Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Tormod Høyland Ulfeng
Sebastian Sneve

Investigating AI-Based IDS Trained on Public Datasets

Bacheloroppgave i Digital infrastruktur og cybersikkerhet
Veileder: Olav Alseth Skundberg
Mai 2024

NTNU
Kunnskap for en bedre verden

Tormod Høyland Ulfeng
Sebastian Sneve

Investigating AI-Based IDS Trained on Public Datasets

Bacheloroppgave i Digital infrastruktur og cybersikkerhet
Veileder: Olav Alseth Skundberg
Mai 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Investigating AI-Based IDS Trained on Public Datasets

Tormod Høyland Ulfeng
Sebastian Sneve

May 16, 2024

Abstract

The successful integration of AI into network IDS has the potential to strengthen network security. By improving the accuracy of threat detection and reducing response times, AI-powered systems can better protect sensitive data, critical infrastructure, and national security interests.

Our research investigates the effectiveness of an AI-based IDS trained on publicly available data when tested in a simulated network scenario. This allows us to assess the capabilities and limitations of AI-based network security technologies.

The results gives insight into the significant challenges of using a publicly available dataset for training an AI that is intended for use in a real world scenario. Furthermore, the results give directions for further research related to AI-based IDS.

Sammendrag

Integrasjon av kunstig intelligens i IDS-løsninger har potensial til å forbedre sikkerheten i ethvert datanettverk. Ved å øke presisjonen av trusseldeteksjon og redusere responstid, kan AI-drevne IDS løsninger bidra til bedre beskyttelsen av sensitive data, kritisk infrastruktur og andre nasjonale sikkerhetsinteresser.

Denne oppgaven undersøker hvor effektiv en AI-basert IDS trent på et offentlig tilgjengelig datasett er når den testes i et simulert nettverksscenario. Dette muliggjør vurdering av styrkene og svakhetene til AI-basert IDS.

Våre resultater avdekker betydelige utfordringer ved å bruke et offentlig tilgjengelig datasett for å trene en AI som skal brukes i et virkelig scenario. Videre gir resultatene retning for videre forskning relatert til AI-basert IDS.

Preface

This project serves as the final thesis for the Bachelor´s Degree in Digital Infrastructure and Cybersecurity at The Norwegian University of Science and Technology (NTNU).

The project investigates the viability of using publicly available data to train an AI-based IDS-solution for network applications. It explores whether public network data can be used to produce an efficient AI, or whether it is necessary to go the route of system specific network data.

Research on this topic is motivated by the state of the cybersecurity landscape, as well as the recent breakthroughs in artificial intelligence. During the course of our studies AI has been made available to the general public by companies such as OpenAI, Anthropic and Google. This has made the topic of machine learning even more relevant, as companies in all sectors are investing heavily in artificial intelligence. As threat actors within the cyberspace starts utilizing the artificial intelligence, it becomes increasingly important to remain focused on security. In order to combat the growing number of threat actors, organisations will likely need to employ artificial intelligence themselves in their security mechanisms; one of them being intrusion detection systems.

Our aim is that the research and experiments conducted in this project can help organisations make an informed decision of how to use AI as a part of their network defence mechanisms, especially when it comes to choosing an IDS solution fit for their needs. Hopefully, our findings and conclusions can help organisations avoid some of the pitfalls of artificial intelligence.

We would like to thank Kongsberg Defence & Aerospace for providing the topic of research, as well as their continued feedback and guidance on the project. We would also like to thank the Faculty of Information Technology and Electrical Engineering at NTNU, more specifically the Department of Computer Science, for the teachings and support over the past three years. Finally, a special thanks to our advisor at the department, Olav A. Skundberg. Your advice has been much appreciated.

Contents

List of Figures

List of Tables

Code Listings

1 Introduction

1.1 Problem Area

The sphere of cybersecurity is ever-changing. Due to new and more advanced threats traditional IDS solutions are getting outdated. To solve this problem an increasing number of organisations are looking for novel ways of identifying network intrusions. One promising solution is AI-based IDS.

Using artificial intelligence to sort network traffic has the potential to improve network security. Developing an artificial intelligence for this requires data, and in this case it requires network data.

Despite the data requirement, companies are hesitant about sharing their network data for the development of AI systems. This is due to privacy and confidentiality concerns. The result of this hesitation is that system specific data is difficult to obtain. This makes it difficult for outsiders to train models on data that is similar to that of which it can expect to process when deployed in production.

Therefore, when researching AI-based IDS solutions without access to internal datasets, researchers must resolve to using public datasets. There exists publicly available datasets online that can be used for this purpose. The problem with this approach is that the training data may not be perfectly suited for the task the model is being built for. This problem raises an interesting question for investigation:

- How well does an AI-based IDS trained on publicly available data perform when tested in a real-world scenario?

In order to weigh in on the topic with an informed answer we have developed two machine learning models using a public dataset. Furthermore we have tested them against a test dataset of our own creation consisting of typical network attacks, meant to simulate a real-world scenario. The method and findings of this research is put forward in this paper.

1.2 Partner Organization

This research project was conducted in collaboration with Kongsberg Defence & Aerospace (KDA). The general research topic was suggested by KDA, however no requirements nor limitations in terms of method or technology was put in place by KDA.

Being a supplier of defence products and related communications systems, KDA's business activities are clearly linked with cybersecurity and security in network communications. IDS is an essential tool for securing networks, thus KDA is motivated to constantly explore new methods of improving such systems.

Due to confidentially concerns no data from KDA was used during the research; not during training, validation or testing of the developed machine learning models.

1.3 Scope and Limitations

The project goal was to determine if it is worthwhile to train machine learning models, intended for network intrusion detection, on publicly available datasets. The reason we have researched this topic is due to the challenges of gaining access to datasets from real world networks.

For our purposes the publicly available LuFlow dataset has been used. Our training data is therefore limited to the content of the LuFlow dataset.

Regarding model development we have limited our research to two types of machine learning algorithms: Random Forest and Convolutional Neural Network.

In terms of testing we have created a custom testing dataset in order to verify or refute the efficiency of our model. The content of our custom dataset has been limited to three common network attacks, as well as benign network data.

1.4 Thesis Outline

This research paper is divided into the following chapters:

1. **Introduction** provides the background for this research project. It describes the problem area and defines the relevant research question.

2. **Theory** describes the theoretical background for our research area and method. This includes definitions of terms and technologies which are important for understanding our methodology and results.

3. **Method** details how our experimentation was carried out. It explains how we have used the technologies defined in the theory chapter to develop and test the machine learning models.

4. **Results** presents and explains the statistical findings from our testing.

5. **Discussion** answers the research question, based on the presented results. The findings are also discussed in a broader context. An evaluation of our methodology is conducted and avenues for further research is presented.

6. **Conclusion** summarizes our findings.

2 Theory

The theoretical framework of our project is presented in this chapter. Firstly, the main concepts relevant to our research are introduced. Then, given the importance of intrusion detection systems (IDS) and artificial intelligence (AI) to our research, these terms are introduced in more detail. Finally, the technologies used for AI development and testing are presented.

2.1 Main Concepts

2.1.1 Computer Network

Within network science, a computer network can be defined as two or more computers that are connected with one another for the purpose of communicating data electronically [1]. Networks vary in size from local, fully integrated networks to global ones, such as the internet. In today's modern world networks surround us in all aspects of life. Security is integral to the efficient use of networks.

2.1.2 Computer Network Defence Systems

For a network to operate reliably and for its communication to be trusted, network defence is essential. In the context of this thesis, the term "defence systems" refers specifically to computer network defence. Such defence systems can be defined as systems that protect, detect, and react to unauthorized activity within a computer network [2]. This includes implementing intrusion detection systems (IDS) [3].

2.1.3 CIA triad

The CIA triad defines the desired attributes of network data. It consists of the three attributes [4][5]:

- Confidentiality: The information remains undisclosed to those without authorization.
- Integrity: The information remains unaltered, either accidentally or by unauthorized individuals.

- Availability: The information is readily available to authorized individuals when needed.

All network defence systems should strive to uphold these attributes for network data.

2.2 Intrusion Detection System

Intrusion Detection Systems (IDS) are the focus of this project. To define intrusion detection systems, we first need to define what a network intrusion is. An intrusion is a set of actions that threaten the fundamental security attributes of data in a network.

An intrusion detection system can therefore be defined as a system that identifies threats and malicious activity in a network, and then issues appropriate alerts.

It is important to note that IDS are not intended to resolve network attacks. Their function is limited to monitoring network traffic and informing network administrators about the current state of the network [6].

2.2.1 History of IDS

The concept of IDS was first mentioned in 1986, in the academic paper "An Intrusion-Detection Model" [7] written Doreothy E. Denning [8]. Since then, intrusion detection systems have evolved. Today, IDS is a crucial part of network defence systems. The traditional type, which is not based on AI, is often referred to as signature-based IDS.

2.2.2 Signature-based IDS

Signature-based IDS is the traditional type of intrusion detection systems. Signature-based IDS works by looking for known signatures in the data of incoming traffic, and classifying the data accordingly.

This well-established method of sorting for malicious network traffic has certain limitations: it requires maintenance of the rules defining the signatures to look out for, thus new types of network attacks might not be covered by the manually maintained set of rules. This is one drawback that AI/ML-based systems aim to rectify.

2.2.3 Network Intrusion Detection Systems (NIDS)

IDS systems can be classified into two main categories, NIDS and HIDS. Network IDS, or NIDS, are distributed within computer networks and inspect the network traffic traversing the device with the system running. NIDS can be be both software and hardware based [9].

NIDS is the type of intrusion detection system that is relevant to our research, as it is focused on network security.

2.2.4 Host-Based Intrusion Detection Systems (HIDS)

The other main category of IDS is Host-Based IDS, or HIDS. These are end-point based detection systems, focused on monitoring activities happening within the host itself. Such systems analyses the operating system, applications, and network connections. A HIDS typically looks out for computing events such as file permission alterations and other types of malicious system requests [9].

Since HIDS are not network focused, these systems are not researched or evaluated in this paper.

2.2.5 Modern Challenges of IDS

The ever-changing state of modern technology has introduced challenges for traditional IDS solutions.

Firstly, rule-based IDS requires pre-defined rule sets to function. These sets of network rules require frequent updates by network experts. The combination of a static rulebase and ever-changing technology is a security weakness, since novel attacks will not be covered by the rule sets.

Another significant challenge arises from the proliferation of devices connected to networks. Modern Intrusion Detection Systems must efficiently handle vast amounts of network traffic. Mitigating false positives becomes crucial in this context. To illustrate, consider an IDS with a false positive rate of 0.001. In a scenario where it analyzes 100 packets per day, it would report one potential malicious packet every ten days. However, in a larger network scenario where the IDS receives 1 million packets per minute, it would generate 1000 false positives. Consequently, network administrators would face 1000 alerts per minute [10]. Therefore, contemporary IDS solutions must meet the demanding standards of both accuracy and scalability required in real production environments.

The application of AI in IDS is anticipated to resolve these limitations of traditional IDS. In theory, an AI-based system would be able recognise novel network patterns and retrain itself, which would allow for less manual maintenance and increased precision, limiting the amount of false positive classifications.

2.3 Artificial Intelligence (AI)

Artificial Intelligence is defined as the field within computer science that aims to develop machines that are capable of behaving intelligently [11]. AI enables computers and machines to simulate human intelligence and problem-solving capabilities [12].

AI is a broad field with many subfields. Therefore, it is important to highlight some of the different types of AI.

2.3.1 Narrow AI

Narrow AI, also referred to as weak AI, focuses on performing a specific task, such as classifying network traffic. Narrow AI relies on humans to decide its parameters and aid its learning. A narrow AI cannot complete tasks it has not been trained for [13].

Nevertheless, narrow AIs can become extremely efficient at completing certain tasks quickly. The AI application in this paper is strictly narrow.

2.3.2 General AI

General AI, also referred to as strong AI, is a theoretical form of AI, where the computer gains the ability to learn new topics by itself, thus being able to reach broad intelligence equal to that of humans, or even surpassing that of humans, gaining self-awareness and consciousness [13].

This level of AI is still only theoretical. Even advanced AI systems like OpenAI's ChapGPT-4 can only carry out the tasks its programmers have made it capable of. The distinction between narrow and strong AI is important, and this project is only concerned with narrow AI in its development and discussion.

2.3.3 AI in IDS

By utilizing artificial intelligence in IDS researchers and developers aim to replace the manually maintained rule-set with the classification capabilities of machine learning. By doing so the systems will no longer require rule-set maintenance by network specialist. Furthermore, the detection systems will be capable of identifying potential threats stemming from new types of attacks. This is due to the fact the system is no longer signature-based, but rather its based upon pattern recognition by the computer, looking for both known and unknown patterns in network data.

Developing an algorithm with the described capabilities necessitates the use of machine learning.

2.4 Machine Learning (ML)

The terms artificial intelligence and machine learning are often viewed as interchangeable, but its more accurate to view machine learning as the way of developing an artificial intelligence. IBM defines machine learning as a branch of artificial intelligence that focuses on the using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy [14].

In other words, machine learning is the process of which a computer system learns to process and classify data in a way similar to that of humans. The benefit of computers emulating this skill is their ability to look for patterns in enormous dataset, too large for humans to process, and learn from these.

When a developer wishes to use machine learning to solve their problems they have to make some decisions on method as not all AI is created equal. Machine learning comes in several forms, as outline below.

2.4.1 Supervised Learning

One main subcategory of machine learning is called supervised learning, and involves using labeled datasets to train algorithms to classify new data. This form of machine learning requires the training data to be labelled. As the training dataset consists of both the inputs and the correct label, the model can learn which data features that should weigh positively and negatively towards a certain classification.

It is this form of learning that is applied in this paper's research.

2.4.2 Unsupervised Learning

The other main subcategory of machine learning is unsupervised learning. This type involves unlabeled data. Since the data is unlabeled, a model developed using unsupervised learning is unfit for classification problems, but powerful at discovering hidden patterns in data without the need for human intervention. These patterns can then be used to cluster data into groups. Clustering the data can allow for sorting and analysis.

While supervised learning has the goal of predicting outcome of data, unsupervised learning aims to get insights from large volumes of data. Which type to use depends on the dataset at hand and the desired outcome [15].

Due to the nature of the research and data used of this paper, unsupervised learning is not explored.

2.4.3 Machine Learning Algorithms

Machine learning algorithms can be defined as computational techniques that enable computers to learn patterns and relationships from data without being explicitly programmed. In other words, these algorithms allow a programmer to feed it training data, and with little to no further programming, produce a model capable of interpreting data. These algorithms use mathematical and statistical techniques to identify patterns and make predictions or decisions based on the data input.

Many such algorithms have been developed. They vary in method and complexity, and choosing the right algorithm is an important step of machine learning. The ideal algorithm depends on the intended application and use.

For the research of this paper two types of machine learning algorithms have been used, see Section 2.4.5 and 2.4.6.

2.4.4 Neural Networks and Deep Learning

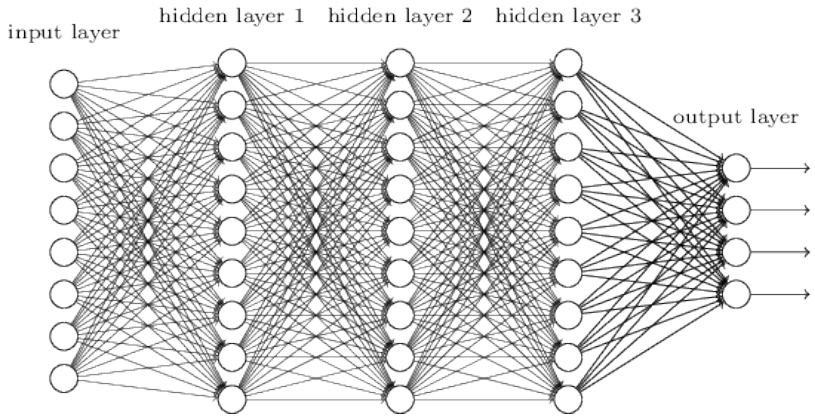
Two other terms often used when discussing machine learning are neural networks and deep learning.

Neural network refers to the type of machine learning algorithms that models artificial neurons, arranged in layers, interconnected by weights. All neural networks consist of an input layer, that takes the raw input; a hidden layer, that processes the data; and a output layer, that ultimately classify the inputted data.

Deep Learning refers to machine learning of this kind, but with many layers. Due to the plurality of layers in its architecture it is viewed as a "deep" network of neurons, thus its called deep learning.

Figure 1 illustrates the inter-connectivity between layers. The figure shows a model with eight input parameters, that classifies data into four classes, illustrated with the four output neurons in the output layer.

Figure 1: Layout of a Typical Deep Neural Network [16]



For each layer of the model the neurons either activate or not, based on the input coming from the previous layer. The machine learning involves tuning all these neurons to activate when appropriate in order to produce the desired output. The addition of more layers (a "deeper" model) allows the model to build multiple layers of abstraction. This gives the model the ability to learn from and recognize more complex features of the data, however this comes at a computational cost so the amount of layers should always be carefully considered.

2.4.5 Convolutional Neural Network (CNN)

Convolutional Neural Network is a type of deep learning algorithm. CNNs architecture is similar to that of Figure 1, yet it does also incorporate pre-processing of the input data.

This pre-processing step of filtering is what gives it the name convolutional. In this context convolution refers to applying a filter to the input. This filter reduces the amount of input parameters, but preserves the complexity of the

input, as well as the position of the input data parameters in relation to each other.

This step makes CNN incredibly powerful at image recognition, as it preserves information about previous pixels and their spatial relationships. Furthermore, CNN has proved effective in other applications as well, for example in classification of network log data [17] [18].

Given the promising results reported by other researchers in using CNNs for network traffic classification, we have selected it as one of the two machine learning algorithms for our research. For a detailed description of our CNN model, see Section 3.3.

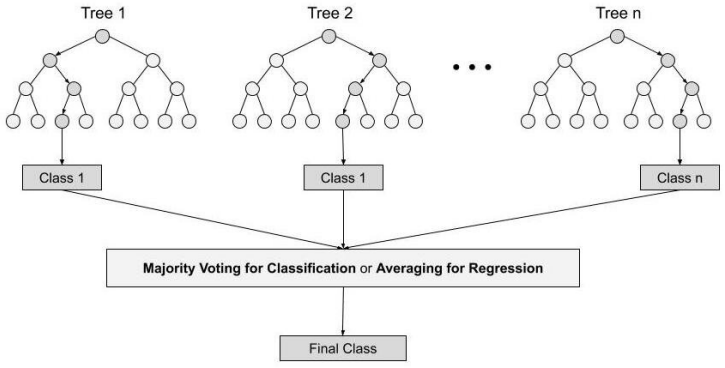
2.4.6 Random Forest (RF)

The other machine learning algorithm chosen for this project is called Random Forest. This model does not rely on a neural network, but combines the output of multiple decision trees to reach a classification of the input.

Random Forest algorithms are not viewed as a form of neural network, but instead is classified as a sub-type of ensemble learning; that is combining multiple models, in this is case combining multiple decisions trees.

A Random Forest consist of many decisions trees, and whereas a neural network is made up of neurons, decisions trees are made up of nodes. Each tree start at one node, and then branches into two or more nodes, with its own branches. The resulting schematic resembles a tree, as illustrated in Figure 2.

Figure 2: Random Forest Illustration [19]



At each node of the tree, a decision is made based on the value of a feature. The algorithm selects the best feature to split the data at each node. This

splitting process can be repeated until a certain criteria is met. The tree then terminates at a terminal node that represents a final classification. In layman terms, a decision tree is a series of questions regarding the ingested data, finally reaching a conclusion of what the data is.

The idea behind a Random Forest is that a series of these trees, independent of each other but each with one vote, will vote forward the correct classification. It is important to note that when training a RF model the trees are not built equally. The trees are grown from randomly selected features of the data. This randomness give the algorithm its name.

2.5 Dataset

Another crucial component in machine learning and artificial intelligence is data. In this context, dataset refers to a collection of data that is used to train and test algorithms and models [20]. The datasets are essential in AI because they form the basis for training, evaluation, and improvement of machine learning models. The better the dataset is, the more likely the model is to perform well in addressing real-world problems.

Due to the importance of the dataset in AI development, the choice of data used in training is crucial in order to produce an efficient AI. A common issue is simply the lack of available datasets, evidenced by the motivation behind this research project.

2.5.1 Challenge of Domain Specific Datasets

When developing a narrow AI with the goal of effectively solving real-world problems, a dataset containing data from that domain is preferred. This can be refereed to as a domain specific dataset.

The availability of such domain specific dataset cannot be guaranteed. Existing datasets can be kept private by organisations due to confidentiality and privacy concerns, and new datasets might be challenging to collect and label without time intensive manual work.

Because of this challenge, we have opted to investigate model performance when using of a public dataset for training. See 2.5.2 for a description of the public dataset chosen for this research project.

2.5.2 LuFlow Network Intrusion Detection Dataset

The LuFlow Network Intrusion Detection Dataset is a public dataset, available on Kaggle, consisting of flow-based network traffic captured from 2020 to 2022 [21]. The dataset was developed by Ryan Mills, a PhD candidate at Lancaster University. LuFlow was created with the intention of promoting research into detection mechanisms suitable for emerging network threats, thus its a suitable dataset for our intended purposes.

The total dataset is over 20 GB large, containing over 200 million network flows in the format of .CSV-files organised into months and years. These network flows, consisting of groups of network packets, have been captured using Cisco Joy, see Section 2.6.3. The network data has been captured at honeypots placed within Lancaster University's address space.

The labelling of this data is autonomous, and is supported by third party cyber threat intelligence sources. This process is explained in detail in Mill's dissertation describing how the dataset was generated [22]. The data entries of the set are labelled as benign, malicious or as a outlier. Due to its content the dataset is well-suited for AI development related to intrusion detection systems.

Furthermore, Mills' dissertation describes how destination port analysis was done on the captured data in order to identify what kinds of network attacks the honeypots captured. One of the ports that were extensively targeted were port 22, exclusively used for by the Secure Shell (SSH) Protocol.

Additionally, Mills conducted evaluation of the detection capabilities of a machine learning model he developed. The results showed that the algorithm could with a high degree of accuracy detect DDoS attacks, Port Scan attacks and Brute Force attacks in his own simulated environment. The aforementioned attacks are especially relevant for our thesis as they correspond with the types of network attacks we tested our models on. The fact that the LuFlow dataset contains and have been successfully tested for these types of attack, yet in a different virtual environment, makes the LuFlow dataset well-suited for our own application.

For more specific information on how we used the dataset in our research, see Chapter 3, which explain the methodology of our model development.

2.6 Technology

This project has utilized several technologies to carry out the required development, experimentation and testing. In this section those technologies are introduced on a theoretical level.

This section is split into three parts: AI development; outlining the technologies used to develop our models, Virtual Environment; describing the tools used to configure and run our development and testing environment, and Other Software; listing the software used to capture data and simulate network attacks.

2.6.1 AI Development

The following technologies have been utilized to develop the AI models in this project.

Python 3.11.8 Python is a programming language extensively used in AI development due its large ecosystem of libraries and frameworks. Python 3.11.8, released in February of 2024, has been used in this project.

TensorFlow TensorFlow is Python machine learning framework developed by Google. It is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms [23]. TensorFlow provides a flexible architecture that allows for both high-level model building with prebuilt layers and low-level customization for advanced users.

Keras The Keras framework is a high-level API which can use a number of backends, one of them being TensorFlow. It provides a user-friendly interface for building and training neural networks, hiding the lower level operations of TensorFlow.

Scikit-learn Also known as sklearn, Scikit-learn is an open-source machine learning library for Python. It provides simple and efficient tools for machine learning tasks such as classification. It is more focused on the classical algorithms, such as random forest trees and linear regression. Its easy-to-use API makes it an efficient library for developing such models.

2.6.2 Virtual Environment

The following technologies have been used to set up and run our development and testing environment.

OpenStack SkyHiGh Openstack is a virtualisation platform originally developed by Rackspace Hosting and NASA. The Norwegian University of Science and Technology (NTNU) uses OpenStack to serve students and staff with access to virtualization resources.

One of NTNU's computer installations are located in Gjøvik, Norway and is called SkyHiGh. It is maintained by The Department of Information Security and Communication Technology. Resources from SkyHiGh was used to develop and test the machine learning models of this project.

Kali Linux 2023.3 Kali Linux is a Linux distribution designed for digital forensics and penetration testing. The 2023.3 release of Kali Linux is the operating system installed on all the virtual machines in our OpenStack cloud. Kali comes pre-installed with a number of penetration testing tools fit for our purposes.

2.6.3 Other Software and Utilities

The following software has been used to conduct and log the network attacks.

Cisco Joy Developed by Cisco, Joy is an open-source software package for collecting and analyzing network data. It reads raw network traffic and outputs a JSON file, suitable for further processing.

It captures network data in flows, defined as a set of set of network packets with common characteristics. The grouping of packets depends on source, destination, and timing. Joy can also capture bidirectional flows which consists of a pair of unidirectional flows whose source and destination addresses, as well as port number, are reversed, and whose time spans overlap [24].

It is the output from Cisco Joy that is processed and used to train and evaluate the models described in this paper. Joy was used to log the data in the LuFlow dataset and we have used it to generate our custom testing dataset. See 3.4 for more details on how we used Cisco Joy.

Nmap Nmap is an open source tool for network exploration and security auditing [25]. It is a software tool that can used for scanning the network ports in a network or on a single host. See Section 3.4.2 for our use of the tool.

Hydra Developed by Marc "van Hauser" Heuse, Hydra is password hacking tool. It supports attacking of numerous protocols, one of them being SSH. This tool can be used to carry out a SSH Brute Force attack, as described in Section 3.4.2.

MHDDoS MHDDoS is a Distributed Denial of Service attack script available on GitHub. It includes the functionally of many types of DDoS attacks, including TCP Flood, UDP Flood and many others. For details on how MHDDoS was used in this project, see Section 3.4.2.

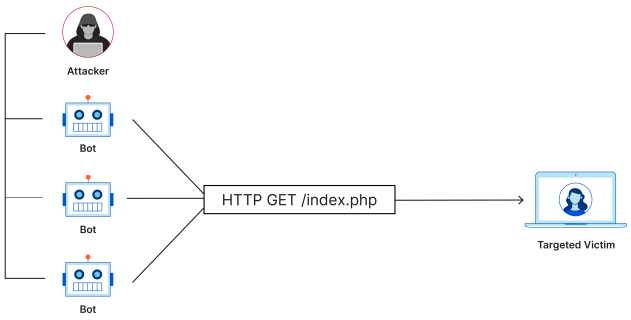
2.7 Cyber Attacks

Cyber attacks are becoming increasingly prevalent and they come in many forms. The three types of cyber attacks used in our research are introduced in this section.

2.7.1 Distributed Denial of Service (DDoS)

A Distributed Denial of Service (DDoS) attack affects a target server by sending copious amounts of request, eventually overwhelming the service and denying others access to the network resource. Such attacks are often carried out by a network of many computers, thus its refereed to as a distributed attack. The process is illustrated in the figure below.

Figure 3: Illustration of DDoS Attack[26]



As Figure 3 shows, one attackers uses a multiple computers, or botnet, to sends network requests to a target server. By utilising a botnet the attacker is able to distribute the attack and multiply the amount of server request. Eventually the targeted victim can no longer process the stream of request and the service starts to slow down. A sustained DDoS attack might result in the service shutting down completely [27]. Such an attack affects the availability-attribute of the CIA triad. If data access is crucial, a DDoS attack can cause significant damage.

2.7.2 Brute Force

A brute force attack attempts to guess a password or key by automated trial and error. The most known type of brute force attacks are dictionary attacks [28], where a list of possible passwords are tried using a script. If the target employs poor password routines brute force attacks can be effective at gaining unauthorized access to computer systems.

Using brute forcing techniques to gain SSH protocol access is a common network attack. If successful, an unauthorised SSH login can compromise the confidentiality and integrity of the computer data.

2.7.3 Port Scan

A port scan scans one or more of the target's networks port, in an attempt reveal information regarding what services are running on a server. The information gathered from a port scan can be used to uncover vulnerabilities for exploitation [29].

A port scan is oftentimes used as a preliminary step in an network attack. Therefore, depending on the ensuing attack type, all the principles of the CIA-triad are at risk.

2.8 Machine Learning Evaluation

This subsection defines the different metrics used to evaluate the performance of machine learning classification models.

2.8.1 Accuracy

Accuracy measures the accuracy of both positive and negative predictions. It takes both true positives and true negatives, and divides it by the whole set. It describes how well the model performs across both classes of a binary classification problem. Accuracy is calculated as follows:

$$\text{Accuracy} = \frac{\text{True Pos.} + \text{True Neg.}}{\text{True Pos.} + \text{True Neg.} + \text{False Pos.} + \text{False Neg.}}$$

It is important to note that the accuracy metric might be misleading when dealing with imbalanced datasets, since it gives equal weight to the model's ability to predict both classes. For example, if a model is developed to identify pictures of dogs, but there are few dog pictures in the dataset the model might simply classify all pictures as not a dog and achieve a high accuracy. Yet, the model is now of little worth since its intended use is to classify dog pictures. Due to this, accuracy might be close to 1.00, but the model might not work for its intended use. This is why other metrics are needed as well, and this is where precision and recall comes in.

2.8.2 Precision

Precision measures the accuracy of positive predictions. In other words, it takes the true positives, and divides it by all the positive predictions of the model. This tell us how accurate the model is at being right, when it first decides to classify data as positive. If precision turns out be 1.00, it means there are no false positives. It is calculated as follows:

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

For example, if a model is developed to identify pictures of dogs, it may be very precise, meaning all the pictures it does classify as dog pictures is in fact dog pictures. Yet, this doesn't mean it doesn't miss some of the dog pictures in the data. Yet, lets say it does miss a few pictures, this does not affect precision since this counts as a false negative which is not included in the calculation. This is where recall becomes important.

2.8.3 Recall

Recall measures the ability of the model to correctly identify all the positive instances in the dataset. Since recall also take false negatives into account it tells us how proficient the model is at identifying all the positives in the dataset. If recall turns out to be 1.00, it means there are no false negatives. It is calculated as follows:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positives} + \text{False Negatives}}$$

Using the same dog classification example as in 2.8.1 and 2.8.2, if the model has a high recall score it means that the model has not missed any of the actual positives in the dataset. That is, the model has correctly classified all the dog pictures. Yet, the weakness of recall is that it does not take false positives into account. This means that if the model classifies the whole dataset as positive it will achieve a recall of 1.00, but the precision score would suffer as many would be false positives.

2.8.4 F1-score

Due to the inherent strengths and weaknesses of precision and recall, they should always be considered together. This is what F1-score does. This metric is the harmonic mean of precision and recall. This means that the F1-score takes both false positives and false negative into account. It is calculated as follows:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-score does also range from 0 to 1, where 1 indicates perfect performance of the model. A score of 1 means that is has no false positives or false negatives.

This means that all its positive identifications were actually correct and all actual positives were identified.

2.8.5 Support

Support is another metric often seen in the output of evaluation data is support. This represents the number of samples belonging to that class in the dataset. This number is important for understanding the distribution of classes in the dataset.

2.8.6 Confusion Matrix

A confusion matrix is a tabular representation of the classification performance of a machine learning model. Table 1 gives an example of how a confusion matrix is represented.

Table 1: Confusion Matrix Example

		Predicted Label	
		Positive	Negative
Actual Label	Positive	TP	FN
	Negative	FP	TN

As seen, the table compares the predicted labels to the actual labels of the dataset. The model therefore shows the number of true and false positives and negatives. These metrics forms the confusion matrix.

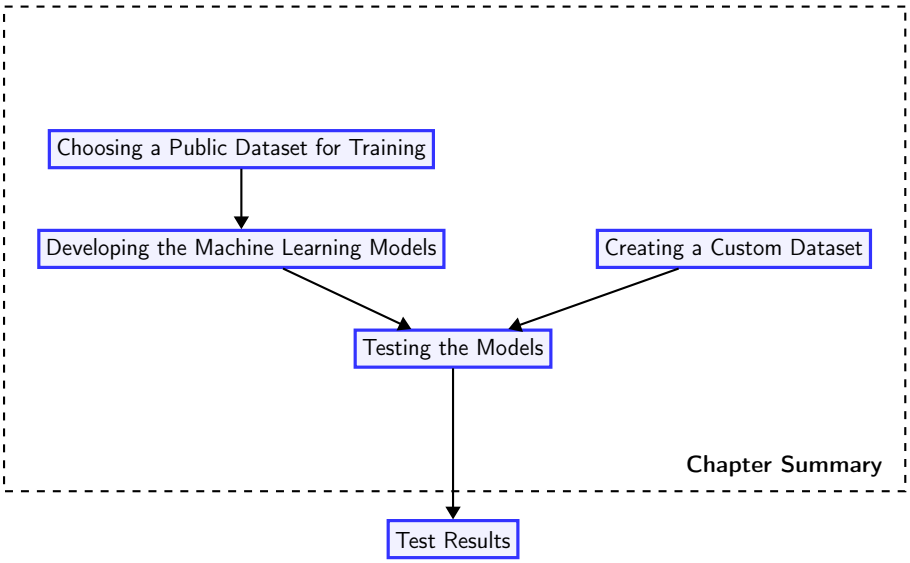
3 Method

This chapter describes the methodology of our research. It goes into depth about how we developed the machine learning models using the public dataset that we chose as training data. The chapter also details how we designed and carried out the required model testing in order to yield the test results that are presented in Chapter 4.

3.1 Chapter Outline

The figure below illustrates the key steps of our methodology. These steps are essential in our research. It is by completing these steps that we are able to give an informed opinion on the research question at hand.

Figure 4: General Overview of Method



This chapter is structured such that each key step has its own section. Figure 4 shows that there is a specific order of execution to our method, indicated by the arrows. The chapter sections follows this order of operation and is as follows:

1. **Choosing a Public Dataset** describes the reasons as to why the LuFlow dataset was chosen to serve as the data foundation to train our machine learning models. Other candidate datasets are also presented.

2. **Developing the Machine Learning Models** describes how we developed the CNN and RF models. This section goes into depth on procedure and choice of technology in all stages of development.

3. **Creating a Custom Dataset** explains how we constructed the custom dataset used to simulate a real-world network scenario. It details the creation of the virtual environment that was used. Furthermore, it describes how the network attacks were carried out and logged. The steps taken to prepare the dataset for classification by the models are also outlined.

4. **Testing the Models** explains how testing of the machine learning models was carried out.

5. **Chapter Summary** gives a short overview of the complete methodology. It merges the aforementioned sections and gives a complete understanding of our method.

6. **Test Results** are presented in chapter 4.

3.2 Choosing a Public Dataset

Having decided that for this research we would train our AI-based IDS prototypes on public data, we investigated different datasets for their applicability.

3.2.1 Contenders

Four datasets were seriously considered. The ones listed below were evaluated thoroughly, but in the end, not chosen due to the existence of a more suited dataset.

MAWILab is a database that assists researchers to evaluate their traffic anomaly detection methods [30]. The dataset is wast, containing data from 2007 to 2024, and its labelling mechanism is autonomous, using graph-based methodology that compares and combines different and independent anomaly detectors. The dataset consist of the data found in the MAWI archive, also known as Traffic Data Repository at The Wide Project [31].

In contrast to the other datasets introduced in this chapter, MAWILab is not available on Kaggle, but instead is available on its own web page. Furthermore,

the data is sorted into single days, making it more troublesome to concatenate a longer series of data. Also, the web traffic consists of ten data fields, many of which are difficult to reproduce in new network data. This difficulty makes the dataset unfit for our purpose, since we aim to assign the same data values to our own dataset, making it suitable for classification with our own models.

BETH was created in 2021 by four researchers across different institutions. It consists of data captured at 23 honeypots set up in an unnamed major cloud provider. It includes both kernel-level process calls and network logs, mainly DNS logs [32].

The BETH dataset is available on Kaggle, making it easy to access, yet it was not chosen due to the content of the dataset. For our research we are not concerned about kernel-level process logs, as this is out of scope. Furthermore, the DNS network logs that are present in the dataset is less applicable to our use case, since we have chosen to focus on network flows.

CIC-IDS2017 is a dataset built by The Canadian Institute for Cybersecurity. The data was captured in July of 2017 and consists of five days of simulated network attacks. The dataset contains 79 feature columns in total, where the final column indicates if traffic is benign or part of a specified network attacks [33].

The dataset contains a range of typical network attacks, yet a weakness of the dataset is that it is synthetically created, meaning it does not include real world data. Furthermore, the extensive feature set makes the dataset less readable and computationally heavier. And as with MAWILab, the range of features makes reproduction of these fields in a new dataset more challenging. Finally, the age of the data made it unfit for our purposes.

3.2.2 LuFlow

After having considered the available options the choice of dataset settled on the LuFlow Network Intrusion Detection Dataset. Introduced in section 2.5.2, the dataset's features is explained in more detail here.

To reiterate, the LuFlow Dataset, developed by PhD candidate Ryan Mills from Lancaster University, is a public dataset [22] available on Kaggle [34]. The network data collection spans from 2020 to 2022, thus its content is relatively new compared to the other alternatives. The dataset contains more than

206 million network flows stored in .CSV format, totaling over 20 gigabyte of data, organized by months and years. These network flows, captured using Cisco Joy, originate from honeypots within Lancaster University's network. The data is labeled autonomously with support from third-party cyber threat intelligence sources, categorizing entries as benign, malicious, or outliers.

Since the network telemetry has been captured using Cisco Joy, the data features of the dataset originate from the features Joy applies to the network flows. These data features are explained in Table 2.

Table 2: LUFlow Dataset Features

Feature Name	Description
src_ip	The source IP associated with the flow.
src_port	The source port number associated with the flow
dest_ip	The destination IP associated with the flow.
dest_port	The destination port number associated with the flow
proto	The protocol number associated with the flow. For example TCP is 6.
bytes_in	The number of bytes transmitted from source to destination.
bytes_out	The number of bytes transmitted from destination to source.
num_pkts_in	The packet count from source to destination.
num_pkts_out	The packet count from destination to source.
entropy	The entropy in bits per byte of the data fields within the flow.
total_entropy	The total entropy in bytes over all of the bytes in the data fields of the flow.
mean_ipt	The mean of the inter-packet arrival times of the flow.
time_start	The start time of the flow in seconds since the epoch.
time_end	The end time of the flow in seconds since the epoch.
duration	The flow duration time, with microsecond precision.
label	The label of the flow. Either benign, outlier, or malicious.

As Table 2 describes, the LuFlow's data features are relatively easy to understand. The data features are diverse and captures many details regarding

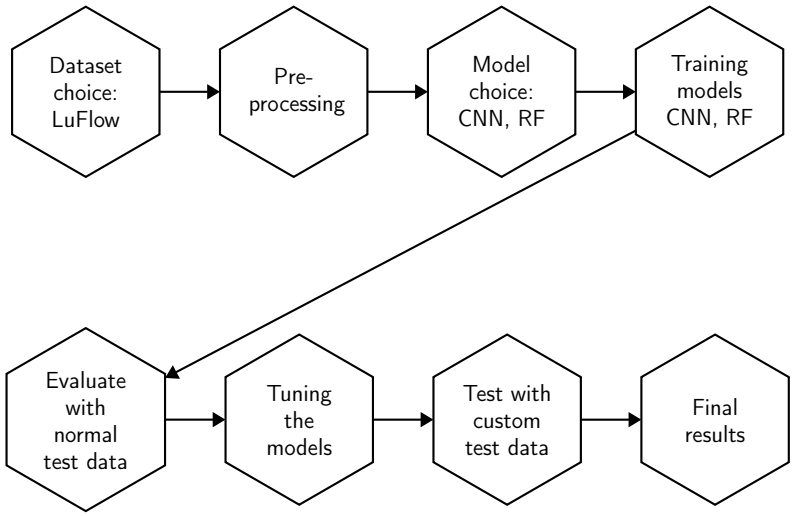
each data flow. Also, it is easy to assign the same features to new data flows using Cisco Joy. This is important for our work, since we will assign the same features to our custom dataset.

Due to LuFlow's content of real-world captured data, its ease of availability, and good documentation, it was chosen as the public dataset for our research. Our pre-processing and use of the dataset is detailed in Section 3.3.1.

3.3 Developing the Machine Learning Models

The steps required to develop the machine learning models are outlined in Figure 5. The first step is choosing the dataset that the models will be trained and evaluated on. As stated in Section 3.2 our chosen dataset is LuFlow.

Figure 5: Model Development Process



The next step is to pre-process the data so that it is suitable for model training. This involves removing columns that are irrelevant for the models. All the steps taken during data preparation are described in Section 3.3.1.

The third step is to choose the machine learning algorithms to use. CNN and RF models were chosen as they have showcased success in earlier research when used for AI-based IDS [17] [18].

Having chosen the machine learning algorithms, the next step is to train the models. This is a crucial step, because this where the models process the data and become able to make predictions based on the gained knowledge.

The specific model architectures are determined in this step. Detail regarding of the architectures is described in section 3.3.2. Another part of training and evaluation is parameter tuning. Tuning the parameters of the model architectures can improve the machine learning process and the resulting performance of the model.

When training is complete, model testing is necessary. It is essential to know whether the models perform well or not. Classification tests on data the models have not seen before reveals their efficacy. The tests will consist of classifying unseen test data from LuFlow and from our custom dataset. The custom dataset is described in Section 3.4, while details regarding the evaluation is found in Section 3.3.3.

The steps described above have been conducted on a virtual machine nicknamed "Compute" in the SkyHiGh cloud. Table 3 shows the specification, while the details of the SkyHiGh virtual environment setup is described section 3.4.1.

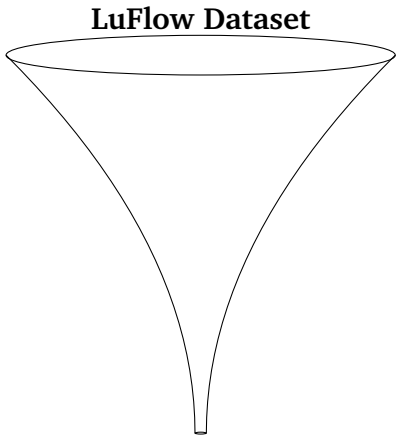
Table 3: Specification of Model Development Computer

Name	**OS**	**IPs**	**VCPUs**	**RAM**	**Storage**
Compute	Kali Linux 2023.3	192.168.0.61, 10.212.172.77	16	128GB	240GB

3.3.1 Data Pre-Processing

As mentioned, the LuFLow dataset contains more than 200 million network flows, that all have 16 data features. Training our ML models based on every flows with all features is not appropriate due resource limitations. Optimization of the dataset is possible by pre-processing. Figure 6 describes the basic concept of the pre-processing.

Figure 6: Illustration of Pre-Processing



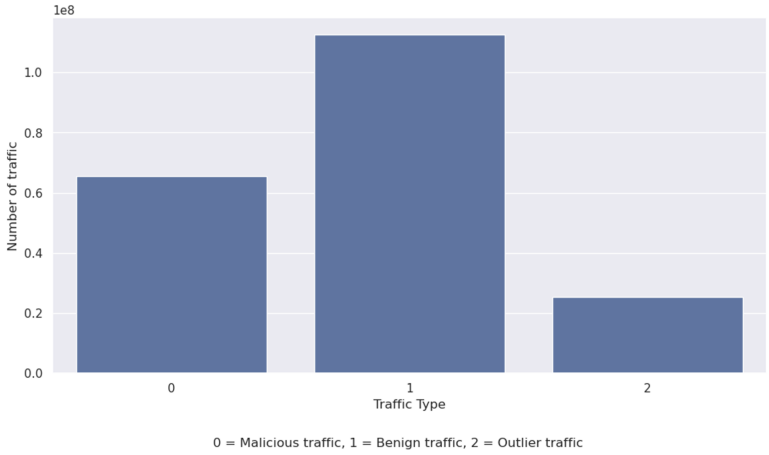
Several steps have been taken to sort and process the LuFlow dataset into a pre-processed dataset ready for model training. These steps are described below:

- **Removing duplicates.** In the LuFlow dataset, there were approximately 491 000 duplicates. These are removed as they do not contribute to model learning.
- **Removing the null values.** This step removes the network flows that are missing some data features. In total, over 2 700 000 rows were removed.
- **Removing columns "time_end" and "time_start".** These columns are unique for every row, since they describe the time for which the flow was recorded. Since this data feature is unique for each row the data feature does not contribute to the pattern recognition of the ML models.
- **Redefining the "label" column** The column "label" is the column that defines if the flow is categorized as malicious, benign or outlier. This features is redefined into an integer for future ease of coding. Table 4 and Figure 7 describes the distribution of the different labels.

Table 4: Label Distribution of LuFlow Dataset

Label	**Label (int)**	**Number of flows**
Malicious	0	65 391 995
Benign	1	112 611 684
Outlier	2	25 294 621

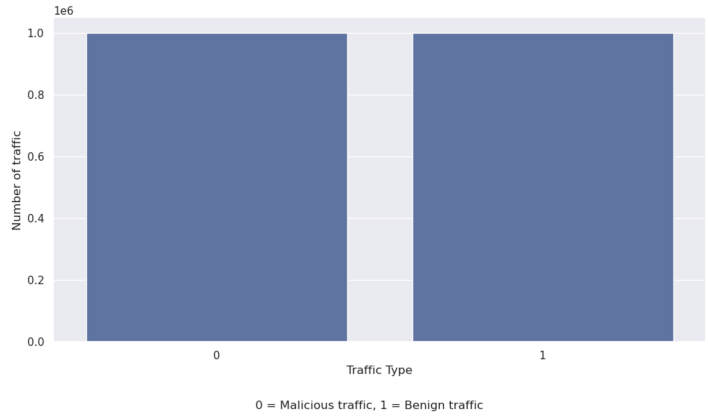
Figure 7: LuFlow Label Distribution Chart



- **Undersampling to a balanced dataset.** A balanced dataset is desirable for model training. By sampling the dataset such that there is an equal amount of benign and malicious network flows we make sure our training data is balanced. Given that we are developing models intended to solve a binary classification task, the outlier label is not of interest. This category is therefore removed from the dataset.

  Also, in order to decrease the computational requirements for training we have under-sampled our training dataset. One million flows each of benign and malicious network flows have been randomly sampled to make up our training dataset. Figure 8 shows the resulting dataset.

Figure 8: Balanced and Under-sampled Dataset



Due to the different characteristics and properties of the machine learning models, the remaining data pre-processing steps are different. These differences are described in the paragraphs below. Table 5 outlines the content of the final datasets used for training, validation and testing of the CNN and RF models.

Table 5: Balanced Datasets Splits

Model	**CNN**		**RF**	
	Benign	**Malicious**	**Benign**	**Malicious**
Full dataset	1,000,000	1,000,000	1,000,000	1,000,000
Training set	799,877	800,123	800,331	799,669
Validation set	100,068	99,932	N/A	N/A
Test set	100,055	99,945	199,669	200,331

CNN

The balanced dataset is divided into X_train, X_validation, X_test, Y_train, Y_validation and Y_test sets (80/10/10). These sets are then scaled using the StandardScaler found in the Sci-kit Learn library.

Random Forest

Dividing the balanced dataset for the RF model requires only splitting into train and test sets (80/20), as the RF model is validated using a different method.

Finally, the RF subsets are scaled using the MinMaxScaler in the Sci-kit Learn library

3.3.2 Model Architectures

This section describes the technical choices taken to code the machine learning models. Two types of machine learning models have been developed: one based on a Convolutional Neural Network (CNN) algorithm, and one based on the Random Forest (RF) algorithm. These models are described in section 2.4.5 and 2.4.6, respectively. The CNN model was developed due to its well-documented effectiveness in data classification. Additionally, the RF model was developed due its ease-of-development and for comparisons sake. The comparative results are described in Chapter 4.

CNN
When making a CNN model using TensorFlow and Keras, you need to define the architecture of the model before training. Table 6 describes the layers of our model.

Table 6: CNN Model Architecture

Layer (type)
Input
Conv1D
Batch Normalization
MaxPooling1D
Dropout
Conv1D
Batch Normalization
MaxPooling1D
Dropout
Flatten
Dense
Dropout
Dense
Dense
Output

In any neural network model, an input shape is required. This model has an input shape of 13, equaling the number of features our model will evaluate. Next, there are two iterative sequences of a Conv1D, Batch Normalization, MaxPooling1D and Dropout layers.

The Conv1D layers extracts the features from the sequence and applies a set of filters to the input [35]. In the first sequence there are 64 filters and in the second it is 128 filters. Following each Conv1D, the batch normalization layers normalizes the activations from the layer above: Rectified Linear Unit (ReLU). ReLU makes it possible for the model to learn complex functions, however the batch normalization makes this activation stable [36]. The max-pooling layer reduce the data. As described in section 2.4.5, this is what makes the model so powerful. The sequence goes from 13, to 7 and to 4. The dropout layer helps to prevent over-fitting by randomly changing some of the input to 0.

To be able to get the data to the fully connected layers called Dense layers, we need to add a layer called Flatten. It coverts the multi-dimensional output from the layers above into a single-dimensional array. This means that the the output shape becomes 4*128=512. Then there is three dense layers, with a dropout layer to give the same effect as before. The output shape in these layers changes from 512, to 128, to 64 and finally to 1. The dense layers are fully connected, meaning each input node is connected to a output node. The first two nodes uses the activation function ReLU, as before, but the final activation function is called Sigmoid. This then evaluates whether the traffic is malicious or benign, which is the binary classification.

When developing a CNN model you can tune the hyper parameters depending on the objectives of your model. Our chosen parameters are showed in Table 7.

Table 7: CNN Hyper Parameters

Parameters	**Value**
Optimizer	Adam
Loss Function	Binary Cross Entropy
Activation Function	ReLU
Final Activation Function	Sigmoid
Batch Size	64
No. of Epochs	50

The Adam optimizer is a known as a popular choice for deep learning mod-

els, while the loss function binary cross entropy is well suited for binary classification. The batch size is the number of samples that is fed into the network to train, while the epoch is the number of cycles it will go through the dataset. Our CNN model will be trained on 1 600 000 flows. When the batch size is 64, the training will require 25 000 iterations to achieve one epoch.

Random Forest

Our Random Forest machine learning model was developed using the Scikit-learn library for Python. It allows for fast development of Random Forest classifier models using the built-in class called RandomForestClassifier, which is part of the ensemble-method in the library.

As described in section 3.3.1, the beginning stage of creating the RF models involves using the Scikit-learn scaler called MinMax to scale the input data from 0 to 1. After scaling, the processed data is split into a training and testing datasets using a 80/20 split.

With the training dataset ready for use, the next required step is hyper parameter tuning. The class RandomForestClassifier accepts a series of parameters meant to tune the algorithm. The Scikit-learn API includes several optimization methods, one of which is called RandomizedSearchCV. This method does a randomized search on the specified hyper parameters. The number of parameter settings that are tried is decided by the programmer. We used this method to find the optimal parameter settings for the parameters listed in Table 8. Also illustrated in Table 8 is the optimal parameter setting identified by the optimizer. The optimizer was run a total of 20 times, optimizing the parameters for best possible model performance. These optimal parameters are then saved using the best_params_ method.

Table 8: Random Forest Model Parameters

Parameter Description	Tested Range	Optimal Parameter Setting
Number of trees in the forest	50 - 1000	805
Maximum depth of the trees	2 - 10	8
Min. number of samples required to split an internal node	2 - 20	17
Min. number of samples required to be at a leaf node	1 - 10	1
Number of features to consider when looking for the best split	sqrt, log2, None	None

Having identified the optimal parameter settings, showed in Table 8, we proceed by using the Scikit-learn class RandomForestClassifier to train the model, using the hyper parameters identified in the previous step.

The fit method of the RandomForestClassifier class is used to initiate training. Upon completion, the predict method is used to evaluate the resulting model on the test dataset. See Section 3.3.3 for the evaluation results.

3.3.3 Model Evaluation

Before we can proceed with testing the models against our custom dataset we have created, we want to make sure the development has been successful. This is done by seeing how well the models perform when tasked with classifying unseen data from the LuFlow dataset.

CNN

When evaluating the development of the CNN model both the validation set and tests set are used.

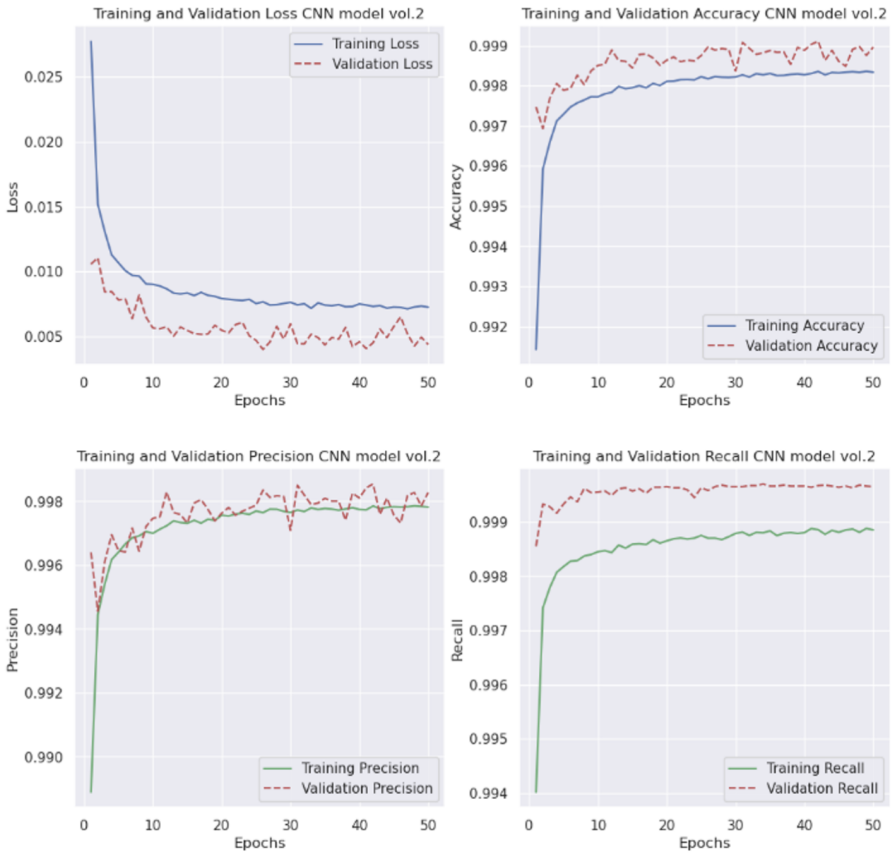
The TensorFlow evaluation method, model.evaluate, uses the test dataset to see how well the model performs when used to classify the test dataset. Table 9 shows the evaluation results. The loss is low, and accuracy, precision and recall are very close to 1.

Table 9: CNN Evaluation Results

Loss	Accuracy	Precision	Recall
0.005	0.998	0.998	0.999

The second evaluation method shows how the model training progressed over time. As illustrated in Table 5, we have split the dataset into training, validation and test sets. To evaluate how the training have gone over time, which in this case is the epochs, we inspected if the model is over-fitting or under-fitting. Figure 9 shows four graphs which visualises the training progression of the CNN model. These graphs are drawn based on evaluation done against the validation dataset

Figure 9: Visualization of The CNN Training Progression



- **Training and Validation Loss.** The validation and training loss should decrease over the epochs. The training loss is decreasing significantly during the first 10 epochs. A good sign is that the validation is close to the training loss. If these are too far apart, it could be signs of either under- or overfitting.

- **Training and Validation Accuracy.** In contrast to the loss, the Training and Validation Accuracy should increase over time. The validation and training follow each other pretty close over the epochs.

- **Training and Validation Precision.** Both tests shows high precision over the epochs, this suggest that there will be a low rate of false positives.

- **Training and Validation Recall.** The recall should also increase over the epochs and describes that the model does not fail to identify positive cases.

Model evaluation shows that the CNN training has establishing a solid starting point before proceeding with testing the model against our custom dataset, as detailed in Section 3.5. Further parameter tuning is not considered necessary.

Random Forest

The Random Forest Classifier model is evaluated using the predict method, as well as the metrics module of the Scikit-learn library. The model is tasked with predicting the label of the data flows present in the test dataset. The model has not before seen this data, and infers the correct class based on the teachings from the training phase. The evaluation results are presented in Table 15 in Chapter 4.

As evident the model performs supremely well when presented with the task of classifying data similar to that of which it was trained on. All performance metrics are rounded to 1.00 which indicates that the real score is better than 0.995. In order to get a more detailed look into the performance of the Random Forest model the confusion matrix is displayed in Figure 10.

Figure 10: Confusion Matrix of RF Model Evaluation

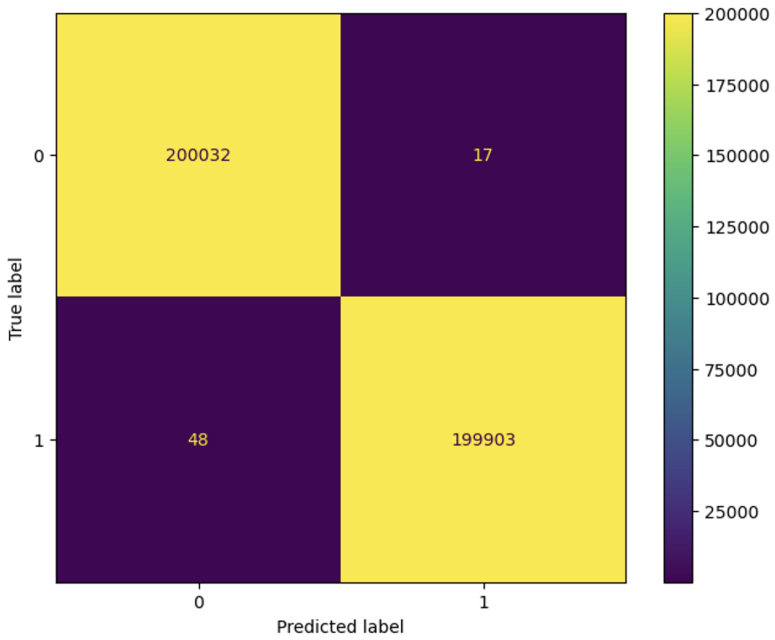


Figure 10 shows how well the model is at predicting the correct label. The Y-axis indicated the true label of the data flows and the X-axis shows the predicted label. Correct predictions are indicated by the yellow boxes, which shows a high number of correct predictions. The model correctly applies the benign and malicious label in all, but 17 and 48 cases, respectively. The small number of false negatives and false positives explains why the reported F1-score is equal to 1.00.

The hyper parameter tuning using RandomizedSearchCV, as described in Section 3.3.2, has ensured that the model performs well and no further tuning is required after evaluation. The Random Forest classifier is now ready to be tested against the dataset we have created ourselves.

3.4 Creating the Custom Dataset

In order to test if our models trained on public data can perform well if deployed in a IDS application, we have created a custom dataset meant to simulate a real-world setting. The machine learning models will be tested on this dataset. This section will present how this dataset was created.

The custom dataset creation process is split into three parts. The first part involves setting up the virtual environment for conducting the network attacks and data logging. The next part involves carrying out the attack scenarios, as well as generating benign data. The final step of the process involves labelling and pre-processing of the captured raw data.

3.4.1 Creating the Virtual Environment

Our custom dataset is created in a virtual computer environment, consisting of one attacker computer and one target computer, as described in Figure 11.

Figure 11: Illustration of Virtual Computer Environment

Both computers are set up as virtual machines in SkyHiGh with the specifications described in Table 10.

Table 10: Specification of VMs in Virtual Environment

Name	OS	IPs	VCPUs	RAM	Storage
Target	Kali Linux 2023.3	192.168.0.30, 10.212.173.130	4	4GB	40GB
Attacker	Kali Linux 2023.3	192.168.0.216, 10.212.172.226	4	8GB	40GB

Whereas both computer are installed with Kali, there are a differences in the additional software used. On the attacker computer, the tools to deploy the attacks are installed. The use of these tools are described in Section 3.4.2. The target computer has the Cisco Joy tool installed. This enables the target computer to capture and log the incoming network attacks stemming from the attacker.

A private network was set up in order for the attacker to connect to the target computer. When conducting the attacks, the computers will be able to communicate over the IPv4 192.168.0.0/24 network. Access to the virtual

environment requires a connection to the university network. In addition to the private network address, the virtual machines are equipped with a public address. The public enables remote control of the virtual machines, via secure shell (SSH) or remote desktop protocol (RDP). Both protocols have been used to configure and issue commands to the virtual machines.

3.4.2 Conducting and Capturing the Network Attacks

In order to generate the raw network data that is to be labelled as malicious in our custom dataset, tree types of common network attacks were conducted. The method of execution is described below. The generation of benign data is also described. Table 11 shows the final content and distribution of our customs dataset.

Table 11: Content of Custom Dataset

Type of Traffic	**Label**	**No. of Flows**	**Share of Total Dataset**
DDoS Attack	Malicious	12	1.14 %
Brute Force Attack	Malicious	47	4.48 %
Port Scan	Malicious	5	0.48 %
Browser Activity	Benign	986	93.90 %
Total	Mixed	1050	100 %

DDoS

The MHDDoS tool, described in Section 2.6.3, was used to carry out the DDoS attack. This tools was chosen due to it easy-of-use and options for tailoring the attack. The following command was used to execute the attack:

```
# DDoS attack using MHDDoS
python3 start.py TCP 192.168.0.30:22 100 3600
```

Listing 1: DDoS Attack Command

The command options are described below:

- **python3 start.py** This command initiates the Python script that executes the DDoS attack. The script takes in parameters that specifies the attack options.
- **TCP** This option specifies the type of DDoS attack. Our choice was to execute a TCP flood attack, due to its commonality.
- **192.168.0.30:22** The target IP-address and port.
- **100** The number of connections used in the attack.
- **3600** The duration of the attack (seconds).

The DDoS attack was then logged by Cisco Joy at the target computer, which generated a total of 12 network flows with features, outputted in a JSON format. It is due to Cisco Joy's grouping of network packets into flows which results in the attack consisting of 12 only flows.

Brute Force

The brute force attack was conducted using Hydra, described in Section 2.6.3. A password list consisting of 200 common passwords was used in the brute force attack. The Hydra command is listed below:

```
# Brute Force attack using Hydra
hydra -l kali -P
   /home/kali/hydra/attack/password_list.txt
   192.168.0.216 ssh -t 4
```

Listing 2: Brute Force Attack Command

The command options are described below:

- **hydra** Initiates the Hydra tool.
- **-l kali** Specifies the username to use in the login attempts.
- **-P /home/kali/hydra/attack/password_list.txt** Path to list.
- **192.168.0.216** Target IP-address.
- **ssh:** Specifies the protocol targeted in the attack.

- **-t 4** Specifies the number of parallel tasks to run. It is set to 4, meaning Hydra will try up to four passwords simultaneously.

The brute force attack was captured using Cisco Joy on the target computer. The attack provided a JSON file consisting of 47 network flows with features.

Port Scan

The port scan was conducted using the Nmap tool. Using a shell script, Hydra was initiated with a range of different parameters. The script is listed below, with the options explained in the code comments.

```
# Basic nmap command
nmap 192.168.0.30

# Nmap ping scan
nmap -sp 192.168.0.30

# Nmap scan all ports
nmap -p 1-65535 192.168.0.30

# nmap most popular ports
nmap --top-ports 20 192.168.0.30

# Nmap scan for OS
nmap -A 192.168.0.30

# Nmap scan for version
nmap -sV 192.168.0.30

# TCP and UDP
nmap -sT
nmap -sU

# CVE detection
nmap -Pn --script vuln 192.168.0.30
```

Listing 3: Port Scan Attack Script

As with the previous attacks, also the port scans were captured using Cisco Joy on the target computer. These port scans generated a total of five network flows containing the packets that made up the port scan.

Normal traffic
In addition to generating the malicious network traffic, normal traffic was also generated in order to supplement our custom dataset with benign data. This was done since the correct labeling of benign data is equally as important as the malicious data.

In this scenario only the target computer is used. The Cisco Joy tool was then used to capture the network data from the computer's internet browser. The browsing included typical work-related activities such as sending email, browsing news sites and logging into Microsoft services such as SharePoint using two-factor authentication. In total, this network activity, which is known to be benign, created 986 flows.

3.4.3 The Data Cleaner

The logging tool Cisco Joy generates a log file with some data features that are not needed for the model testing. Additionally, some of the features require a change of format. The dataset also needs to be converted from JSON to CSV.

In order to conduct the aforementioned data cleaning and preparation we created a Python script, see appendix A.1. The script cleaner.py cleans the raw JSON file by doing the following:

- **Removing the configuration data** The first line in all the JSON files is the configuration data of Cisco Joy. The configuration data is removed.
- **Changes name of some of the features.** For example: "dp", "sp" and "pr" in the JSON file, should be "dest_port" "dest_port" and "proto" in the final CSV file.
- **Standardizing the IP addresses.** In the LuFlow dataset, every IP-address is in integers format. Thus, the cleaner also standardizes the IP-addresses to the same anonymized integer address as the training dataset.
- **Creating the feature "label".** This label decides whether the traffic is malicious or benign. If malicious, the label is 0, while for benign data the label is 1. The network attacks are labelled as malicious, while the normal browser traffic is labelled as benign.
- **Creating the feature "duration".** By default, Cisco Joy does not create the duration feature. This is calculated by subtracting the value of

"time_start" from "time_end" and appending it to the network flows' features.

- **Creating the feature "avg_ipt".** If the inter packet time (ipt) is available, we can calculate the average ipt.
- **Converting to CSV file.** Finally, the script converts the file into a CSV-file which is processable by the machine learning models.

There is one step missing, which is done when creating the test files, which is defining the "src_ip" and "dest_ip" as type float64. After this step, our custom dataset is on the correct format and ready to be used for model testing.

3.5 Testing the Models

Testing of the two machine learning models is done using the Scikit-learn methods model.predict and classification_report. To facilitate for ease of testing the custom dataset was transferred to the virtual machine used to develop the CNN and RF model, and testing was done locally on that machine. The final model testing were done in three parts:

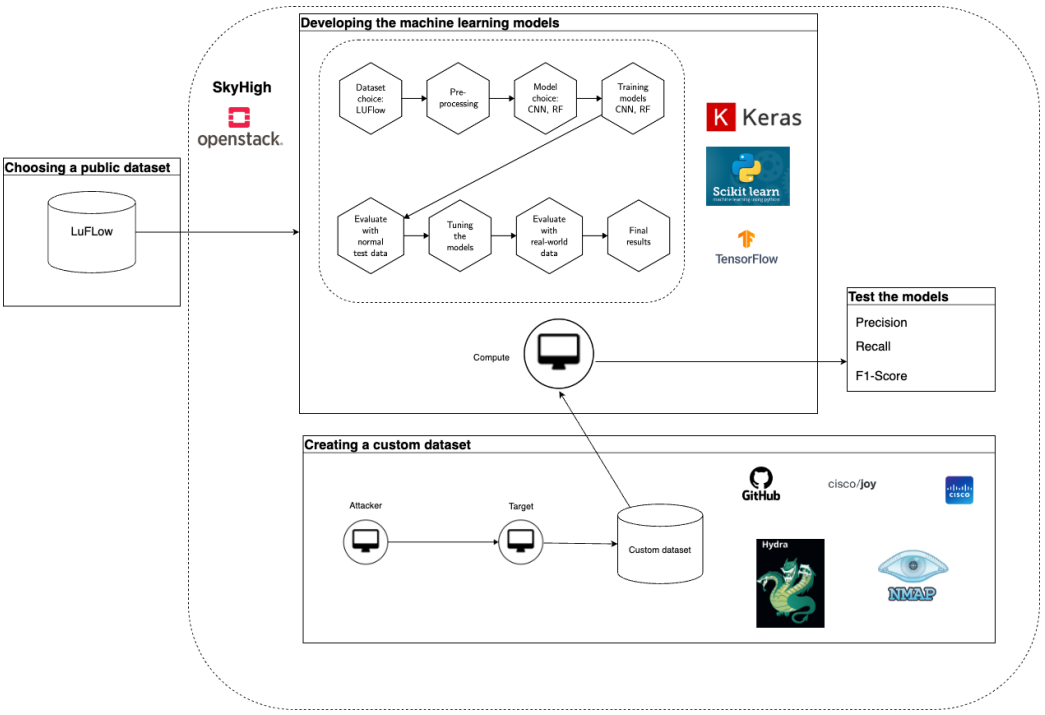
- Testing the models on the LuFlow test data.
- Testing the models on separate parts of the custom dataset.
- Testing the models on the combined custom dataset.

This separation of testing was done in order to yield more detailed results of how the models perform depending on the type of data it is tasked with classifying. This allows for a comparative evaluation of the results. In a real-world scenario the IDS would likely process a combination of mostly normal traffic and some malicious traffic, thus the malicious and benign data was also combined and tested together. The testing results are presented in Chapter 4.

3.6 Summary of Method

Figure 12 summarises the methodology of our research and highlights the technologies used.

Figure 12: Detailed Illustration of Methodology



The figure depicts how the aforementioned parts of this chapter fits together into the method as a whole. By completing the steps described in this chapter our method has yielded the test results presented in the next chapter.

4 Results

The results from our testing is presented in this chapter. The chapter is divided into two parts, one for each machine learning model. The result for each model is presented by three tables and one confusion matrix.

The first table shows the results from model testing against the test data from the LuFlow dataset. This is the test data from taken from the same dataset the models are trained on. The second table presents the results from testing with each of the network attacks separated, as well as the benign traffic. The third table presents the results from when the network attack and normal data is combined into one file. By looking at the three tables together one can establish an understanding of how the machine learning models perform in classifying network data across different datasets. Also, for each model a confusion matrix is presented, which shows the predictions made by the models compared to the actual values of the test data.

4.1 The CNN Model

In this section the results from testing of the convolutional neural network is presented.

4.1.1 CNN Testing on LuFlow Dataset

Table 12 shows the test results from classification testing of the CNN model on the LuFlow dataset.

Table 12: CNN Test Results on LuFlow Dataset

Attack	**Precision**	**Recall**	**F1-Score**	**Support**
Malicious	1.00	1.00	1.00	99945
Benign	1.00	1.00	1.00	100055

As shown in the table above, the CNN models scores perfectly when tested on data similar to that of which it was trained on. Both classification of malicious and benign network traffic is done with high precision and high recall, thus the F1-score is also high. This evaluation results show that the CNN development has been successful and that the model can correctly predict the label on data originating from the same dataset as the training data.

4.1.2 CNN Testing on Our Custom Dataset

Table 13 and 14 showcases how the CNN model performs when tasked with classifying the network data in our custom data set. Table 13 shows the results on a per-attack basis, and Table 14 shows how the model performs when handling the total dataset at once.

Table 13: CNN Test Results on Separated Custom Data

Attack	**Precision**	**Recall**	**F1-Score**	**Support**
DDoS	1.00	1.00	1.00	12
Brute Force	1.00	0.98	0.99	47
Port Scan	1.00	1.00	1.00	5
Normal Traffic	1.00	0.03	0.06	986

As evident by the results presented in Table 13 the model performs well in identifying the malicious attacks as malicious when testing is done on per-attack basis. Both precision and recall is high for the network attacks. Yet, when the model is tasked with classifying the normal network traffic it fails to identify it as benign. Recall is very low, at only 0.03, meaning it fails to classify almost all the normal traffic as benign. This results in a poor F1-score for normal traffic.

Table 14: CNN Test Results on Merged Custom Data

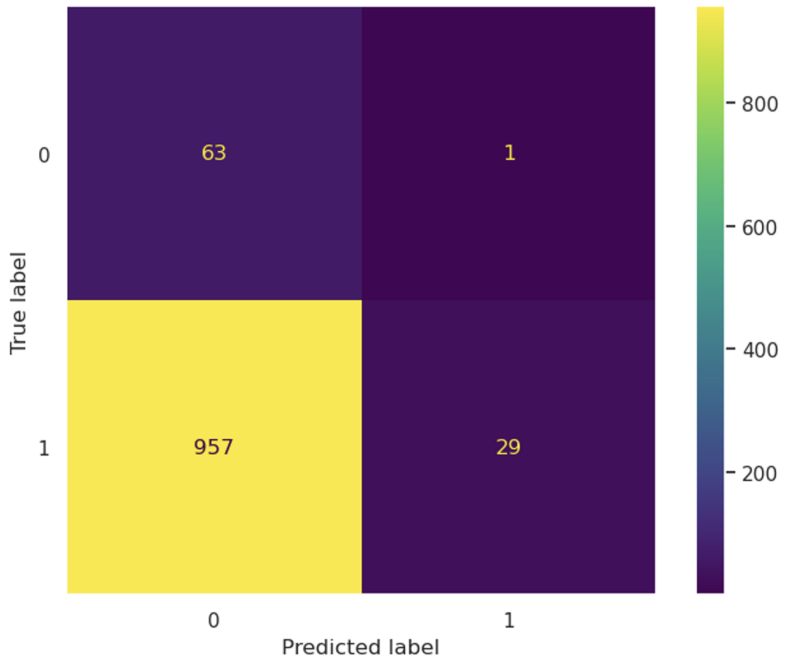
Label	**Precision**	**Recall**	**F1-Score**	**Support**
Malicious	0.06	0.98	0.12	64
Benign	0.97	0.03	0.06	986

When the custom dataset is merged into one file the test results worsen. For malicious traffic the model scores high on recall, but suffers from low precision. For benign traffic the model scores high on precision, but suffers from low recall. The low scores affect the F1-score, resulting in poor overall performance.

The results show that the CNN model correctly identifies most of the malicious packets, but it incorrectly labels benign traffic as malicious, hence the low precision. The opposite is true for benign traffic. Here, the model has high precision, meaning when it does apply the benign label it is correct 97% of the

time, but its mislabels the majority of the benign traffic as malicious. This is visualised by the confusion matrix seen in Figure 13 below.

Figure 13: Confusion Matrix of CNN Model Testing



The confusion matrix shows how the model predicts label 0, representing malicious, in 1020 out of 1050 cases. Unfortunately, this is only correct in 63 cases. It is evident from the confusion matrix that the majority of classifications are false positives of malicious classifications. In summary, the CNN model labels almost all data as malicious when it is tasked with classifying the network traffic in the complete custom dataset.

4.2 The RF Model

In this section the results from testing of the random forest classifier is presented.

4.2.1 RF Testing on LuFlow Dataset

Table 15 shows the test results from classification testing of the RF model on the LuFlow dataset.

Table 15: RF Test Results on LuFlow Dataset

Label	Precision	Recall	F1-Score	Support
Malicious	1.00	1.00	1.00	200049
Benign	1.00	1.00	1.00	199951

As shown in Table 15, the RF model scores an ideal score when tested on the test subset from the LuFlow dataset. Classification of both malicious and benign network traffic is done with high precision and high recall. This gives a perfect F1-score. This performance is similar to that of the CNN model. These evaluation results show that the training of RF model has been successful.

4.2.2 RF Testing on Our Custom Dataset

Table 16 and 17 presents how the RF model performs when attempting classification of the network data in our custom data set. Table 16 shows the results on a per-attack basis, and Table 17 shows how the model performs when handling the total dataset at once.

Table 16: RF Test Results on Separated Custom Data

Attack	Precision	Recall	F1-Score	Support
DDoS	1.00	1.00	1.00	12
Brute Force	1.00	1.00	1.00	47
Port Scan	0.00	0.00	0.00	5
Normal Traffic	1.00	0.01	0.01	986

Evident by the table above, the RF model test shows mixed results. For DDoS and brute force attacks the RF model classifies them correctly as malicious network traffic, yielding a F1-score of 1.00. As for the port scan the RF

model incorrectly labels all the data flows. This results in a precision score of 0.00 and recall of 0.00. For normal traffic the results are also poor. Precision is ideal, but recall is close to zero at 0.01. This mean that the model fails to correctly classify 99% of the normal traffic as benign.

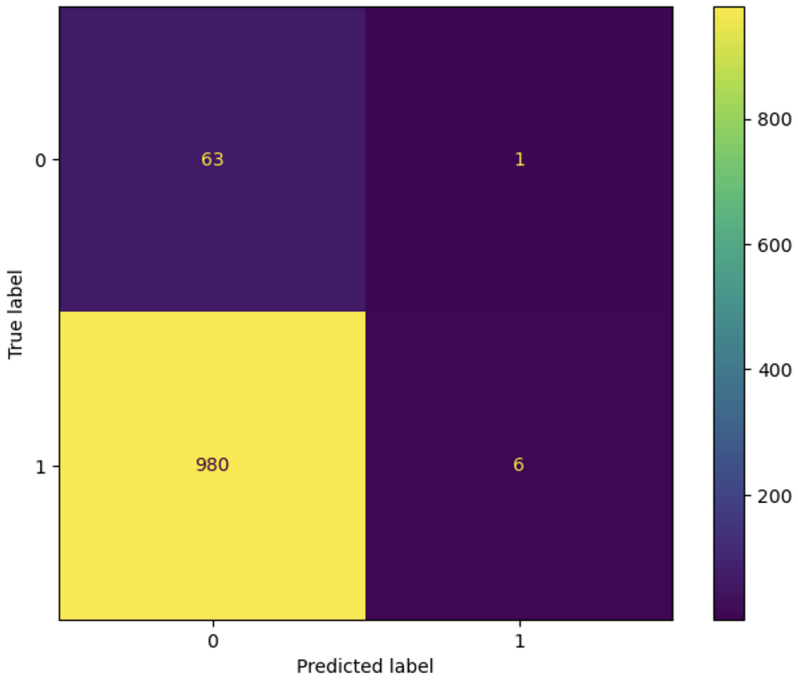
Incorrect classification of benign traffic is also a problem when the RF model is tasked with classifying the data in the merged dataset. This is made evident in Table 17 and Figure 14.

Table 17: RF Test Results on Merged Custom Data

Attack	**Precision**	**Recall**	**F1-Score**	**Support**
Malicious	0.06	0.98	0.11	64
Benign	0.86	0.01	0.01	986

Precision is low for malicious traffic, while recall is high. For benign traffic the opposite is true, meaning the RF model applies the malicious label to the great majority of data.

Figure 14: Confusion Matrix of RF Model Testing



The confusion matrix in Figure 14 demonstrates the classification issue of the RF model. It predicts 1043 cases of malicious traffic, yet it is wrong in 980 of those predictions. This results in the precision of 0.06. Since it incorrectly labels these flows as malicious, the recall for benign traffic is equally poor. These poor metrics results in a weak F1-score that quantifies the models inability to effectively classify data from our custom dataset.

5 Discussion

This thesis attempts to answer the following question: How well does an AI-based IDS trained on publicly available data perform when tested in simulated real-world scenario? To answer this question a discussion of the results presented in Chapter 4 is needed. The methodology used to yield these results is also discussed. By systematically evaluating the methodology, its strength and weaknesses are highlighted. This gives insight into why the results are as presented.

This chapter structure follows the methodology backwards. First, the results are evaluated. Second, the models themselves are discussed. Third and finally, the dataset foundation is evaluated.

5.1 Evaluation of the Test Results

The testing of both models shows the same results. That is, both models perform well when classifying data from the LuFlow dataset, yet they perform poorly when classifying data in our custom dataset. Our test results therefore shows that an AI-based IDS trained on publicly available data performs poorly when deployed in an environment where the network traffic is unfamiliar to model. Much of this chapter attempts to give reasons as to why the performance is as poor as made evident.

5.1.1 Comparative Evaluation of CNN and RF Test Results

In terms of performance there are some minor differences between the models. This section evaluates these.

Evaluation of the models on the LuFlow test dataset yields solid scores for both the CNN and RF model. In fact, both models have a score of 1.00 on precision, recall and F1-Score. Thus, the models are capable of correctly labeling the malicious and benign traffic. These scored verifies that the the development of our models has been successful.

Even so, it is the testing on our custom dataset that sheds light on the research question of this thesis. By keeping the network attacks separated for testing we can gauge how well each models performs on labeling the different types of attacks. These results are presented in Table 13 and 16. For DDoS and Brute Force, both models gets a score of 1.00 on precision, recall and F1-score. The CNN model correctly labels the Port Scan attacks, while the RF model does

not. These results gives evidence to conclude that the CNN models performs better at identifying network attacks.

Additionally, both models were tested if they could label the normal network traffic as benign. This is where both models suffer. The CNN model has a score of 1.00 on precision, but only 0.03 on recall and 0.06 on the F1-score. The RF model scores even worse, yielding results of 1.00 on precision, 0.01 on recall and 0.01 on the F1-score. This inability to correctly label benign traffic has detrimental effect on the models' overall performance.

Finally, the network attacks and the normal traffic is combined into a single dataset for combined testing. This test is meant to simulate how the models would perform when deployed in a production environment. In such a real-world scenario an IDS needs to be able to classify both malicious and benign data. Table 14 and 17 shows these test results. They show that both models applies the malicious label to the majority of the network flows in the dataset. For malicious labeling both models have a precision of 0.06 and recall of 0.98. F1-score is also similar. For benign labelling the CNN model scores 0.97 on precision, 0.03 on recall and 0.06 on F1-score, while the RF scores 0.86, 0.01 and 0.01 on the same metrics.

Despite these slight performance differences, the conclusion regarding both models are the same. Neither model perform well when labeling the custom dataset containing the unseen network traffic that is gathered from a different source than the training data. This indicates that the models are unfit for being put to use in an environment different to that of which its training data was gathered from. The poor performance originates from the incorrect labeling of benign data. Given that the benign data represents 95% of the total dataset the incorrect labeling of the majority of the benign data drastically affect the overall performance in testing. If put into production in this state the results indicate that the models would report an extensive amount of false positives.

These poor test results warrants further evaluation of the methodology in order to understand the underlying reasons, as well as giving advice for further research. This evaluation of methodology is done in the following sections.

5.2 Evaluation of Model Development

A crucial part of the methodology is the development of the machine learning models. As Section 3.3 presents and Figure 5 shows, the process of making the CNN and RF models has been kept as similar as possible. As the comparative analysis in Section 5.1.1 shows, the test result of both models support the

same conclusion. That is, training an AI-model on a public datasets leads to subpar performance when applied in an environment where the network data is dissimilar to the training data.

5.2.1 The Development Process

As discussed earlier, both models were developed using a similar methodology. Firstly, the models were trained on the same dataset. Also, the pre-processing was the same for both models. The only difference prior to model training is in the dataset split. This difference is due to the CNN model's use of a validation set to aid training. The RF model did not employ a validation dataset, hence its training and test splits are larger.

Naturally, the models differ in their inherent nature. CNN is a deep learning model with layers and nodes, while RF is a ensemble learning algorithm consisting of decisions trees. This leads to different model architectures and different use of technology. Whereas the RF model uses the Scikit-learn library, the CNN model also uses TensorFlow and Keras. Model optimization is also different given that the algorithms operates differently. The hyper parameters are different since the model architectures are not the same.

Granted the similarities and differences between the models' development processes they perform comparably. Both models perform well when tasked with classifying test data from LuFlow and subpar when tasked with labelling the data in our custom dataset. The similarity in performance between the two models strengthens the argument that the poor performance is not grounded in methodological errors, but instead in the inherent differences between the training dataset LuFlow and our custom dataset meant to simulate a production setting.

5.2.2 Improvement Areas of the Development Process

Given today's rapid technological improvements within machine learning and artificial intelligence, it is necessary to acknowledge that improvements to our method are possible. Potential improvements are discussed below and their potential impact on the test results are evaluated.

Increasing the size of the training dataset is one possible point of improvement. In general, a larger training dataset improves the model, however there are diminishing returns to consider. Our process was limited by resource availability. The LuFlow dataset as a whole exhausted our available RAM resources,

therefore we decided to use a sample as our training set.

A potential increase of the training dataset would also require an increase in the CPU power. We were restricted by having no GPU resources available for training, and thus had to rely on a multi-core CPU to handle the computational workload. Access to tensor cores, or similar, would allow for faster iteration of the machine learning models.

Despite the resource limitations mentioned above, testing of the ML models on test data from LuFlow showed that the models had close to no room for improvement, as they both scored F1-scores of 1.00. Also, the sampling of the entire LuFlow dataset was done by random sampling, meaning there is no reason to believe that particularly important parts of the dataset were left out of our training sample due to sampling errors.

Given the unsatisfactory performance of our machine learning models, a clear improvement to our process would be to allocate time and resources for feature analysis. This could inform feature engineering efforts and detect data quality issues. Model interpretability would likely be improved by feature analysis, which could aid the development of future iterations of the models. A thorough feature analysis would require time and resources not available to us at this stage. See Section 5.4 for more detail regarding future avenues of research.

5.2.3 Model Development Evaluation Summary

Having reviewed the process of developing the two machine learning models, its strengths and weaknesses have become apparent. The process has showed that training ML models on network data is relatively straightforward, yet the iterative process of tuning and improving the models is a time and resource intensive task. It also requires a thorough understanding of the technology and mathematics that underpins the models.

While the CNN and RF models' performance when labelling our custom dataset is underwhelming, the development process itself has no immediate weaknesses, other than limited time and resources. Yet, we cannot state that our models are optimal, due to the fact this project has not compared a large range of different models with different hyper parameters. Our research aimed at investigating AI-based IDS trained on public datasets, and what we have discovered is that applications of AI such as this, where the training set and testing set comes from different sources, requires thorough analysis and assessment both before and after model training. Given more time, a natural

continuation for our research would be conduct a comprehensive and systematic post-training analysis and developed improved iterations of the models.

5.3 Dataset Evaluation

The datasets are fundamental to the research done in this project, as they laid the foundation for both model development and testing. Due to their importance for the results they require evaluation.

Firstly, its crucial to evaluate how suitable the LuFlow dataset is for model development and training. Secondly, its important to review how representative our custom dataset is of a real-world production setting. Finally, given the poor testing results, it is important to assess if any methodological mistakes were made during data processing and testing.

5.3.1 Use of LuFlow in Model Development

The LuFlow dataset is presented in Section 2.5.2 and the reasoning behind choosing the LuFlow dataset for development is discussed in Section 3.2. To summarize, the dataset was chosen due the fact that the data was gathered from modern, real-world network traffic. Also, the data features of the dataset were regarded as relevant and allowed us to replicate those features in our custom dataset. These strengths made it the best candidate of the datasets considered.

The consideration of other public data sets did not reveal any reason to suspect other candidates would yield a better results. However, due to time and resource constraints this has not been confirmed by technical testing, as this would require re-engineering of the custom dataset in order to match data features.

Furthermore, as described in Section 2.5.2, the dataset consist of relevant attacks and has successfully been used to label attacks similar to those that we have conducted. This indicates that the model training data is appropriate for our project. In a way, this is confirmed by our models' ability to correctly label malicious attacks. Yet, it is the incorrect labelling of benign data that hurts our models' performance. This reveals the necessity for evaluation of our custom dataset, and mainly the benign data.

5.3.2 Evaluation of Our Custom Data Set

Essential to our research is the custom dataset we have created for testing and validating the models trained on the public data. By validating our models' classification performance on this data we are able to give an informed opinion on the research question at hand. Given that the test results are poor, driven by low classification performance on benign data, it is critical that the dataset is evaluated. The method of how we assembled our custom dataset is described in Section 3.4. This section evaluates the dataset content in a qualitative manner.

As discussed earlier, our custom dataset contains 984 benign and 64 malicious network flows. The overweight of benign flows is intentional and is due to the inherent overweight of benign network data in an ordinary network. This unbalance does affect the performance in a negative manner as the statistical metrics are weighted heavier by the incorrectly labeled benign data. By reducing the proportion of benign data we could have artificially boosted the models' performance, yet this would obfuscate the true performance of the developed models. We intended for our custom dataset to closely resemble the real-world conditions the model could expect to handle if deployed in a production environment. Therefore, we found it important to highlight the models' weaknesses, and not hide them in favourably sampled test data.

Given the poor classification results it is worthwhile to discuss the benign data. The benign data generation and capture is described in Section 3.4.2. Since the capture took place in an isolated virtual environment we are certain the benign data is not contaminated with malicious data, thus we can be sufficiently certain that the data is not mislabelled in our custom dataset.

As already mentioned, if time and resources allowed for it, the logical next step would be to conduct a thorough feature analysis of the dataset and models, as this might give insight into which data features that leads to the incorrect labeling. This is a potential avenue for further research. As for this project, having concluded that the capture of benign data was done to the best of our ability, the final step is to evaluate if any methodological errors were done in the dataset processing stage, subsequent to the data capture and prior to the model testing. This is evaluated in the next section.

5.3.3 Evaluation of Dataset Treatment Method

As described in Section 3.4, following the data capture of the network attacks, the data was pre-processed and prepared for testing. The intention of this data treatments was to ensure that our custom dataset contained the same features as the LuFlow dataset used for training. Given the poor testing results it is imperative to assess if any methodological errors were made in this part of the process.

A comparative dataset analysis has been conducted. Datasets samples have been compared and the dataset composition have been analysed. The only finding was that our custom dataset had four data features out-of-order when compared to the LuFlow dataset. Theoretically, the order of features should not affect the models in any significant way. To confirm this, the custom dataset was reordered such that it is an exact match of the LuFlow dataset structure, and testing was conducted again. See Appendix A.2 for the analysis procedure. The CNN model showed no change in performance, and the RF models change was not significant. Thus, it can be deduced that the limited mismatch in feature order present during testing did not affect the results in any significant way. Furthermore, it can be concluded that the pre-processing and data treatment of our custom dataset did not lead to methodological errors of any significance to the models' performance.

5.3.4 Dataset Evaluation Summary

The above evaluations of the application, capture and treatment of our dataset does not expose any immediate weaknesses to the methodology. Therefore, the results of our testing indicates that it is the intrinsic differences between the network traffic in the training data and in our custom dataset that affects the machine learning algorithms' performance.

This finding underpins our answer to the research question at hand, namely that the training data of an AI-based IDS solution is critical for its performance in production. The training dataset should be similar to the real traffic the IDS is expected to process. Ideally, the machine learning model should be trained on data from the same network it is intended to be deployed in.

5.4 Future Development

The result of our research has highlighted two avenues for future development and research.

Feature analysis is one natural path forward. By experimenting with feature engineering and feature importance, as well as dimensionality reduction and correlation analysis one might uncover ways of improving the dataset processing and the model performance. With the appropriate resources it would provide an interesting extension of the research done in this paper. If feature analysis is conducted, and there is no significant improvement in model performance, it would reinforce the argument that real-world domain-specific data is imperative in the development of AI models for network applications.

Comparative analysis with both public and domain-specific training data is another such avenue of potential research. This approach would require access to domain-specific data which proved unattainable for our research due to privacy and security concerns. However, if this challenge was to be overcome, one would be able to compare if public data can be used as a substitute for domain-specific data. Furthermore, this approach would allow for classification testing against data captured in a operational network. If the models trained on the public datasets exhibit significantly inferior performance compared to those trained on domain-specific data, it would serve as evidence that domain-specific data is a necessity for achieving optimal performance in an AI-based IDS solution.

Both of the above directions for further investigations would serve to validate our findings or demonstrate the opposite. Since our research is not conclusive by itself, both research directions are regarded as beneficial to the research topic at hand.

6 Conclusion

6.1 Summary

This thesis investigates the performance of AI-based Intrusion Detection Systems trained on publicly available data when tested against a custom dataset intended to simulate a real network setting.

Our research used the public LuFlow dataset as the training dataset for a Convolutional Neural Network (CNN) and a Random Forest (RF) classifier. To test the models we have used a testing subset of the LuFlow dataset and also created our own custom dataset, meant to simulate a real-world network by conducting common network attacks against a logging target.

Our results show that both the CNN and RF models perform well in correctly labeling the LuFlow dataset. However, the effectiveness of both models significantly decreases when applied to our custom dataset.

This serves as evidence that AI-models intended to solve classification tasks on network data should be trained on data from the specific network it will be deployed in. Public datasets appear inherently unsuitable for such applications.

6.2 Further work

Future work related to our study can be divided into the scope of our research and the broader context of AI-based IDS.

Our research suggest further investigation into feature analysis. Depending on the findings, further evaluation could determine whether the differences between the datasets are the significant factor leading to the inefficacy of the AI-models.

In the greater context of AI and IDS research, our project highlights that access to real-world datasets is a challenge. If this obstacle was overcome a comparative analysis could be carried out, which could provide a more robust conclusion. Hopefully, this challenge can be solved such that academical research in the field can continue and lead to improvements in AI-based network security.

Bibliography

[1] The Editors of Encyclopedia Britannica, "Computer network," in *Encyclopedia Britannica*. Mar. 2024.

[2] I. Winkler and A. T. Gomes, "Chapter 2 - cyberwarfare concepts," in *Advanced Persistent Security*, I. Winkler and A. T. Gomes, Eds., Syngress, 2017, pp. 15–19, ISBN: 978-0-12-809316-0. DOI: https://doi.org/10.1016/B978-0-12-809316-0.00002-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128093160000026.

[3] Sangfor Technologies, *What is computer network defense (cnd)*, Accessed: Jan 30, 2024, Oct. 2023. [Online]. Available: https://www.sangfor.com/glossary/cybersecurity/what-is-cnd-computer-network-defense.

[4] Microsoft, *Hva er informasjonssikkerhet (infosec)?* Accessed: Jan 30, 2024. [Online]. Available: https://www.microsoft.com/nb-no/security/business/security-101/what-is-information-security-infosec.

[5] R. W. Håkon Bergsjø and L. Øverlier, *Digital sikkerhet: en innføring*. Oslo: Universitetsforlaget, 2020, pp. 21–23, ISBN: 9788215034225.

[6] A. S. Ashoor and S. Gore, "Difference between intrusion detection system (ids) and intrusion prevention system (ips)," in *Advances in Network Security and Applications*, D. C. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, and D. Nagamalai, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 497–501, ISBN: 978-3-642-22540-6.

[7] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 222–232, 1987. [Online]. Available: https://api.semanticscholar.org/CorpusID:262088592.

[8] A. Mailewa and S. Thapa, "The role of intrusion detection/prevention systems in modern computer networks: A review," pp. 1–10, 2020.

[9] J. Burton, I. Dubrawsky, V. Osipov, C. T. Baumrucker, and M. Sweeney, "Chapter 2 - cisco intrusion detection," in *Cisco Security Professional's Guide to Secure Intrusion Detection Systems*, Burlington: Syngress, 2003, pp. 39–73, ISBN: 978-1-932266-69-6. DOI: https://doi.org/10.

1016/B978-193226669-6/50022-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781932266696500227.

[10] C. Chio, *Machine learning & security : Protecting systems with data and algorithms*, eng, Sebastopol, Calif, 2018.

[11] I. Strümke, *Maskiner som tenker : Algoritmenes hemmelighet og veien til kunstig intelligens*, nob, Oslo, 2023.

[12] IBM. "What is ai?" (2024), [Online]. Available: https://www.ibm.com/topics/artificial-intelligence (visited on 04/05/2024).

[13] IBM. "What is strong ai?" (2024), [Online]. Available: https://www.ibm.com/topics/strong-ai (visited on 04/05/2024).

[14] IBM. "What is ml?" (2024), [Online]. Available: https://www.ibm.com/topics/machine-learning (visited on 04/10/2024).

[15] IBM. "Supervised vs. unsupervised learning: What's the difference?" (2024), [Online]. Available: https://www.ibm.com/blog/supervised-vs-unsupervised-learning/ (visited on 04/12/2024).

[16] M. A. Nielsen. "Neural networks and deep learning." (2024), [Online]. Available: http://neuralnetworksanddeeplearning.com/chap5.html (visited on 04/16/2024).

[17] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 151–158. DOI: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037.

[18] N. Chockwanich and V. Visoottiviseth, "Intrusion detection by deep learning with tensorflow," Feb. 2019, pp. 654–659. DOI: 10.23919/ICACT.2019.8701969.

[19] D. Cowan. "The science of machine learning & ai." (2024), [Online]. Available: https://www.ml-science.com/models-and-modeling-overview (visited on 04/16/2024).

[20] Encord. "Datasets." (2024), [Online]. Available: https://encord.com/glossary/datasets-definition/ (visited on 04/17/2024).

[21] R. Mills, *Luflow network intrusion detection data set*, 2024. DOI: 10.34740/KAGGLE/DSV/8106645. [Online]. Available: https://www.kaggle.com/dsv/8106645.

[22] R. Mills, *Enhancing Anomaly Detection Techniques for Emerging Threats*. Lancaster University, 2022. [Online]. Available: https://books.google.no/books?id=MjGHzwEACAAJ.

[23] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[24] P. Perricone, B. Hudson, B. Anderson, B. Long, and D. McGrew, *Joy, a package for capturing and analysing network features*. Cisco Systems, 2018.

[25] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Sunnyvale, CA, USA: Insecure, 2009, ISBN: 0979958717.

[26] Cloudfare. "What is a ddos attack?" (2024), [Online]. Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ (visited on 04/17/2024).

[27] Microsoft. "What is a ddos attack?" (2024), [Online]. Available: https://www.microsoft.com/en-us/security/business/security-101/what-is-a-ddos-attack (visited on 04/17/2024).

[28] Cloudfare. "What is a brute force attack?" (2024), [Online]. Available: https://www.cloudflare.com/learning/bots/brute-force-attack/ (visited on 04/17/2024).

[29] P. A. Networks. "What is a port scan?" (2024), [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-port-scan (visited on 04/17/2024).

[30] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *ACM CoNEXT '10*, Philadelphia, PA, Dec. 2010.

[31] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the WIDE project," in *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, San Diego, CA: USENIX Association, Jun. 2000. [Online]. Available: https://www.usenix.org/conference/2000-usenix-annual-technical-conference/traffic-data-repository-wide-project.

[32] K. Highnam, K. Arulkumaran, Z. D. Hanif, and N. R. Jennings, "Beth dataset: Real cybersecurity data for anomaly detection research," 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:237258147.

[33] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4707749.

[34] R. Mills, *Luflow network intrusion detection data set*, 2024. DOI: 10.34740/KAGGLE/DSV/8389692. [Online]. Available: https://www.kaggle.com/dsv/8389692.

[35] geeksforgeeks. "Introduction to convolution neural network." (Mar. 2024), [Online]. Available: https://www.geeksforgeeks.org/introduction-convolution-neural-network/ (visited on 05/13/2024).

[36] DeepChecks. "Relu." (2024), [Online]. Available: https://deepchecks.com/glossary/rectified-linear-unit-relu/ (visited on 04/29/2024).

A Additional Code and Output

A.1 Cleaner.py

The python code presented below works as a cleaner. A detailed description is described in section 3.4.3.

```
import json
import csv
import socket
import struct

# Function to convert an IP address from string to an integer
def ip_to_int(ip):
    if ip and isinstance(ip, str):
        try:
            return struct.unpack("!I", socket.inet_aton(ip))[0]
        except socket.error:
            # Handles end cases
            return 0
    else:
        return 0

csv_file_name = 'portscan_v2.csv'

# Open the JSON file and the CSV file for writing
with open('portscan_.json', 'r') as json_file, open(csv_file_name, 'w', newline='') as csv_file:
    fieldnames = ['avg_ipt', 'bytes_in', 'bytes_out', 'dest_ip', 'dest_port', 'entropy', 'proto',
                  'src_ip', 'src_port', 'num_pkts_out', 'num_pkts_in', 'total_entropy', 'label', 'duration']

    # Create the CSV writer object
    csv_writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

```

```
    csv_writer.writeheader()

    # Skip the first line (configuration data)
    next(json_file)

    for line in json_file:
        event = json.loads(line)

        row = {
            'bytes_in': event.get('bytes_in', 0),
            'bytes_out': event.get('bytes_out', 0),
            'dest_ip': 786,  # Make it a standard value
            'dest_port': event.get('dp', 0),
            'entropy': event.get('entropy', 0.0),
            'proto': int(event.get('pr', 0)),
            'src_ip': 786,  # Make it a standrad value
            'src_port': event.get('sp', 0),
            'num_pkts_out': int(event.get('num_pkts_out', 0)),
            'num_pkts_in': int(event.get('num_pkts_in', 0)),
            'total_entropy': event.get('total_entropy', 0.0),
            'label': 0,  # 0 = benign, 1 = malicious, update it accordingly
            'duration': event.get('time_end', 0) - event.get('time_start', 0)
        }

        # Calculate avg_ipt
        packets = event.get('packets', [])
        if packets:
            total_ipt = sum(packet.get('ipt', 0) for packet in packets)
            row['avg_ipt'] = total_ipt / len(packets) if len(packets) > 0 else 0.0
        else:
            row['avg_ipt'] = 0.0
```

```
58
59          csv_writer.writerow(row)
60
61 print(f"Data has been successfully written to
      {csv_file_name}")
```

Listing 4: Cleaner.py

A.2 Dataset Comparison

The Python code and output used in the discussion is presented below.

```
1 ds_luflow.info()
2 ds_custom.info()
```

Listing 5: Dataset Feature Comparison

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 203298300 entries, 0 to 203298299
Data columns (total 15 columns):
 #   Column         Dtype
---  ------         -----
 0   index          int64
 1   avg_ipt        float64
 2   bytes_in       int64
 3   bytes_out      int64
 4   dest_ip        int64
 5   dest_port      float64
 6   entropy        float64
 7   num_pkts_out   int64
 8   num_pkts_in    int64
 9   proto          int64
 10  src_ip         int64
 11  src_port       float64
 12  total_entropy  float64
 13  label          int64
 14  duration       float64
dtypes: float64(6), int64(9)
memory usage: 22.7 GB
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1050 entries, 490 to 851
Data columns (total 15 columns):
 #   Column         Dtype
---  ------         -----
 0   index          int64
 1   avg_ipt        float64
 2   bytes_in       int64
 3   bytes_out      int64
 4   dest_ip        int64
 5   dest_port      float64
 6   entropy        float64
 7   proto          int64
 8   src_ip         int64
 9   src_port       float64
 10  num_pkts_out   int64
 11  num_pkts_in    int64
 12  total_entropy  float64
 13  label          int64
 14  duration       float64
dtypes: float64(6), int64(9)
memory usage: 131.2 KB
```

As seen in the output, the data features in column 7 to 11 is out of order compared to the LuFlow features. This is changed by the code showed below. An exact match is show in the code output following the rearrangement.

```
1 import pandas as pd
2
3 new_order = ['index', 'avg_ipt', 'bytes_in',
     'bytes_out', 'dest_ip', 'dest_port', 'entropy',
     'num_pkts_out', 'num_pkts_in','proto' ,'src_ip',
     'src_port', 'total_entropy', 'label', 'duration']
4
5 ds_custom_reordered = ds_custom[new_order]
6
7 ds_luflow.info()
```

```
8 ds_custom_reordered.info()
```

Listing 6: Dataset Column Reordering

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 203298300 entries, 0 to 203298299
Data columns (total 15 columns):
 #   Column         Dtype
---  ------         -----
 0   index          int64
 1   avg_ipt        float64
 2   bytes_in       int64
 3   bytes_out      int64
 4   dest_ip        int64
 5   dest_port      float64
 6   entropy        float64
 7   num_pkts_out   int64
 8   num_pkts_in    int64
 9   proto          int64
 10  src_ip         int64
 11  src_port       float64
 12  total_entropy  float64
 13  label          int64
 14  duration       float64
dtypes: float64(6), int64(9)
memory usage: 22.7 GB

<class 'pandas.core.frame.DataFrame'>
Index: 1050 entries, 490 to 851
Data columns (total 15 columns):
 #   Column         Dtype
---  ------         -----
 0   index          int64
 1   avg_ipt        float64
 2   bytes_in       int64
 3   bytes_out      int64
 4   dest_ip        int64
 5   dest_port      float64
```

```
 6   entropy        float64
 7   num_pkts_out   int64
 8   num_pkts_in    int64
 9   proto          int64
 10  src_ip         int64
 11  src_port       float64
 12  total_entropy  float64
 13  label          int64
 14  duration       float64
dtypes: float64(6), int64(9)
memory usage: 131.2 KB
```

Tests sets were then created using the reordered custom dataset and testing was conducted on the RF and CNN model. New classifications report were generated for comparison with the original testing.

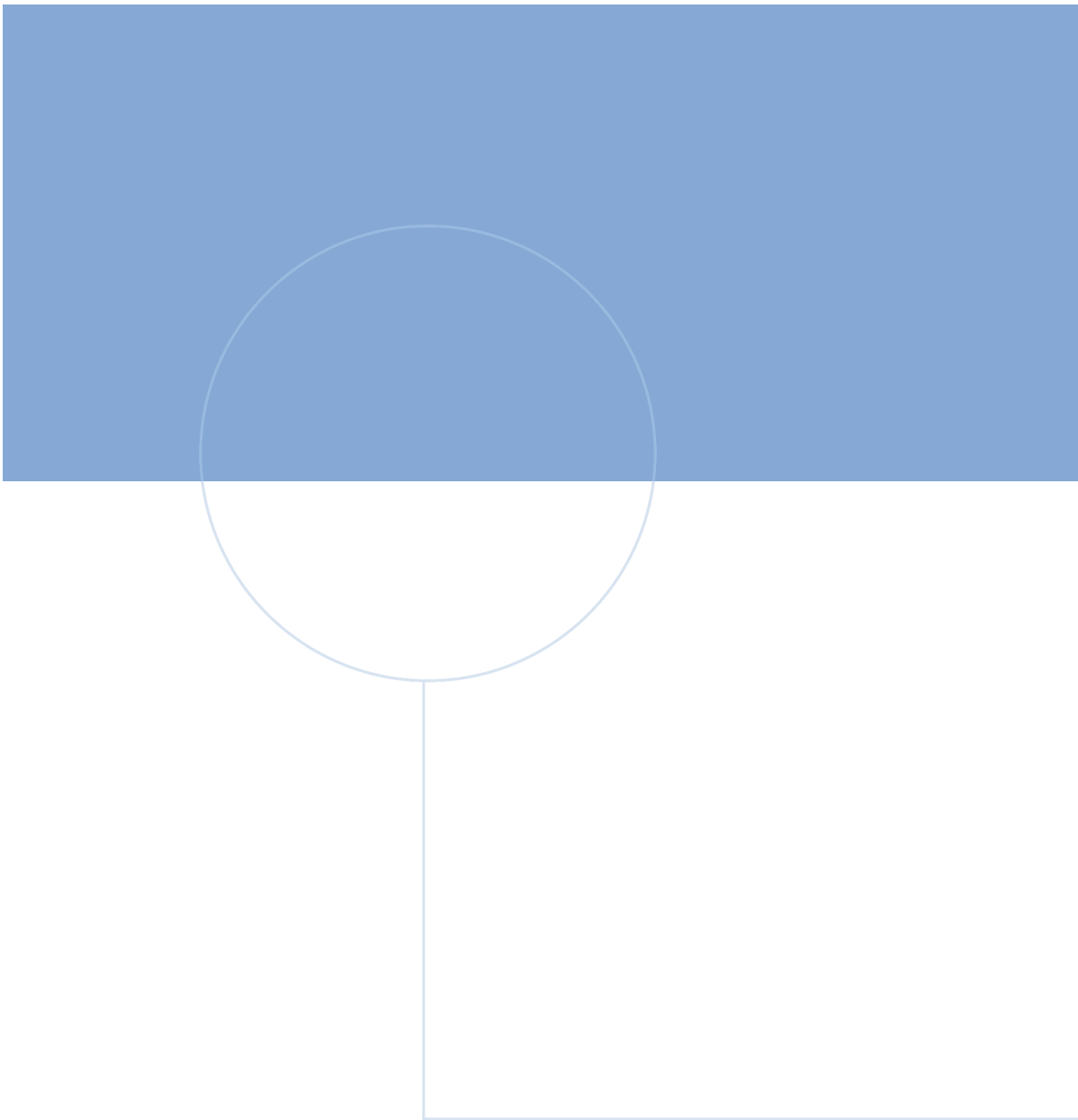
```
# RF classification report
from sklearn.metrics import classification_report
target_names = ['0 - Malicious', '1 - Benign']
print(classification_report(ds_custom_reordered_y_test,
   reordered_y_pred_rf, target_names=target_names))
```

```
                precision recall    f1-score  support

 0 - Malicious      0.06      1.00      0.12        64
    1 - Benign      1.00      0.00      0.00       986

     accuracy                           0.06      1050
    macro avg       0.53      0.50      0.06      1050
 weighted avg       0.94      0.06      0.01      1050
```

```
# CNN classification report
from sklearn.metrics import classification_report
target_names = ['0 - Malicious', '1 - Benign']
print(classification_report(y_cnn, y_pred_cnn,
   target_names=target_names))
```

```
                precision recall    f1-score  support
```

0 - Malicious	0.06	0.98	0.12	64
1 - Benign	0.97	0.03	0.06	986
accuracy			0.09	1050
macro avg	0.51	0.51	0.09	1050
weighted avg	0.91	0.09	0.06	1050



NTNU

Kunnskap for en bedre verden