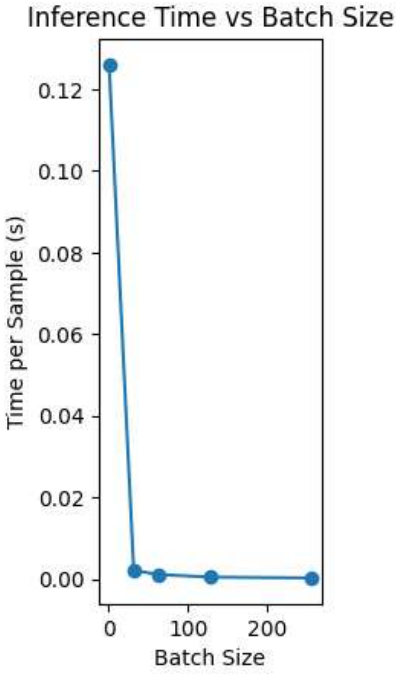

Fig. 11: Time vs Batch Size. Source: Own development.

The architecture of the model is well-balanced in terms of performance and resources. With only 18,246 trainable parameters, this model only requires a memory footprint of 725.55 MB and has CPU utilization of just 4.6 %. Its low weight is especially useful for real-time network monitors to save on resources. The architecture of the model is efficient and shows that network intrusion detection does not have to be complex or require too many resources.

Table 10: CPU Usage

Metric	Value
CPU Usage	4.6%
Memory Usage	725.55 MB

Time Analysis:

- Each epoch took ~3-4ms/step
- Each fold ran for ~20-25 epochs on average

Each fold took approximately:

- Training: ~190-235 seconds per epoch × ~20 epochs ≈ 3800-4700 seconds
- Evaluation: ~75-80 seconds
- Total runtime for all 5 folds: approximately 5-6 hours

Longest Running Components:

- Model training
- 2. SMOTE resampling
- Model evaluation and prediction
- Feature preprocessing

4.3.4 Learning Dynamics and Training Stability

The convergences of the training metrics remain similar across the validation folds. The model learns quickly during its initial training, with noticeable performance improvements in the first 5 to 10 epochs. After updating the model continually, the model can fine-tune the task. The early stopping mechanism successfully stopped overfitting; most runs stabilized from epochs 15-25. The model architecture and choice of hyperparameters are well-balanced according to this efficient convergence pattern.

4.3.5 Operational Implications

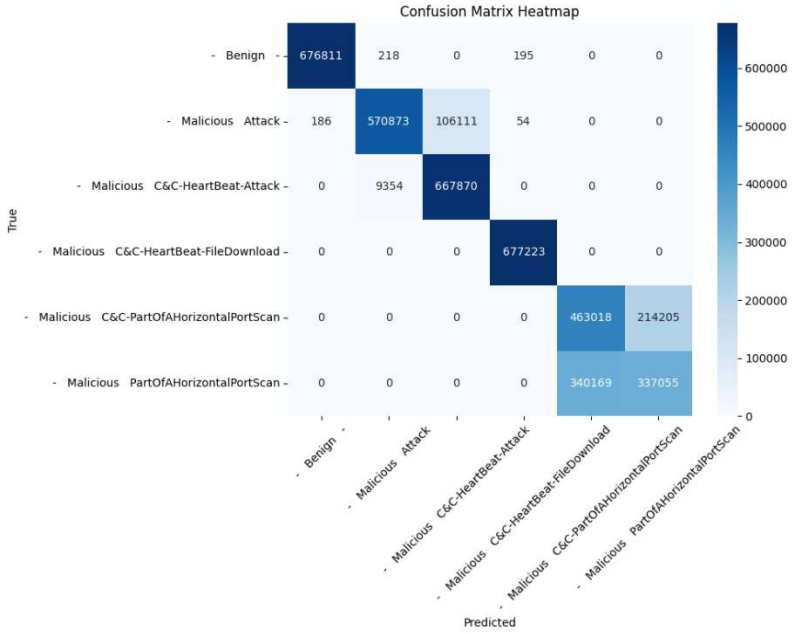

Fig. 12: Confusion Matrix heat map. Source Own development.

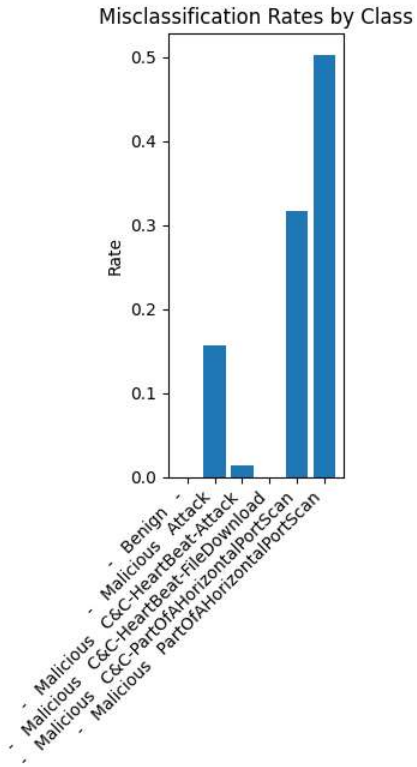

Fig. 13: Misclassification rates. Source Own development.

The confusion matrix analysis shows you useful insights to operate. The model's ability to detect benign traffic without error (100%) avoids false alarms in real-world use cases. The efficiency in identifying C&C-HeartBeat attacks (97-99% recall) implies their usefulness against command-and-control threats. However, the low accuracy in PortScan detection suggests the specialized detection mechanism is preferable for this attack.

4.3.6 Technical Implementation and Scalability

Table 11: Model architecture table

Layer	Output Shape	Units/Rate	Activation	Regularization	Function
Input	Input Shape	-	-	-	Accepts input features, preparing them for processing.
Flatten	(None, 74)	-	-	-	Flattens multi-dimensional input to a 1D array.
Dense 1	(None, 128)	128	ReLU	L2(0.001)	Learns complex patterns with non-linearity and regularization to prevent overfitting.
Dropout 1	(None, 128)	40%	-	-	Randomly drops 40% of neurons to enhance generalization.
Dense 2	(None, 64)	64	ReLU	L2(0.001)	Further extracts features while reducing dimensionality, with L2 regularization.
Dropout 2	(None, 64)	30%	-	-	Drops 30% of neurons to reduce model overfitting.
Output	(None, 6)	6	Softmax	-	Produces class probabilities for multi-class classification.

- **Input & Flatten:** Prepares and reshapes data for dense layers.
- **Dense Layers:** Perform feature extraction and learning, with ReLU providing non-linearity.
- **Dropout:** Prevents overfitting by introducing random neuron deactivation.
- **Output Layer:** Converts outputs into probabilities for classification

Scalability features of implementation metrics look promising. The model can process 256 samples at a time and produces the same inference time on differing batch sizes. The CPU utilization of 4.6% when it was running means it could run at the same time or on a lower-end machine. The model adds dense layers with a dropout of 0.3–0.4.

4.3.7 Future Optimization Potential

Although the model performs well overall, our analysis shows that it has weaknesses. The difference in detection rates for attack types suggests an overall hierarchical or ensemble approach for PortScan detection, without sacrificing the existing capabilities of the model. Low resource usage also indicates that model expansion or feature engineering may be possible without losing deployment practicality.

4.4 Misclassifications & False Positives

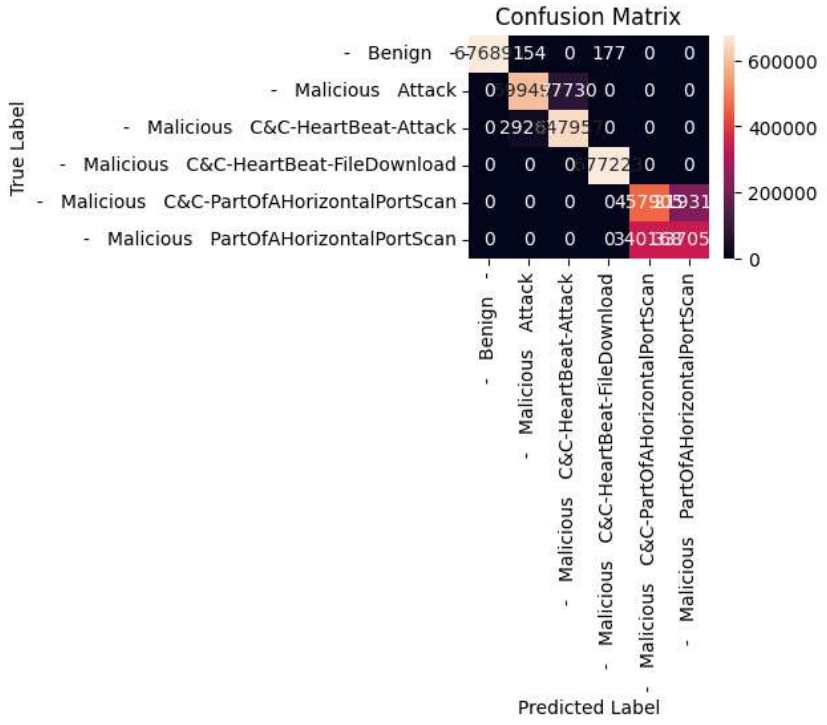

Fig. 14: Confusion matrix. Source Own development.

- Shows a clear diagonal pattern, which is good as it indicates many correct classifications
- The largest confusion occurs between PortScan types (bottom right corner)
- Very few misclassifications of benign traffic (top row)
- Notable confusion between Malicious Attack and C&C-HeartBeat-Attack (visible in the upper left portion)

The colour intensity indicates the number of cases, with darker colours representing fewer cases and brighter colours representing more cases.

Table 12: Classification accuracy

Total samples	4,063,342
Correctly classified	3,396,528
Misclassified	666,814
Accuracy	83.59%

According to the model, **4,063,342** network samples were analyzed. It successfully classified **3,396,528** cases with an accuracy of **83.59%**. This is very accurate, but there were **666,814** cases which were misclassified. The misclassifications take place because network behavior is complex, and it is difficult to tell the difference between some benign and malicious ones, in particular, the similarity between some types of attacks can present a significant challenge, as can the resemblance of benign traffic to it on occasion.

4.4.1 Top 10 Most Influential Features

Table 13: Top features

Features	**Rates**
history_S	0.9132
conn_state_S0	0.8934
orig_bytes	0.5728
conn_state_SF	0.4934
service_http	0.4121
history_ShADadttfF	0.4121
resp_ip_bytes	0.4082
resp_bytes	0.4075
proto_tcp	0.4004

4.4.2 Feature Descriptions and Relevance

- **history_S(Score:0.9132)**
  This represents the **connection history** of a session, typically indicating whether a connection was successfully established.
    - A high value (close to 1.0) suggests that the feature is a strong predictor, particularly for activities such as **port scans** where a sequence of similar connection patterns is observed.
    - Example: Continuous successful SYN packets (S) may signify a reconnaissance scan.
- **conn_state_S0(Score:0.8934)**
  This indicates the **connection state**, specifically the state when a **SYN packet** is sent but no response is received (state S0).
    - A high value often correlates with **unanswered connection attempts**, which are characteristic of **failed port scans** or **denied access attempts**.
    - Significance: Attackers might scan multiple ports, resulting in numerous S0 states when targets refuse or drop connections.
- **orig_byte(Score:0.5728)**
  This is the number of **bytes sent by the originator** of the connection (client).
    - Lower values are often associated with **malicious activity** since reconnaissance scans or lightweight bo t commands may not transfer significant data.
    - A difference in byte counts has been noted between benign traffic (~0.9675) and malicious traffic (~0.0334), making it a moderately important predictor.
- **conn_state_SF(Score:0.4934)**
  This denotes a **fully established connection** where data was sent both ways and the connection was closed properly (SF).
    - Higher values for benign traffic indicate legitimate communication, whereas malicious traffic tends to have incomplete or abrupt connections.
- **service_http(Score:0.4121)**
  This feature identifies if the connection is associated with the **HTTP protocol**.
    - Malicious activity targeting web servers or engaging in command-and-control (C&C) often involves HTTP, making it a relevant, though less critical, feature.

- **history_ShADadttfF(Score:0.4121)**
  This is a **compound connection history** feature representing various states: S (SYN), h (handshake), A (ACK), D (Data), and F (FIN).
    - It reflects a more complex interaction pattern, often indicative of **sophisticated attacks** or **legitimate but complex sessions**.
- **resp_ip_bytes(Score:0.4082)**
  This is the number of **bytes sent by the responder** (server or target).
    - Higher values may indicate legitimate data transfer, while lower values suggest quick denials or responses in scan-like behaviours.
- **resp_bytes(Score:0.4075)**
  Similar to resp_ip_bytes, this represents the total bytes exchanged by the responder.
    - The feature holds moderate importance, as malicious sessions often have significantly fewer bytes transferred.
- **proto_tcp(Score:0.4004)**
  This specifies the **TCP protocol** as the communication medium.
    - While nearly all network traffic relies on TCP, malicious behaviours like **SYN floods** or **TCP-based port scans** can be differentiated by examining TCP-specific attributes.

The analysis of variable importance shows that the past connection (history_S) and current connection state (conn_state_S0) are the most important variables with scores above 0.89. The meaning of some features is very informative because they describe an active normal or attack network behavior. Byte counts matter a little bit since the correlation is 0.57. This also contributes to model decisions though not as predictive (0.41) as the features related to the connection. It is therefore contextual and not an important indicator.

4.5 Feature Patterns in Misclassified Cases

Table 14: Misclassification pattern table

True Label	Predicted Label	Count	Feature	Misclassified Avg	Correct Class Avg	Difference
Benign	**Malicious Attack**	154	missed_bytes	0	0	0
			history_S	0	0	0
			conn_state_S0	0	0.0456	-0.0456
			orig_bytes	4.3976	0.9675	3.43
			conn_state_SF	1	0.3238	0.6762

Benign	**Malicious C&C-HeartBeat-FileDownload**	177	missed_bytes	0	0	0
			history_S	0	0	0
			conn_state_S0	0	0.0456	-0.0456
			orig_bytes	6.0472	0.9675	5.0797
			conn_state_SF	1	0.3238	0.6762
Malicious Attack	**Malicious C&C-HeartBeat-Attack**	77,730	missed_bytes	0	0	0
			history_S	0	0	0
			conn_state_S0	0	0	0
			orig_bytes	0	0.0334	-0.0334
			conn_state_SF	0	0.0149	-0.0149
Malicious C&C-HeartBeat-Attack	**Malicious Attack**	29,267	missed_bytes	0	0	0
			history_S	0	0	0
			conn_state_S0	0	0	0
			orig_bytes	0.0001	0.0153	-0.0153
			conn_state_SF	0	0	0
Malicious C&C-PartOfAHorizontalPortScan	**Malicious PartOfAHorizontalPortScan**	219,318	missed_bytes	0	0	0
			history_S	1	1	0
			conn_state_S0	1	1	0
			orig_bytes	0	0	0
			conn_state_SF	0	0	0
Malicious PartOfAHorizontalPortScan	**Malicious C&C-PartOfAHorizontalPortScan**	340,168	missed_bytes	0	0	0
			history_S	1	1	0
			conn_state_S0	1	1	0
			orig_bytes	0	0	0
			conn_state_SF	0	0	0

Table 15: Detailed table

True Label	Predicted Label	Count
Benign	MaliciousAttack	154

Benign	MaliciousC&C-HeartBeat-FileDownload	177
Malicious Attack	Malicious C&C-HeartBeat-Attack	77,730
Malicious C&C-HeartBeat-Attack	Malicious Attack	29,267
Malicious-C&C-PartOfAHorizontalPortScan	Malicious PartOfAHorizontalPortScan	219,318
Malicious PartOfAHorizontalPortScan	Malicious C&C-PartOfAHorizontalPortScan	340,168

4.5.1 Port Scan Related

Table 16: Port Scan

Misclassification Type	Misclassified Cases
C&C-PartOfAHorizontalPortScan → PartOfAHorizontalPortScan	219,318
PartOfAHorizontalPortScan → C&C-PartOfAHorizontalPortScan	340,168

Root Cause: Both types show identical patterns in key features:

Table 17: Root cause

Feature	Value for Both Classes
history's	1
conn_state_S0	1
orig_bytes	0

One important classification error is between different port scans. The model confuses categories C&C-PartOfAHorizontalPortScan and PartOfAHorizontalPortScan among themselves in 219,318 and 340,168 cases respectively. The confusion arises from the same values for history_S, conn_state_S0 and orig_bytes for both these scans. The overlap indicates both port scans are similar and are performing essentially the same scan. Without any other indicators that could expose intent, such as behavior, there is no way of differentiating them.

4.5.2 Attack Related

Table 18: Attach class table

Misclassification Type	Misclassified Cases

Malicious Attack → C&C-HeartBeat-Attack	77,730
C&C-HeartBeat-Attack → Malicious Attack	29,267

Root Cause:

Table 19: Root cause

Feature	Value for Class 1	Value for Class 2
orig_bytes	0.0334	0.0153
conn_state_SF	0.0149	0

Moreover, there is confusion between Malicious Attack and C&C-HeartBeat-Attack due to 77,730 and 29,267 misclassifications in the respective cases. Because these attacks have similar values for features such as orig_bytes (0.0334 vs. 0.0153) and conn_state_SF (0.0149 vs. 0.0000), the model has trouble distinguishing them. The difference in type of each attack and not a measure of network activity shows a gap in the feature set. Perhaps a few additional context-sensitive metrics are in order.

4.5.3 Benign Misclassifications

Root cause: Benign traffic has distinct patterns:

Table 20: Root cause

Feature	Normal Benign Value	Misclassified Benign Value	Comparison to Malicious
orig_bytes	0.9675	5.2797	Much higher in benign
conn_state_SF	0.3238	1	Higher in benign

One positive point is that benign traffic is hardly misclassified as malicious. Only 154 samples are labeled Malicious Attack and 177 as C&C-HeartBeat-FileDownload. Misclassifications here are often due to atypical behavior in benign traffic. In particular, orig_bytes and conn_state_SF features spike like malicious samples. For instance, the orig_bytes value jumps from a normal benign average of 0.9675 to 5.2797 when misclassified. A few such false classification are encountered, and they show that the model is successful in minimizing false alerting which is a requirement to prevent unnecessary alerting in real-time application.

4.6 Root Causes of Misclassification:

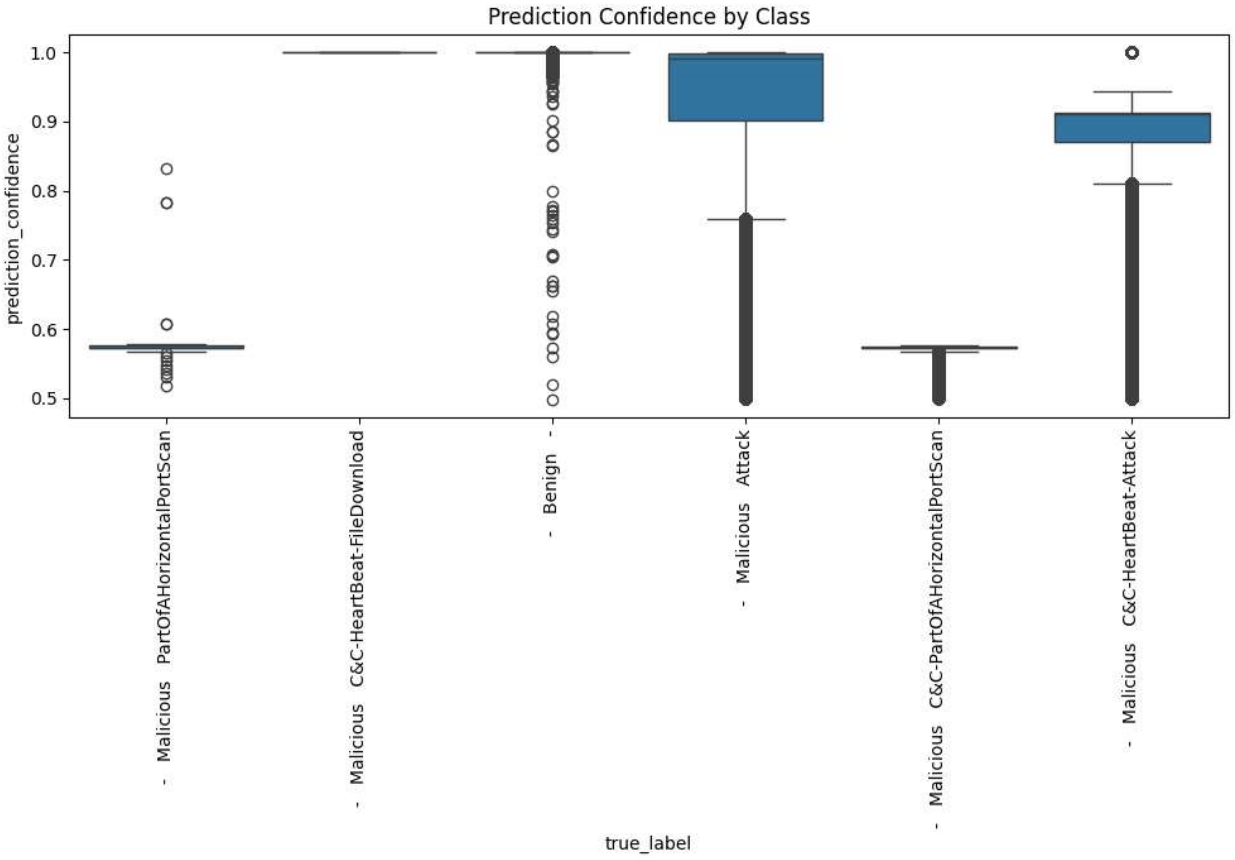

Fig. 15: Prediction confidence by class and correctness. Source: Own development

- Blue boxes show confidence for correct predictions
- Most correct predictions have high confidence (>0.9)
- Incorrect predictions generally show lower confidence (0.5-0.8)
- PortScan-related classifications show particularly high confidence, even when incorrect
- There are some outliers (circles) in both correct and incorrect prediction

1. Technical Similarity: Similar network behaviors despite different intents

2. Feature Limitations: Current features don't capture some crucial differences

3.Pattern Overlap: Some legitimate activities naturally resemble malicious patterns

4.6.1 Misclassification Insights

Overall, the model performs adequately, but there is still room for improvement in differentiating between technically similar but intention-driven differences, especially in port scan and attack classifications. Misclassifications don't happen for no reason – they happen because of certain patterns, usually because of overlaps in the features. Using the same network activities to measure them, instead of using signs that tell us what's going on, limits the model's ability to separate activities that look technically the same but have different intentions.

5. Conclusions

To summarize, the deep learning model shown exhibits strong overall performance, with strong metrics across different evaluative parameters. It attained a mean accuracy of 83.66% through 5-fold cross-validation coupled a ROC-AUC score of 0.9714 showing consistent reliability and stability. The analysis of per-class performance highlights model capability in correctly classifying benign traffic and some attacks like C&C-HeartBeat-FileDownload with perfect precision and recall. Apart from that, the set goal demonstrated a clear challenge problem of PortScan attack variants detection which footprint the scope of improvement.

The design of the model is a perfect balance between performance and complexity. It uses 18246 trainable parameters with a memory footprint of 725.55 MB. The CPU usage is 4.6% which is very less. These features make it ideal for real-time network monitoring applications to detect attacks, without burdening the system resources. Analysis of the training point out that the model learns quickly. You get to see the improvement in the initial epochs and great stabilization by epoch 25. Early stopping also effectively stops overfitting. Among other insights, the confusion matrix analysis shows the ability of the model to keep the false positive rate low with high precision benign traffic detection thus avoiding real-world false alarms. The C&C-HeartBeat detection accuracy is quite good. However, the comparatively lower precision and recall rates of PortScan detection indicate the need for more specialized solutions, perhaps through hierarchical or ensemble methods.

The measures that indicate operational efficiency show that the model can scale with the deployment on less-powerful systems with consistent inference times and low resource usage.

5.1 Setup

The experiment was conducted on a Windows 11 64-bit operating system, Intel® Core™ i5-9300H CPU @ 2.40GHZ, 16GB of RAM. The graphics card was NVIDIA GeForce RTX 2060 6GB. The algorithms implemented were from Python 3.8 64-bit, data loading was possible due to ‘Pandas’ library. Models and metrics were implemented by ‘Sklearn’ library.

5.2 Project Limitations

During my project, I face many limitations that hamper the outcome of my project. A major problem was the large class imbalances in the dataset, which could cause the model to act erratically and not classify the various types of traffic properly. The high volume of data I was working with was also troublesome, particularly with the low computational capacity I had. It took a long time to prepare the data and train the model. This has affected the experimentation. It was difficult to iterate quickly on the models. Because I was not well versed with machine learning

concepts, I could not dive into more advanced techniques, which could have contributed to better model performance. As a result of this, I had to face hurdle after hurdle throughout the project, impacting what I was able to analyze, and the insights that came out of it.

Bibliography

[1] MLA Style: World Economic Forum. "Threats to IoT Devices Are Constantly Evolving, but Is Security Keeping Up?" World Economic Forum, 2021,

www.weforum.org/stories/2021/08/threats-to-iot-devices-are-constantly-evolving-but-is-security-keeping-up/.

[2] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357. Available at: https://www.jair.org/index.php/jair/article/view/10302.

[3] Scikit-learn Developers. (2024). Cross-Validation: Model Evaluation. Scikit-learn Documentation. Available at: https://scikit-learn.org/stable/modules/cross_validation.html.

[4] Keras Developers. (2024). Early Stopping Callback. Keras Documentation. Available at: https://keras.io/api/callbacks/early_stopping/.

[5] Denning, D. E. (1986). *An Intrusion-Detection Model*. IEEE Transactions on Software Engineering. Available at: Computer Science Department - CSU

[6] ChaosGenius. (2024). *Anomaly Detection History: Techniques, Tools, and Use Cases*. Available at: ChaosGenius

[7] Stratosphere IPS. (2020). *Aposemat IoT-23: A Labeled Dataset with Malicious and Benign IoT Network Traffic*. Stratosphere IPS. Available at: https://www.stratosphereips.org/datasets-iot23

[8] Stratosphere IPS. (2020). *IoT-23 Dataset: A Labeled Dataset of Malware and Benign IoT Traffic*. Stratosphere IPS. Available at: https://www.stratosphereips.org/blog/tag/IoT-23Dataset

[9] Stratosphere IPS. (2020). *Aposemat IoT-23: A Labeled Dataset with Malicious and Benign IoT Network Traffic*. Available at: https://www.stratosphereips.org

[10] McCulloch, W. S., & Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics.

[11] Nauna, K. "Week 7B: Neural Network." *Portland State University*, n.d., https://web.pdx.edu/~nauna/week7b-neuralnetwork.pdf.

[12] Vardhaman Institute of Technology. "Neural Networks." 2021, https://vardhaman.org/wp-content/uploads/2021/03/Neural-Networks.pdf.

[13] Khan, M. S., et al. "Anomaly-Based Intrusion Detection Systems: A Survey." *ResearchGate*, 2023, https://www.researchgate.net/publication/358334600_Anomaly-Based_Intrusion_Detection_Systems_A_Survey.

[14] Moustafa, N., et al. "A Survey of Machine Learning Techniques for Intrusion Detection Systems." *ScienceDirect*, 2020, https://www.sciencedirect.com/science/article/pii/S1877056819300271.

[15] Liu, J., et al. "Machine Learning Algorithms for Intrusion Detection: A Review." *IEEE Access*, vol. 8, 2020, pp. 12340-12356. https://ieeexplore.ieee.org/document/9059701.

[16] "Neural Networks in Intrusion Detection Systems." *Vardhaman College of Engineering*, 2021, https://vardhaman.org/wp-content/uploads/2021/03/Neural-Networks.pdf.

[17] "Machine Learning in Intrusion Detection Systems." *Portland State University*, 2020, https://web.pdx.edu/~nauna/week7b-neuralnetwork.pdf.

[19] Amoretti, M., et al. "Reinforcement Learning Techniques in Intrusion Detection Systems: A Survey." *SpringerLink*, 2022, https://link.springer.com/article/10.1007/s11042-022-12852-2.

[20] Gao, J., et al. "Reinforcement Learning for Anomaly Detection in Intrusion Detection Systems." *MDPI Sensors*, 2021, https://www.mdpi.com/1424-8220/21/7/2311.

[21] He, X., et al. "Ensemble Learning Techniques for Intrusion Detection Systems." *ACM Computing Surveys*, vol. 52, no. 5, 2019, pp. 1-35. https://dl.acm.org/doi/abs/10.1145/3318139.

[22] Mishra, D., et al. "Security Challenges in the Internet of Things: A Survey." *IEEE Internet of Things Journal*, vol. 8, no. 4, 2021, pp. 2969-2980. https://ieeexplore.ieee.org/document/9312465.

[23] N. Kumar, et al. "Challenges and Solutions in Securing IoT Networks: A Survey." *SpringerLink*, 2021, https://link.springer.com/chapter/10.1007/978-3-030-41518-3_14.

[24] Elber Libombo, “Using Machine Learning to detect and divert bad network traffic to a honeypot” Cardiff University