UiA University of Agder

Master

A Machine Learning Approach for Intrusion Detection

AMIR SADIQ ABOHAIKEL AND ERIK TOKHEIM

SUPERVISOR
Morten Goodwin

University of Agder, 2020
Faculty of Engineering and Science
Department of ICT

Acknowledgements

Conducting a machine learning approach to achieve a proper detection performance of a Network Intrusion Detection System; we want to thank our supervisor, Dr Morten Goodwin, for guidance and feedback. We would also thank Sigurd Brinch for helping us getting access to the hardware resources used in this thesis.

I, Amir Sadiq Abohaikel, I would like to thank my family. Namely, my parents and my siblings for supporting me spiritually in writing this thesis.

I, Erik Tokheim, I would like to give an appreciation to my family for supporting me finishing this thesis.

Abstract

Securing networks and their confidentiality from intrusions is crucial, and for this reason, Intrusion Detection Systems have to be employed. The main goal of this thesis is to achieve a proper detection performance of a Network Intrusion Detection System (NIDS). In this thesis, we have examined the detection efficiency of machine learning algorithms such as Neural Network, Convolutional Neural Network, Random Forest and Long Short-Term Memory. We have constructed our models so that they can detect different types of attacks utilizing the CICIDS2017 dataset. We have worked on identifying 15 various attacks present in CICIDS2017, instead of merely identifying normal-abnormal traffic. We have also discussed the reason why to use precisely this dataset, and why should one classify by attack to enhance the detection. Previous works based on benchmark datasets such as NSL-KDD and KDD99 are discussed. Also, how to address and solve these issues. The thesis also shows how the results are effected using different machine learning algorithms. As the research will demonstrate, the Neural Network, Convulotional Neural Network, Random Forest and Long Short-Term Memory are evaluated by conducting cross validation; the average score across five folds of each model is at 92.30%, 87.73%, 94.42% and 87.94%, respectively. Nevertheless, the confusion metrics was also a crucial measurement to evaluate the models, as we shall see.

Keywords: Information security, NIDS, Machine Learning, Neural Network, Convolutional Neural Network, Random Forest, Long Short-Term Memory, CICIDS2017.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, the use of the Internet is growing tremendously. As of January 2020, the number of people using the Internet reached about 4.54 billions [36]. It indicates that the number of computers and systems connected to the outside world is significant, which introduces vital security concerns. Since there are no perfectly secured systems, security components such as Network Intrusion Detection Systems (NIDS) have to be introduced.

A significant challenge that has been a concern since the first IDS was introduced is the False Positive Rate [30]. Researchers have been working with this issue as it adds much burden to security analysts due to the high number of false alarms. Such a burden can lead analysts to ignore severe cyberattacks unintentionally. IDSs have to get continuously improved as networks change all the time. Hence, when changes occur in the network, new types of attacks emerge.

To solve the mentioned challenges, researchers have been working on improving intrusion detection systems by introducing machine learning techniques. In the case of intrusion detection systems, machine learning algorithms rely on analyzing massive data sets to gather useful information, such that they can detect abnormal behavior on the network. The information gathered from the data sets can be used to enhance detection systems. It can be achieved by training the algorithms; hence, allowing the security analysts to gain the desired level of satisfaction in regards to the False Positive Rate. Additionally, machine learning algorithms do not rely on the knowledge of the domain, making them easy to design and construct.

The purpose of this thesis is to benefit from some of the popular machine-learning-based IDSs algorithms. Namely, Neural Network (NN), Convolutional Neural Network (CNN), Random Forest (RF) and Long Short-Term Memory (LSTM). The reason behind that is many research papers such as [81][28][62][17] use old datasets (KDD99 and NSL-KDD) with a small number of features and old attacks. Moreover, some of these papers train only a small proportion of the dataset and the analysis merely relies on normal and abnormal traffic, which leads to higher and unreasonable results. The utilized dataset in this thesis is called CICIDS2017. It is experimented using the mentioned machine learning algorithms to predict future attacks by training the models so that they are able to detect the various types of attacks, and not only normal and abnormal traffic. This dataset is newer, contains more sophisticated attacks, more prominent and realistic. Additionally, a more significant proportion (80%) of the dataset is utilized for the training part, which leads for more reasonable results.

This dataset was produced by the Canadian Institute for Cybersecurity (CIC) in 2017.

1.1 Motivation

Today's networks are not perfectly secured [30], as new technologies emerge, and continuous change in network infrastructure occurs, new security challenges appear. Therefore, to cope with these challenges, multiple layers of security have to be designed securely, i.e., an appropriately defence-in-depth infrastructure has to be deployed.

One of these security layers is the Network Intrusion Detection System. An IDS helps in notifying if there is an ongoing sophisticated attack. Alternatively, if an attack was conducted earlier and by whom, indicating that it also helps in identifying the adversary and its actions.

Intrusion Detection Systems have to be continuously enhanced so that they are up-to-date and can detect new attacks. However, although a high number of works have aimed to improve intrusion detection, there are still challenges to build such systems with high efficiency. For this reason, this thesis will mainly focus on performing multi-class detection, not just to merely detecting bad connections but also to detect the attack type, benefiting from different machine learning algorithms.

1.2 Goal

Intrusion Detection Systems (IDS) suffer from high False Positive Rates and unpredictable attacks. Thus, the thesis mainly aims to improve the predictability in terms of the anomaly-based intrusion detection systems. The goals are to analyze a dataset and make use of machine learning to predict future intrusions and achieve more reasonable detection results.

1.2.1 Field of research

Which type of Machine Learning algorithms plays a vital role in how accurate the detection is. However, many researchers have conducted different approaches to improve Intrusion Detection Systems. Each algorithm has both unique pros and cons, accuracy, and a distinct level of efficiency. The focus of the thesis is on which machine learning technique is most efficient in regards to traffic types represented in the CICIDS2017 dataset.

1.3 Hypotheses - Statement of the problem

- Statement 1: How to enhance the network intrusion detection system using Machine Learning algorithms?

- Statement 2: Why CICIDS2017? How are the results going to be affected when training the dataset?

1.4 Contributions

As briefly stated in the introduction, many papers do only use a small proportion of the dataset (a typical proportion is 10%), which leads to high and unreasonable accuracy. In this thesis, instead of using old benchmarks datasets such as Cup KDD'99 and NSL-KDD, the group focuses on using a more significant proportion for training of a newer (new attacks and more features) dataset witch is the CICIDS2017. It is expected that the results lead to more reasonable outcome compared to other papers. Additionally, when the classification is based on identifying normal and abnormal traffic, the detection is then more straightforward. Therefore, a better approach would be a classification by attack, i.e., identifying what type of intrusion is occurring, which is covered in this thesis.

1.5 Report structure

Chapter one briefly introduces the IDS, why it is needed, and the challenges associated with it along with how this thesis aims to solve these challenges.

Chapter 2, or background, talks about all background information required in this thesis. Background defines information security, intrusion detection systems and their working procedures, types of attacks, and both commercial and open-source IDSs. Additionally, what machine learning is, most popular types and how they are utilized.

Furthermore, State-of-the-art (Chapter 3) discusses the taxonomy of anomaly detection types and how they are implemented in machine learning. It describes different algorithms types, how they are implemented in IDS, and their benefits and limitations. Additionally, five distinct research examples (Neural Networks, Convolutional Neural Network, Random Forest, Long Short-Term Memory and Bayesian Network) and their results are demonstrated and reviewed.

Moreover, Chapter 4 studies the CICIDS2017 dataset. This chapter provides a comprehensive analysis of CICDS2017 dataset. It analyzes the attacks CICIDS2017 consists of, including detailed information about each attack and when the attacks were conducted. In addition, details about how the data is gathered, what systems used to collect the data, the features and records of the dataset, and mainly why CICIDS2017.

The following chapter, which is approach, describes a design science approach that is followed to deliver a comprehensive explanation of the taken steps to construct the models. This chapter explains the preprocessing procedures, the implementation steps of each algorithm, how the models are evaluated and what tools used to develop the models.

Testing and evaluation is then shown in Chapter 6, it shows both confusion metrics and cross-validation results.

Finally, Chapter 7 discusses the conclusions drawn from the work and observations of this thesis. Further work is also included.

Chapter 2

Background

2.1 Information security

Information security revolves around going through certain phases to strengthen the security posture in a system [38]. As a goal, information security attempts to protect Confidentiality, Integrity, and Availability (the CIA triad). Confidentiality means the information is not readable by those who are not authorized, where the goal of integrity is to protect data from being modified. Availability is the ability to access certain information when needed by those who are authorized.

2.1.1 Information security process

In the process of information security, there are three main categories, which are prevention, detection, and response. In order to have a secure system, each phase requires maintenance, analysis, and organizing strategies to move to the next phase.

Prevention

In this phase, security policies, awareness training, and access controls must be designed and conducted to prevent attacks [52]. These procedures have to be implemented early on as they are related to each other. Security policies are high-level security measures conducted by organizations to achieve desired security objectives. Moreover, security policies are based on three main categories, which are physical controls, logical controls, and administrative controls.

Awareness training is a very critical control [51]. Organizations always try to educate their employees to avoid being victims of cyberattacks. Awareness training programs highlight the importance of security, how to avoid being a subject for attacks, providing knowledge of best practices (passwords, email, remote work, secure browsing, etc.), how to report a security issue, and so on.

Access control [38] provides an identity and a specific level of authentication and authorization to each user. An identity is a unique identifier, and in order to use specific resources in a system, the identity has to be authenticated or validated by three main factors, "something you know, something you are and something you have." Based on the provided information, a certain level of authorization will be given.

Detection

This phase is an essential one, as defending the network against malicious attacks is one of the most critical procedures and must be handled by network administrators and security analysts [38]. One technology that can be used to discover intrusions is an intrusion detection system (IDS). As mentioned earlier, intrusion detection systems always have to be improved because no matter how secure the system is, there will exist attacks that are capable of compromising the system. An IDS is capable of detecting a conducted attack, for instance, by checking the signatures, and modified files and configuration. Nevertheless, when an attack or a breach occurs, the IDS alerts the network's administrators, then they have to follow a response plan, as will be discussed shortly.

Response

Organizations have always to be prepared for an incident to defend their systems [50]. It can be achieved by establishing an incident response strategy. A response plan must describe which procedures to be taken during an incident. Furthermore, for each type of incident, there must be a specific type of response depending on the threat level. During the response phase, several steps must be conducted, including containment, eradication, and recovery. These steps revolve around selecting a strategy to contain an attack, gathering shreds of evidence to support incident response documentation, identifying attackers, and eradicating the incident impact on business operations.

2.2 Intrusion detection systems

As shown in Figure 2.1, Intrusion Detection System classes are based on three main modules, data source, data analysis, and response [5]. The first module is the data source, this is where the data or the traffic can be gathered using two main techniques, host-based IDS, where the data gathered is only about one individual, and network-based, where the data is obtained from the entire network. The second module is the data analysis. Here, there are two main techniques, misused-based (signature-based) and anomaly-based in which will be discussed in detail in forthcoming sections. The third module is called a response, and this is where an appropriate response is performed based on given data, and the response could be either active or passive.

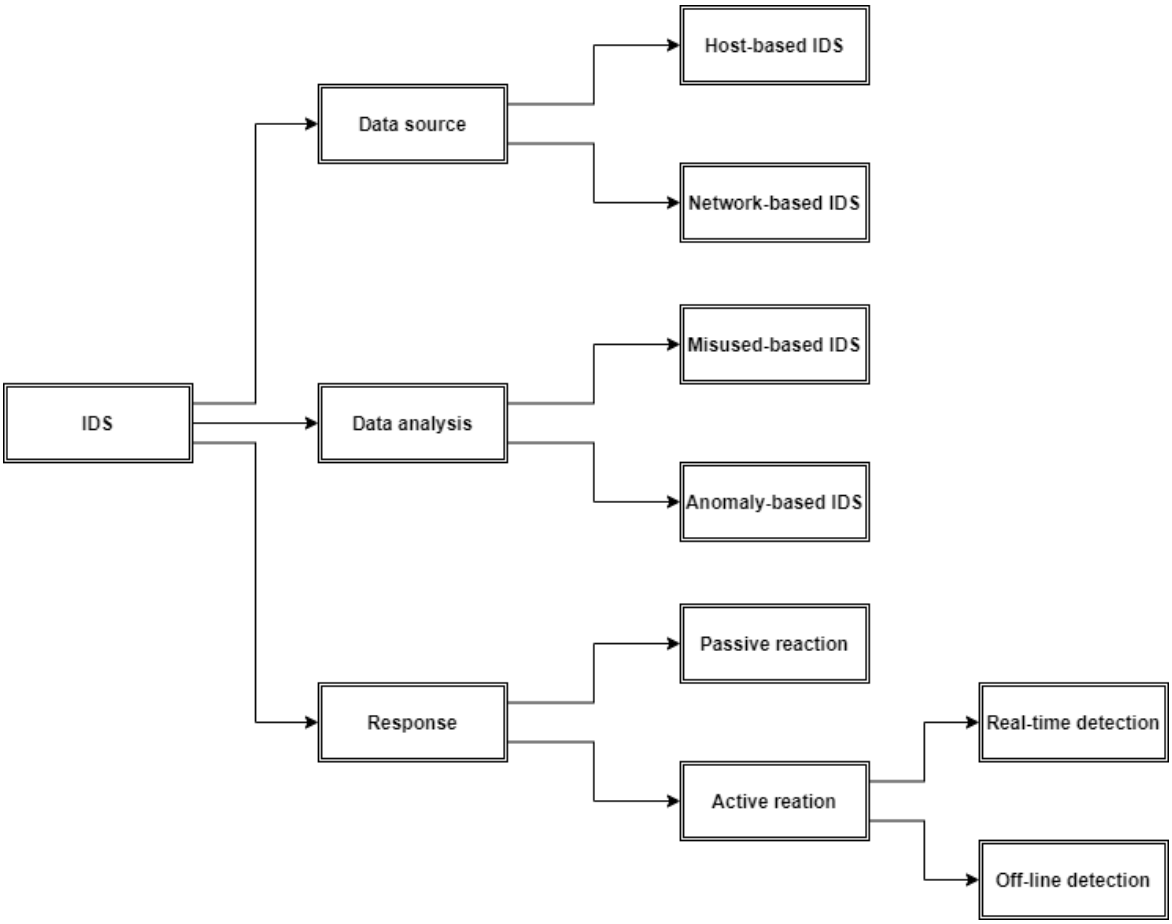

Figure 2.1: IDS classes

2.2.1 Data source

Network-based IDS

In general terms, a Network Intrusion Detection System (NIDS) [94] is placed in a location where it is able to read all incoming traffic, such as the network's edge. The reason is to passively detect attacks that target the most vulnerable points. In order to detect a particular attack, different traffic patterns are compared to a list of known attacks.

Host-based IDS

A Host Intrusion Detection System (HIDS) [94] relies mainly on being installed on end-points. Thereby, all devices connected to the network are monitored. A HIDS provides the ability to take a closer look at the traffic generated at each device. As well as being an additional layer of security that detects malicious traffic that was failed to be detected by the NIDS.

The way a HIDS operates is it takes a look at the entire system and bases its conclusion on a snapshot taken previously. If in any case, it detects suspicious behavior that is not normal, it notifies the administrator. However, suspicious behavior could be data and settings modification, data loss, and other things.

NIDS vs HIDS

	NIDS	**HIDS**
Pros	- The detection is done using network packets. - One does need to install the software on each host. - Checks multiple hosts at the same time. - Detects all network protocols.	- Checks encrypted communication behavior. - One does need extra hardware to use HIDS. - Intrusions are detected using filesystem, events and system calls. - Checks item by item, not streams only.
Cons	- Encrypted traffic is a problem, as it is challenging to identify attacks. - Hardware is required. - Analysis challenges in high-speed networks. - The biggest threat are insider attacks.	- Takes time to report attacks. - Requires resources from the host. - Has to be installed on each host. - Monitors attacks on where it is installed.
Data source	- Simple Network Management Protocol (SNMP). - TCP/UDP/IP. - Management Information Base (MIB). - Netflow.	- System calls. - Rule patterns. - Logs. - Application Program Interface (API). - Audits records.

Table 2.1: Network Intrusion Detection System vs Host Intrusion Detection System [6]

2.2.2 Data analysis

Anomaly-based IDS

Anomaly-based IDS is based on identifying unusual data traffic that diverge from the expected behavior [59]. Anomaly detection has many applications; for instance, in business, if a suspicious pattern is detected, it could be a sign of a hack. Anomaly detection has other purposes in healthcare systems, such as detection of any unusual patterns in an MRI scan. This technique can also detect faults in operating environments.

There are three main types of anomalies that are categorized as follows:

- **Point anomalies**: if a single point of data is away far from the rest, then it is anomalous. For instance, say in business, this detection type discovers credit card fraud based on the spent sum.
- **Contextual anomalies**: as the name implies, this detection type is context-based. If one usually spends 100$ during the holidays, then it would not be normal otherwise.
- **Collective anomalies**: it is based on collecting data to analyze the anomalies. For instance, if an attacker conducts an infiltration attack (i.e., stealing data from a system), this action is then flagged as a cyberattack.

Misuse-based IDS

Misuse-based IDS (signature-based IDS) [66] looks for pre-defined signatures stored in a big database to identify attacks. This detection model requires to be continually updated with new signatures to detect new attack patterns. An example would be sending an e-mail containing a specific virus. As long the signature is known by the IDS, it will be detected.

Misuse-based IDS can be implemented based on of the following approaches:

- **Expert systems**: this is based on the if-then approach, implying that if all if-conditions are satisfied, an alarm of misuse will be activated.
- **Model based reasoning systems**: an existing database that comprises different attack patters is used to distinguish an attack scenario. In this approach, the IDS will gather data about a series of various behaviors that describe a particular attack.
- **State transition analysis**: a series of state-transitions that describe the attacks based on state-transition diagrams [77].
- **Key stroke monitoring**: this technique is based on registering what is typed on the keyboard. The reason is to detect if there is an attempt to violate security policies. It particularly looks for a sequence of commands that caused a violation in the system. For example, it is capable of detecting a series of commands that lead to a system compromise [77].

Misuse-based IDS vs anomaly-based IDS

	Anomaly-based IDS	**Misuse-based IDS**
Pros	- Efficient against unknown attacks and Zero-day attacks. - Monitors any data source. - Identifies rogue users.	- Not complicated. - Detects known attacks. - Efficient if the signatures are well known.
Cons	- Many false positives. - It requires the analysts to figure out what triggered an alarm.	- All signatures must be up-to-date. - Large number of signatures. - Can only detect known attacks. - If a known attack has many variations, it may not be detected.

Table 2.2: Anomaly-based IDS vs miseuse-based IDS [66][59]

2.2.3 Response

Passive reaction

A passive IDS [23] analyzes the logs that an attack generates, and notifies the system's administrator. This type of reaction is not capable of any actions, such as correcting

issues caused by an attack. Thus, it is not ordinarily susceptible to attacks itself.

Active reaction

An active IDS [23] is capable of making decisions and performing actions against an attack. Such a system is called Intrusion Prevention System (IPS). An IPS is capable of blocking attacks automatically if an attack occurs. IPSs are generally placed at the boundary of the network. Thus, the IPS itself is susceptible to cyberattacks. Moreover, it can be used as a part of a Denial Of Service attacks by making it sending a vast number of alarms. Therefore, a well-designed alarm system has to be configured, such that it does not send many false alarms. However, such a system has an advantage where it detects attacks in real-time and reacts to them accordingly.

2.2.4 IDS architecture

In Intrusion Detection Systems, there are a diversity of techniques to gather data [7], but in general, as shown in Figure 2.2, IDSs consist of the following:

- **Data gathering (sensors)** is device that is responsible for gathering information from the system.
- **Detector ID - Engine** analyzes the data collected from the sensors to identify any attacks.
- **Knowledge base (database)** is the component where the IDS contains information about traffic collected by the sensors. Security professionals usually provide such information.
- **Configuration device** tells something about the state of the Intrusion Detection System.
- **Response component** is responsible for initiating actions when an attack or intrusion is detected. The responses are either passive or active.

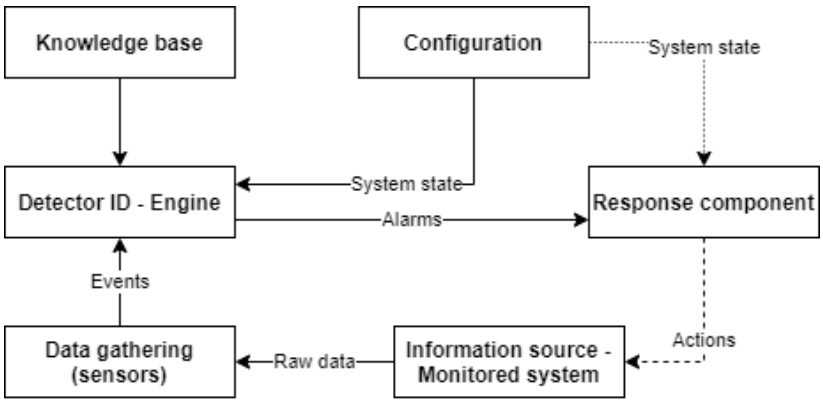

Figure 2.2: A simplified IDS architecture

2.2.5 Efficiency of IDS

Prediction performance

Two criteria have to be satisfied to have an excellent prediction performance. Firstly, the IDS must not identify a legitimate behaviour as an intrusion. Secondly, it must correctly be able to identify if a behaviour is an intrusion. Such measures can be evaluated by determining the False Positive Rate and Detection Rate (also called Recall or True Positive Rate [27]). Detection Rate is calculated by identifying the number of detected attacks. In contrast, False Positive Rate is measured by identifying the number of normal connections that are classified as attacks. Practically, these two measures are challenging to evaluate due to the difficulty to have a global knowledge of existing attacks. For this reason, an IDS is evaluated by performing Receiver Operating Characteristics (ROC) analysis, or on what is called F1-score, depending on the scenario [37]. The ROC analysis is a representation of the trade-off between both False Positive Rate and Detection Rate. As shown in Figure 2.3, when the ROC is close to the upper left corner, it means that the IDS detection is effective [7].

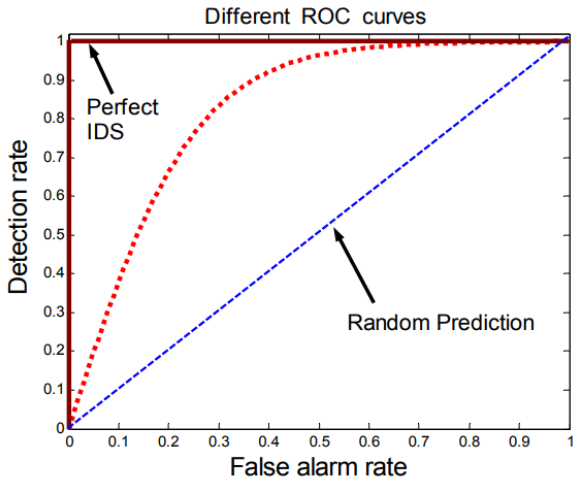

Figure 2.3: Recover Operating Characteristics (ROC) curves

		Predicted connection label	
		Normal connections	**Attacks**
Actual connection label	**Normal connections**	True Positive (TP) - The prediction is correct.	False positive (FP) - The prediction is positive but it is false.
	Attacks	False Negative (FN) - The prediction is negative and it is false.	True Negative (TN) - The negative prediction is correct.

Table 2.3: Intrusions evaluation

Time performance

Time performance [7] depends on the total time the IDS needs in order to detect an attack (intrusion). This time is measured using two values; propagation time and processing time. The propagation time is the time it takes for processed information to be sent to cybersecurity analysts. The processing time depends on the IDS's speed to audit occurred events. These two times have to be as short as possible so that analysts have the sufficient time to react to the adversary's actions.

Fault tolerance

Intrusion detection systems have to be fault-tolerant [7], meaning that IDSs have to be robust and resistant to attacks conducted by adversaries. Moreover, IDSs must be able to recover from an attack and get back to normal and provide secure services immediately. IDSs have also to be resistant to attacks that target the IDS itself, such as buffer overflow attacks and similar attacks that can shut down the IDS. However, it is also essential to have the ability to be resistant to attacks that aim to make the IDS generate a massive number of false alarms which causes a denial of service.

2.2.6 Evaluation Metrics

Confusion Matrix

As the name suggests, confusion matrix gives an output matrix that describes the performance of the model [8]. As shown in 2.3, a confusion matrix rely upon four important terms; TN, FP, FN and TP. To measure the accuracy of a model, one has to use Equation 2.1, where *TotalNumberOfSampels* is equal to *TN+FP+FN+TP*.

$$Accuracy = \frac{TP + FN}{TotalNumberOfSamples} \tag{2.1}$$

Area Under Curve (AUC)

Area Under Curve [8] tells something about the capability of the model of distinguishing between classes. The higher the AUC, the better the model. In order to calculate the AUC, one must calculate True Positive Rate (TPR) and False Positive Rate (FPR) as shown in Equations 2.2 and 2.3, respectively.

TPR corresponds to the number of positive data points that are correctly considered as positive. However, FPR corresponds to the number of negative data points that are considered as positive.

$$TPR(Sensitivity) = \frac{TP}{TP + FN} \tag{2.2}$$

$$FPR(Specifity) = \frac{FP}{FP + TN} \tag{2.3}$$

Both TPR and FPR have values located between 0 and 1. AUC is the area under the curve as shown in Figure 2.3 in section 2.2.5. In addition, the greater the value the better the performance.

Precision, Recall and F1-score

The score of F1 precisely tells how many instances were correctly classified. In addition, it tells how robust the model is as it does miss only a minimal number of instances [8]. However, Precision is the number of true positives divided by the number of true positives and false negatives (number of positive results). Furthermore, Recall is the number of true positives divided by true positives and false negatives (all relevant samples that have been identified as positive). Note that Recall and TPR (sensitivity) are the same. The equation of F1 score is shown in 2.4, and the precision and recall are shown in 2.5 and 2.6 respectively.

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}} \tag{2.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

2.2.7 Differences between firewall, IDS and IPS

The main difference between these three technologies [94] is that a firewall blocks and filters traffic. In contrast, an IPS or IDS detects and notifies the security analyst/administrator; it can also prevent attacks depending on the configuration. The placement of each device in a network is shown in 2.4.

A firewall is configured such that it contains a set of rules that describe what kinds of traffic are allowed to pass through. Firewall rules rely mainly on source and destination IP address, in addition to port numbers. If a packet does not meet the specified criteria (rules), then it is denied.

As mentioned in section 2.2.3, an IDS is a passive device that captures and analyzes the traffic coming into the network. In terms IPS, it is an active device that is able to prevent attacks by blocking them.

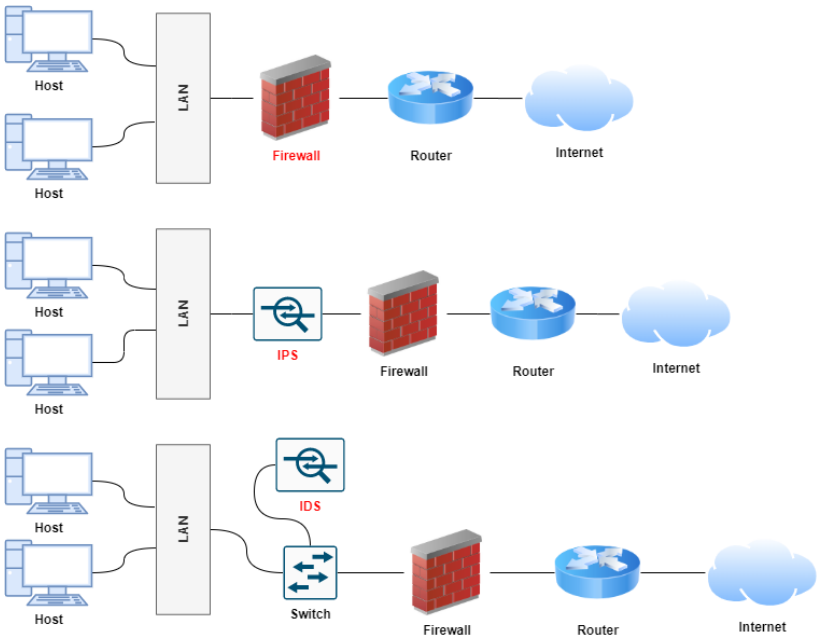

Figure 2.4: The placement of IDS, IPS and FW

	Firewall	IPS	IDS
Definition	A device that filters incoming and outgoing traffic based on specified rules.	A device that detects, inspects and distinguish the traffic. Then, it proactively stops malicious packets.	A device that monitors a certain system. If it finds any suspicious behaviour, whether it is a malicious or policy violation, it will alert the administrator that there is an issue.
Functionality	The traffic is filtered or blocked based on IP addresses and port numbers.	It inspects the traffic and make a decision if there is an attack. If an attack is occurring, then it is prevented by the IPS.	The same as IPS, but the difference here is that instead of preventing attacks, it generates alerts.
Configuration	Layer 3 mode. Transparent mode.	Layer 2 mode. Inline mode.	Inline mode. End host mode.
Placement	Inline at the edge of the network. Firewall should be first line of defense.	Inline, typically after the FW.	Non-inline . IDS should be placed after the FW.
Traffic patterns	Traffic patterns in FW are not analyzed.	Traffic patterns in IPS are analyzed.	Traffic patterns in IDS are analyzed.
Actions	Blocks the traffic based on specific rules.	Prevents the traffic on anomaly detection.	Alerts on anomaly detection.
Related terms	- Traffic is permitted or blocked based on IPs and ports. - Stateful packet filtering.	- Signature-based detection. - Anomaly-based detection. - Blocking the attacks. - Zero day attacks.	- Signature-based detection. - Anomaly-based detection. - Zero day attacks. - Monitoring. - Alarm.

Table 2.4: Firewall vs IPS vs IDS [24]

2.3 Types of attacks

2.3.1 Scanning Attacks

A scanning attack [94] is an attack that attempts to send packets of information to a network system to gather information about the topology. It involves looking for ports which are either open or closed, what type of traffic is permitted and not permitted, which hosts are active or even the type of hardware running on different devices. For instance, a type of attack that finds weak points in a network is Blind SQL injection attacks. A Blind SQL injection attack is an attempt to ask a database questions that make it respond by a Boolean value to find vulnerabilities. These types of attacks often attempt to find open ports to be exploited by injecting malicious code or malware.

2.3.2 Asymmetric Routing

When packets take a specific route to the destination, and a different route back to the source, this behaviour is called asymmetric routing [94]. This behaviour is normal in general, but it is unwanted. The reason behind that is adversaries can benefit from asymmetric routing by sending malicious data through particular parts of the network to bypass security systems, depending on firewalls configuration. If the network is allowed to perform asymmetric routing, then it is exposed to attacks such as SYN flood attacks. An SYN flood attack is an attack that attempts to open many connections without closing them (half-open attack), which leads to a total consumption of system or server resources so that it becomes unresponsive. This attack is a DDoS attack type, and one reason to deactivate asymmetric routing in the network.

2.3.3 Buffer Overflow Attacks

Buffer overflow [94] attacks attempt to replace normal data with malicious data in penetrated memory parts, such that a malicious code gets executed later on. In generic terms, a buffer overflow attack writes more data in the memory's buffer than it can handle; performing this action results in making the data overflow into the neighbouring memory.

2.3.4 Protocol-Specific Attacks

As the name implies [94], these attacks is conducted using specific protocols such as ICMP (Internet Control Message Protocol), TCP (Transmission Control Protocol) and ARP (Address Resolution Protocol).

ICMP is a protocol that can be used the devices to communicate with each other. For instance, the ping and traceroute commands use ICMP packet to determine if a device is on a network and to determine the path to a certain device, respectively. ICMP protocol is used by attackers to conduct attacks called ping floods. These attacks revolve around over flooding the network by these packets so it becomes unresponsive by consuming the bandwidth. Additionally, ICMP is also used to what is called tunnelling attacks where the firewalls are bypassed, such as smurf attacks (an attack where the source address is spoofed) and port scanning attacks (the error messages generated by the ICMP are used to determine if a port is open or not).

TCP is one protocol that is often used to conduct attacks. Such a protocol can be used to conduct, for instance, a SYN flood attack as mentioned in 2.3.2.

ARP is also used for flooding attacks. For instance, an adversary can send a massive number of ARP requests/packets to fill up ARP tables with data. An attack called ARP poisoning is often used to link the adversary's device to victims device by luring the network into believing that the adversary owns a MAC address which is not his. By performing this action, the adversary is able to intercept packets directed to the legitimate owner of the MAC address. It results in making the attacker modify or stop these packets.

2.4 Commercial and open source intrusion detection systems

Many software variants provide IDS/IPS functionalities. Therefore, this section revolves about describing only two of the most popular commercial and open source systems, which are McAfee NSP and Snort, respectively.

2.4.1 Snort

Snort [10] is open-source software for intrusion prevention and detection. Snort performs real-time analysis detection and packet logging. It is capable of detecting malware, probe attacks, exploits and harmful threats. There are three main methods to configure snort; network intrusion detection, packet logger and sniffer. In network intrusion detection mode, snort will monitor the traffic in real-time and compare it with the rules determined by the administrator. Packet logger mode logs network packets and stores the data. In the sniffer mode, the program will inspect the packets and show the information to the administrator. A snort rule is represented in Figure 2.5.

What makes Snort a good software is it is open-source. It provides the simplicity in writing rules that determine what traffic is allowed to pass through. Moreover, the flexibility of deployment and good community support to solve issues rapidly.

On the other hand, many people may be a bit frustrated because there is no GUI to set up the rules. Besides, it is relatively slow in terms of processing network packets.

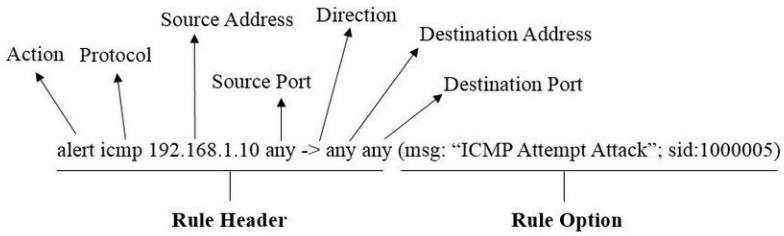

Figure 2.5: A snort rule [55]

2.4.2 McAfee NSP

McAfee NSP (The McAfee Network Security Platform) [49] is a commercial tool used for intrusion and threat prevention. It provides many features such as SSL encryption, learning engines and behavioural analysis, automation, high performance, and so on. In simple terms, McAfee NSP is like any IPS/IDS system, but the difference is it performs better as it is developed by a more prominent company with many professionals.

2.5 Machine learning methods

2.5.1 Supervised learning vs Unsupervised learning

Supervised learning [80] is a technique used to train a machine using labelled datasets. As the name implies, it means that some of the labelled data is tagged with the correct answer. The way supervised learning works is by training the labeled data, one can predict unforeseen results. For instance, suppose that one wants to train a machine to predict how long it does take between from a location to another; specific data must be gathered and analyzed. Namely, such data may include weather conditions, holidays, time of the day, and route chosen. All these details are considered as inputs. Naturally, if it is raining, one assumes that it will take a longer time to reach the desired location, but a machine needs statistics. Consequently, in order to create a dataset that can get trained, specific data such as the total time it takes from a start location and corresponding data that includes time, weather condition, route, and so one. Based on the given information, the machine will be able to see the relationship between different data and predict the time it takes to travel from a location to another.

Unsupervised learning [80] is a technique where a machine learning model does not need to be supervised. The model will instead try to discover the information by itself, which means that unsupervised learning deals with unlabeled datasets. In unsupervised learning, the machine can find all types of data patterns, and it also helps in identifying the features one needs to categorize the data.

	Supervised learning	**Unsupervised learning**
Process	Input and output data are given.	Only input data is given.
Input data	The machine is trained using labeled data.	The machine is not given unlabeled data.
Algorithms used	SVM, NN, Random Forest, Linear and Logistics regression, Classification trees.	Different categoriez: K-means, Cluster algorithms, Hierarchical clustering, and so on.
Computational complexity	Simple.	Complex.
Use of data	Uses training data and relate input and output results.	Does not use output data.
Accuracy of results	Accurate and trustworthy.	Less accurate and trustworthy.
Real time learning	Learning is offline.	Real-time.
Number of classes	Known.	Unknown.
Main drawbacks	Big data is a challenge.	No precise information in regards to data sorting, and the output is not known.

Table 2.5: Supervised learning vs unsupervised learning

Semi-supervised learning

There is another type of learning, which is called semi-supervised learning [80]. This technique makes use of both supervised learning and unsupervised learning. It combines some of the labeled data with a massive amount of unlabeled data during the training phase.

2.5.2 Neural Networks (NN)

In simple terms, a Neural Network [53] is a network that contains a number of neurons used to process information. A Neural Network consists of three main components; input layer, hidden layers and output layer, as shown in Figure 2.6. Moreover, each of these layers consists of neurons, connections and weights, propagation function and a learning rule.

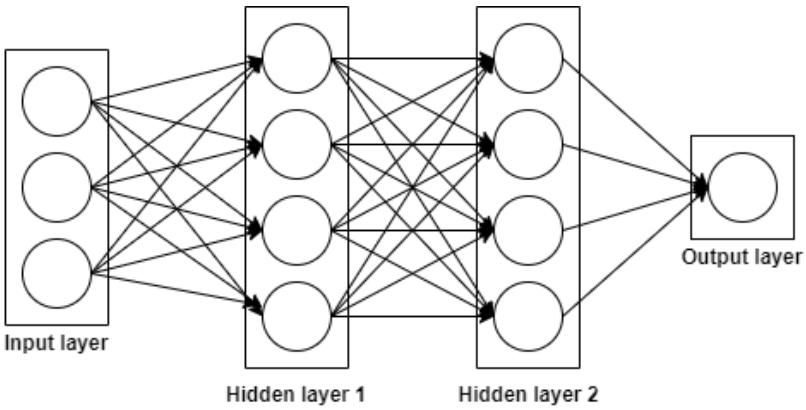

Figure 2.6: Neural network components

Neurons (nodes) are what the Neural Networks are built upon. Each neuron in a Neural Network has an input and an output. It means that input neurons and output neurons represent a Neural network. In regards to the input layer, the neurons it consists of have outputs but does not have inputs. However, when it comes to the output layer neurons, they have inputs but no successor outputs.

The connections and weights are essential parts of the Neural Network. The connections transfer the output from, say hidden layer 1, to hidden layer 2, which in this case, the input of hidden layer 2 neurons. Indeed, each of these connections has a weight which represents in simple terms the importance of the input.

A propagation function is a function used to compute a neuron's inputs. This function is typically used when training the data.

The learning rule function is essential when determining and modifying the weights of the connections. It helps in achieving the desired output from the Neural Network. This functions, just as the propagation functions, is used during the training phase.

The way a Neural Network operates is [42]; firstly, the input layer is the layer where information is given to the network, and each of these circles represents a feature. Secondly, the hidden layers are those layers that process the data given. The number

of layers may vary; one can choose as many layers as needed to process the data. In generic terms, the more hidden layers, the more accurate the results. Each of these layers consists of neurons that receive information from the previous layer's nodes. When the information is received, the information is then multiplied with weight, and a bias is added. Finally, output layers gather the information processed in the last hidden layer of the network and produces the desired output. As shown in Figure 2.7 and equation 2.7, a mathematical description of NN is represented, where w is the weight, x is input vector, b is neuron bias, $\odot$ is the element-wise multiplication, f is the activation function and y is the neuron output.

$$y(x) = f(w \odot x + b)[45] \tag{2.7}$$

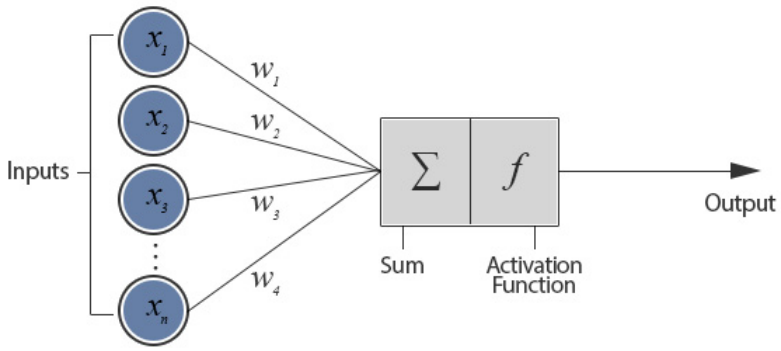

Figure 2.7: How a neurons operate [45]

2.5.3 Long Short-Term Memory (LSTM)

Before describing what an LSTM is, Recurrent Neural Networks (RNN) have to described as the LSTM is a variant of the RNN. The main idea behind RNNs is to benefit from the sequential information [60]. In regular Neural Networks, the assumption is that inputs and outputs are independent of each other, and in many case scenarios, this is a bad idea. The reason is that if one wants to predict a particular value, information about the previous one is essential to have. The RNN is called recurrent due to its ability in performing computations based on the given information in the "memory", including information about what has been considered in the calculation so far.

LSTM [21] is an RNN variant and can learn from long term dependencies. As the name implies (Long Short-Term Memory), this algorithm is capable of remembering given information for long periods. It operates by performing three main step processes called gates; Forget gate, Input gate and Output gate. A complete overview of LSTM is shown in 2.8.

- **Forget gate** is what makes a decision about how much of the past to remember. It determines what information to remove from the cell in a particular timestamp which decided by the *sigmoid* function (or a squashing function which limits the output to a range between 0 and 1 in order to predict the probability). As shown in Figure 2.9, it checks the previous state h_{t-1} and the given input x_t, then it decides whether to delete or keep the information by outputting a number

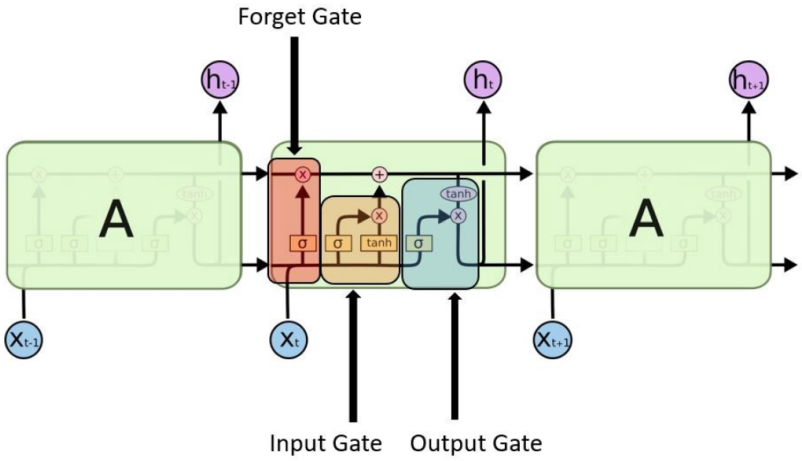

Figure 2.8: LSTM gates [60]

between 0 (delete this) and 1 (keep this) for each number in the cell state C_{t-1}. The forget gate equation is represented in 2.8.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.8}$$

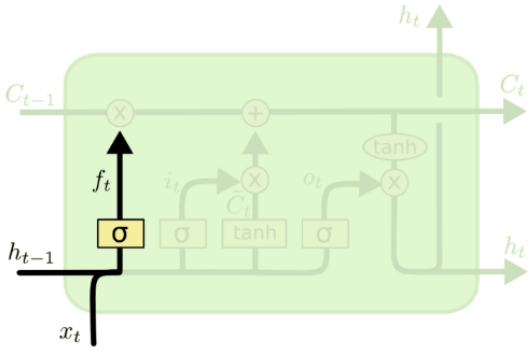

Figure 2.9: LSTM forget gate [60]

- **Update gate/input gate** is what decides how much of a particular piece of information to add to the current state. In this gate, just as in the forget gate shown in equation 2.9, the *sigmoid* function decides which value will go through and which will not. In addition, a *tanh* function, as shown in equation 2.10, is used to weight the values passed to determine their importance, represented by a specific value from -1 and 1. LSTM update gate/input gate is shown in Figure 2.10.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.9}$$

$$\tilde{C}_t = tanh(W_X \cdot [h_{t-1}, x_t] + b_C) \tag{2.10}$$

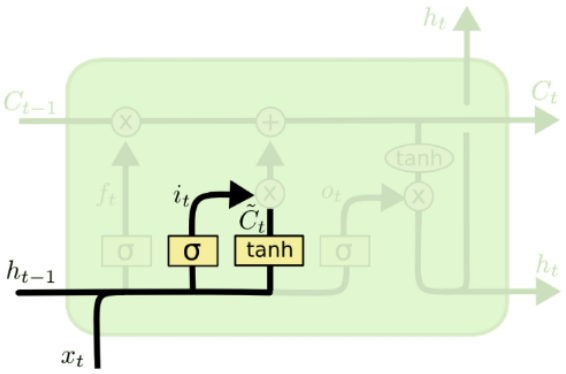

Figure 2.10: LSTM update gate/input gate [60]

- Finally, the **output gate** shown in 2.11, is the gate used to decide which piece of information will make it to the output. The *sigmoid* and the *tanh* functions are used for the same purpose as in both forget and input gates. Both of the equations are represented in 2.11 and 2.12, respectively.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.11}$$

$$h_t = o_t * tanh(C_t) \tag{2.12}$$

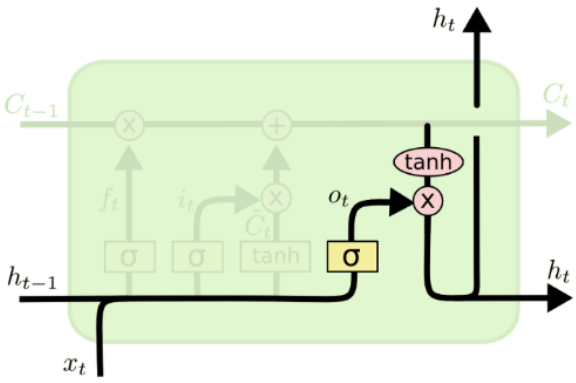

Figure 2.11: LSTM output gate [60]

2.5.4 Convolutional Neural Network (CNN)

CNN [31] is like Neural Networks as it is made up of neurons with weights and biases. Each neuron has inputs and outputs. The inputs have weights that represent the importance of the data; this data gets passed through an activation function and respond with an output.

The main difference is that CNNs function over volumes and this means that the CNN takes a multi-channelled image as the input in contrast to NN where the input is a vector. There are three main components in a CNN [82]; input layer, convolutional layer and output layer, as shown in Figure 2.12. However, CNN is capable of dealing with text as well; this will be shown in Chapter 5.

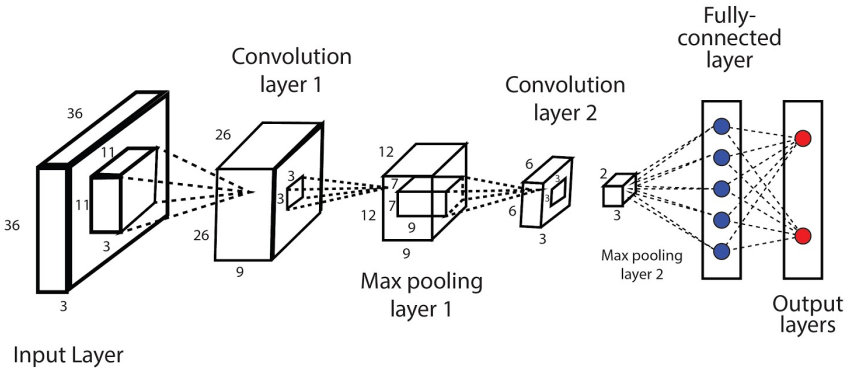

Figure 2.12: The architecture of CNN [82]

Input layers in CNN are directly connected to convolutional layers that are considered as a building block of CNNs. The primary purpose of convolutional layers is to extract the features from an image and then learn these features to help in detection techniques. The input layer contains pixel values that have both height and weight; these values are used by the filters to convolve around the input layer and provide results which return the features with fewer dimensions.

One essential function of CNN is padding. Padding is the number of pixels added to an image while processing the image by the kernel. It works by extending the image by a border of pixels of an image. The kernel of CNN moves across the image and scans each pixel and then converts the data it finds into a smaller or bigger format. This process assists the kernel is processing the image it scans, and the padding is used to add more data to the frame of the image so that the kernel can cover more space of the image. The padding of an image increases the accuracy of image analysis.

To increase the computational power in a CNN, one has to reduce the parameters, and either average pool or max-pooling does this. As the name implies, max-pooling operates by extracting the maximum value from the filter, whereas the average pooling operates by extracting the average from the filter. Hence, pooling is performed when reducing the dimensionality, whereas padding is used only when necessary.

2.5.5 Random Forest (RF)

RF is a supervised machine learning algorithm [16] that mainly operates based on the classification to solve problems. This algorithm combines multiple Decision Trees, and the more trees, the more accurate the results. These decision trees are feed with data and trained to produce outputs (predictions); the Random Forest algorithm will then choose the best prediction (solution) based on voting. Figure 2.13 shows an example of a decision tree.

A Random Forest algorithm starts with selecting random data samples from a

dataset. Then, for each chosen sample, a decision tree is built, and the prediction results are gathered from each one. When the results are gathered, a voting process is performed in order to select the best prediction result as a final solution. A simple illustration of the functionality is shown in Figure 2.14.

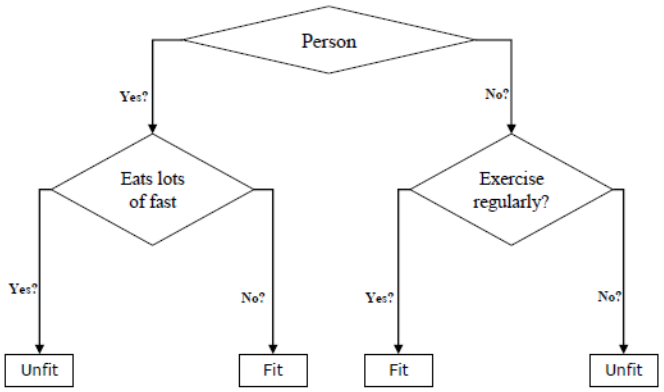

Figure 2.13: Decision tree [15]

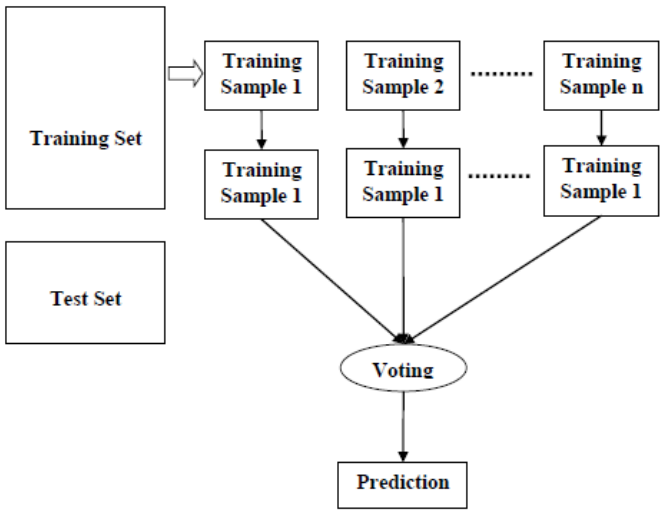

Figure 2.14: Random Forest [16]

A decision tree [64] consists of three components; the nodes where each node represents an attribute or a feature, the links that represent the rules (decisions), and leaves that represent the outcomes.

2.5.6 A comparison between NSL-KDD and KDD Cup 99 (KDD99)

KDD Cup 99

This dataset is the most known and most used one by researchers [19][54]. It is utilized by the researchers to conduct experiments on anomaly detection using machine learning algorithms. The traffic in this dataset was gathered using a virtual environment and is a subset of a dataset called 1998 DARPA (Defense Advanced Research Projects Agency).

The KDD Cup 99 is a benchmark dataset and contains three main components; the whole KDD Cup 99 dataset which contains different attacks and normal connections, 10% proportion which is used for training of classifiers, and the KDD test dataset used for testing. In total, the whole dataset consists of 4 898 431 single connection records, and each record composes 41 features labelled as attacks or normal traffic. These features are divided into four categories:

- **Basic features**: are gathered from the packet header without checking contents such as duration, protocol type, service, flag and number of bytes.
- **Content features**: the content of TCP packet analyzed and determined.
- **Time features**: provides information about the time taken (duration) from a source IP to reach the destination (i.e., target IP address).
- **Traffic features**: are based on a window that has a number of connections. Suitable for prolonged attacks.

Furthermore, KDD Cup 99 consists of four main attack categories:

- **Probe**: attacks that used to collect information about the system to find known vulnerabilities. Information gathered about such vulnerabilities is used to conduct later attacks.
- **DoS**: Denial of Service attacks that prevent authorized users from accessing the system.
- **U2R**: User to Root attacks used to exploit weaknesses in the system in order to acquire administrator privileges. For instance, the attacker starts by compromising a user account, then looks for weaknesses to escalate his privileges.
- **R2L**: Root to Local attacks are used to get access to the remote system without a user account.

	Whole dataset		10% training set		test set	
Attack category	**Number of instances**	(%)	**Number of instances**	(%)	**Number of instances**	(%)
Normal	492 798	19.86%	97 278	19.69%	60 593	19.48%
Probe	41 102	0.84%	4 107	0.83%	4 166	1.34%
DoS	3 883 370	79.30%	391 458	79.24%	229 853	73.94%
U2R	52	0.00%	52	0.01%	70	0.02%
R2L	1 126	0.02%	1 126	0.23%	16 347	5.26%

Table 2.6: A description of the KDD Cup 99 dataset

There are many criticisms to this dataset, and this is due to several issues it has:

- Calculations are complex.
- Both the training set and testing set have a high level of complexity.
- The redundancy impacts the machine learning algorithms.
- Attack traffic is high compared to normal traffic.
- Attack categories relationship is not realistic.
- The accuracy of detecting the distribution of attacks is low.

NSL-KDD

As the KDD Cup 99 contains both redundant (78%) and duplicate (75%) records, an enhanced dataset so-called NSL-KDD was released in 2009 to eliminate these problems [19]. NSL-KDD eliminated problems such as redundant records in the training dataset and duplicated records in the testing dataset by selecting a reasonable number of features from the KDD99 dataset. NSL-KDD dataset helps in increasing the performance of the classifier as it does not have any duplicated records; hence, it will not be biased with other techniques when training the machine. Additionally, since both training and testing sets contain a smaller number of instances, it would be more efficient to conduct experiments on the whole dataset instead of randomly choosing a small portion.

In the NSL-KDD dataset, there are 37 attacks in total, where 21 attacks are present in the training dataset and all of them in the test dataset. Just like in KDD99 dataset, there are four main categories of attacks in NSL-KDD; Probe, DoS, U2R and R2L.

The amount of normal traffic the training dataset consists of is 67 343 instances, and this number is out of 125 973 instances in total. In the test dataset, there are 9 711 normal traffic instances out of 22 850 instances in total.

Dataset type	Number of records					
	Records	**Normal**	**DoS**	**Probe**	**U2R**	**L2R**
NSL-KDD Train+ (full NSL-KDD train set)	125 973	67 343	45 927	11 656	52	995
NSL-KDD Train+ 20% (A 20% subset of the KDD Train+)	25 192	13 449	9 234	2 289	11	209
NSL-KDD Test+ (The full NSL-KDD test set)	22 544	9 711	7 458	2 421	200	2 754
NSL-KDD Test-21 (A subset of the KDDTest+)	11 850	2 152	4 342	2 402	200	2754

Table 2.7: Details of NSL-KDD dataset

Summary

Dataset	Features	Advantages	Disadvantages
DARPA 1998	-	- Broad range of attacks. - First dataset for evaluating IDSs	- Generating the traffic was simple due to models used - Does not reflect how real networks operate.
KDD Cup 99	- 41 features: (32 numeric) & (9 categorical)	- Used to evaluate anomaly-based systems. - Attacks in the training set are distinguished from those in the testing set.	- Redundant records and duplicate records. - Very old.
NSL-KDD 2009	- 41 features: (32 numeric) & (9 categorical)	- No redundancy and no duplication in terms of records. - Reasonable number of records compared to KDD99.	- Not good enough to represent real-world networks.

Table 2.8: A comparison between KDD Cup 99 and NSL-KDD 2009

Chapter 3

State-of-the-art

In this chapter, the taxonomy of anomaly detection types and how they are implemented in regards to machine learning will be represented. Moreover, a summary of the latest research in this field will be considered. As mentioned earlier, this thesis revolves around how to improve the anomaly detection using machine learning algorithms. However, the classification of anomaly-based detection is pretty broad, and many machine learning techniques can be applied [39][20].

3.1 Anomaly-based NIDS architecture

According to [78], a simplified architecture of Anomaly-based Network Intrusion Detection System (A-NIDS) that describes the workflow of such a system, is shown in Figure 3.1.

This architecture includes three main stages, which are parameterization, training, and detection.

- **Parameterization** focuses on collecting data to be modelled from the monitored network.

- **Training** stage analyzes the normal or abnormal behavior of the system and based on given data, a corresponding model is built.

- **Detection** stage compares the built model with incoming traffic to detect if something is suspicious. If the behaviour exceeds a pre-defined thresh hold, an alarm will be triggered.

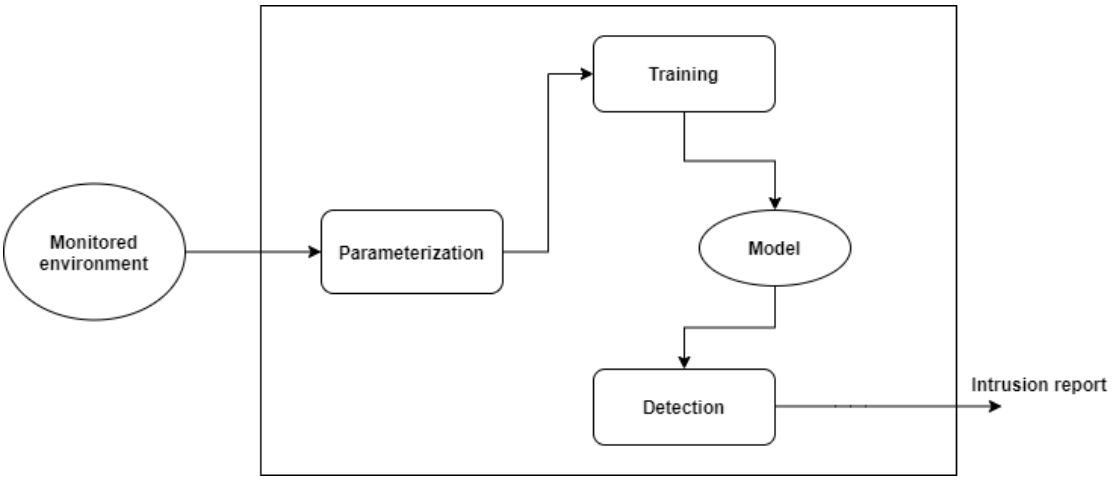

Figure 3.1: A simplified anomaly-based NIDS architecture

3.2 Anomaly-based ML techniques

3.2.1 Distance-based methods

According to [46], distance-based methods are categorized into two primary categories, clustering-based anomaly detection and nearest neighbour-based anomaly detection approaches. These methods are based on the similarity function between data instances.

Nearest neighbour-based methods

This approach is capable of detecting an irregular data point based on its distance from other points (neighbours) or on the density [47].

One of these methods is k-Nearest Neighbours (kNN). This method calculates the anomaly score, which is the distance between the neighbours (data instances). If the score exceeds a specific score level, then it is anomalous.

Density-based identify whether a point is anomalous or not by checking if the density around a data point is low or high. If the density is low, then the data point is abnormal, and normal if the density is high. For example, an algorithm called Local Outlier Factor (LOF) is a method used in combination with k-nearest neighbours to compute the average density ratio of the points and anomaly score, so that it identifies if there are any abnormal behaviours.

Indeed, LOF works better than kNN in terms of detecting the density in large datasets. However, both techniques have scalability issues when used for large datasets. The reason is that both need to compute the distance between data points in order to determine nearest-neighbours. Moreover, using these two techniques increases the computational complexity in both training and testing phases.

Clustering-based methods

As the name implies, this technique arranges similar objects into groups [47]. For instance, the well-known k-means algorithm is a technique used for anomaly detection.

It works by assuming that anomalous data points are far from their clusters, or that they are not assigned to any cluster at all. In terms of clustering-based anomaly detection methods, there are three main categories:

- The first category relies on assuming that normal data points (instances) belong to a cluster, whereas anomalies do not (noise).
- The techniques used in the second category assume that normal instances are close to the centroid of the cluster. In the case of anomalies, they lie far away from the centroid. In order for anomaly to work, there are two requirements, an algorithm to cluster the data and then compute the anomaly score for each data instance based on the distance between this instance and its cluster centroid.
- In the third category, the issue with the previous two methods is addressed after the anomalies clusters are formed. The reason is that normal data points are grouped into large and dense clusters, whereas anomalies belong to scattered or small clusters.

Clustering-based anomaly detection methods also suffer from scalability issues. However, the test phase is much faster compared to the Nearest neighbour-based methods, as the algorithm only needs to compare a few number of clusters with each other. There are efficient variants used such as heuristic techniques (e.g., k-means), approximate clustering and advanced-indexing techniques for data partitioning.

3.2.2 Ensemble-based methods

According to [14], there are several ensemble-based methods used for anomaly detection. Ensemble-based methods category, there are classical anomaly detection models such as the Local Outlier Factor (LOF) to combine the resulting score when using sets of hyperparameters.

Isolation Forest (IF) is another method among ensemble-based methods used for anomaly detection. The algorithm works by building decision trees for data instances classifying. Furthermore, the average score (anomaly score) is calculated from the root to the sample location based on the path lengths. Using this algorithm, to detect the anomalies, they are isolated based on path lengths. If the path is shorter than normal instances, then the data instance is anomalous. This approach is efficient and can be used to capture the anomalies in streaming data. In [70], they have showed a variant of Isolation Forest to improve anomaly scores by correcting the bias resulted when using the classical Isolation Forest.

The disadvantage [11] with ensemble methods is when the model is complicated enough, a reduction of interpretation ability of the system decreases. Besides, the computation and design time is pretty high, which means that ensemble methods are not good enough for real-time applications. Also, it is essential to mention that creating an ensemble model is quite complicated. On the other hand [11], in terms of mathematics, ensemble methods are able to give a degree of freedom when it comes to bias and variance tradeoff, which allows complicated problems to reach a particular hypothesis. Moreover, ensemble methods are unlikely to overfit.

3.2.3 Statistical methods

Statistical methods [41] are based on the probability. In terms of anomaly detection, these methods estimate and assume that normal data reside in high probability density areas, whereas anomaly data reside in regions that have low probability density. The probability distribution is calculated using the training data, and the threshold is used to differentiate normal and anomalous data instances. Some of the techniques that are widely used are:

- **Gaussian mixture models (GMM)**: these probabilistic models are used in anomaly detection by calculating the distance between the data instance and the estimated mean, and the distance, in this case, is the anomaly score. However, if an instance gets a score beyond the threshold, then it is considered anomalous.

- **Independent component analysis (ICA)**: ICA is a statistical technique that finds the factors, latent variables or sources of anomalies data. This is done by increasing the statistical independence of the estimated components to the maximum level.

- **Regression model-based**: such methods are a two-step approach. The regression model has first be fitted on the training data. Next, the result is used to differentiate the real value and predicted value in the test phase. Vector Auto-Regressive (VAR), Autoregressive Integrated Moving Average (ARIMA) and Recurrent Neural Network (RNN) are examples of statistical methods.

The disadvantage with statistical methods is their assumption mainly rely on that the data is generated from a specific distribution. Since anomaly detection is pretty wide and complex field of research, the assumption is usually not correct [89].

3.2.4 Domain-based methods

Domain-based methods [88] base their results on separating the data into two domains, normal and anomalies based on the training data. Support Vector Machines (SVM) is the most and the only used technique in domain-based approaches. More precisely, the One-Class Support Vector Machines (OC-SVM) variant.

SVM works by assuming that the training data represent the normal data; thus, the normal data region is well defined; consequently, if the data drop outside the defined normal domain, then it is considered anomalies.

The benefit using of such techniques, particularly multi-class techniques, is the ability to make use of powerful algorithms that can discriminate between different data instances that belong to different classes. Furthermore, the testing phase is fast because each instance would be compared to the pre-computed model. However, the downside here is that these techniques depend on the availability of accurate labels for different normal traffic classes, which is, in most cases not possible. Additionally, it becomes a disadvantage when labelling each test instance because one desires a meaningful anomaly score for these instances [86].

3.2.5 Reconstruction-based methods

Recurrent Neural Networks (RNN)

RNN [41] is a Neural Network type that is suited for time series processing. One issue with RNN is that it suffers from learning long term patterns [85]. The reason is so-called a gradient vanishing problem (a problem that occurs when training the data. What happens is that the gradient may be so small so that it prevents the weight from changing its value). It happens when applying Backpropagation Through Time (BPTT) (a supervised algorithm used to correct the network when specific errors occur) algorithm when training the data. Therefore, the standard RNN is not usually implemented in terms of real-world applications. One RNN approach or variant is Long Short-Term Memory (LSTM).

The fundamental idea behind the use of LSTM for anomaly detection is the system checks the past values over a certain amount of time and tries to predict the behaviour for the upcoming minute. If the behaviour in the next minute belongs to normal behaviours, then it is normal, and anomaly otherwise [74] [22].

Stacked LSTM RNN is used for anomaly detection, as showed in [43]. In this approach, the model is designed to accept only one-time step as input and the LSTM state is maintained across the input sequence. The data is trained with normal time series instead of training on normal data only. Thus, for each observation, there are multiple predictions made at different times previously. The predictions information is then gathered to calculate error vectors using a multivariate Gaussian distribution to detect an anomaly.

Nevertheless, there also other NN methods used for anomaly detection such as Convolutional Neural Networks (CNN). Research examples are discussed in **??**.

3.2.6 Research examples

Neural Networks (NN)

One approach that was implemented by researchers is Deep Learning for Intrusion Detection in Software Defined Network (SDN) [81]. As demonstrated in Figure 3.2, the experiment revolved around building a Deep Neural Network with an input layer (dimension: 6), three hidden layers (neurons: 12, 6, and 3) and an output layer (dimension: 2).
The dataset used in the experiment is NSL-KDD. Researchers have been using this dataset to evaluate the performance of NIDS. Even it is an old dataset (2009), it is yet to be an excellent dataset to compare with NIDS models. There are 125973 and 22554 network traffic samples in KDDTrain+ and KDDTest+, respectively. Additionally, each traffic sample has 41 features, and it contains four main types of attacks, which are DoS (Denial of Service attacks), Probe (Probing attacks), U2R (User To Root attacks), and R2L (Remote To Local attacks). Each one of these attack families consists of several types of attacks, and the number of features chosen is 6 (out of 41). The six features are duration (connection length in seconds), protocol_type (TCP, UDP, ICMP, etc.), src_bytes (number of data bytes from source to destination), dst_bytes (number of data bytes from destination to source), count (number of connections to the same host) and srv_count (number of connections to the same service).

Regarding the evaluation metrics, Accuracy (AC), precision (P), recall (R), and

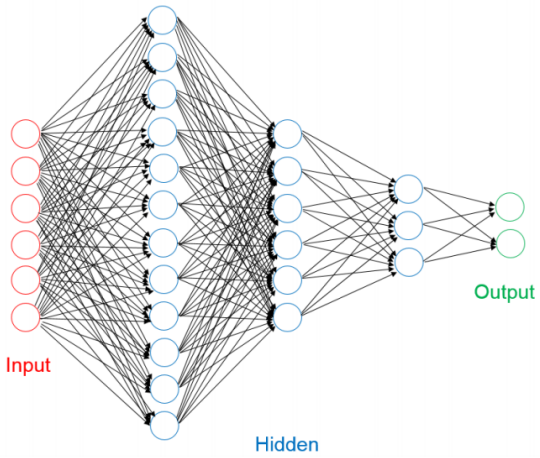

Figure 3.2: Deep Learning Network Model [81]

F-measure (F) were used. Indeed, a confusion matrix was used in order to calculate each one.

The experiments conducted were based on two approaches, normal and abnormal traffic. As an attempt to optimize the model, four learning rate values were used (0.1, 0.01, 0.001, 0.0001). According to [81], as shown in Table 3.1 and Table 3.2, in terms of accuracy metrics, loss, and accuracy, when choosing the learning rate as 0.001, the best results are achieved.

Learning Rate	**Train Set**		**Test Set**	
	Loss (%)	**Accuracy (%)**	**Loss (%)**	**Accuracy (%)**
0.1	11.49	88.04	31.26	72.05
0.01	8.41	90.9	20.15	73.03
0.001	8.26	91.62	19.51	75.75
0.0001	7.45	91.7	20.3	74.67

Table 3.1: Loss and accuracy evaluation for different learning rates

Learning Rate	**Precision (%)**	**Recall (%)**	**F1-score (%)**
0.1	79	72	72
0.01	82	73	72
0.001	83	76	75
0.0001	83	75	74

Table 3.2: Accuracy metrics for different learning rates

What is missing?

Firstly, comparing this research to others, the accuracy achieved is less than the accuracy obtained using other machine learning algorithms. As demonstrated in Table 3.3, the only algorithm that resulted in less accuracy was Support Vector Machine (SVM). The reason is they have used only six features for training and testing, and

this number is minimal, considering that there are 41 features in total.

Algorithm	**Accuracy (%)**
J48	81.05
Naive Bayes (NB)	76.56
NB Tree	81.59
Random Forest	80.67
Random Tree	81.59
Multi-layer Perception	77.41
Support Vector Machine (SVM)	69.52
The DNN approach	**75.75**

Table 3.3: Accuracy comparision of different algorithms

Secondly, the main focus of the research was based on normal and abnormal traffic, meaning they did not consider the attack type. Moreover, in order to detect sophisticated attacks, more features, and a more significant proportion of the dataset have to be selected. Hence, the sub-dataset given to the DNN algorithm was not enough to generalize the characteristics of different attacks.

Finally, the number of features selected to train the machine plays a vital role in terms of loss and accuracy. Therefore, one of the goals in this thesis is to show that considering a more extensive training dataset and selecting a higher number of features will result in achieving better numbers.

Bayesian Network

Bayesian Network is one of the most used ML techniques in intrusion detection systems. In [28], a dataset called DARPA KDD99 was used to test the IDS. This dataset is an accessible dataset among researchers due to various types of attacks it contains. This dataset contains three subsets, which are "Whole KDD," "10% KDD," and "Corrected KDD." In their experiment, they have only used 10% KDD and corrected KDD as their training and testing sets, respectively.

The KDD dataset contains four main types of attacks, which are DoS (Denial of Service attacks), Probe (Probing attacks), U2R (User To Root attacks), and R2L (Remote To Local attacks). Both training and testing sets contain 22 and 17 attacks, respectively, meaning that there are 39 attacks in total.

The Bayesian technique was used as a filter, and it worked by making the machine recognize that different features have different probabilities in terms of attacks and regular TCP traffic. The filter was able to recognize different attacks after adjusting the probabilities of the features. The main idea was to provide information about each TCP connection so it can tell if it is normal or abnormal (attack) traffic.

As shown in Figure 3.3, the IDS-based Bayesian system is represented. Each labeled input represents the type of connection. After training, the system is tested using the corrected dataset.

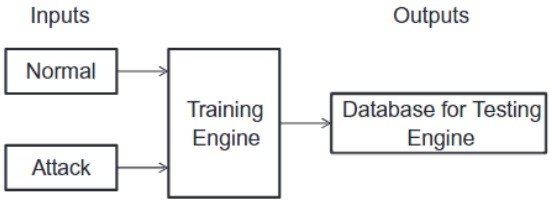

Figure 3.3: The training engine [28]

In the training phase, KDD 10% that contains 494020 records are used as the training dataset. The records to be trained include both normal traffic and abnormal data traffic (DoS, Probe, U2R, and R2L). Several experiments were conducted using this training engine. In each experiment, one attack type was selected and compared to normal traffic. The reason is to recognize how each attack type would affect the results.

- Using DoS records and normal traffic: TN = 99.6%, TP = 99.24%, FN = 0.4%, FP = 0.76% and DR = 99.24%.
- Using Probing records and normal traffic: TN = 99.4%, TP = 81.9%, FN = 0.6%, FP = 18.1% and DR = 81.9%.
- Using U2R records and normal traffic: TN = 99.7%, TP = 93%, FN = 0.3%, FP = 7% and DR = 93%.
- Using L2R records and normal traffic: TN = 68.03%, TP = 85.35%, FN = 31.97%, FP = 14.65% and DR = 85.35%.

What is missing?

In the case of this approach, the issue here is that only 10% of the dataset was used for training, but in contrast to the DNN approach, the 41 features were utilized. Hence, it would be reasonable to achieve high accuracy. So again, what if a higher proportion of the dataset is used, and how would the result be?

Random Forests (RF)

A Random Forest based system was proposed in [62]. As shown in Figure 3.4, the system uses the NSL-KDD dataset to train the model. The NSL-KDD dataset (full dataset) is split into two parts, 75% for training and 25% for testing. The training data in the proposed model is labelled, whereas the testing data is not. Next, perform data preprocessing and then select all appropriate features. The Random Forest is then applied to obtain the trained model. Finally, the performance evaluation of the model is given.

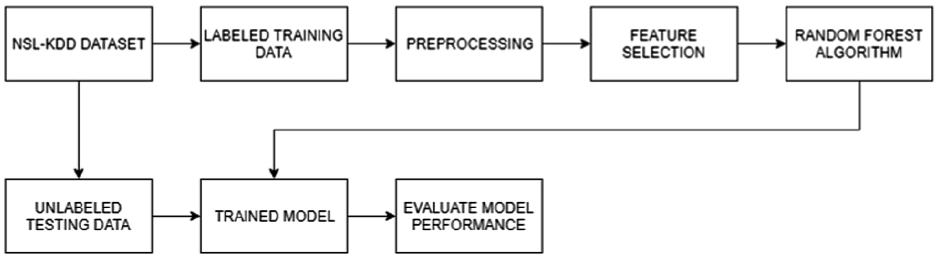

Figure 3.4: Proposed model [62]

The feature selection in this approach is performed using Gini importance (Mean Decrease in Impurity MDI - it calculates the importance of each feature). The reason is that even though the RF has an implicit feature selection, using Gini importance increases the performance of the algorithm. Additionally, noise and less important features are eliminated. However, the resulted subset of features is then evaluated to determine if it is more efficient than the whole. This operation was done multiple times to eliminate the features with less importance. Finally, the used subset is the one with the best performance.

When the subset is selected, the training data proportion is then given to the RF algorithm. The number of used decision trees was 1000 due to the high number of features. As shown in Table 3.4, the accuracy of the proposed model is higher than the basic one.

Parameters	**Basic RF model**	**Proposed model**
Accuracy (%)	99.752	99.880
Inaccuracy (%)	0.248	0.120

Table 3.4: Evaluation results

What is missing?

This approach is based on an old dataset (NSL-KDD) which consists of some disadvantages as mentioned in Table 2.8, Section 2.5.6, Chapter 2. Moreover, the proposed approach does not classify by attack, which one of the main issues to be solved in this thesis.

Convolutional Neural Networks (CNN)

One approach that aimed to use CNN for anomaly detection is [97]. The main aim was to design a method that is capable of converting NSL-KDD data format to visual image type so that CNN can handle it. This goal was achieved by mapping various types of features into binary vector space. The binary vector space is then transformed into an image. For instance, one-hot encoder mapping was used to map symbolic features such as protocol_type, flag and service into binary vectors, as shown in Figure 3.5. As an example, protocol_type feature has three values "TCP, UDP, ICMP", and these values can be turned into "100, 010, 100" using the one-hot encoder.

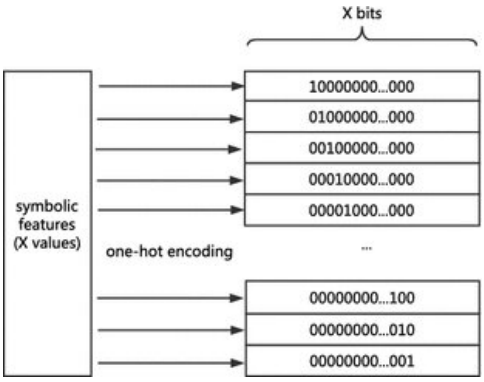

Figure 3.5: Mapping symbolic features into binary vectors

Additionally, continuous features that include integer and float types were normalized into a range from 0 to 1. For this purpose, a standard scaler was utilized (scaling data to a specific interval). The normalization method used in the approach is shown in Equation 3.1. According to the equation, x stands for numeric feature value, $\mathrm{x_{min}}$ stands for minimal value of the feature, $\mathrm{x_{max}}$ stands for the max value and $\mathrm{x_{new}}$ stands for the value after normalization. When the normalization process is finished, scaled continuous value is discretized into 10 intervals. The one-hot encoder is then used to encode the order number into 10 binary vectors, as demonstrated in Figure 3.6.

$$x_{\mathrm{new}} = \frac{x - x_{\mathrm{min}}}{x_{\mathrm{max}} - x_{\mathrm{min}}} \tag{3.1}$$

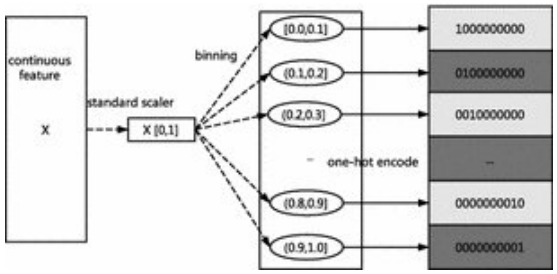

Figure 3.6: Discretization and binarization on continuous features

When the NSL-KDD dataset is preprocessed and turned into a binary vector with 464 dimensions, each 8 bits are turned into grayscale pixel. The binary vector with 464 dimensions is transformed into 8*8 grayscale image with vacant pixel padded by 0. As demonstrated in Figure 3.7, one can see how different samples were transformed into images.

Images (a) and (b) represent normal data, images (c) and (d) represent DoS attacks, images (e) and (f) represent probe attacks, images (g) and (h) represent R2L (Remote to Local), and images (i) and (j) represent U2R (User to Root).

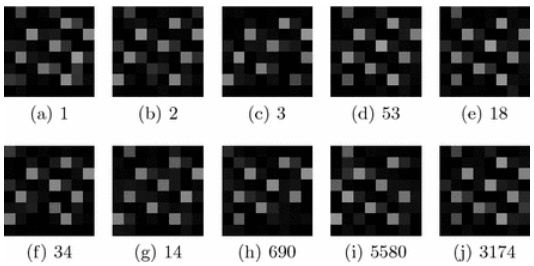

Figure 3.7: Images of different NSL-KDD samples

To conduct the experiments, ResNet 50 and GoogLeNet were used as CNN models. To train ResNet 50, 100 epochs with 256 batch size were used. In regards to GoogLeNet, 50 and 100 epochs with 64 batch size were used. Additionally, to evaluate the accuracy of the model; Precision, Recall, and F1 score are utilized as evaluation metrics.

As stated, NSL-KDD dataset was used in this approach. More precisely, the data files used were NSL-KDD Train+, NSL-KDD Test+ and NSL-KDD-21. The details of each set is demonstrated in Table 2.7, Section 2.5.6. It is also necessary to mention that each file of these files has a corresponding binary labeled files which are utilized to conduct the CNN-based anomaly detection approach. The performance of binary labeled class using ResNet 50 and GoogLeNet is shown in Table 3.5.

	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 score (%)**
ResNet 50 NSL-KDD Test+	79.14	91.97	69.41	79.12
ResNet 50 NSL-KDD Test-21	81.57	81.81	99.63	89.85
GoogLeNet NSL-KDD Test+	77.04	91.66	65.64	76.50
GoogLeNet NSL-KDD Test-21	81.84	81.84	100	90.01

Table 3.5: Performance of binary labelled class

As the results show, the Recall is highest when it comes to NSL-KDD Test-21. It is due to the difficulty of this dataset, and the uneven distribution of data within it. As one can see in Figure 2.7, the number of normal data records is small, which leads the CNN to consider the normal data as an attack class data. However, Table 3.6 compares the accuracies of proposed methods (ResNet50 and GoogLeNet) with other methods conducted by [54], and the results of confusion metrics are shown in Figure 3.8.

Classifier	Accuracy on NSL-KDD Test+ (%)	Accuracy on NSL-KDD Test-21 (%)
J48	81.05	63.97
Naive Bayes	76.56	55.77
NB Tree	82.02	66.16
Random Forest	80.67	62.26
Random Tree	81.59	58.51
Multi-Layer Perceptron	77.41	57.34
SVM	69.52	42.29
Proposed method ResNet50	79.14	81.57
Proposed method GoogLeNet	77.04	81.84

Table 3.6: Accuracy comparison of different algorithms

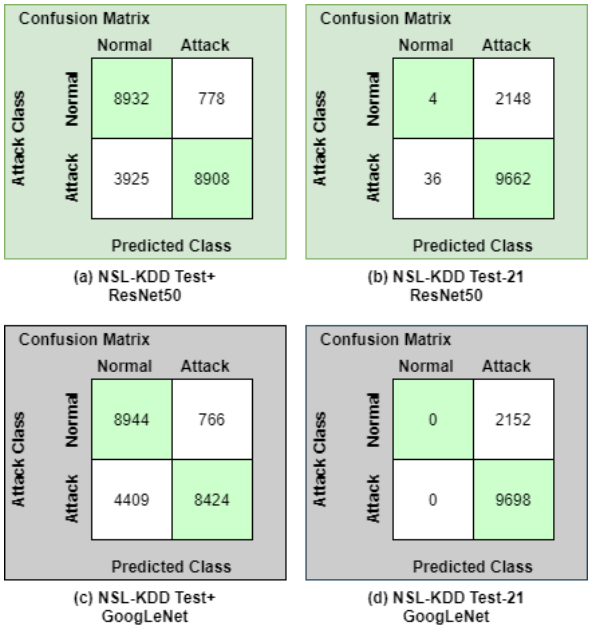

Figure 3.8: Confusion matrices of the binary test

What is missing?

As in other papers, the proposed model uses an old dataset (NSL-KDD) that has a less number of features and is not realistic compared to a dataset like CICIDS2017. Moreover, the proposed model does not classify by attack (which is one of the main objectives to solve in this thesis).

3.2.7 A relevant Machine-Learning-based IDS work for CICIDS2017

The authors (I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani) of CICIDS2017 [33] have conducted experiments applying different machine learning algorithms. The

authors used a Random Forest Regressor to conclude which features are best to use to detect each attack family. Then, 8000 randomly selected benign records and 2000 attack flows for each type of attack from the training dataset are selected (a comprehensive analysis of the dataset is done in Chapter 4). This number is little in comparison to the number of total records in the complete dataset. Consequently, the authors have conducted experiments using selected features by utilizing various machine learning algorithms such as Random Forest (RF), K-Nearest Neighbour (KNN), Naive Bayes (NB), Iterative Dichotomiser 3 (ID3), Multi-Layer Perceptron (MLP) and Quadratic Discriminant Analysis (QDA). The achieved results are shown in Table 3.7.

Algorithm	**Precision (%)**	**Recall (%)**	**F1 score (%)**
KNN	96	96	96
RF	98	97	97
ID3	98	98	98
Adaboost	77	84	77
MLP	77	83	76
Naive-Bayes	88	4	4
QDA	97	88	92

Table 3.7: The performance examination results

What is missing?

We believe that using only 8000 and 2000 records for both benign and attack flows, respectively, is not sufficient, which means that results shown in Table 3.7 are not reasonable. Hence, in this thesis, the whole dataset (all records) is used to evaluate the models.

Chapter 4

CICIDS2017 - Dataset

CICIDS2017 [34] [33] is an intrusion detection evaluation dataset released by the Canadian Institute for Cybersecurity in 2017. This dataset is a realistic dataset that contains both benign (normal) and malicious traffic (most up-to-date common attacks) gathered and saved in PCAP files. Additionally, this dataset is labelled with data flows that are based on the timestamp, source IP, destination IP, source port, destination port, used protocols and types of attacks (CSV files). It was built by what is called B-profile system, which means this dataset was made based on the human's interactions in the network. CICIDS2017 was made based on the behaviour of 25 users who used different type of protocols which are HTTP, HTTPS, FTP, SSH, and other email protocols. However, more details about why this dataset was chosen for this thesis is discussed in Section 4.4.

4.1 CICIDS2017 attacks

As this dataset is a realistic one, it contains a variety of attack scenarios. It contains eight attack profiles that represent different attack categories; Web-based, Brute force, DoS, DDoS, Infiltration, Heart-bleed, botnet and Scan are covered in this dataset.

4.1.1 Description of attack types

Web-based attack

Web-based attacks [69] aim to breach both confidentiality and integrity by targeting websites to exploit existing vulnerabilities. In this dataset, the SQL Injection attack was conducted to make the database return sensitive information. Cross-Site Scripting (XSS) is another attack that is done by an adversary by injecting scripts; this happens when the developers of a website do not sanitize their code. Brute Force over HTTP attack is also conducted; this attack aims to find the administrator's password by trying a list of passwords.

Brute force attack

Brute force attacks [90] are widespread attacks that aim not only to crack passwords but also to find hidden pages and hidden contents in web pages. It is based on trial

and error principle until the desired outcome is achieved. The brute force attacks were conducted against FTP, SSH and HTTP in this dataset.

DoS/DDoS attack

DoS (Denial-of-Service) attacks [92] aims in the first place to make systems not accessible so that legitimate users cannot access the resources. DoS attacks can be achieved by flooding the target system with data.

The DDoS (Distributed Denial-of-Service) attack [93] is a variant of DoS attacks; as instead of initiating an attack from only one system, the attack is initiated from multiple systems or computers to flood the victim's network or bandwidth with a massive amount of traffic.

DoS/DDoS attacks conducted in this dataset are DoS slowloris, DoS Slowhttptest, DoS Hulk and DoS GoldenEye.

Infiltration attack

Infiltration attacks [76] revolve around exploiting a vulnerable software, and then make a backdoor so that attacker is able to conduct various attacks on the victim's network using this backdoor.

Heart-bleed attack

Heart-Bleed [83] is an attack that is based on a bug in the OpenSSL cryptography library. This library is used to implement the Transport Layer Security (TLS) protocol. By sending a malicious request with a payload to and a large length field to make the server leak sensitive information. The information may include various types of data such as passwords, credit card numbers, medical records, and private email contents or even social media messages.

Scan attack

A scan attack [63] is an attempt to send data requests to a number of ports or addresses in order to find exploitable open ports.

Botnet attack

This attack is a form of DDoS attacks [13]. By having control over multiple Internet-connected devices, the adversary can perform many actions, for instance, stealing data and sending spams.

4.2 CICIDS2017 dataset details

The capturing process started at 09:00 on Monday, which is July 3rd and gathered all traffic data for 5 days. The capturing process continued until 17:00 on Friday, July 7th. As shown in Table 4.1, a list of different attacks and at what day each attack was executed is represented. Nevertheless, a list of used tools to conduct the variety

of attacks is shown in Table 4.3, and the size of each one of the 14 attacks and benign traffic is shown in Table 4.2.

Days	Labels
Monday	Benign.
Tuesday	Brute force, SFTP and SSH.
Wednesday	DoS and Heartbleed attacks. Slowloris, Slowhttptest, Hulk and GoldenEye.
Thursday	Web and Infiltration attacks. Web Brute Force, XSS and Sql Injection, Infiltration Dropbox Download and Cool disk.
Friday	DDoS LOIT, Botnet ARES, PortScans (sS, sT, sF, sX, sN, sP, sV, sU, sO, sA, sW, sR, sL and B).

Table 4.1: Related attacks to each day [34]

Traffic type	Size (number of records)
Benign	2273097
DoS Hulk	231073
Port Scan	158930
DDoS	128027
DoS GoldenEye	10293
FTP Patator	7938
SSH Patator	5897
DoS Slow Loris	5796
DoS Slow HTTP Test	5499
Botnet	1966
Web Attack: Brute Force	1507
Web Attack: XSS	652
Infiltration	36
Web Attack: SQL Injection	21
HeartBleed	11

Table 4.2: The size of each attack class [34]

Attack	Tools
Brute force attack (Tuesday morning-afternoon)	- Hydra, Medusa, Ncrack, Metasploit and Nmap NSE scripts to crack passwords. - Hashcat and hash pump for cracking hashed passwords. - Patator (the used one as it is one of the best multi-threaded tools).
DoS attack (Wednesday morning)	LOIC, HOIC, Hulk, GoldenEye, Slowloris, and Slowhttptest. This attack scenario was conducted using these tools excluding first two.
Dos attack (Wednesday afternoon)	Heartleech; a tool used to exploit heartbleed vulnerability, and scan the system to discover the vulnerabilities.
Web attack (Thursday morning)	DVWA (Damn Vulnerable Web App) used to implement the attack scenarios. DVWA is vulnerable PHP/MySQL web application.
Infiltration attack (Thursday afternoon)	Metasploit was used for implementing this attack scenario, as it is most common in terms of verifying vulnerabilities and security issues. When the infected file was downloaded on the victim's machine from either Dropbox or an infected USB flash, Nmap was used by the attacker to scan the second level of the victim's network.
Botnet attack (Friday morning)	Grum, Windigo, Storm and Ares. Ares was the used tool to conduct the attack, as it is able to capture screenshots, provide a remote shell, keylogging or even download/upload files.
DDoS attack and Port-Scan (Friday afternoon)	High Orbit Ion Canon (HOIC), Low Orbit Ion Canon (LOIC), and DDoSIM. However, LOIC was used in this scenario by sending a huge amount of UDP, TCP or HTTP requests to the victim's server.

Table 4.3: Tools used to conduct the attack scenarios [33]

The testing architecture used to make this dataset is shown in Figure 4.1. As demonstrated, there are two separated networks; a victim-network and a an attack-network. In the victim's network, necessary components which are one router, one firewall, two switches, three servers and ten PCs are deployed. Additionally, different operating systems such as Windows, Linux and Macintosh are installed. On the other side, the attack-network consists of one router, one switch and four PCs that run Kali, and Windows 8.1 operating systems. Moreover, Table 4.4 shows different workstations, firewalls and servers along with their both private and public IP addresses.

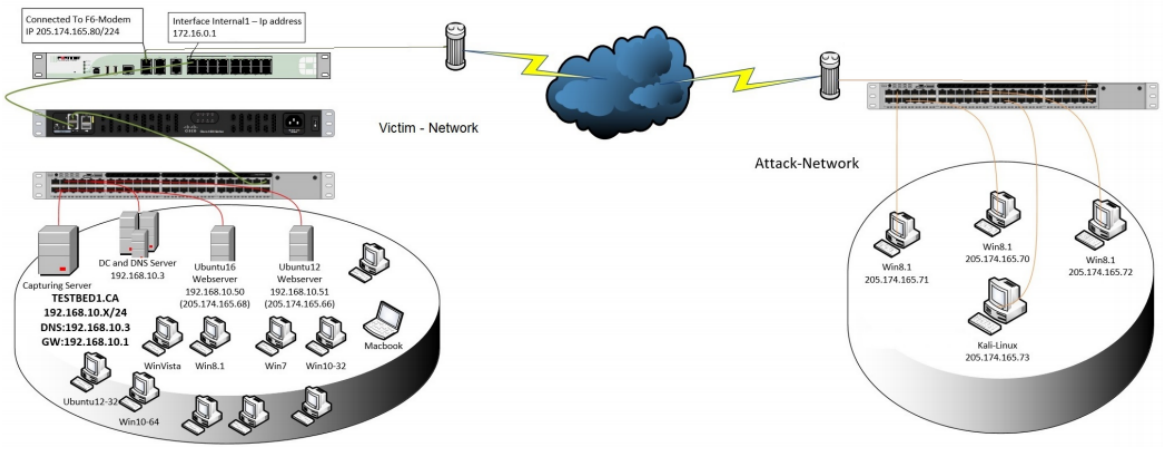

Figure 4.1: Networks used to make the dataset [33]

	Machine	OS	IPs
Victim-Network	***Servers***	Win Server 2016 (DC and DNS)	192.168.10.3
		Ubuntu 16 (Web Server)	192.168.10.50 and 205.174.165.68
		Ubuntu 12	192.168.10.51 and 205.174.165.66
	PCs	Ubuntu 14.4 (32, 64)	192.168.10.19 and 192168.10.17
		Ubuntu 16.4 (32-64)	192.168.10.16 and 192.168.10.12
		Win 7 Pro	192.168.10.9
		Win 8.1-64	192.168.10.5
		Win Vista	192.168.10.8
		Win 10 (Pro 32-64)	192.168.10.14 and 192.168.10.15
		Mac	192.168.10.25
	Firewall	Fortinet	
Attack-Network	***PCs***	Kali	205.174.165.73
		Win 8.1	205.174.165.69
		Win 8.1	205.174.165.70
		Win 8.1	205.174.165.71

Table 4.4: Victim-Network and attackers [34]

4.3 Attack scenarios details

Monday, July 3, 2017

Benign traffic

Tuesday, July 4, 2017

Brute Force

- FTP-Patator (9:20 - 10:20 a.m.)
- SSH-Patator (14:00 - 15:00 p.m.)

Attacker: Kali, 205.174.165.73
Victim: WebServer Ubuntu, 205.174.165.68 (Local IP: 192.168.10.50)
NAT Process on Firewall:
Attack: 205.174.165.73 -> 205.174.165.80 (IP Valid Firewall) -> 172.16.0.10 -> 192.168.10.50
Reply: 192.168.10.50 -> 172.16.0.1 -> 205.174.165.80 -> 205.174.165.73

Wednesday, July 5, 2017

DoS/DDoS

- DoS slowloris (9:47 - 10:10 a.m.)
- DoS Slowhttptest (10:14 - 10:35 a.m.)
- DoS Hulk (10:43 - 11 a.m.)
- DoS GoldenEye (11:10 - 11:23 a.m.)

  Attacker: Kali, 205.174.165.73

  Victim: WebServer Ubuntu, 205.174.165.68 (Local IP192.168.10.50)

  NAT Process on Firewall:

  Attack: 205.174.165.73 -> 205.174.165.80 (IP Valid Firewall) -> 172.16.0.10 -> 192.168.10.50

  Reply: 192.168.10.50 -> 172.16.0.1 -> 205.174.165.80 -> 205.174.165.73

Heartbleed

- Heartbleed Port 444 (15:12 - 15:32)

  Attacker: Kali, 205.174.165.73

  Victim: Ubuntu12, 205.174.165.66 (Local IP192.168.10.51)

  NAT Process on Firewall:

  Attack: 205.174.165.73 -> 205.174.165.80 (IP Valid Firewall) -> 172.16.0.11 -> 192.168.10.51

  Reply: 192.168.10.51 -> 172.16.0.1 -> 205.174.165.80 -> 205.174.165.73

Thursday, July 6, 2017

MORNING
Web attack

- Brute Force (9:20 - 10 a.m.)
- XSS (10:15 - 10:35 a.m.)
- Sql Injection (10:40 - 10:42 a.m.)

  Attacker: Kali, 205.174.165.73

  Victim: WebServer Ubuntu, 205.174.165.68 (Local IP192.168.10.50)

  NAT Process on Firewall:

  Attack: 205.174.165.73 -> 205.174.165.80 (IP Valid Firewall) -> 172.16.0.10 -> 192.168.10.50

  Reply: 192.168.10.50 -> 172.16.0.1 -> 205.174.165.80 -> 205.174.165.73

AFTERNOON
Infiltration - Dropbox download

- Meta exploit Win Vista (14:19 and 14:20-14:21 p.m.) and (14:33 -14:35)

  Attacker: Kali, 205.174.165.73

  Victim: Windows Vista, 192.168.10.8

- Infiltration - Cool disk - MAC (14:53 p.m. - 15:00 p.m.)

  Attacker: Kali, 205.174.165.73

  Victim: MAC, 192.168.10.25

Infiltration - Dropbox download

- Win Vista (15:04 - 15:45 p.m.)

  First Step:

  Attacker: Kali, 205.174.165.73

  Victim: Windows Vista, 192.168.10.8

  Second Step (Portscan + Nmap):

  Attacker:Vista, 192.168.10.8

  Victim: All other clients

Friday, July 7, 2017

MORNING
Botnet ARES

- Botnet ARES (10:02 a.m. - 11:02 a.m)

  Attacker: Kali, 205.174.165.73

  Victims: Win 10, 192.168.10.15 + Win 7, 192.168.10.9 + Win 10, 192.168.10.14 + Win 8, 192.168.10.5 + Vista, 192.168.10.8

AFTERNOON
PortScan

- **Firewall rule on** (13:55 - 13:57, 13:58 - 14:00, 14:01 - 14:04, 14:05 - 14:07, 14:08 - 14:10, 14:11 - 14:13, 14:14 - 14:16, 14:17 - 14:19, 14:20 - 14:21, 14:22 - 14:24, 14:33 - 14:33, 14:35 - 14:35)
- **Firewall rules off** (sS 14:51-14:53, sT 14:54-14:56, sF 14:57-14:59, sX 15:00-15:02, sN 15:03-15:05, sP 15:06-15:07, sV 15:08-15:10, sU 15:11-15:12, sO 15:13-15:15, sA 15:16-15:18, sW 15:19-15:21, sR 15:22-15:24, sL 15:25-15:25, sI 15:26-15:27, b 15:28-15:29)

  Attacker: Kali, 205.174.165.73

  Victim: Ubuntu16, 205.174.165.68 (Local IP: 192.168.10.50)

  NAT Process on Firewall:

  Attacker: 205.174.165.73 -> 205.174.165.80 (IP Valid Firewall) -> 172.16.0.1

DDoS LOIT

- DDoS LOIT (15:56 - 16:16)
  Attackers: Three Win 8.1, 205.174.165.69 - 71

  Victim: Ubuntu16, 205.174.165.68 (Local IP: 192.168.10.50)

  NAT Process on Firewall:

  Attackers:205.174.165.69, 70, 71 -> 205.174.165.80 (IP Valid Firewall) -> 172.16.0.1

Day and file size (GB)	Activity
Monday - 11GB	Normal activity
Tuesday - 11GB	Attacks and normal activity
Wednesday - 13GB	Attacks and normal activity
Thursday - 13GB	Attacks and normal activity
Friday - 8.3GB	Attacks and normal activity

Table 4.5: A description of dataset files [34]

4.4 Why CICIDS2017?

The reason why CICIDS2017 dataset was chosen is it fulfils the most important criteria [32].

- **Complete Network Configuration**: the attacks were conducted against a network with a realistic configuration. The network contains multiple devices such as computers, servers, routers and firewalls.
- **Complete Traffic**: the traffic is generated from different types of devices. Hence, traffic generation technique results in having realistic traffic in the dataset. The reason is the pseudo-realistic traffic contains real-world traffic such as simulated human behaviour on the network, in addition to realistic attack scenarios.
- **Labelled Dataset**: since this dataset is labelled with different types of attacks, it provides much more comfort when analyzing it. Indeed, it is desirable to have a dataset with accurate and informative data/labels about each attack and not only benign or malicious. Having a labelled dataset helps particularly in avoiding the calculation and analyzing the features.
- **Complete Interaction**: a complete interaction within and between LANs (Local Area Network) was required in order to get an accurate interpretation of the results.
- **Complete Capture**: capturing all the traffic is essential to calculate the false-positive percentage of the intrusion detection system. Some of the other datasets remove traffic that is not labelled or not functional.
- **Available Protocols**: as there are many types of attacks that function using different types of protocols, it is essential to have data generated from a variety of protocols to get the best results from the IDS testing.
- **Attack Diversity**: attack diversity is essential in a dataset. The reason is as the technology gets improved and more sophisticated, more attacks appear. Hence, new attacks for evaluating IPS and IDS are vital to have. According to 2016 McAfee report, attacks are categorized into seven groups; browser-based, brute force, DoS, Scan, backdoors, DNS and other attack types, for instance, Heartbleed, shellshock and Apple SSL library bug.
- **Anonymity**: other datasets than the CICIDS2017 have removed the payloads because of some privacy issues. The impact of this procedure is that it decreases the usefulness of some detection mechanisms such as deep packet inspection.
- **Heterogeneity**: it is essential to have different sources of traffic to conduct proper IDS research. The reason is to cover all aspects of the detection process. In contrast to heterogeneity, a homogeneous dataset uses only one source of traffic, and this is only useful for testing particular detection systems.
- **Metadata**: many datasets lack sufficient documentation. In regards to the CICIDS2017 dataset, it is one of the best documented datasets available as it provides information about network configuration, OS for both the attacker and victim machines, attack scenarios and much more.

As shown in Table 4.6, based on the discussed criteria, it is evident that CICIDS2017 is better than DAPRA, KDD'99 and NSL-KDD for IDS testing and evaluation. Moreover, in regards to the number of features in CICIDS2017, there are 85 features which is much higher than the number of features in DAPRA (-), KDD'99 (41 features) and NSL-KDD 2009 (41 features). All features that CICIDS2017 contains are shown in Table 4.7.

It is also necessary to mention that the CICIDS2017 dataset consists of two zipped files which are MachineLearningCSV and GeneratedLabelledFlows. The difference is that the first one (MachineLearningCVE) consists of 79 features, whereas the second one (GeneratedLabelledFlows) consists of 85 features as just stated. It means that there are 6 removed features from the dataset. These extra 6 features in the second dataset identify the flow as stated by the authors of the dataset. The 6 deleted features are "FlowID, SourceIP, SourcePort, DestinationIP, Protocol and timestamp". One reason why these features were removed is that, for instance, IP addresses and source ports change continuously. Moreover, features like "SourceIP, SourcePort, DestinationIP and DestinationPort" are considered repeated features because the FlowID feature includes all of them. Hence, in this thesis, MachineLearningCVE dataset, which consists of 79 features, is going to be used in the experiments.

		DAPRA	**KDD'99/NSL-KDD**	**CICIDS2017**
Network		✓	✓	✓
Traffic		✗	✗	✓
Labelled dataset		✓	✓	✓
Complete interaction		✓	✓	✓
Complete capture		✓	✓	✓
Protocols	**HTTP**	✓	✓	✓
	HTTPS	✗	✗	✓
	SSH	✓	✓	✓
	FTP	✓	✓	✓
	Email	✓	✓	✓
Attack diversity	**Browser**	✗	✗	✓
	Brute force	✓	✓	✓
	DoS	✓	✓	✓
	Scan	✓	✓	✓
	Back door	✗	✗	✓
	DNS	✗	✗	✓
	Other	✓	✓	✓
Anonymity		✗	✗	✓
Heterogeneity		✗	✗	✓
Feature Set		✗	✗	✓
Metadata		✓	✓	✓

Table 4.6: A comparison between most known datasets [34]

No	Feature	No	Feature	No	Feature	No	Feature
1	Flow ID	22	Flow Packets/s	43	Fwd Packets/s	64	Fwd Avg Packets/Bulk
2	Source IP	23	Flow IAT Mean	44	Bwd Packets/s	65	Fwd Avg Bulk Rate
3	Source Port	24	Flow IAT Std	45	Min Packet Length	66	Bwd Avg Bytes/Bulk
4	Destination IP	25	Flow IAT Max	46	Max Packet Length	67	Bwd Avg Packets/Bulk
5	Destination Port	26	Flow IAT Min	47	Packet Length Mean	68	Bwd Avg Bulk Rate
6	Protocol	27	Fwd IAT Total	48	Packet Length Std	69	Subflow Fwd Packets
7	Timestamp	28	Fwd IAT Mean	49	Packet Length Variance	70	Subflow Fwd Bytes
8	Flow Duration	29	Fwd IAT Std	50	FIN Flag Count	71	Subflow Bwd Packets
9	Total Fwd Packets	30	Fwd IAT Max	51	SYN Flag Count	72	Subflow Bwd Bytes
10	Total Backward Packets	31	Fwd IAT Min	52	RST Flag Count	73	Init_Win_bytes _forward
11	Total Length of Fwd Packets	32	Bwd IAT Total	53	PSH Flag Count	74	Init_Win_bytes _backward
12	Total Length of Bwd Packets	33	Bwd IAT Mean	54	ACK Flag Count	75	act_data_pkt _fwd
13	Fwd Packet Length Max	34	Bwd IAT Std	55	URG Flag Count	76	min_seg_size _forward
14	Fwd Packet Length Min	35	Bwd IAT Max	56	CWE Flag Count	77	Active Mean
15	Fwd Packet Length Mean	36	Bwd IAT Min	57	ECE Flag Count	78	Active Std
16	Fwd Packet Length Std	37	Fwd PSH Flags	58	Down/Up Ratio	79	Active Max
17	Bwd Packet Length Max	38	Bwd PSH Flags	59	Average Packet Size	80	Active Min
18	Bwd Packet Length Min	39	Fwd URG Flags	60	Avg Fwd Segment Size	81	Idle Mean
19	Bwd Packet Length Mean	40	Bwd URG Flags	61	Avg Bwd Segment Size	82	Idle Std
20	Bwd Packet Length Std	41	Fwd Header Length	62	Fwd Header Length_1	83	Idle Max
21	Flow Bytes/s	42	Bwd Header Length	63	Fwd Avg Bytes/Bulk	84	Idle Min
85	Lable						

Table 4.7: CICIDS2017 Features

Chapter 5

Approach

In an attempt to achieve the goals described in Section 1.2, Chapter 1, several experiments were conducted by tuning the parameters of each model, following typical machine learning procedures [25]. Trying various parameters such as the number of layers, batch size, epochs, dimensions and others, have assisted in achieving the goals we aimed for. Hence, finding out which algorithm is most efficient regarding attack types represented in CICIDS2017.

Notwithstanding, this chapter explains each step taken to build the models and their determined parameters. Besides, it describes different aspects such as preprocessing of the dataset, the structure of each algorithm and the tools used to develop the models. Further, the results of each model will be shown in Chapter 6. Questions such as "were the attacks predicted correctly? And which model did get the best and most reasonable results?" are going to be discussed in Chapter 6 as well.

The approach is going to be based on supervised machine learning as each traffic type is labelled with a name. However, intrusion detection systems learn by examples; the models that are going to be constructed will be able to extract each attack's features, making it possible to classify each one.

Each utilized algorithm learns, classifies and then detects. It is essential to extract more worthy and distinguish features to help the models differentiate an attack from the others (multi-class classification). Machine learning in intrusion detection systems is a necessity. The reason behind it is if one desires to detect a sophisticated attack, an intelligent IDS is required. An intelligent IDS needs a good foundation, meaning that an IDS has to be feed with a sufficient amount of data which the system can learn. Consequently, CICIDS2017 is utilized as it contains 15 attacks (including benign traffic) and 79 features.

5.1 Constructing the models

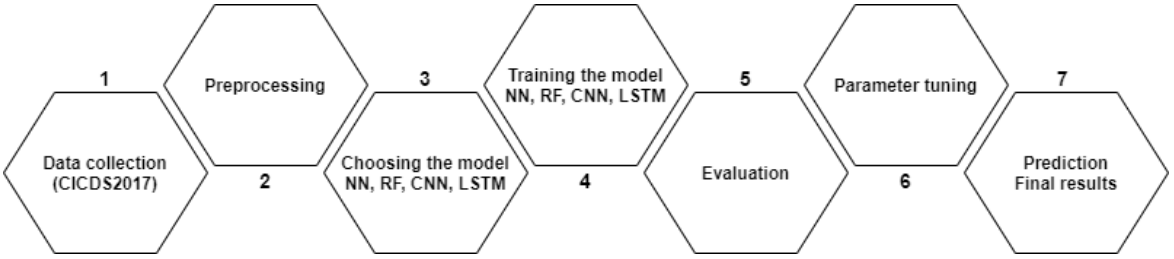

Figure 5.1: A general overview of the steps taken to build the models.

Figure 5.1 shows the steps taken to construct the models. The first step is to collect the data needed to build the models. The reason why CICIDS2017 is chosen as a dataset to utilize in this approach is discussed in Section 4.4, Chapter 4.

The following step is to preprocess the data so it is ready to be used to train the models. In this step, unnecessary and incomplete information, in addition to white spaces are removed from the dataset, and the labels are vectorized. Then, the data is both normalized and sampled, and split into both train and test sets. More information about each step is going to be reviewed in Section 5.1.1.

The subsequent step is to choose the model. There are several models to take advantage of regarding intrusion detection systems. However, NN, RF, CNN and LSTM are four popular algorithms used for intrusion detection systems, as stated in Chapter 3. Consequently, they were chosen to get utilized for training the models.

Training step is a vital step in machine learning. The training step is where the data is utilized to gradually enhance the models' ability to predict the type of attack by discriminating the classes. Further explanation of which parameters selected before starting the training and how each model was built is shown in Section 5.1.2.

After the training is finished, the models have to be evaluated to see if they perform well. It allows for testing the model against data that it has never seen before, i.e., used for training. However, the cross-validation is utilized to evaluate the models. See Section 5.2.

When the evaluation is done, parameter tuning is conducted to see if it is possible to get any better results. As an example, one can modify the number of times (epochs) the model runs through the dataset during the training phase, rather than just once, in addition to adjusting the learning rate (how fast the model adapts to the problem), which may lead to better results. The parameters determined for each algorithm are shown in Section 5.1.2.

Finally, the prediction step. When the models are trained, and the results are achieved, it is the time to answer questions. It is where the worth of machine learning accomplished, and to find out if the goals of this thesis are attained.

5.1.1 Pre-processing

In machine learning, preprocessing the data correctly can have a huge impact on the results of the experiments. As mentioned earlier, CICIDS2017 contains 79 features.

This dataset was incomplete since it consists of several Nan and whitespace that had to be deprecated. The CICIDS2017 dataset has initially been in several files separated by day but was concatenated into one file. The new file created was split into X_train, Y_train, X_test and Y_test in a ratio of 80/20. X_train consists of only float values that will be passed into the model and contains X features. Y-train on the other hand consists of 15 various attacks (included normal traffic).

- **Removing information**: Flow Bytes/s and Flow Packets/s were removed from the dataset because they contain both Nan (missing features) and infinity values.

- **Removing white spaces**: In CICIDS2017, there are white spaces that have to be removed. If there are white spaces in the beginning and at the end of a string, the data is then not the same as the actual value. Removing white spaces also results in having a more consistent dataset with a smaller size.

- **Label encoding**: The Label feature is a multi-class feature that includes string values (attack names). To make the model learn about this data type, the Label data was transformed into integer values. The values range from 0 to 14 (number of each attack including benign traffic).

- **Normalization**: Because the features have different ranges, normalization is required. In this approach, the normalization value was assigned to be between -1 and 0.

- **Sampling**: To solve the imbalance problem in the dataset, sampling had to be used. On the one hand, all small classes were oversampled so that their population are equal to the mean number of samples per class. On the other hand, all big classes were undersampled so that their population are equal to the mean number of samples per class. As one can see in Table 4.2 (Chapter 4), the number of records associated to each attack is not balanced.

5.1.2 Models architecture

Neural Network (NN)

After the preprocessing stage, the data is feed to the neural network model. As mentioned in Section 5.1.1, the CICIDS2017 is not balanced; hence it was essential to solve this problem to balance the data (Section 5.2 describes the problems regarding unbalanced datasets). However, the data is then split into train and test sets, then shuffled using Sklearn library. The test proportion is set to 20%, which is used for validation and test, and the train proportion is 80%. Next, the hyperparameters [84] Batch Size (describes the number of samples feed to the network to train), Optimizer (adaptive learning rate algorithm), Epochs (the number of times the data passed to the network), Learning Rate, and Regularization (attempts to modify the algorithm slightly to make it perform better, by adding a penalty to the loss function; hence, avoiding overfitting problem) are shown in Table 5.1.

Hyperparameters	Values
K-Folds	5
Batch Size	81920
Optimizer	Adam
Epochs	100
Learning rate	0.001
Regularization	1e-6
Activation function	Rectified Linear Unit (ReLU)

Table 5.1: Neural network hyperparameters

To normalize the inputs in the model, Batch Normalization is utilized to have an approximately equal distribution in every training step in the Neural Network. Batch normalization solves the *internal covariate shift* problem, which in simple terms means slowing the training process because every layer in the NN has to adapt itself to the new distribution, in every training step [26]. As an activation function, the ReLU is utilized, which is a nonlinear and the most used activation function for both NN and CNN. The formula of ReLU is demonstrated in Equation 5.1 [79].

$$y = max(0, x) \tag{5.1}$$

Additionally, to avoid the overfitting problem and have a good generalization, Dropout is used [2]. Dropout refers to dropping neurons in a Neural Network. It is needed during the training phase to drop a random set of neurons by "ignoring" them.

Moreover, the number of layers used were 3, where 1 of them is a hidden layer, 1 input layer and 1 output layer. The value of the hidden dimension is 1024; the reason is that the dataset is quite large, and a significant amount of data has to be processed. Furthermore, the input dimension is set to 76 (number of features), and the output dimension is set to 15 (number of classes), as shown in Table 5.2. Finally, the model is trained and tested. Chapter 6 demonstrates the results for each attack, its Precision, Recall (True Positive Rate [27]) and F1-score. However, more details about how Neural Networks work can be located in Section 2.5.2, Chapter 2.

Parameter	Value
Input dimension	76
Hidden dimension	1024
Output dimension	15
Dropout	0.5

Table 5.2: Neural network model

Convolutional Neural Network (CNN)

In the CNN approach, Batch Normalization and ReLU activation function are also utilized. Since the input is text, the used CNN operation is what is called 1D convolution [12]. However, more information about CNN is discussed in 2.5.4. Table 5.3 shows two convolution layers with their parameters which are used in this approach. In addition to these two layers, one Fully-Connected layer is also added to them to

classify the features. The input of the Fully-Connected layer is 9728 (128*76 where 128 is input dimension and 76 is number of features), and the output dimension is 15 (number of classes). The kernel size assists the 1D convolution in processing more data by padding data to an array to increase the accuracy. Choosing 3x3 as a padding size is quite common when utilizing CNN [3]. Stride is set to 1 to encode more information. Mainly because it performs better when identifying the same data in the next layer. Having a small stride value assists in avoiding overlapping.

Parameter	**Conv. layer 1**	**Conv. layer 2**
Input channels	1	64
Output channels	64	128
Kernel size	3	3
Padding size	1	1
Stride	1	1

Table 5.3: Convolutional model

The hyperparameters used before training the CNN model is shown in Table 5.4. However, the confusion metrics for each attack are also measured. The reasons that these parameters were used are discussed in Section 5.1.2. Regarding batch size utilized here, 20480 provided better results than a batch size of 81920.

Hyperparameters	**Values**
K-Folds	5
Batch Size	20480
Optimizer	Adam
Epochs	100
Learning rate	0.001
Regularization	0.001
Activation function	Rectified Linear Unit (ReLU)

Table 5.4: Convolutional Neural network hyperparameters

Random Forest (RF)

In RF, after the data is normalized and balanced, the utilized hyperparameteres are shown in Table 5.5. The number of estimators represents how many Decision Trees used. In addition, Min Samples Split, which is the minimum number required to spilt and internal node, and Min Samples Leaf, which is the minimum number of samples required to be at a leaf node, are set to their default values [72].

Hyperparameters	**Values**
K-Folds	5
Estimators	20
Min Samples Split	2
Min Samples Leaf	1

Table 5.5: Random Forest hyperparameters

The Random Forest utilized in this approach consists of 20 decision trees to classify the data. The default value of the decision trees is 10. However, since increasing the number of trees typically improves the performance of the classifier [57], we have increased it to 20 trees, as it depends on the number of votes made by each one.

Furthermore, Min Samples Split can constrain the tree. The reason is the tree has to consider more samples at each node, which can result in underfitting. Hence, the default value is chosen based on the experiments we have conducted. Moreover, this problem applies also for Min Samples Leaf; meaning that increasing the value might cause underfitting [48]. Determining the values of these two parameters depend on the experiments conducted. Mainly because these values depend on the size and the complexity of the data. We have decided to use the hyperparameters shown in Table 5.5 after conducting several experiments, and we have concluded that these are the optimal values. Section 6.3.2, Chapter 6, is going talk more about which attempts we have conducted. For more information about Random Forest, review Section 2.5.5, Chapter 2.

After these hyperparameters are determined, the classifier is used to generate the confusion metrics, and to calculate Precision, Recall and F1-Score for each attack.

Long Short-Term Memory (LSTM)

After the prepossessing stage is accomplished, the data is feed to the LSTM model. As in previous models, the test proportion is set to 20%, which is used for validation and test, and the train proportion is 80%, as this is a quite common ratio in machine learning.

The parameters utilized to build the LSTM model are shown in Table 5.6, and these parameters are based on a standard LSTM approach. The input size represents the number of features in the CICIDS2017. Hidden size is the number of units/neurons in each of the 4 layers in the model [75]. The dataset used in this thesis is quite large and complex. Therefore, it would be beneficial to use a larger number of layers. In a standard machine learning approach, it is quite common to use 1 or 2 layers. As shown in [29], LSTM performs better when increasing the number of layers, depending on the size of the dataset.

Dropout in LSTM is also utilized to avoid the overfitting problem. Finally, the number of output features is 15, which represents the number of classes in CICIDS2017. For more details regarding LSTM, see Section 2.5.3, Chapter 2.

Parameter	**Value**
Input size	76
Hidden size	32
Number of layers	4
Dropout	0.3
Output features	15

Table 5.6: LSTM model

Hyperparameters used in the model could be found in Table 5.7. A common used learning rate in machine learning fields is 0.01. Regarding the epochs and batch

size, these hyperparameters uses a more unconventional approach that is going to be discussed more in Section 6.4, Chapter 6.

Hyperparameter	Value
Epochs	50
Batch size	4096
Learning rate	0.01

Table 5.7: LSTM hyperparameters

5.2 Evaluating the models

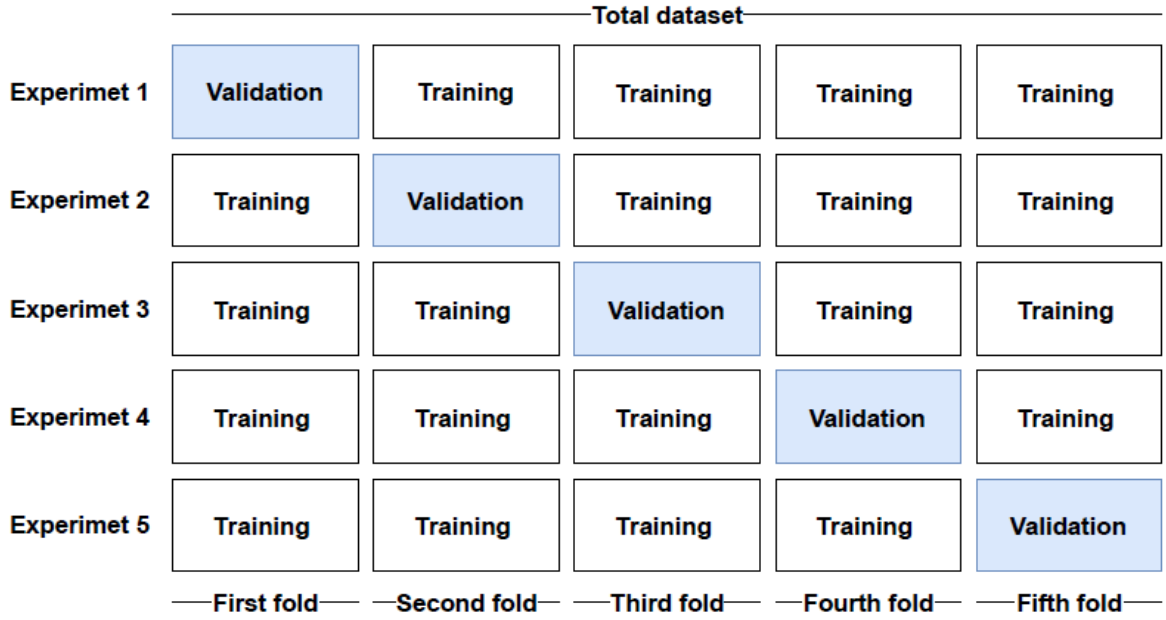

Figure 5.2: Cross validation with K=5

Evaluating the performance of each algorithm is essential. Hence, cross validation approach is utilized to test the accuracy on the test set. Cross validation method is used to validate the performance of machine learning models. CICIDS2017 is going to be divided into five (K) subsets; four subsets for training, and one subset for testing. Each time an algorithm runs, the next subset (which is 20%) is going to be used as a test set, and the rest are for training. All results achieved using this technique are added to each other. Furthermore, the sum is divided by 5 to get the overall conclusion that describes the performance of the model. Figure 5.2 shows how cross validation works.

Since CICIDS2017 is not balanced, typical optimization metrics such as accuracy is not good enough as it does not show the actual performance of the models (even the dataset is balanced in the preprocessing step) [61]. There are several classes with very few records compared to other classes in the dataset. Hence, these events are infrequent, making it very difficult to find the data that would be used to balance

the class distribution. It can also introduce a disadvantage which is bias, where the classifier is more sensitive to detect classes with a high number of records. Thus, F1-score is a better measurement score than overall accuracy. That said, most research examples discussed in Section 3.2.6, Chapter 3, utilize the accuracy metric, which is not enough to evaluate the models, knowing that NSL-KDD and KDD99 are also unbalanced datasets. However, since it is desired that no positive sample will be classified as negative, the average Recall across all five experiments is the right metric to evaluate the model as it is equal to the True Positive Rate [9][27]. Section 2.2.5, Chapter 2, mentions that True Positive Rate is used to evaluate the performance of an IDS.

In terms of intrusion detection systems, Precision is the portion of the data that is predicted as positive, and it is indeed positive. However, Recall evaluates the lacking part in precision, which is the proportion of the genuine attack covered by the classifier. Thus, the classifier should have a high Recall value. On the other hand, F1-score is the harmonic mean between Recall and Precision, and it is desirable to use this metric when one accuracy metric is wanted for the evaluation. The formulas of F1-score, Precision and Recall are shown and explained in Section 2.2.6, Chapter 2.

5.3 Development tools

5.3.1 Software

The implementation of the approach was developed in Python using the following libraries:

- **NumPy** is a Python library that provides a multidimensional array object, and tools used to implement these arrays [56].
- **Pandas** is a Python library used for data manipulation and analysis. More specifically, it provides the ability to manipulate numerical tables and time series [58].
- **PyTorch** is used instead of NumPy to use GPU power. It is a deep learning platform that affords maximum flexibility and speed [95].
- **Matplotlib** is a Python library used for visualization. It is used to create static, interactive, and animated visualizations [87].
- **Scikit-learn** or so-called sklearn is a python library that is built on NumPy, SciPy and Matplotlib, and used for predictive data analysis. In addition, it is capable of providing classification, regression, clustering, dimensionality reduction, model selection and preprocessing features [68].

5.3.2 Hardware

To conduct the experiments, it was crucial to use a lot of power to train the algorithms. The experiments were done using the university's powerful computer that has the specifications demonstrated in Table 5.8.

Hardware	Specification
GPU	128 NVIDIA Tesla V100 GPUs
Memory	12 TB system memory
Storage	256 TB storage
GPU memory	4 TB GPU memory

Table 5.8: The specifications used for the training

Chapter 6

Testing and evaluation

This chapter revolves about the experiments conducted and the achieved results. Recall, Precision and F1-score obtained from each algorithm are going to be demonstrated in the form of tables and heat maps. Furthermore, the best K fold result of each algorithm is going to be represented, and the rest is moved to the appendixes. Additionally, for each model, the cross validation average score is also going to be demonstrated and discussed to evaluate the performance. A further explanation of the results achieved and why we believe that the results ended up as they are will also be reviewed. Finally, a performance comparison between algorithms is going to be conducted. The purpose is to find out which one performs best in terms of Recall, Precision, F1-score and cross validation, when detecting each attack found in the CICIDS2017 dataset.

It is essential to observe and distinguish the model's results to discover which one does perform best in terms of multi-class classification detection. Observing the outcomes from each one would answer questions such as which algorithm performs best regarding multi-class classification intrusion detection? Which algorithm performs best in terms of True Positive Rate (Recall)? And which one provides the best Precision score? How would F1-score be affected based on both Recall and Precision? And finally, is it beneficial to detect each attack, or is it better to detect the attack type (family)?

By the end of this chapter, the Statements in Section 1.3, Chapter 1, will be answered.

6.1 Neural network (NN)

6.1.1 Results

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.94	0.97	454620
Botnet	0.05	1.00	0.10	393
DDoS	0.95	1.00	0.97	25606
DoS GoldenEye	0.87	1.00	0.93	2059
DoS Hulk	0.94	1.00	0.97	46214
DoS Slowhttptest	0.82	1.00	0.90	1100
DoS Slowloris	0.78	0.99	0.87	1159
FTP Patator	0.84	1.00	0.91	1587
Heartbleed	0.50	1.00	0.67	2
Infiltration	0.00	1.00	0.01	7
PortScan	0.76	1.00	0.86	31786
SSH Patator	0.52	0.90	0.66	1180
Web Attack Brute Force	0.18	0.53	0.27	302
Web Attack SQL Injection	0.01	1.00	0.02	4
Web Attack XSS	0.06	0.74	0.11	130

Table 6.1: Neural Network Fold #1. The cross validation result is 93.91.

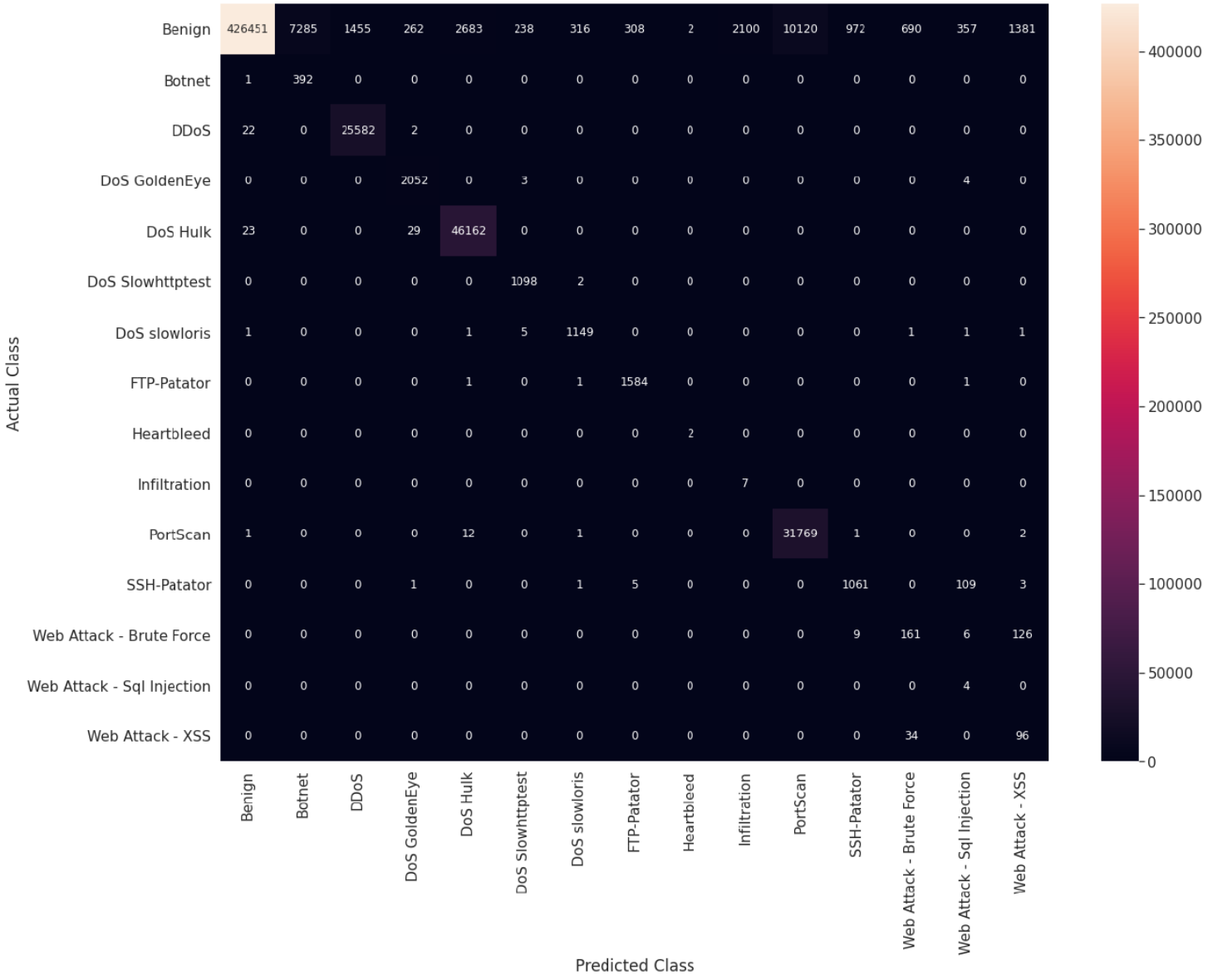

Figure 6.1: Neural Network Confusion Metrics Heat map Fold #1

Folds	**Cross validation**
Fold #0	93.90
Fold #1	93.91
Fold #2	91.97
Fold #3	90.46
Fold #4	91.30
Average	92.30

Table 6.2: Neural Network cross validation results

6.1.2 Discussion

After many attempts with different options, hyperparameters, dimensions and layers, the Neural Network model has lead to the results shown in Table 6.1 and 6.2. We have tried different batch sizes (4096, 40960, 81920 and 122880), a various number of epochs (20, 50 and 100) and other commonly used learning rates (0.1, 0.01 and 0.001). Furthermore, we have also examined different numbers of layers (3 and 5), and a different number of dimensions (128, 256, 512 and 1024). However, utilizing the values shown in Table 5.1 and Table 5.2 resulted in the most significant outcome in terms of True Positive Rate and cross validation.

As shown in Table 6.1, the test dataset has 566148 records, which is 20% of the entire dataset. Out of these records, 454620 records belong to Benign traffic, and the remaining belong to malicious traffic. Figure 6.1 shows that Neural Network algorithm was able to correctly classify 426451 (93.80%) records out of 454620 records that belong to Benign traffic, and the remaining is classified as something else. Confusion metrics heat map shows the correctly classified records of each attack.

As a reminder, Precision is the proportion of attack cases that were correctly predicted relative to the predicted size of the attack class. However, some results show low Precision scores (even after sampling), and this is expected due to the number of records, which are very low compared to other records.

Regarding Precision scores, it is shown that the smaller the class, the worse the Precision score. For instance, Botnet, Brute Force, SQL Injection, XSS and Infiltration have very few records compared to others. The Precision score of these classes shows that the model suffers when detecting them.

The reason why we believe that Botnet's Precision is pretty low is that detecting Botnet traffic is a challenging task. The data traffic that gets generated by each bot is legitimate because the bots themselves are legitimate devices [91]. As an example, the Precision of botnet is calculated by 392/(392+7285), which is the precision formula; 392 are the true positives, and 7285 are false positives. It means that 7285 records of the benign traffic were classified as a botnet.

Moreover, we have realized that for Neural Networks to be very efficient in intrusion detection, a massive amount of data has to be feed to the network, and this is due to its training complexity. That said, since some classes have a few numbers of records (the dataset is highly unbalanced), it indeed does need refinements so that it can get better and more accurate results regarding Precision [1]. How such a problem can be solved is going to be discussed in Conclusions, Chapter 7.

However, in terms of Recall scores, the model performs well and solves the lacking part in Precision, which is the proportion of the genuine attack covered by the classifier. It means that even the Precision is low in some classes, Recall makes it possible to detect the attack as it deals with false negatives. For instance, botnet Recall (True Positive Rate) is approximately 100%, and it is calculated by 392/(392+1) where 1 is the false negative (the prediction is negative, and it is false, meaning that 1 record was classified as benign), and 392 are true positives.

As mentioned earlier, F1-Score is based on both Precision and Recall, that is, a good Precision score means that we have both False Positive Rate and False Negative Rate, which means that the model is able to classify the attack and is not disturbed by false alarms. In our case, F1-score is a better metric score than accuracy because

the dataset is unbalanced [65]. As shown in Table 6.1, the F1-score is the weighted average between Precision and Recall (the formula is shown in section 2.2.6, chapter 2).

Folds 0, 1, 2, 3 and 4 are shown in Table 6.2 (for more details about each fold check the Appendixes). As the tables show, each fold provides different results. The reason is that each class has a different number of records, and dividing the dataset into several splits makes the difference. The highest cross validation score achieved, which is Fold #1, is shown in Table 6.2, and the average score across all 5 folds is 92.30%, which we believe is a good result. More information about each fold can be found in A.1.

6.2 Convolutional Neural Network (CNN)

6.2.1 Results

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.93	0.96	454620
Botnet	0.04	1.00	0.08	393
DDoS	0.91	1.00	0.95	25606
DoS GoldenEye	0.80	0.99	0.88	2059
DoS Hulk	0.92	1.00	0.96	46214
DoS Slowhttptest	0.81	0.98	0.88	1100
DoS Slowloris	0.51	0.99	0.67	1159
FTP Patator	0.86	1.00	0.92	1587
Heartbleed	0.12	1.00	0.21	2
Infiltration	0.00	1.00	0.01	7
PortScan	0.82	1.00	0.90	31786
SSH Patator	0.35	0.99	0.52	1180
Web Attack Brute Force	0.14	0.56	0.23	302
Web Attack SQL Injection	0.00	1.00	0.00	4
Web Attack XSS	0.06	0.81	0.11	130

Table 6.3: Convolutional Neural Network Fold #2. The cross validation result is 94.99%.

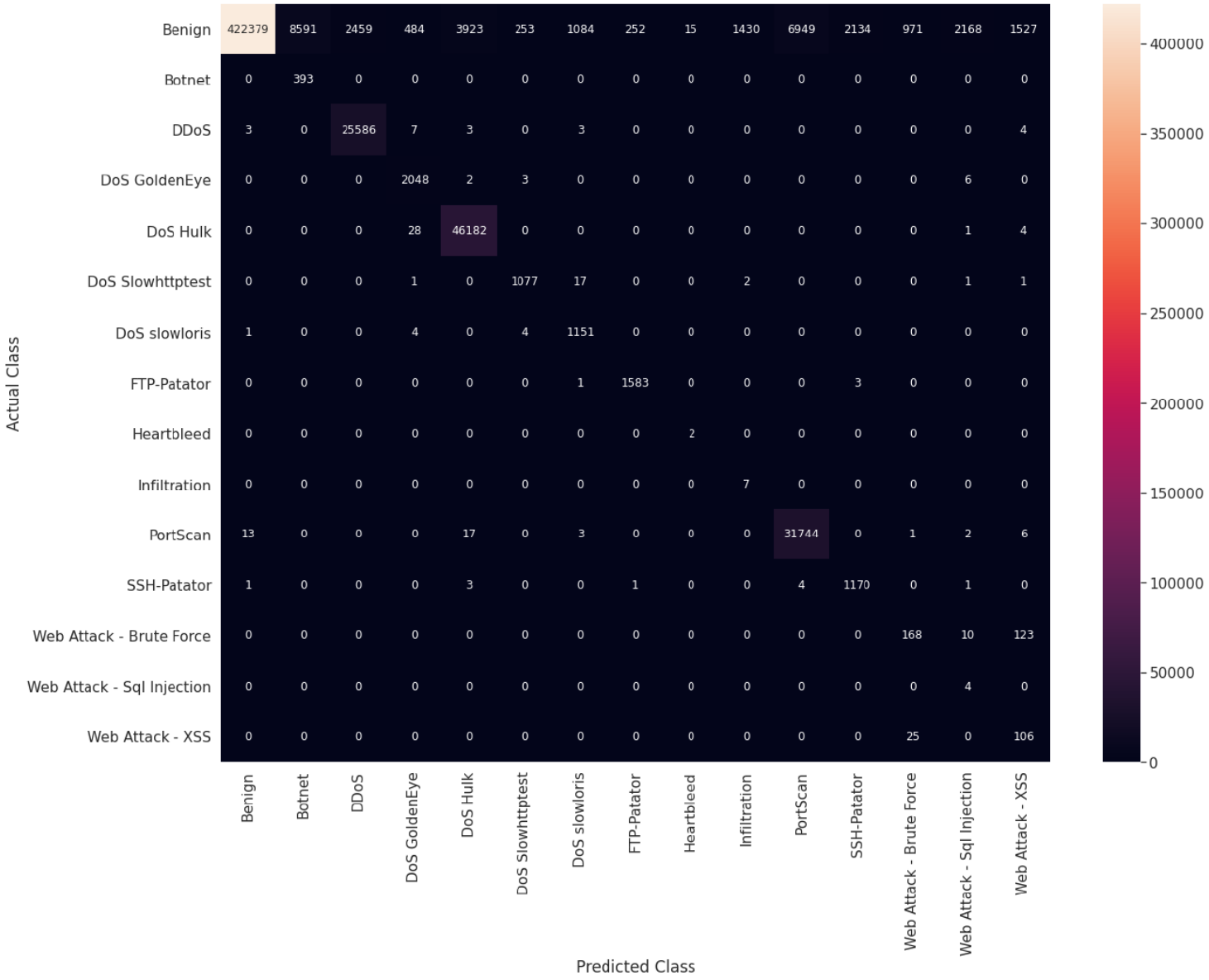

Figure 6.2: Convolutional Neural Network Confusion Metrics heat map Fold #2

Folds	Cross validation
Fold #0	94.08
Fold #1	94.46
Fold #2	94.99
Fold #3	94.53
Fold #4	94.07
Average	94.42

Table 6.4: Convolutional Neural Network cross validation results

6.2.2 Discussion

The results achieved are based on the parameters shown in Section 5.1.2, Chapter 5. Indeed, these results are achieved after conducting several different experiments. Namely, different batch sizes (1024, 4096, 20480 and 81920), various number of epochs (50, 100 and 150), learning rates (0.1, 0.01 and 0.001) and other experiments in regard to the number of layers (output and input channels, kernel size, etc.).

However, just as the Neural Network model, some Precision scores are low because of the number of samples associated with each attack. For instance, Botnet, DoS Slowloris, Heartbleed, SSH Patator, Brute Force, SQL Injection and XSS have few records where the latter two are the worse in comparison to other attacks, and Benign which is the most significant one. It is also important to mention that the difference between DoS attacks is that they are tools used to conduct DoS attacks (see Table 4.3, Wednesday morning). Hence, we assume that would affect the result of DoS Slowloris (notice that DoS Slowloris has less more records than Slowhttpstest but still has higher precision score). Generally, based on the results we have achieved, we realize that the fewer the number of records, the worse the Precision.

Recall score of each attack except Brute Force is pretty satisfying. For some reason, Brute Force attack did not get a high Recall, and this applies for all CNN folds even though we believe that Web Attack Brute Force is easy to detect since the traffic of login attempts is unique (the DVWA framework was used to conduct this attack. Review Table 4.3, Section 4.2, Chapter 4). However, it could also be due to the number of records.

As stated earlier, F1-score is the harmonic mean between Precision and Recall. Hence, the F1-score results would be affected by them. However, some classes such as Benign, DDoS, DoS attacks, FTP Patator and PortScan provide good results, where the rest does not, due to the reason we have mentioned.

Convolutional Neural Network's Fold #2 was able to provide the best cross validation fold result across all folds among all utilized models. As demonstrated in Table 6.3, the cross validation result is 94.99%. The heat map of CNN fold #2 is shown in Figure 6.2. Additionally, the average score (94.42%) and the result of each fold is shown in Table 6.4. More details about each fold are demonstrated in A.2.

6.3 Random forest (RF)

6.3.1 Results

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.99	1.00	454619
Botnet	0.13	1.00	0.23	393
DDoS	1.00	1.00	1.00	25604
DoS GoldenEye	0.99	1.00	0.99	2059
DoS Hulk	0.98	1.00	0.99	46215
DoS Slowhttptest	0.98	0.99	0.99	1099
DoS Slowloris	0.99	0.99	0.99	1160
FTP Patator	1.00	1.00	1.00	1587
Heartbleed	1.00	1.00	1.00	2
Infiltration	0.75	0.86	0.80	7
PortScan	0.99	1.00	1.00	31786
SSH Patator	1.00	1.00	1.00	1180
Web Attack Brute Force	0.73	0.73	0.73	301
Web Attack SQL Injection	0.29	0.50	0.36	4
Web Attack XSS	0.31	0.40	0.35	131

Table 6.5: Random Forest Fold #2. The cross validation result is 89.72%.

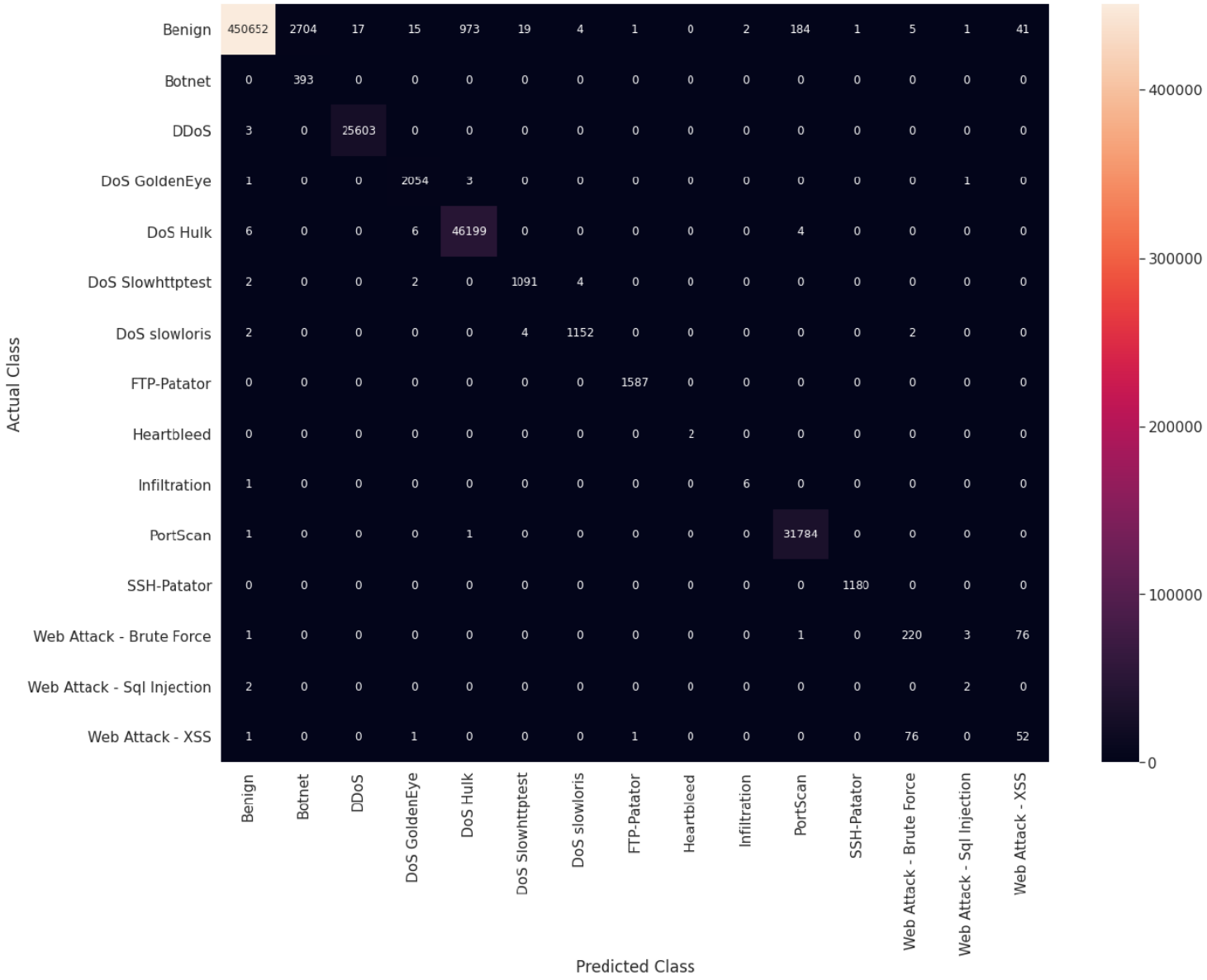

Figure 6.3: Random Forest Confusion Metrics heat map Fold #2

Folds	Cross validation
Fold #0	88.68
Fold #1	86.12
Fold #2	89.72
Fold #3	85.76
Fold #4	88.41
Average	87.73

Table 6.6: Random Forest cross validation results

6.3.2 Discussion

Getting the most out of the Random Forest required trying different parameters to achieve the best possible results. We have tried different numbers of Decision Trees (10, 20, 25 and 30) in combination with the default values of Min Samples Split and Min Samples Leaf, which is 2 and 1 respectively. Furthermore, we have combined each number of the Decision Trees with different numbers of both Min Samples Split and Min Samples Leaf. However, we have ended up choosing the values of each parameter as shown in Table 5.5; namely, 20 trees, 2 (default) Min Samples Split and 1 (default) Min Samples Leaf.

Table 6.5 shows the best cross validation result (89.72%), and Figure 6.3 shows its heat map. The Random Forest classifier was able to provide satisfying results even some attacks have few numbers of records, such as Heartbleed and Infiltration. Classes such as Web attack XSS, SQL injection and Botnet did get pretty poor results compared to other attacks. It could be because of their small number of records, and other issues such as those mentioned in Section 6.1.2.

Generally, based on the results Table 6.5 shows, we realize that large, and the most of small classes often have great Precision compared to NN, CNN and LSTM (as we will see shortly). The results we have achieved are quite satisfying for the most part, regardless the issues mentioned in Section 6.1.2. F1-score performs exceptionally well in big classes, and in classes with a fewer number of records such as Infiltration and Heartbleed.

In a Random Forest architecture, each Decision Tree gives a vote that indicates the tree's decision. The dataset utilized in this thesis is quite large; it contains 76 various features (after removing two features and vectorizing the label feature, as discussed in Section 5.1.1, Chapter 5). Random Forest has the advantage that it runs effectively on large and unbalanced datasets compared to other algorithms. One of the reasons we thought Random Forest would become a success is that it has no nominal data problem and does not over fit the data. In Random Forest, one does not need to provide cross validation or test-set for predicting the test error [40]. However, we have performed cross validation to determine the optimal values for hyperparameters.

Regarding the cross validation results, they are not the best so far, but the results are acceptable. Table 6.6 shows the results of the cross validation, where the average result is 87.73%. More details about each fold can be found in A.3.

6.4 Long Short-Term Memory (LSTM)

6.4.1 Results

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.95	0.97	454620
Botnet	0.07	0.99	0.12	393
DDoS	0.95	1.00	0.97	25606
DoS GoldenEye	0.88	1.00	0.94	2059
DoS Hulk	0.93	1.00	0.97	46214
Dos Slowhttptest	0.81	0.99	0.89	1100
DoS Slowloris	0.77	0.99	0.87	1159
FTP Patator	0.79	1.00	0.88	1587
Heartbleed	1.00	1.00	1.00	2
Infiltration	0.00	0.86	0.01	7
PortScan	0.83	1.00	0.91	31786
SSH Patator	0.58	0.99	0.73	1180
Web Attack Brute Force	0.16	0.43	0.24	302
Web Attack SQL Injection	0.01	0.75	0.01	4
Web Attack XSS	0.07	0.81	0.13	130

Table 6.7: Long Short-Term Memory Fold #1. The cross validation result is 91.78%.

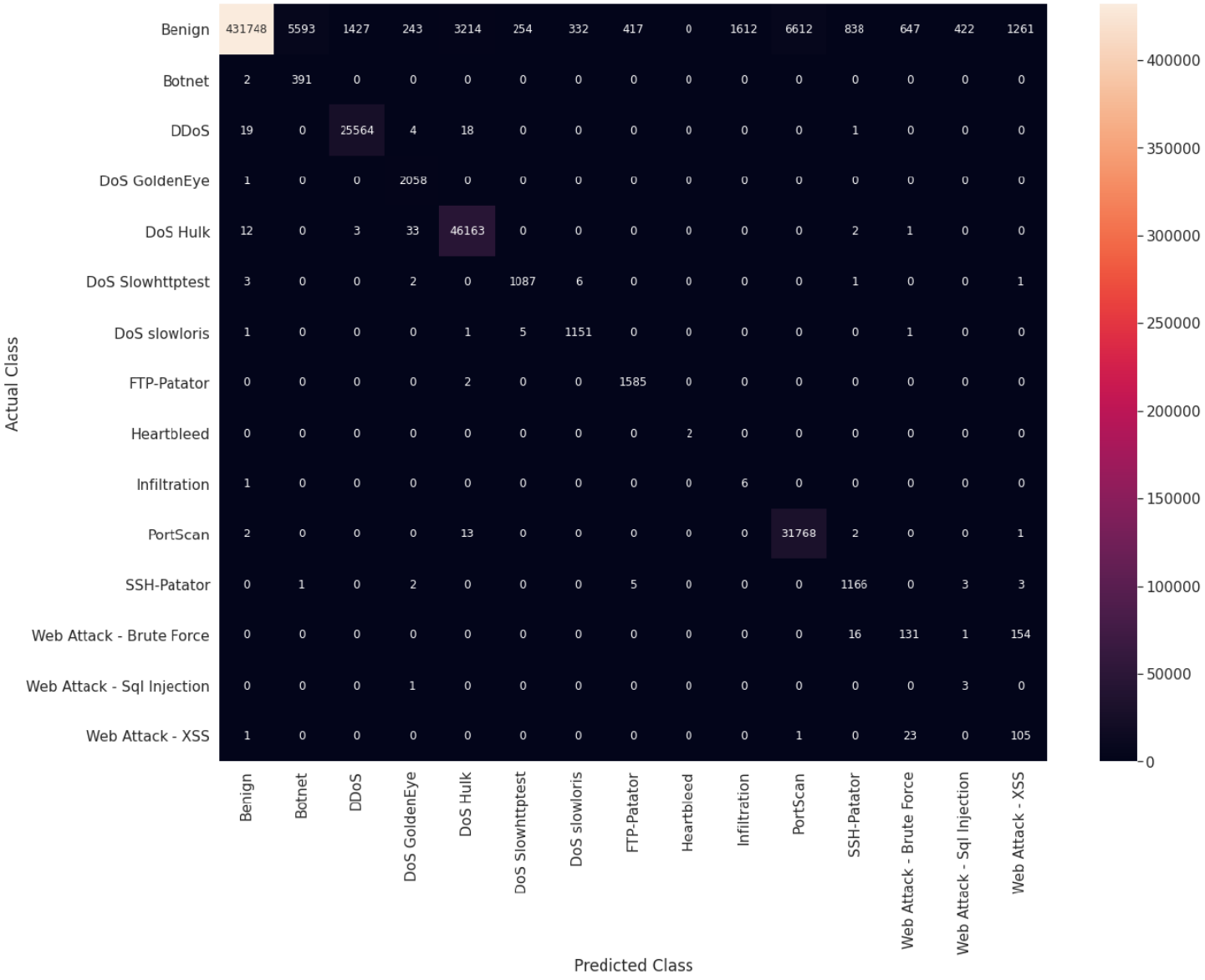

Figure 6.4: Long Short-Term Memory Confusion Metrics heat map Fold #1

Folds	**Cross validation (%)**
Fold #0	84.78
Fold #1	91.78
Fold #2	86.42
Fold #3	86.85
Fold #4	89.88
Average	87.94

Table 6.8: Long Short-Term Memory cross validation results

6.4.2 Discussion

An unavoidable disadvantage of LSTM is it takes a lot of time to train. Hence, it was reasonable to use a large batch size (81920) and 100 epochs. However, the results have shown that the larger the batch size, the worse the results, and since the number of epochs is 100, it took more time than expected to train.

The next attempt was the decrease both batch size to 4096 and the number of epochs to 50 (if the number of epochs remained the same, training the algorithm would have taken a much longer time), and a commonly used learning rate value which is 0.01, see Table 5.7. Using these adjusted parameters have resulted in better results as shown in Table 6.7 and Figure 6.4.

The Precision of LSTM suffers when it comes to small classes. For instance, classes such as Botnet, SSH Patator, Web Attack Brute Force, Web Attack SQL Injection and Web Attack XSS get low Precision scores due to the number of classes and the issues mentioned in Section 6.1.2, as we believe. Additionally, Section 5.2, Chapter 5, also talks about the balance problem that would affect the results.

True Positive Rate or Recall score shows pretty good results when detecting most attacks except Web Attack Brute Force and Web Attack SQL Injection. However, by taking both Precision and Recall into account, some of the F1-scores seem to be satisfying. Classes such as Botnet, Infiltration, and all web attacks got terrible results, and this is due to the low scores of Precision.

Table 6.8 shows the results of each fold. The score of Fold #1 was able to reach 92.78%, and its Table and Heat Map are shown in 6.7 and 6.4, respectively. However, the average score across five-folds was 87.84%, which we believe is acceptable. More details about each fold and the heat maps can be found in A.4.

6.5 Comparing the algorithms

6.5.1 Precision

Attack	**Precision NN-Fold#1**	**Precision RF-Fold#2**	**Precsion CNN-Fold#2**	**Precision LSTM-Fold#1**
Benign	1.00	1.00	1.00	1.00
Botnet	0.05	0.13	0.04	0.07
DDoS	0.95	1.00	0.91	0.95
DoS GoldenEye	0.87	0.99	0.80	0.88
DoS Hulk	0.94	0.98	0.92	0.93
Dos Slowhttptest	0.82	0.98	0.81	0.81
DoS Slowloris	0.78	0.99	0.51	0.77
FTP Patator	0.84	1.00	0.86	0.79
Heartbleed	0.50	1.00	0.12	1.00
Infiltration	0.00	0.75	0.00	0.00
PortScan	0.76	0.99	0.82	0.83
SSH Patator	0.52	1.00	0.35	0.58
Web Attack Brute Force	0.18	0.73	0.14	0.16
Web Attack SQL Injection	0.01	0.29	0.00	0.01
Web Attack XSS	0.06	0.31	0.06	0.07
Average	***0.55***	***≈0.81***	***≈0.49***	***0.59***

Table 6.9: Precision comparison

6.5.2 Recall

Attack	Recall NN-Fold#1	Recall RF-Fold#2	Recall CNN-Fold#2	Recall LSTM-Fold#1
Benign	0.94	0.99	0.93	0.94
Botnet	1.00	1.00	1.00	0.99
DDoS	1.00	1.00	1.00	1.00
DoS GoldenEye	1.00	1.00	0.99	1.00
DoS Hulk	1.00	1.00	1.00	1.00
Dos Slowhttptest	1.00	0.99	0.98	0.99
DoS Slowloris	0.99	0.99	0.99	0.99
FTP Patator	1.00	1.00	1.00	1.00
Heartbleed	1.00	1.00	1.00	1.00
Infiltration	1.00	0.86	1.00	0.86
PortScan	1.00	1.00	1.00	1.00
SSH Patator	0.90	1.00	0.99	0.99
Web Attack Brute Force	0.53	0.73	0.56	0.43
Web Attack SQL Injection	1.00	0.50	1.00	0.75
Web Attack XSS	0.74	0.40	0.81	0.81
Average	***≈0.94***	***≈0.90***	***0.95***	***≈0.92***

Table 6.10: Recall comparison

6.5.3 F1-score

Attack	F1-Score NN-Fold#1	F1-Score RF-Fold#2	F1-Score CNN-Fold#2	F1-Score LSTM-Fold#1
Benign	0.97	1.00	0.96	0.97
Botnet	0.10	0.23	0.08	0.12
DDoS	0.97	1.00	0.95	0.97
DoS GoldenEye	0.93	0.99	0.88	0.94
DoS Hulk	0.97	0.99	0.96	0.97
Dos Slowhttptest	0.90	0.99	0.88	0.89
DoS Slowloris	0.87	0.99	0.67	0.87
FTP Patator	0.91	1.00	0.92	0.88
Heartbleed	0.67	1.00	0.21	1.00
Infiltration	0.01	0.80	0.01	0.01
PortScan	0.86	1.00	0.90	0.91
SSH Patator	0.66	1.00	0.52	0.73
Web Attack Brute Force	0.27	0.73	0.23	0.24
Web Attack SQL Injection	0.02	0.36	0.00	0.01
Web Attack XSS	0.11	0.35	0.11	0.13
Average	***≈0.61***	***≈0.83***	***≈0.55***	***≈0.64***

Table 6.11: F1-Score comparison

6.5.4 Cross validation

Algorithm	Cross validation
Neural Network	92.30
Random Forest	87.73
Convulotional Neural Network	94.42
Long Short-Term Memory	87.94

Table 6.12: Average cross validation comparison

6.5.5 Discussion

Table 6.9 shows a comparison between our models. Random Forest model has proven that it has the best Precision scores. However, all models suffer in terms of Botnet detection due to the type of traffic as we have mentioned earlier. Although the little number of records both Infiltration and Heartbleed have, Random Forest was able to get the highest Precision score compared to the rest of the models. We can also observe that it has a much better score regarding web attacks.

The best average score across all models in terms of Precision is the result of Random Forest (81%). The rest of the algorithms performed poorly in terms of average score. We are going to discuss how to avoid this problem and get better results in Conclusions, Chapter 7.

The Recall score (True Positive Rate) provides good results. Concerning the average Recall score (95%), Convolutional Neural Network had the best score, as demonstrated in Table 6.11. Despite that it had the best average score, Random Forest does have a much better score in terms of Web Attack Brute Force detection. CNN has also gotten a lower result than the other models regarding Benign traffic. Random Forest average Recall score had the lowest score among the others, and it is due to the poor performance in detecting web attacks such as SQL injection and XSS. Both LSTM and CNN perform best when detecting XSS, whereas NN and CNN perform best when detecting SQL Injection. In generic terms, we can observe that web attacks' detection provides the poorest results (excluding SQL Injection when utilizing NN and CNN) compared to the detection of other attacks due to the issues we mentioned earlier.

Nevertheless, F1-score gets impacted significantly due to the low scores of Precision. This applies for all implemented models, expect Random Forest.

As shown in Table 6.12, observe that the Convolutional Neural Network provides the best cross validation result among all other applied algorithms. Hence, we can conclude that CNN performs best, then comes NN. Indeed, LSTM and RF are acceptable.

Answering Statement 1 in Section 1.3, Chapter 1, improving the detection of an intrusion detection system relies on various factors; the form of data that is feed to the model (i.e. balanced, imbalanced, types of features, labelled or not, and so on), the size of the data, number of classes, whether the detection is a binary or multi-class classification or not, how the models are built and their hyperparameters, and other factors.

By examining those four models, we have discovered how they would perform in terms of multi-class classification. As mentioned in Statement 1, Section 1.3, we desired to find out how each of the algorithms would perform using this exact dataset (CICIDS2017) by detecting each attack. Identifying each attack could be beneficial if the desire is to discover how the attacks are conducted using which tool. The ability to detect each attack makes it possible to specify the attack conducted at what time, and which countermeasures should be considered to prevent such an attack. However, Table 2.2 in Section 2.2.2, Chapter 2, mentions that in an anomaly-based IDS, security analysts are required to figure out what triggered an alarm. Hence, we would say that our approach would solve such a problem.

Sometimes, the main goal is to stop the attack. For instance, the purpose of

conducting DoS attacks is to make systems unavailable. We believe that preventing a DoS attack is much more important than identifying the type. Hence, a better approach that could be carried out is to concatenating all DoS attacks with each other. It can be done by relabelling each of them so that all of them are called "DoS" (this approach will be explained in Chapter 7 as a Further work).

Nevertheless, we believe that achieving high Recall values is an indicator that the models perform well. The reason is that True Positive Rate (which is the same as Recall) is quite high, meaning that most of the attacks get classified correctly. Thus, although the Precision score is low, Recall will overcome this issue by correctly classifying the attacks.

Regarding Statement 2 in Section 1.3, Chapter 1, we have answered the questions in both Section 4.4, Chapter 4, and this chapter where we have demonstrated and discussed the results.

Chapter 7

Conclusions

In this thesis, we have utilized the CICIDS2017 dataset provided by Canadian Institute for Cybersecurity for machine learning. The dataset is quite large, as it contains 2830743 records, 79 features and 15 types of attacks.

This research has focused on four different algorithms; Neural Network, Convolutional Neural Network, Random Forest and Long Short-Term Memory which are quite common to utilize in this filed of research.

As the research has demonstrated, the Neural Network, Convulotional Neural Network, Random Forest and Long Short-Term Memory models have achieved average scores across five-folds at 92.30%, 94.42%, 87.73% and 87.94%, respectively.

By analyzing the results, we could observe that average Recall score of each model models is quite high, where the lowest average is achieved by Random Forest, which is 90%, as observed in Table 6.11.

Regarding the Precision results, we have observed that there are some issues when detecting classes with few records. As we believe, the reason is due the imbalance of the dataset. Hence, an approach that could be done to overcome this challenge will be discussed in Further Work. Besides, since many attacks do have low Precision score, F1-Score is automatically affected.

In this research field, many papers merely classify on normal and abnormal traffic, such as those presented in Section 3.2.6, Chapter 3. Contrary to our case, we have performed prediction on every attack scenario in the dataset. Therefore, the results we have achieved became less accurate than theirs. Additionally, in contrast to other research papers, our experiments utilize the whole CICIDS2017 dataset, which contains real world data.

Further Work

If this research were to continue, there would be improvements that could enhance the results. As an example, to achieve a higher Precision score, instead of detecting each class, what could be done is to detect attacks based on which family they belong to, using CICIDS2017. This approach might be conducted by combining all DoS attacks and have them as one class. Similarly, all web attacks can be concatenated with each other, so they are one class. Additionally, FTP Patator and SSH Patator can be one class because they are Brute Force attacks, and PortScan as another class.

Finally, Infiltration and Heartbleed are two other classes (Heartbleed might also be added to DoS due to the minimal number of classes). We believe that reconstructing (relabelling) the data in that way would be beneficial because the number of records will increase (the dataset gets more balanced); hence much better Precision and F1-score results. However, we have stated earlier that it is not what we aim for in this thesis, but it regarded KDD99 and NSL-KDD datasets which are old and have several disadvantages, as discussed in Section 2.5.6, Chapter 2.

An additional improvement that could be done is re-implement the models in C instead of Python, which will impact the execution time tremendously. It is also possible to benefit from Camel case notation, as it is quite a typical code style, and introducing it to our code might be beneficial and more readable for other developers.

Implementing the utilized models presented in this thesis in a live system, and measure the models' performance be beneficial to see how the models perform in the real world.

When implementing the four different models, a lot of different hyperparameters have been tried out. By trying out even further hyperparameters, could the results become even better than we have accomplished?

In terms of CICIDS2017, the dataset contains formerly 79 (before removing two features as we have shown in this thesis) distinct features. By removing even furthers that are nonessential for the result, could the prediction results become even more significant? It is especially important to try out using LSTM because it takes a substantial amount of time to train.

Bibliography

[1] A. A. Hassan, A. F. Sheta , T. M. Wahbi , "Intrusion Detection Using Neural Network: A Literature Review," vol. 6, no. 9, pp. 346, Sep. 2017.

[2] A. Budhiraja, "Learning Less to Learn Better - Dropout in (Deep) Machine learning," *Medium,* 15-Dec-2016. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-mac [Accessed: 01-May-2020].

[3] A. Chazareix, "About Convolutional Layer and Convolution Kernel," *Sicara,* 31-Oct-2018. [Online]. Available: https://www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel. [Accessed: 26-May-2020].

[4] A. Divekar, M. Parekh, V. Savla, R. Mishra and M. Shirole, "Benchmarking datasets for Anomaly-based Network Intrusion Detection: KDD CUP 99 alternatives," *arxiv,* 13-Nov-2018. [Online]. Available: https://arxiv.org/abs/1811.05372. [Accessed: 23-Feb-2020].

[5] A. Jahan and M. A. Alam, "Intrusion Detection Systems based on Artificial Intelligence," *International Journal of Advanced Research in Computer Science,* vol. 8, No. 5, ISSUE No. 0976-5697, 2017.

[6] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity,* 17-Jul-2019. [Online]. Available: https://cybersecurity.springeropen.com/articles/10.1186/s42400-019-0038-7/tables/4. [Accessed: 03-Feb-2020].

[7] A. Lazarevic and J. Srivastava, "INTRUSION DETECTION: A SURVEY," in *Managing Cyber Threats,* vol. 5, V.Kumar, Ed. Boston, MA: Springer, 2005. [Accessed: 15-Feb-2020].

[8] A. Mishra, "Metrics to Evaluate your Machine Learning Algorithms," *Medium,* 24-Feb-2018. [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234. [Accessed: 04-Feb-2020].

[9] A. Mohan, "Cross-Validation for Imbalanced Datasets," *Medium,* 05-Mar-2019. [Online]. Available: https://medium.com/lumiata/cross-validation-for-imbalanced-datasets-9d203ba47e8. [Accessed: 09-May-2020].

[10] A. Nayyar, "The Best Open Source Network Intrusion Detection Tools," *Open-Source,* 10-Apr-2017. [Online]. Available: https://opensourceforu.com/2017/04/best-open-source-network-intrusion-detection-tools/. [Accessed: 29-Feb-2020].

[11] A. Ravanshad, "Ensemble Methods," *Medium,* 28-Apr-2018. [Online]. Available: https://medium.com/@aravanshad/ensemble-methods-95533944783f. [Accessed: 27-Mar-2020].

[12] A. Verma, "Pytorch [Basics] - Intro to CNN," *Medium,* 01-Jan-2018. [Online]. Available: https://towardsdatascience.com/pytorch-basics-how-to-train-your-neural-net-intro-to-cnn-26a14c2ea29. [Accessed: 01-May-2020].

[13] "Botnet," *Wikipedia,* 25-Feb-2020. [Online]. Available: https://en.wikipedia.org/wiki/Botnet. [Accessed: 15-Mar-2020].

[14] C. C. Aggarwal, "Outlier Ensembles," *IBM T. J. Watson Research Center,* vol. 14, No. 2, pp. 49-57, 2013. [Online]. Available: https://dl.acm.org/doi/10.1145/2481244.2481252. [Accessed: 19-Feb-2020].

[15] "Classification Algorithms - Decision Tree," *Tutorialspoint.*[Online]. Available: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_decision_tree.htm. [Accessed: 08-Mar-2020].

[16] "Classification Algorithms - Random Forest," *Tutorialspoint.* [Online]. Available: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_random_forest.htm. [Accessed: 08-Mar-2020].

[17] C. Luo, L. Wang, and H. Lu, "Analysis of LSTM-RNN Based on Attack Type of KDD-99 Dataset," *SpringerLink*, 01-Nov-2018. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-00006-6_29. [Accessed: 22-Feb-2020].

[18] "Cross-validation: evaluating estimator performance," *scikit.* [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html. [Accessed: 08-May-2020].

[19] D. D. Protic, "REVIEW OF KDD CUP '99, NSL-KDD AND KYOTO 2006+ DATASETS," *Center for Applied Mathematics and Electronics.* [Online]. Available: https://www.researchgate.net/publication/326000849_Review_of_KDD_Cup_'99_NSL-KDD_and_Kyoto_2006_datasets. [Accessed: 23-Feb-2020].

[20] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, *SpringerLink,* 27-Sep-2017. [Online]. Available: https://link.springer.com/article/10.1007/s10586-017-1117-8. [Accessed: 06-Feb-2020].

[21] D. Thakur, "LSTM and its equations," *Medium,* 06-Jul-2018. [Online]. Available: https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af. [Accessed: 07-Mar-2020].

[22] E. Korneev, "LSTM Neural Networks for Anomaly Detection," *Medium,* 20-Dec-2018. [Online]. Available: https://medium.com/datadriveninvestor/lstm-neural-networks-for-anomaly-detection-4328cb9b6e27. [Accessed: 21-Feb-2020].

[23] "Examining Different Types of Intrusion Detection Systems," *Dummies - A Wiley Brand.* [online]. Available: https://www.dummies.com/computers/operating-systems/windows-xp-vista/examining-different-types-of-intrusion-detection-systems/. [Accessed: 14-Feb-2020]

[24] "FIREWALL VS IPS VS IDS," *IPWITHEASE,* 14-Sep-2017. [Online]. Available: https://ipwithease.com/firewall-vs-ips-vs-ids/. [Accessed: 26-Feb-2020].

[25] "Frameworks for Approaching the Machine Learning Process." *KDnuggets,* May 2018. [Online]. Available: www.kdnuggets.com/2018/05/general-approaches-machine-learning-process.html. [Accessed: 18-May-2020].

[26] F. Peccia, "Batch normalization: theory and how to use it with Tensorflow," *Medium,* 16-Sep-2018. [Online]. Available: https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad. [Accessed: 01-May-2020].

[27] G. Kumar, "Evaluation Metrics for Intrusion Detection Systems - A Study," vol. 2, no. 11, pp. 12-15, Nov. 2014.

[28] H. Altwaijry and S. Algarny, "Bayesian based intrusion detection system," *Journal of King Saud University - Computer and Information Sciences,* vol. 24, no. 1, pp. 1-6, Nov. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157811000292. [Accessed: 08-Feb-2020].

[29] H. Andrew, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," *arXiv.org,* 05-Feb-2014. [Online]. Available: https://arxiv.org/abs/1402.1128. [Accessed: 27-May-2020].

[30] H. Lui and B. Lang, "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey," *mdpi.* [Online]. Available: https://www.mdpi.com/2076-3417/9/20/4396. [Accessed: 26-Jan-2020].

[31] H. Pokharna, "The best explanation of Convolutional Neural Networks on the Internet!," *Medium,* 28-Jul-2016. [Online]. Available: https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5d [Accessed: 09-Apr-2020].

[32] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "A Detailed Analysis of the CICIDS2017 Data Set," *SpringerLink,* 05-Jul-2019. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-25109-3_9. [Accessed: 11-Mar-2020].

[33] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization", *Proc. 4th Int. Conf. Inf. Syst. Security Privacy,* pp. 108-116, 2018. [online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006639801080116. [Accessed: 10-Jan-2020].

[34] "Intrusion Detection Evaluation Dataset (CICIDS2017)," *UNB University of Brunswick, Canadian Institute for Cybersecurity,* 2017. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html. [Accessed: 09-Mar-2020].

[35] "Intrusion Detection System (IDS)," *GeeksforGeeks,* 16-Jan-2020. [online]. Available: https://www.geeksforgeeks.org/intrusion-detection-system-ids/. [Accessed: 25-Jan-2020].

[36] J. Clement, "Global digital population 2020," *Statista,* 24-Apr-2020. [Online]. Available: https://www.statista.com/statistics/617136/digital-population-worldwide/. [Accessed: 07-Apr-2020].

[37] J. Czakon, "F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?: Neptune's blog," *neptune.ai,* 04-Nov-2019. [Online]. Available: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc. [Accessed: 27-May-2020].

[38] J. LaPiedra, "Global Information Assurance Certification Paper," *GIAC CERTIFICATIONS*. [Online]. Available: https://www.giac.org/paper/gsec/501/information-security-process-prevention-detection-response/101197. [Accessed: 01.02.2020].

[39] J. Sen, V. Jyothsna, and K. M. Prasad, "Anomaly-Based Intrusion Detection System," in *Computer and Network Security,* 11-Jun-2019. [Online]. Available: https://www.intechopen.com/online-first/anomaly-based-intrusion-detection-system. [Accessed: 06-Feb-2020].

[40] J. Zhang and M. Zulkernine, "Network Intrusion Detection using Random Forests," *semanticscholar*. [Online]. Available: https://pdfs.semanticscholar.org/7e55/90bb681cc32f76479984c3e0ff3f35235b8f.pdf. [Accessed: 21-May-2020].

[41] L. Basora, X. Olive, and T. Dubot, "Recent Advances in Anomaly Detection Methods applied to Aviation," *Preprints,* 29-Sep-2019. [Online]. Available: https://www.preprints.org/manuscript/201909.0326/v1. [Accessed: 20-Feb-2020].

[42] L. Bermudez, "Overview of Neural Networks," *Medium,* 14-Nov-2017. [Online]. Available: https://medium.com/machinevision/overview-of-neural-networks-b86ce02ea3d1. [Accessed: 01-Mar-2020].

[43] L. Bontemps, V. L. Cao, J. McDermott, and N. Le-Khac, "Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network," *arxiv,* 28-Mar-2017. [Online]. Available: https://arxiv.org/abs/1703.09752. [Accessed: 21-Feb-2020].

[44] L. Bontemps, V. L. Cao, J. McDermott, and N. Le-Khac, "Collective Anomaly Detection Based on Long Short-Term Memory Recurrent Neural Networks," *SpringerLink,* 23-Oct-2016. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-48057-2_9. [Accessed: 22-Feb-2020].

[45] L. Jacobson, "Introduction to Artificial Neural Networks - Part 1," *The Project Spot,* 15-Dec-2013. [Online]. Available: http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7. [Accessed: 02-Mar-2020].

[46] M. A. F. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing,* 02-Jan-2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016516841300515X?via=ihub. [Accessed: 20-Feb-2020].

[47] M. Amer and M. Goldstein, "Nearest-Neighbor and Clustering based Anomaly Detection Algorithms forRapidMiner," *Department of Computer Science and Engineering, German University in Cairo.* [Online]. Available: https://www.researchgate.net/publication/230856452_Nearest-Neighbor_and_Clustering_based_Anomaly_Detection_Algorithms_for_RapidMiner. [Accessed: 18-Feb-2020].

[48] M. B. Fraj, "InDepth: Parameter tuning for Decision Tree," *Medium,* 20-Dec-2017. [Online]. Available: https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3. [Accessed: 26-May-2020].

[49] "McAfee NSP: IDPS Product Overview and Analysis," *eSecurity Planet: Internet Security for IT Professionals,* 20-Feb-2018. [Online]. Available: https://www.esecurityplanet.com/products/mcafee-nsp.html. [Accessed: 29-Feb-2020].

[50] M. Chapple and D. Seidl, "Building an Incident Response Program" in *CompTIA CySA+ STUDY GUIDE,* New York, United States: John Wiley & Sons Inc, 2017, pp. 148.

[51] M. Chapple and D. Seidl, "Defense-in-Depth Security Architectures" in *CompTIA CySA+ STUDY GUIDE,* New York, United States: John Wiley & Sons Inc, 2017, pp. 311.

[52] M. Chapple and D. Seidl, "Policy and Compliance," in *CompTIA CySA STUDY GUIDE,* New York, United States: John Wiley & Sons Inc, 2017, pp. 285-286.

[53] M. Jain, "The Basics of Neural Networks," *Medium,* 17-Jan-2019. [Online]. Available: https://medium.com/datadriveninvestor/the-basics-of-neural-networks-304364b712dc. [Accessed: 01-Mar-2020].

[54] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," *IEEE Xplore,* 18-Dec-2009. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5356528. [Accessed: 23-02-2020].

[55] N. Khamphakdee, N. Benjamas and S. Saiyod, "Improving Intrusion Detection System Based on Snort Rules for Network Prob Attacks Detection With Association Rules Technique of Data Mining," *Department of Computer Science, Faculty of Science, Khon Kaen University,* vol. 8, No. 3, ISSUE No. 234- 25, pp. 238, 2015.

[56] "NumPy," *NumPy.* [Online]. Available: https://numpy.org/. [Accessed: 01-Apr-2020].

[57] "Optimizing Hyperparameters for Random Forest Algorithms in scikit-learn," *Medium,* 26-Jul-2019. [Online]. Available: https://medium.com/@ODSC/optimizing-hyperparameters-for-random-forest-algorithms-in-scikit-learn-d60b7aa0
[Accessed: 26-May-2020].

[58] "Pandas (software)," *Wikipedia*, 16-Mar-2013. [Online]. Available: https://en.wikipedia.org/wiki/Pandas_(software). [Accessed: 01-Apr-2020].

[59] P. Choudhary, "Introduction to Anomaly Detection," *Oracle AI and Data Science Blog,* 15-Feb-2017. [Online]. Available: https://blogs.oracle.com/datascience/introduction-to-anomaly-detection. [Accessed: 04-Feb-2020].

[60] P. Gudikandula, "Recurrent Neural Networks and LSTM explained," *Medium,* 27-Mar-2019. [Online]. Available: https://medium.com/@purnasaigudikandula/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9. [Accessed: 07-Mar-2020].

[61] P. Huilgol, "Accuracy vs. F1-Score," Medium, 24-Aug-2019. [Online]. Available: https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2. [Accessed: 27-May-2020].

[62] P. Negandhi, Y. Trivedi, and R. Mangrulkar, "Intrusion Detection System Using Random Forest on the NSL-KDD Dataset," *SpringerLink,* 11-Sep-2019. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-13-6001-5_43. [Accessed: 16-Mar-2020].

[63] "Port Scan attacks and its detection methodologies," *VALUE @ Amrita,* 2011. [Online]. Available: http://vlab.amrita.edu/?sub=7&brch=199&sim=362&cnt=1. [Accessed: 15-Mar-2020].

[64] R. Jain, "Decision Tree. It begins here.," *Medium,* 17-Jan-2019. [Online]. Available: https://medium.com/@rishabhjain_22692/decision-trees-it-begins-here-93ff54ef134. [Accessed: 08-Mar-2020].

[65] R. Joshi, "Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures," *Exsilio Blog,* 09-Sep-2016. [Online]. Available: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/.
[Accessed: 21-May-2020].

[66] R. S. Langde and A.P. Wadhe, "Misuse Detection System Using Various Techniques: A Review," *International Journal of Advanced Research in Computer Science,* vol. 4, No.6, ISSUE No. 0976-5697, 2013.

[67] S. Bhatt, "Reinforcement Learning 101," *Medium,* 19-Mar-2018. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292. [Accessed: 22-Mar-2020].

[68] "scikit-learn," *scikit-learn.* [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 03-Apr-2020].

[69] S. Gates, "Web-Based Attacks Threaten Apps and Data," *Dyn,* 19-Jan-2018. [Online]. Available: https://dyn.com/blog/web-based-attacks-threaten-apps-and-data/. [Accessed: 09-Mar-2020].

[70] S. Hariri, M. C. Carrasco, and R. J. Brunner, "Extended Isolation Forest," *arXiv.org,* 05-Nov-2019. [Online]. Available: https://arxiv.org/abs/1811.02141. [Accessed: 22-Mar-2020].

[71] "Simple. Flexible. Powerful.," *Keras.* [Online]. Available: https://keras.io/. [Accessed: 01-Apr-2020].

[72] "sklearn.tree.DecisionTreeClassifier," *scikit.* [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. [Accessed: 01-May-2020].

[73] "sklearn.metrics.balanced_accuracy_score," *scikit.* [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html. [Accessed: 15-May-2020].

[74] S. M. A. Al Mamun and M. Beyaz, "STM Recurrent Neural Network (RNN) for Anomaly Detection in Cellular Mobile Networks," *SpringerLink,* 10-May-2019. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-19945-6_15. [Accessed: 21-Feb-2020].

[75] S. Mujeeb, N. Javaid, M. Ilahi, Z. Wadud, F. Ishmanov, and M. K. Afzal, "Deep Long Short-Term Memory: A New Price and Load Forecasting Scheme for Big Data in Smart Cities," *MDPI*, 14-Feb-2019. [Online]. Available: https://www.mdpi.com/2071-1050/11/4/987/htm. [Accessed: 11-May-2020].

[76] S. Nahari, "How Attackers Infiltrate the Supply Chain & What to Do About It," *IformationWeek IT NETWORK, DARKReading,* 16-Jul-2019. [Online]. Available: https://www.darkreading.com/risk/how-attackers-infiltrate-the-supply-chain-and-what-to-do-about-it/a/d-id/1335234. [Accessed: 23-Mar-2020].

[77] S. Noel, D. Wijesekera and C. Youman, "Modern Intrusion Detection, Data Mining, and Degrees of Attack Guilt," *Center for Secure Information Systems.* [Online]. Available: https://www.researchgate.net/publication/228549884_Modern_Intrusion_Detection_Data_Mining_and_Degrees_of_Attack_Guilt. [Accessed: 04.02.2020].

[78] S. Omar, A. Ngadi and H. H. Jebur, "Machine Learning Techniques for Anomaly Detection: An Overview," *Faculty of Computing, Universiti Teknologi Malaysia,* vol. 79, No. 2, pp. 33-39, 2013. [Online]. Available: https://pdfs.

semanticscholar.org/0278/bbaf1db5df036f02393679d485260b1daeb7.pdf. [Accessed: 06-Feb-2020].

[79] S. Sharma, "Activation Functions in Neural Networks," *Medium,* 06-Sep-2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed: 01-May-2020].

[80] "Supervised vs Unsupervised Learning: Key Differences," *Guru99*. [Online]. Available: https://www.guru99.com/supervised-vs-unsupervised-learning.html. [Accessed: 01-Feb-2020].

[81] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," *IEEE Xplore,* 08-Dec-2016. [Online]. Available: https://ieeexplore.ieee.org/document/7777224. [Accessed: 07-Feb-2020].

[82] T. Balodi, "Convolutional Neural Network (CNN): Graphical Visualization with Code Explanation," *Analytics Steps,* 06-Sep-2019. [Online]. Available: https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphical-visualization-code-explanation. [Accessed: 24-May-2020].

[83] T. B. Lee, "The Heartbleed Bug, explained," *Vox,* 14-May-2015. [Online]. Available: https://www.vox.com/2014/6/19/18076318/heartbleed. [Accessed: 23-Mar-2020].

[84] V. Alto, "Neural Networks: parameters, hyperparameters and optimization strategies," *Medium,* 05-Jul-2019. [Online]. Available: https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0 [Accessed: 22-May-2020].

[85] "Vanishing gradient problem," *Wikipedia,* 19-Jun-2017. [Online]. Available: https://en.wikipedia.org/wiki/Vanishing_gradient_problem. [Accessed: 21-Feb-2020].

[86] V. Chandola, A. Banerjee and V. Kumar, "Anomaly Detection: A Survey," *Department of Computer Science and Engineering, University of Minnesota,* Jul-2009. [Online]. Available: https://www.researchgate.net/publication/220565847_Anomaly_Detection_A_Survey. [Accessed: 28.03.2020].

[87] "Visualization with Python," *Matplotlib*. [Online]. Available: https://matplotlib.org/. [Accessed: 01-Apr-2020].

[88] V. Vapnik, "The Nature of Statical Learning Theory," *Springer science & business media,* 2013. [Accessed: 19-Feb-2020].

[89] V. V. Khandagale and Y. R. Kalshetty, "Review and Discussion on different techniques of Anomaly Detection Based and Recent Work," *Semantic Scholar,* 2013. [Online]. Available: https://www.semanticscholar.org/paper/

Review-and-Discussion-on-different-techniques-of-Khandagale-Kalshetty/6b5432ae827da420b43ba21b0ce7ff74da5909e7. [Accessed: 20-Feb-2020].

[90] "What's a Brute Force Attack?," Kaspersky. [Online]. Available: https://www.kaspersky.com/resource-center/definitions/brute-force-attack. [Accessed: 09-Mar-2020].

[91] "What is a DDoS Attack?," *Cloud Flare*. [Online]. Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/. [Accessed: 19-May-2020].

[92] "What is a denial of service attack (DoS) ?," *Paloalto NETWORKS*. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos. [Accessed: 09-Mar-2020].

[93] "What is a Distributed Denial of Service Attack (DDoS)?," *Paloalto NETWORKS*. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-ddos-attack. [Accessed: 09-Mar-2020].

[94] "What Is an Intrusion Detection System? Definition, Types, and Tools," *DNSstuff,* 18-Oct-2019. [Online]. Available: https://www.dnsstuff.com/intrusion-detection-system#types-of-intrusion-detection-system. [Accessed: 03-Feb-2020].

[95] "What is PyTorch?," *PyTorch*, 2017. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html. [Accessed: 01-Apr-2020].

[96] Y. Sani, A. Mohammedou, K. Ali, A. Farjamfar, M. Azman, and S. Shamsuddin, "An overview of neural networks use in anomaly Intrusion Detection Systems," *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/5443289/authors#authors. [Accessed: 25-Jan-2020].

[97] Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye, "Intrusion Detection Using Convolutional Neural Networks for Representation Learning," *SpringerLink,* 14-Nov-2017. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-70139-4_87. [Accessed: 22-Feb-2020].

Appendix A

Appendixes

A.1 Neural Network (NN) results

A.1.1 Neural Network - Folds 0, 2, 3 and 4

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.94	0.97	454620
Botnet	0.05	1.00	0.10	394
DDoS	0.95	1.00	0.97	25605
DoS GoldenEye	0.87	1.00	0.93	2059
DoS Hulk	0.94	1.00	0.97	46214
DoS Slowhttptest	0.84	0.99	0.91	1100
DoS Slowloris	0.78	0.99	0.87	1159
FTP Patator	0.86	0.99	0.92	1588
Heartbleed	0.25	1.00	0.40	2
Infiltration	0.00	1.00	0.01	7
PortScan	0.76	1.00	0.86	31786
SSH Patator	0.51	0.92	0.66	1179
Web Attack Brute Force	0.18	0.54	0.27	302
Web Attack Sql Injection	0.01	1.00	0.02	4
Web Attack XSS	0.06	0.71	0.11	130

Table A.1: Neural Network Fold #0. The cross validation result is 93.90.

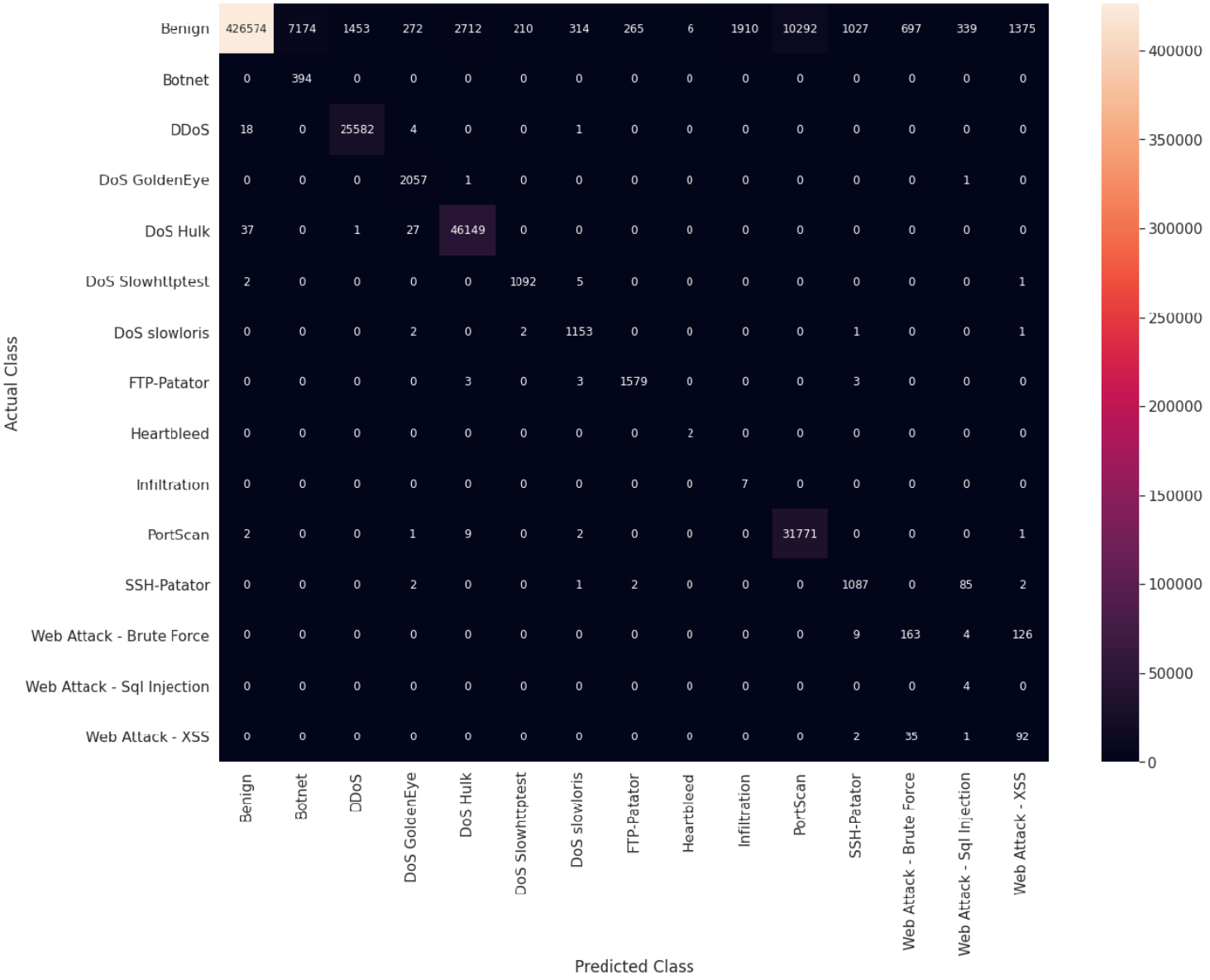

Figure A.1: Neural Network Confusion Metrics heat map Fold #0.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.94	0.97	454619
Botnet	0.05	1.00	0.10	393
DDoS	0.95	1.00	0.97	25606
DoS GoldenEye	0.87	1.00	0.93	2059
DoS Hulk	0.94	1.00	0.97	46215
DoS Slowhttptest	0.82	0.99	0.90	1099
DoS Slowloris	0.79	0.99	0.88	1160
FTP Patator	0.84	1.00	0.91	1587
Heartbleed	0.33	1.00	0.50	2
Infiltration	0.00	1.00	0.01	7
PortScan	0.76	1.00	0.86	31786
SSH Patator	0.51	0.90	0.65	1180
Web Attack Brute Force	0.19	0.54	0.29	301
Web Attack SQL Injection	0.01	0.75	0.01	4
Web Attack XSS	0.06	0.69	0.10	131

Table A.2: Neural Network Fold #2. The cross validation result is 91.97.

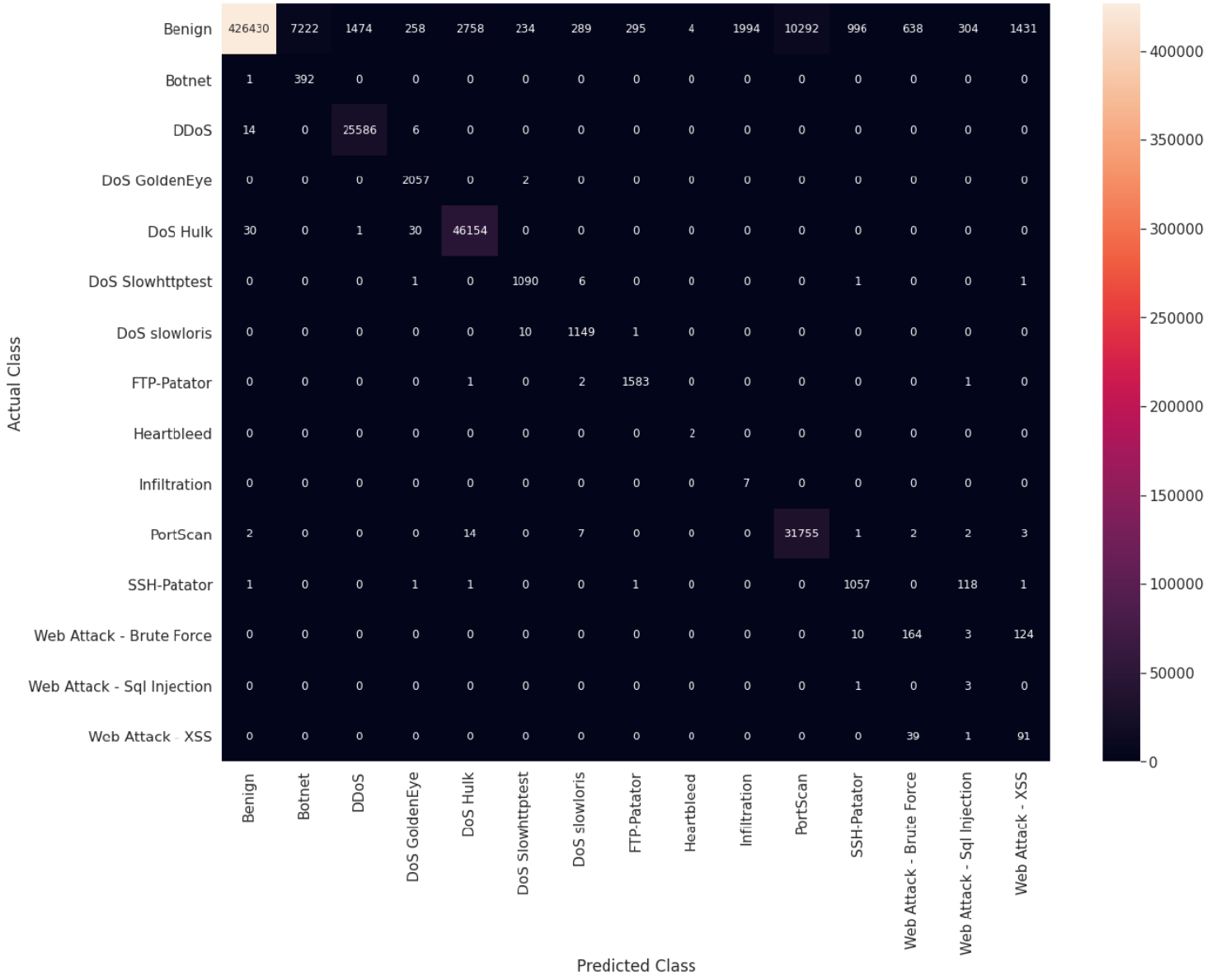

Figure A.2: Neural Network Confusion Metrics heat map Fold #2.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.94	0.97	454619
Botnet	0.05	1.00	0.10	393
DDoS	0.95	1.00	0.97	25605
DoS GoldenEye	0.87	1.00	0.93	2058
DoS Hulk	0.95	1.00	0.97	46215
DoS Slowhttptest	0.83	0.99	0.90	1100
DoS Slowloris	0.79	0.99	0.88	1159
FTP Patator	0.85	1.00	0.92	1588
Heartbleed	0.43	1.00	0.60	3
Infiltration	0.00	1.00	0.01	7
PortScan	0.76	1.00	0.86	31786
SSH Patator	0.51	0.91	0.66	1179
Web Attack Brute Force	0.18	0.59	0.28	301
Web Attack SQL Injection	0.00	0.50	0.01	4
Web Attack XSS	0.05	0.66	0.10	131

Table A.3: Neural Network Fold #3. The cross validation result is 90.46.

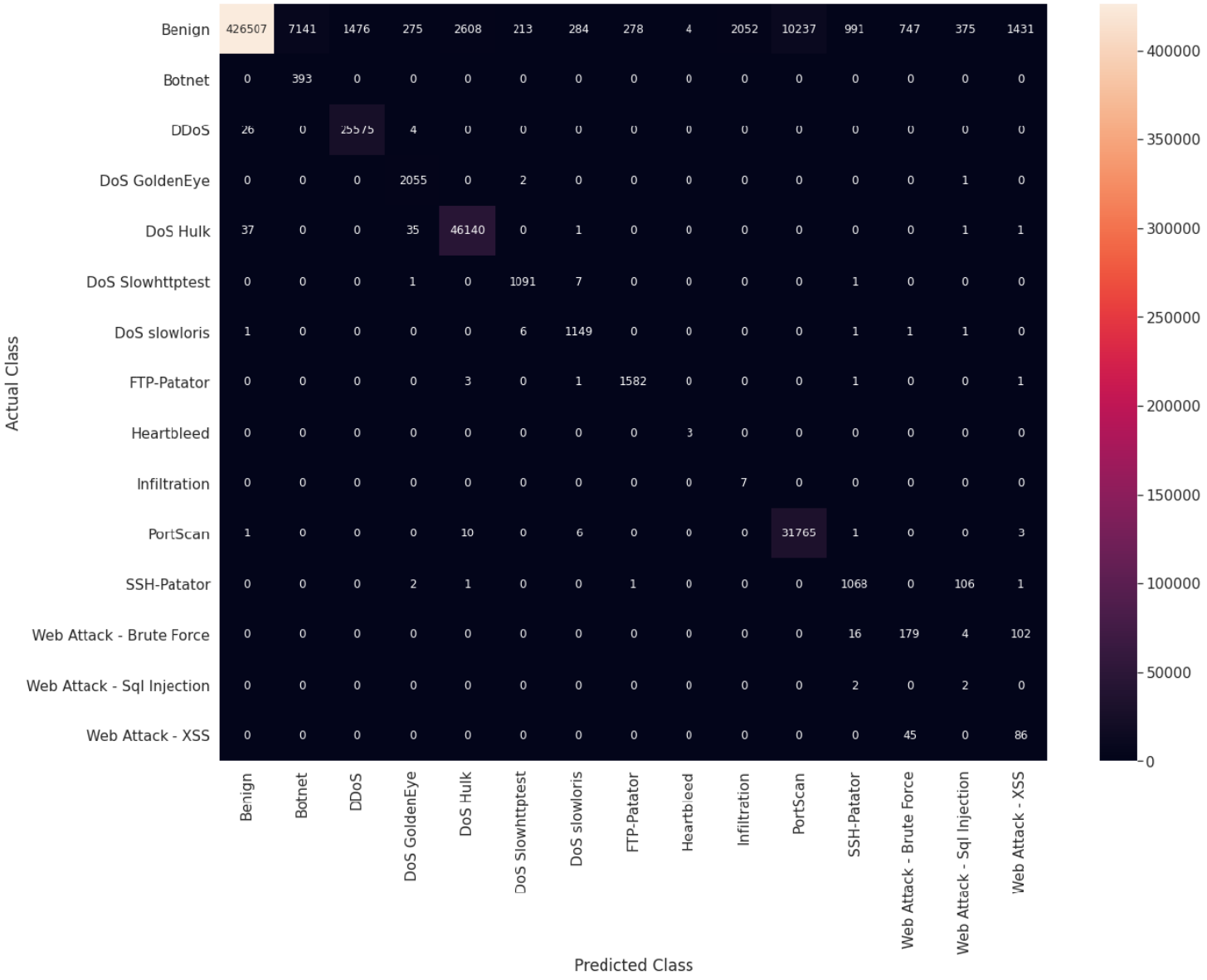

Figure A.3: Neural Network Confusion Metrics heat map Fold #3.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.94	0.97	454619
Botnet	0.05	1.00	0.10	393
DDoS	0.94	1.00	0.97	25605
DoS GoldenEye	0.87	1.00	0.93	2058
DoS Hulk	0.94	1.00	0.97	46215
DoS Slowhttptest	0.81	0.99	0.89	1100
DoS Slowloris	0.79	1.00	0.88	1159
FTP Patator	0.85	1.00	0.92	1588
Heartbleed	0.67	1.00	0.80	3
Infiltration	0.00	1.00	0.01	8
PortScan	0.75	1.00	0.86	31786
SSH Patator	0.50	0.89	0.64	1179
Web Attack Brute Force	0.17	0.51	0.26	301
Web Attack SQL Injection	0.01	0.60	0.01	5
Web Attack XSS	0.06	0.77	0.11	130

Table A.4: Neural Network Fold #4. The cross validation result is 91.30.

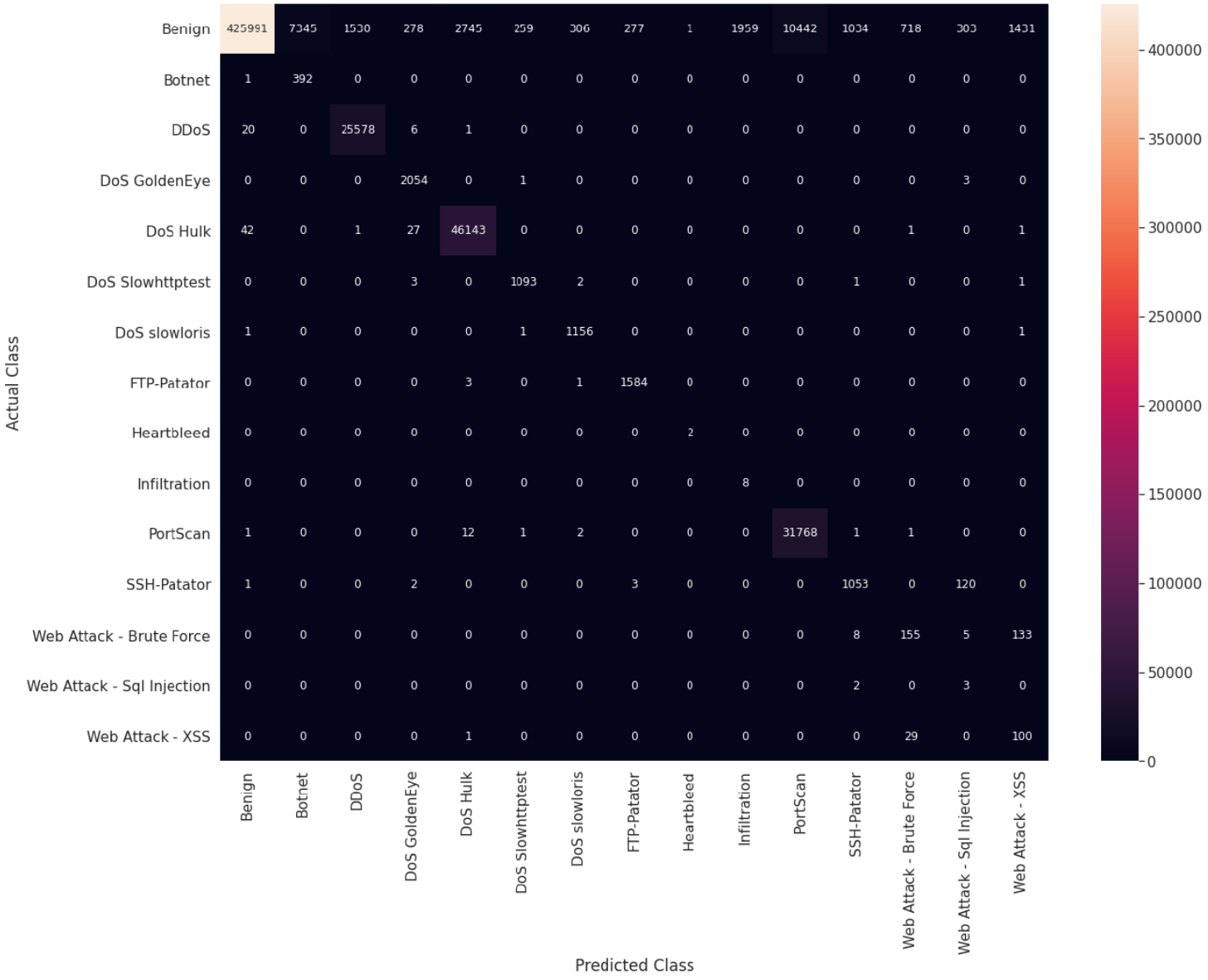

Figure A.4: Neural Network Confusion Metrics heat map Fold #4.

A.2 Convolutional Neural Network (CNN) results

A.2.1 Convolutional Neural Network - Folds 0, 1, 2 and 4

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.93	0.96	454620
Botnet	0.04	1.00	0.08	394
DDoS	0.91	1.00	0.95	25605
DoS GoldenEye	0.80	0.99	0.89	2059
DoS Hulk	0.92	1.00	0.96	46214
DoS Slowhttptest	0.81	0.98	0.89	1100
DoS Slowloris	0.49	0.99	0.66	1159
FTP Patator	0.87	1.00	0.93	1588
Heartbleed	0.11	1.00	0.20	2
Infiltration	0.00	1.00	0.01	7
PortScan	0.82	1.00	0.90	31786
SSH Patator	0.36	0.99	0.53	1179
Web Attack Brute Force	0.13	0.49	0.21	302
Web Attack SQL Injection	0.00	1.00	0.00	4
Web Attack XSS	0.05	0.74	0.10	130

Table A.5: Convolutional Neural Network Fold #0. The cross validation result is 94.08%.

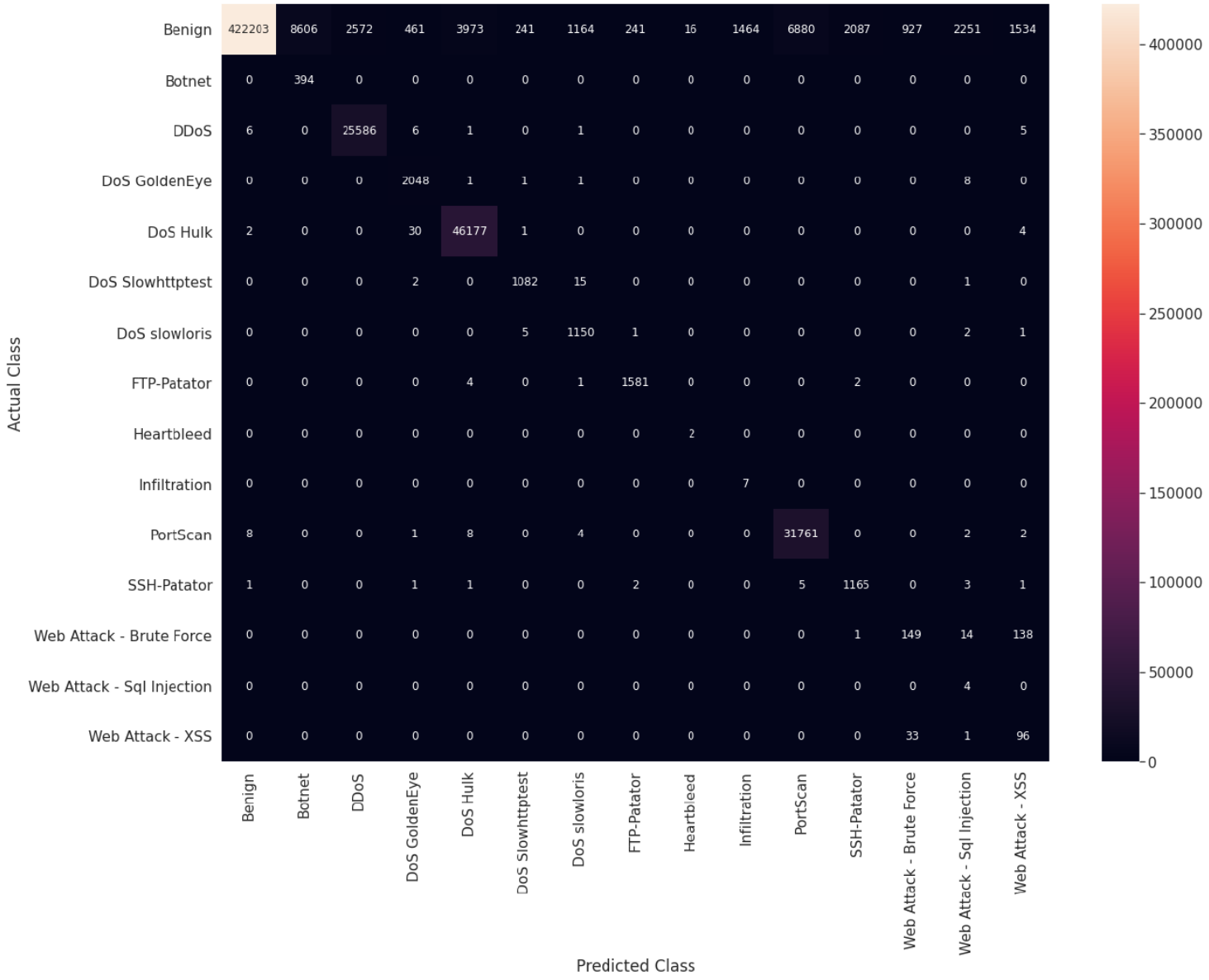

Figure A.5: Convolutional Neural Network Confusion Metrics heat map Fold #0.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.93	0.96	454620
Botnet	0.04	1.00	0.08	393
DDoS	0.91	1.00	0.95	25606
DoS GoldenEye	0.79	1.00	0.88	2059
DoS Hulk	0.92	1.00	0.96	46214
DoS Slowhttptest	0.80	0.97	0.88	1100
DoS Slowloris	0.51	0.99	0.67	1159
FTP Patator	0.87	1.00	0.93	1587
Heartbleed	0.14	1.00	0.25	2
Infiltration	0.00	1.00	0.01	7
PortScan	0.82	1.00	0.90	31786
SSH Patator	0.36	0.99	0.53	1180
Web Attack Brute Force	0.13	0.52	0.21	302
Web Attack SQL Injection	0.00	1.00	0.00	4
Web Attack XSS	0.06	0.78	0.11	130

Table A.6: Convolutional Neural Network Fold #1. The cross validation result is 94.46%.

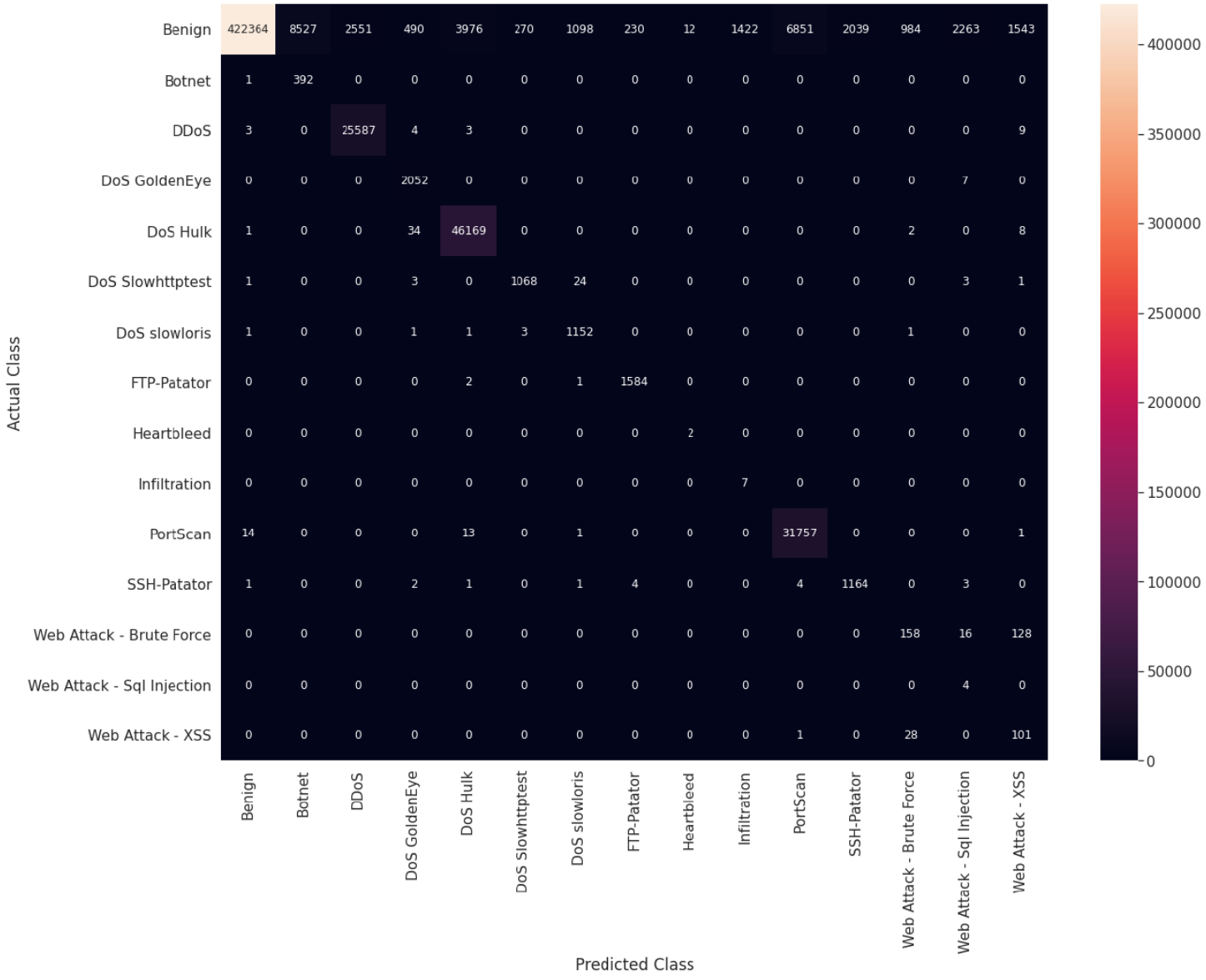

Figure A.6: Convolutional Neural Network Confusion Metrics heat map Fold #1.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.93	0.96	454619
Botnet	0.04	0.99	0.08	393
DDoS	0.91	1.00	0.95	25605
DoS GoldenEye	0.80	1.00	0.89	2058
DoS Hulk	0.92	1.00	0.96	46215
DoS Slowhttptest	0.79	0.98	0.88	1100
DoS Slowloris	0.51	0.99	0.67	1159
FTP Patator	0.86	1.00	0.93	1588
Heartbleed	0.12	1.00	0.22	3
Infiltration	0.00	1.00	0.01	7
PortScan	0.83	1.00	0.90	31786
SSH Patator	0.36	0.99	0.53	1179
Web Attack Brute Force	0.13	0.48	0.20	301
Web Attack SQL Injection	0.00	1.00	0.00	4
Web Attack XSS	0.06	0.82	0.11	131

Table A.7: Convolutional Neural Network Fold #3. The cross validation result is 94.53%.

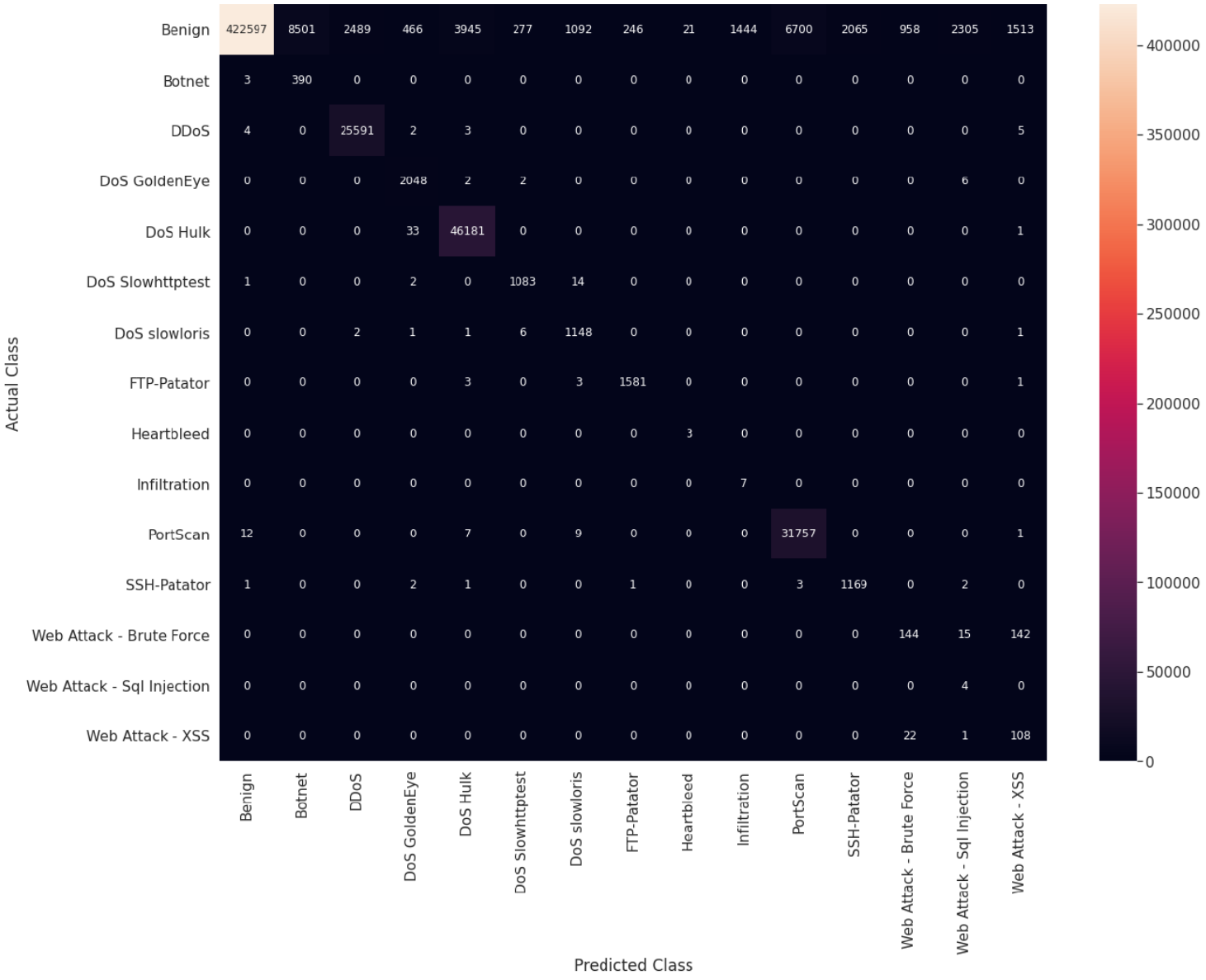

Figure A.7: Convolutional Neural Network Confusion Metrics heat map Fold #3.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.93	0.96	454619
Botnet	0.04	1.00	0.08	393
DDoS	0.91	1.00	0.95	25605
DoS GoldenEye	0.80	0.99	0.88	2058
DoS Hulk	0.92	1.00	0.96	46215
DoS Slowhttptest	0.77	0.99	0.86	1100
DoS Slowloris	0.51	0.99	0.67	1159
FTP Patator	0.86	1.00	0.92	1588
Heartbleed	0.14	1.00	0.25	2
Infiltration	0.01	1.00	0.01	8
PortScan	0.82	1.00	0.90	31786
SSH Patator	0.36	0.99	0.53	1179
Web Attack Brute Force	0.13	0.48	0.20	301
Web Attack SQL Injection	0.00	1.00	0.00	5
Web Attack XSS	0.05	0.75	0.10	130

Table A.8: Convolutional Neural Network Fold #4. The cross validation result is 94.07%.

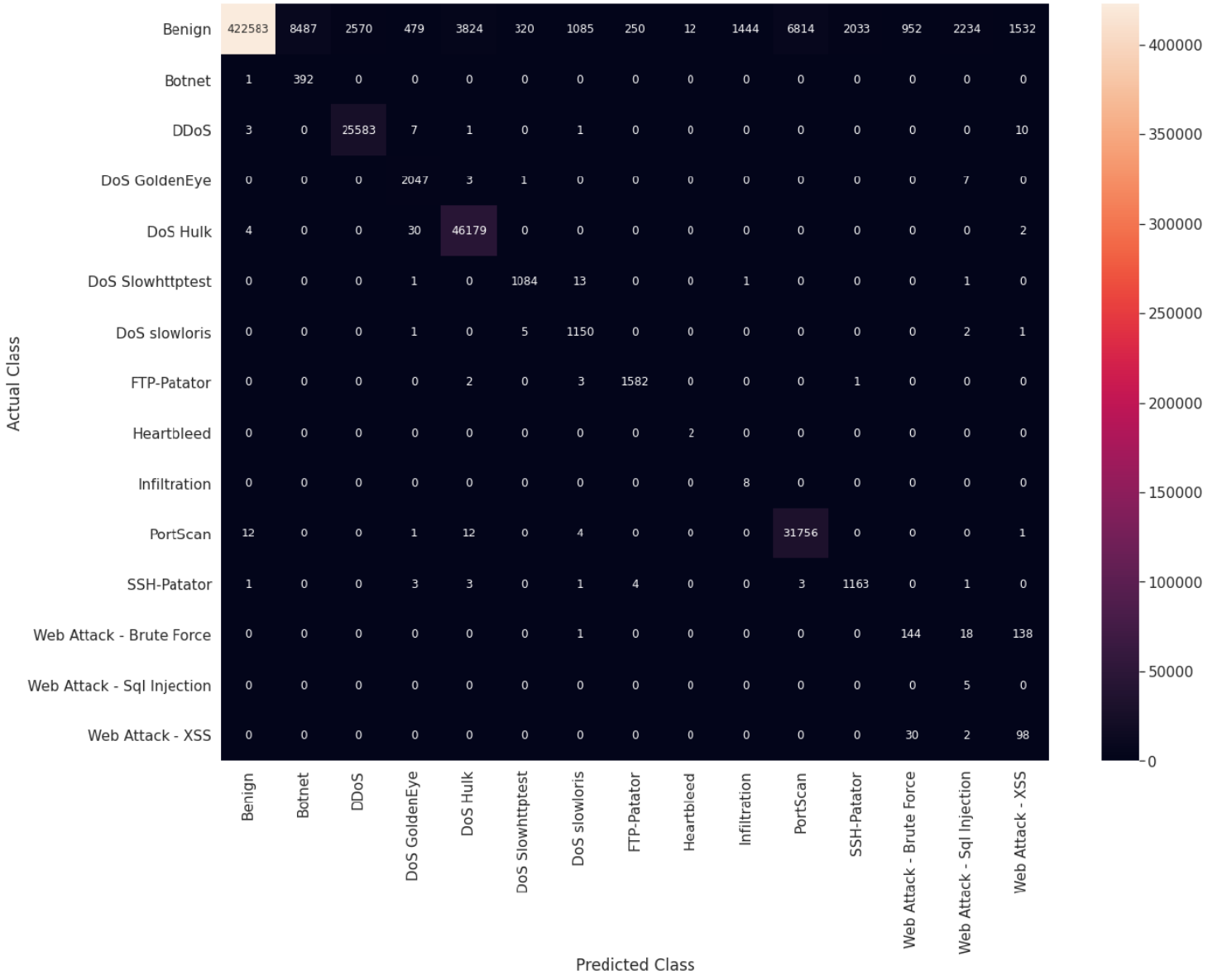

Figure A.8: Convolutional Neural Network Confusion Metrics heat map Fold #4.

A.3 Random Forest (RF) results

A.3.1 Random Forest - Folds 0, 2, 3 and 4

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.99	1.00	454620
Botnet	0.14	1.00	0.25	394
DDoS	1.00	1.00	1.00	25605
DoS GoldenEye	0.98	1.00	0.99	2059
DoS Hulk	0.98	1.00	0.99	46214
DoS Slowhttptest	0.98	0.99	0.99	1100
DoS Slowloris	0.99	1.00	0.99	1159
FTP Patator	1.00	1.00	1.00	1588
Heartbleed	1.00	1.00	0.67	2
Infiltration	0.50	0.71	0.59	7
PortScan	0.99	1.00	1.00	31786
SSH Patator	1.00	1.00	1.00	1179
Web Attack Brute Force	0.73	0.71	0.72	302
Web Attack SQL Injection	0.11	0.50	0.18	4
Web Attack XSS	0.42	0.41	0.41	130

Table A.9: Random Forest Fold #0. The cross validation result is 88.68.

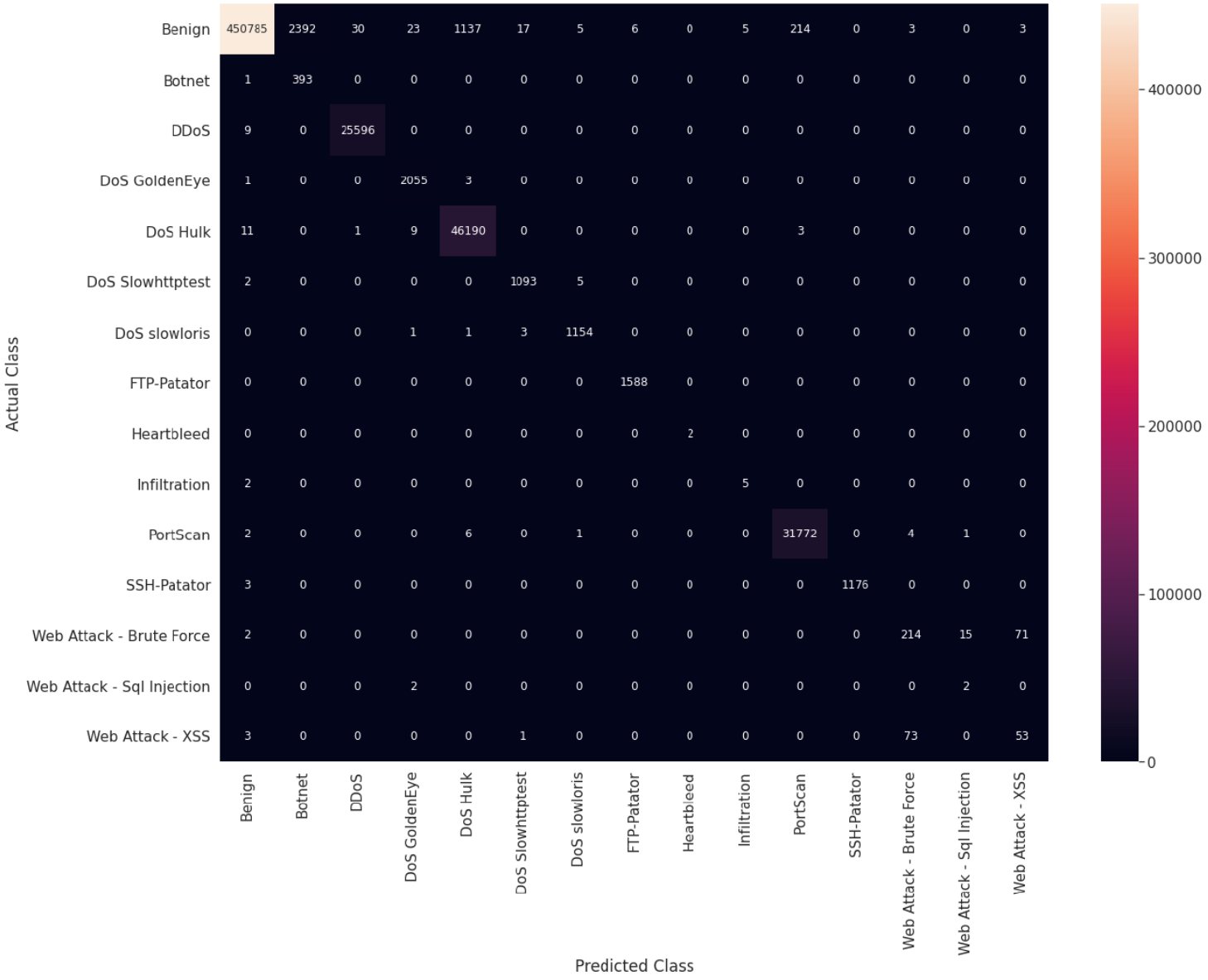

Figure A.9: Random Forest Confusion Metrics heat map Fold #0.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.99	1.00	454620
Botnet	0.12	0.99	0.22	393
DDoS	1.00	1.00	1.00	25606
DoS GoldenEye	0.99	1.00	0.99	2059
DoS Hulk	0.97	1.00	0.99	46214
DoS Slowhttptest	0.98	1.00	0.99	1100
DoS Slowloris	1.00	1.00	1.00	1159
FTP Patator	1.00	1.00	1.00	1587
Heartbleed	0.50	1.00	0.67	2
Infiltration	1.00	0.57	0.73	7
PortScan	0.99	1.00	1.00	31786
SSH Patator	1.00	1.00	1.00	1180
Web Attack Brute Force	0.72	0.73	0.72	302
Web Attack SQL Injection	0.33	0.25	0.29	4
Web Attack XSS	0.36	0.39	0.38	130

Table A.10: Random Forest Fold #1. The cross validation result is 86.12.

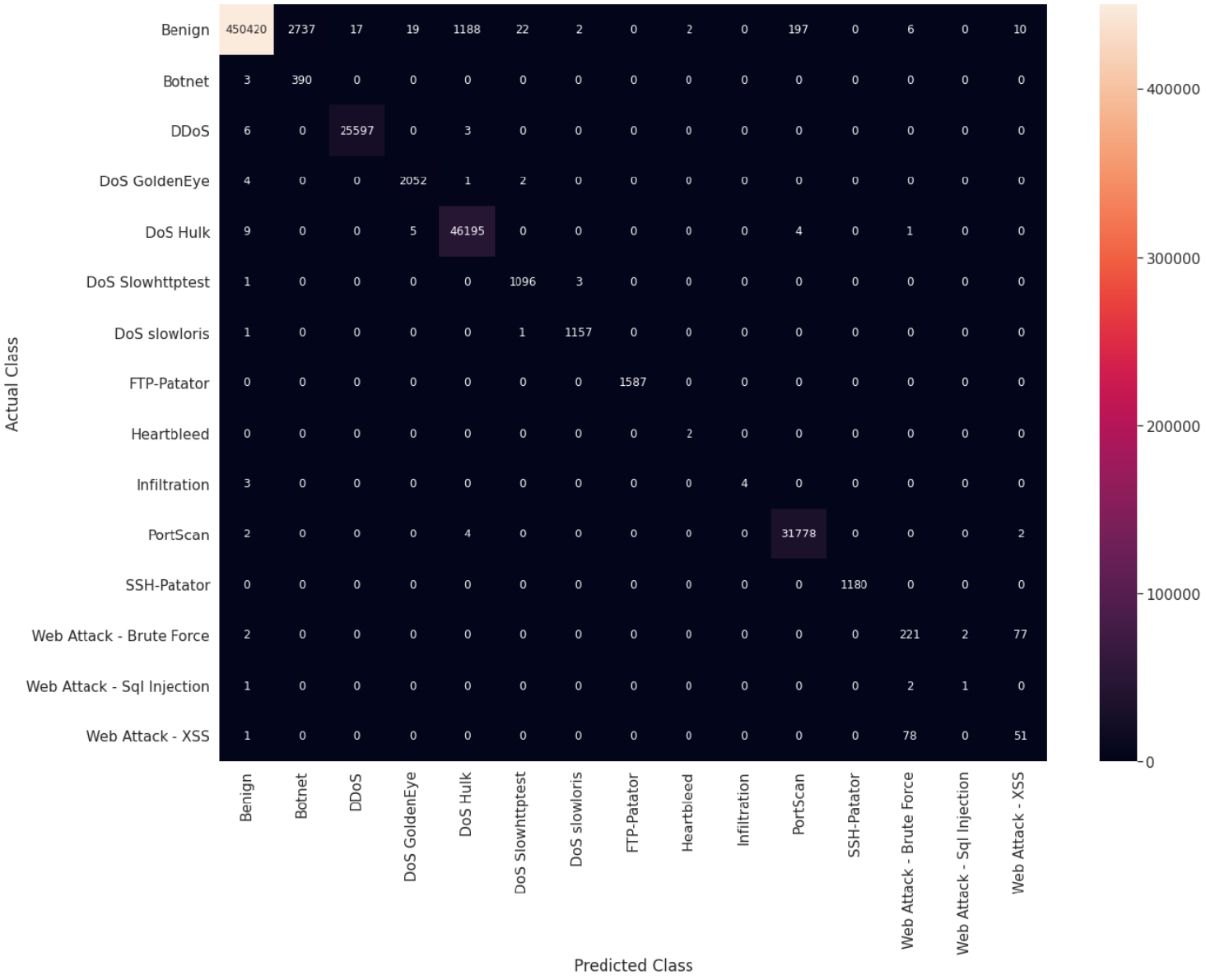

Figure A.10: Random Forest Confusion Metrics heat map Fold #1.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.99	1.00	454619
Botnet	0.16	0.98	0.27	393
DDoS	1.00	1.00	1.00	25605
DoS GoldenEye	0.99	1.00	0.99	2058
DoS Hulk	0.97	1.00	0.99	46215
DoS Slowhttptest	0.98	0.99	0.99	1100
DoS Slowloris	1.00	0.99	0.99	1159
FTP Patator	1.00	1.00	1.00	1588
Heartbleed	1.00	0.67	0.80	3
Infiltration	1.00	0.57	0.73	7
PortScan	0.99	1.00	1.00	31786
SSH Patator	1.00	1.00	1.00	1179
Web Attack Brute Force	0.73	0.74	0.74	301
Web Attack SQL Injection	0.67	0.50	0.57	4
Web Attack XSS	0.32	0.43	0.37	131

Table A.11: Random Forest Fold #3. The cross validation result is 85.76.

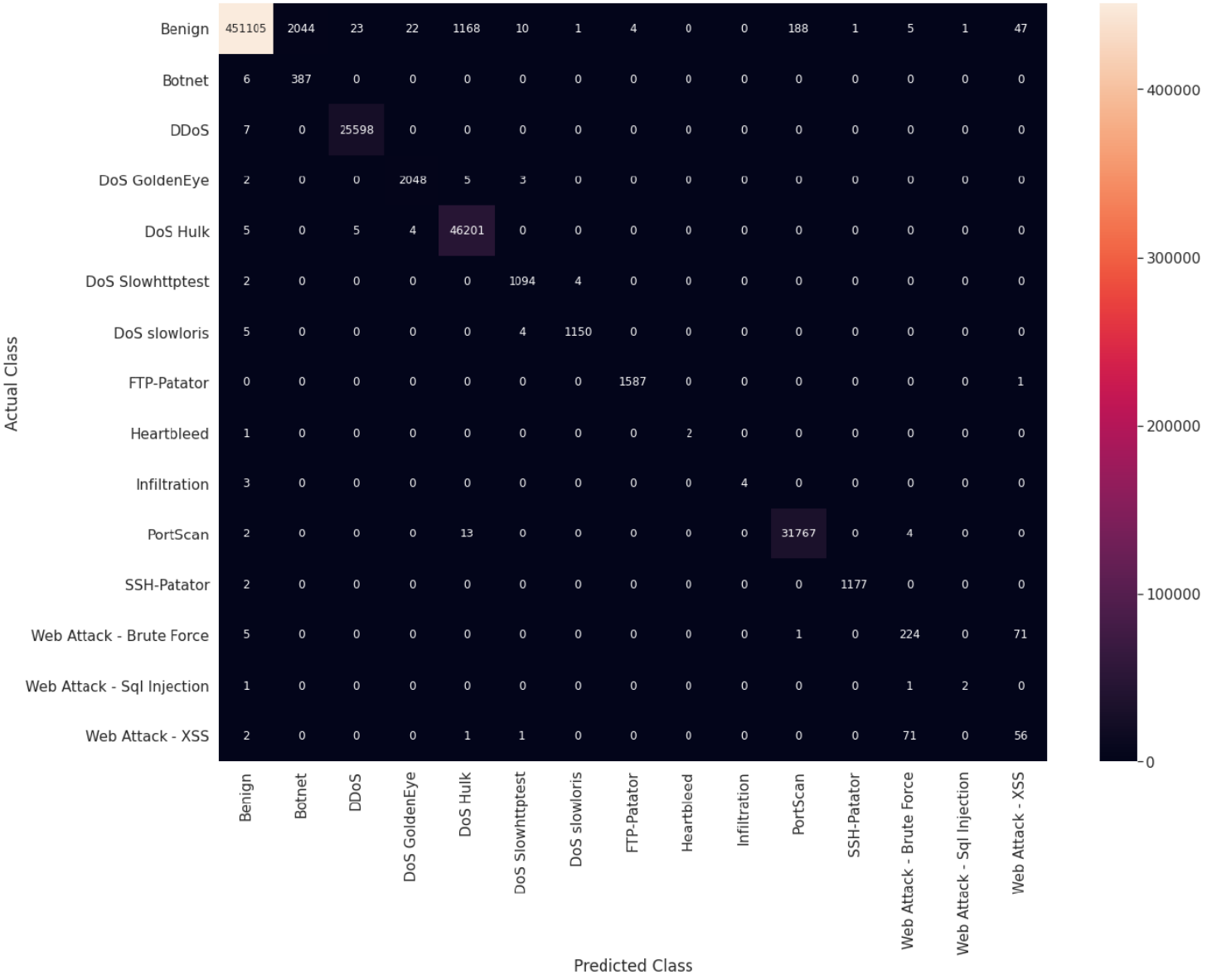

Figure A.11: Random Forest Confusion Metrics Heat map Fold #3.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.99	1.00	454619
Botnet	0.12	1.00	0.22	393
DDoS	1.00	1.00	1.00	25605
DoS GoldenEye	0.99	1.00	0.99	2058
DoS Hulk	0.97	1.00	0.99	46215
DoS Slowhttptest	0.99	0.99	0.99	1100
DoS Slowloris	0.99	0.99	0.99	1159
FTP Patator	1.00	1.00	1.00	1588
Heartbleed	0.67	1.00	0.80	2
Infiltration	1.00	0.62	0.77	8
PortScan	0.99	1.00	1.00	31786
SSH Patator	1.00	1.00	1.00	1179
Web Attack Brute Force	0.73	0.75	0.74	301
Web Attack SQL Injection	0.25	0.60	0.35	5
Web Attack XSS	0.29	0.32	0.31	130

Table A.12: Random Forest Fold #4. The cross validation result is 88.41.

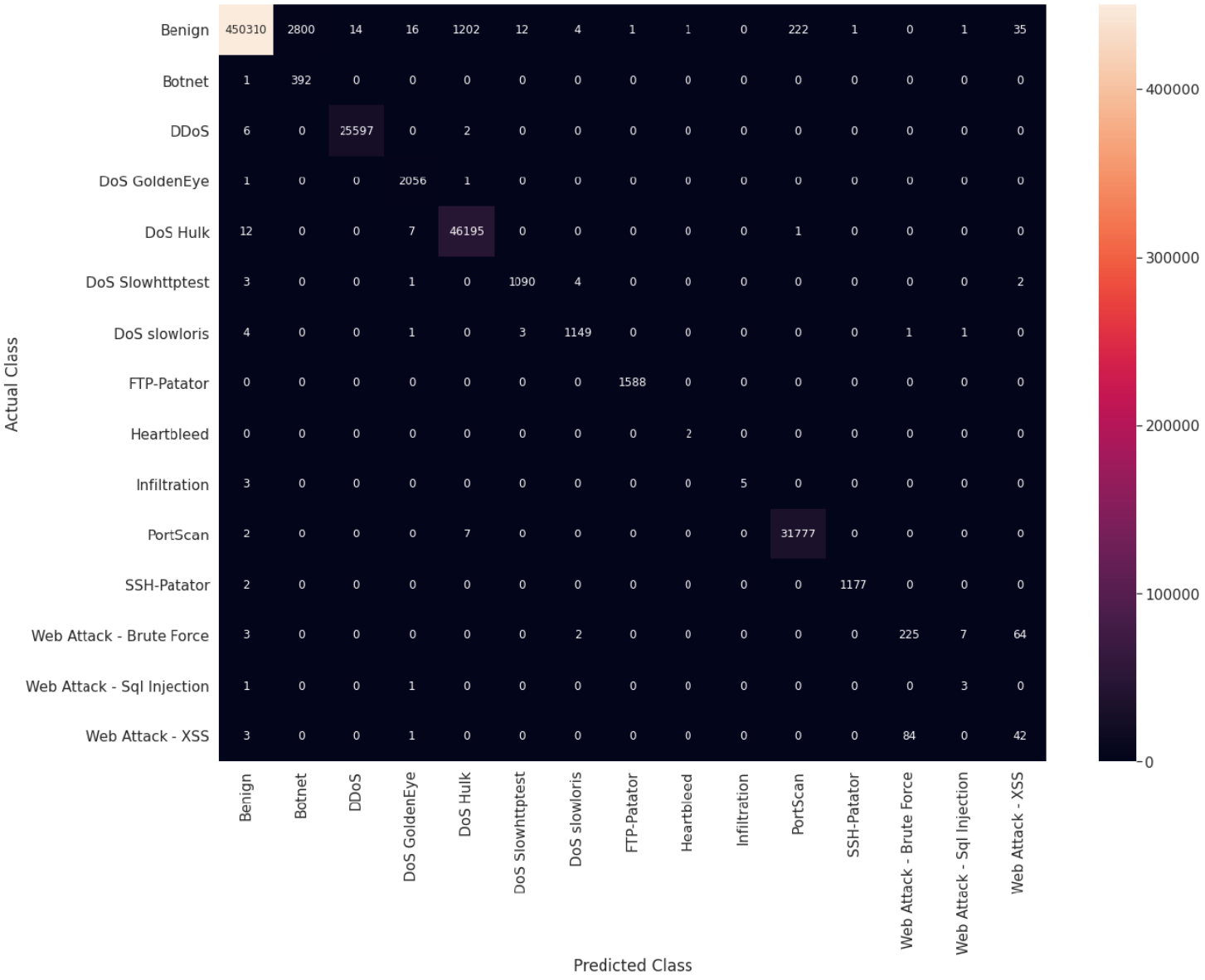

Figure A.12: Random Forest Confusion Metrics heat map Fold #4.

A.4 Long Short-Term Memory (LSTM) results

A.4.1 Long Short-Term Memory - Folds 0, 2, 3 and 4

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.95	0.97	454620
Botnet	0.07	1.00	0.14	394
DDoS	0.94	1.00	0.97	25605
DoS GoldenEye	0.89	1.00	0.94	2059
DoS Hulk	0.94	1.00	0.97	46214
Dos Slowhttptest	0.72	0.99	0.83	1100
DoS Slowloris	0.79	0.99	0.88	1159
FTP Patator	0.85	1.00	0.92	1588
Heartbleed	0.67	1.00	0.80	2
Infiltration	0.00	0.43	0.00	7
PortScan	0.82	1.00	0.90	31786
SSH Patator	0.46	0.92	0.61	1179
Web Attack Brute Force	0.15	0.49	0.23	302
Web Attack SQL Injection	0.01	0.25	0.01	4
Web Attack XSS	0.06	0.72	0.11	130

Table A.13: Long Short-Term Memory Fold#0. The cross validation result is 84.78%.

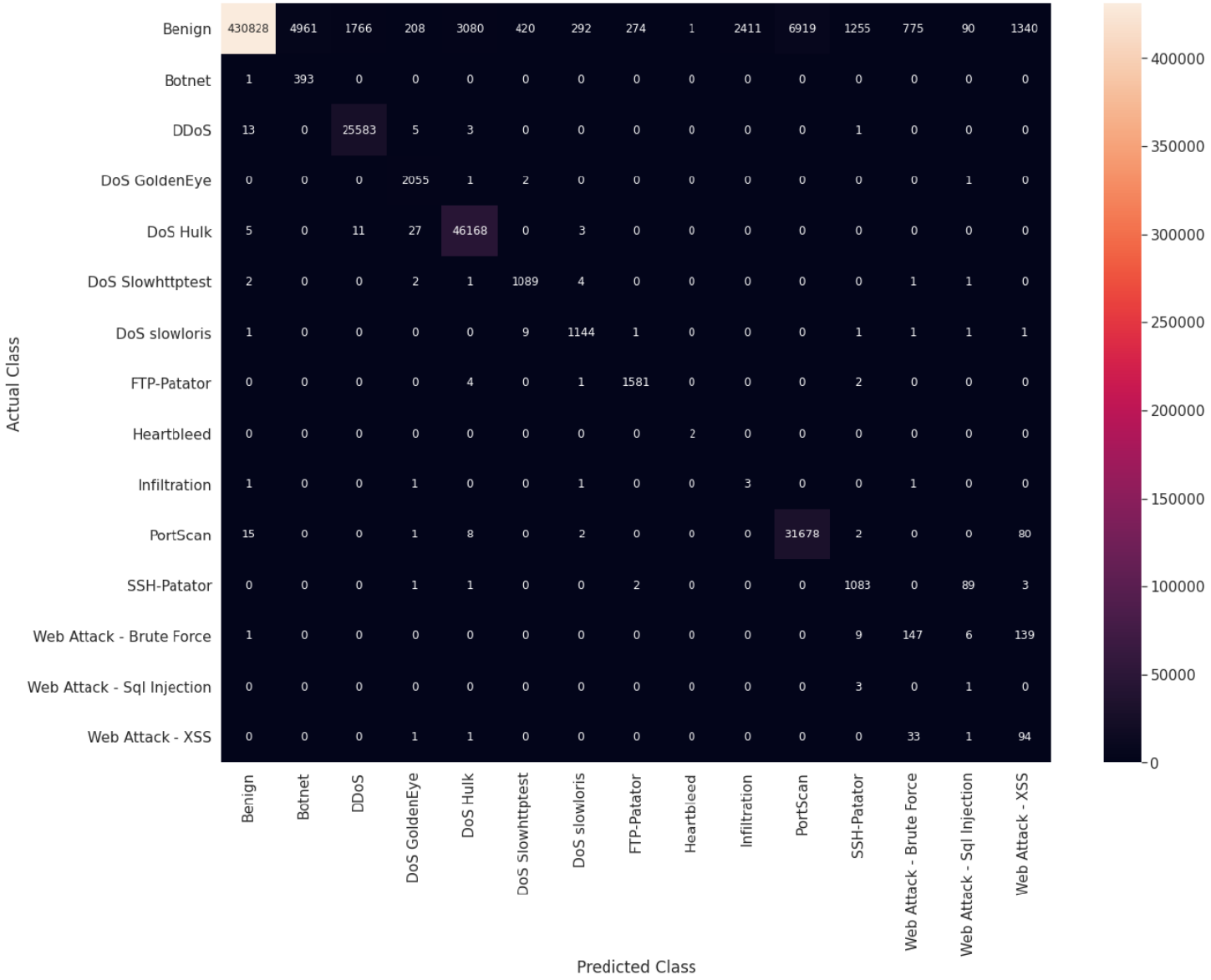

Figure A.13: Long Short-Term Memory Confusion Metrics heat map Fold #0.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.95	0.97	454619
Botnet	0.08	0.99	0.15	393
DDoS	0.93	1.00	0.96	25606
DoS GoldenEye	0.91	0.99	0.95	2059
DoS Hulk	0.94	0.99	0.97	46215
Dos Slowhttptest	0.77	0.99	0.87	1099
DoS Slowloris	0.81	0.99	0.89	1160
FTP Patator	0.79	1.00	0.88	1587
Heartbleed	0.50	1.00	0.67	2
Infiltration	0.00	0.57	0.00	7
PortScan	0.83	1.00	0.90	31786
SSH Patator	0.52	0.99	0.68	1180
Web Attack Brute Force	0.20	0.40	0.27	301
Web Attack SQL Injection	0.00	0.25	0.00	4
Web Attack XSS	0.08	0.84	0.15	131

Table A.14: Long Short-Term Memory Fold #2. The cross validation result is 86.42%.

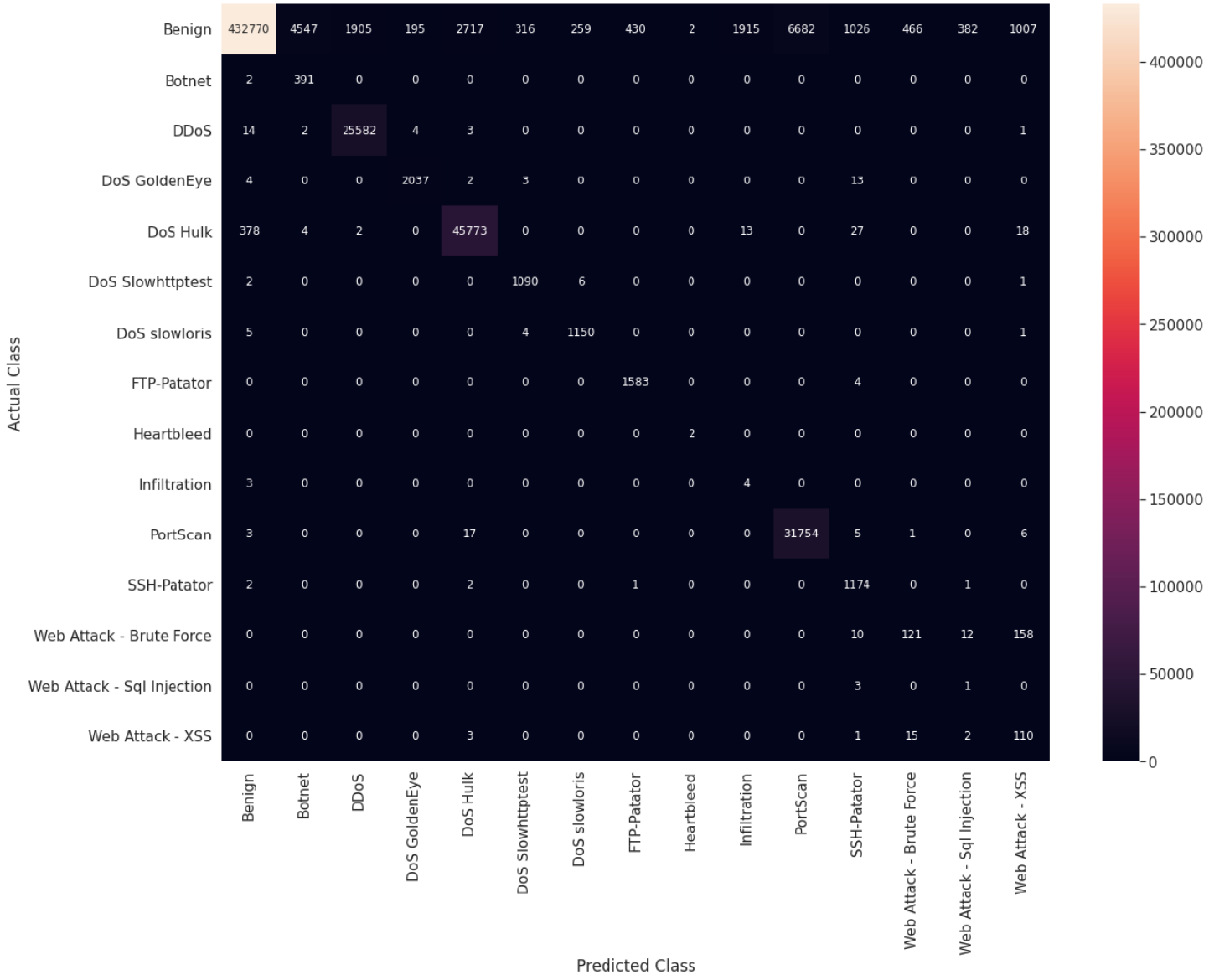

Figure A.14: Long Short-Term Memory Confusion Metrics heat map Fold #2.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.96	0.98	454619
Botnet	0.11	0.99	0.19	393
DDoS	0.96	1.00	0.98	25605
DoS GoldenEye	0.91	1.00	0.95	2058
DoS Hulk	0.95	1.00	0.97	46215
Dos Slowhttptest	0.79	0.99	0.88	1100
DoS Slowloris	0.86	0.99	0.92	1159
FTP Patator	0.89	1.00	0.94	1588
Heartbleed	0.67	0.67	0.67	3
Infiltration	0.04	0.71	0.08	7
PortScan	0.83	1.00	0.91	31786
SSH Patator	0.67	0.95	0.78	1179
Web Attack Brute Force	0.17	0.39	0.23	301
Web Attack SQL Injection	0.01	0.50	0.03	4
Web Attack XSS	0.08	0.88	0.14	131

Table A.15: Long Short-Term Memory Fold #3. The cross validation result is 86.85%.

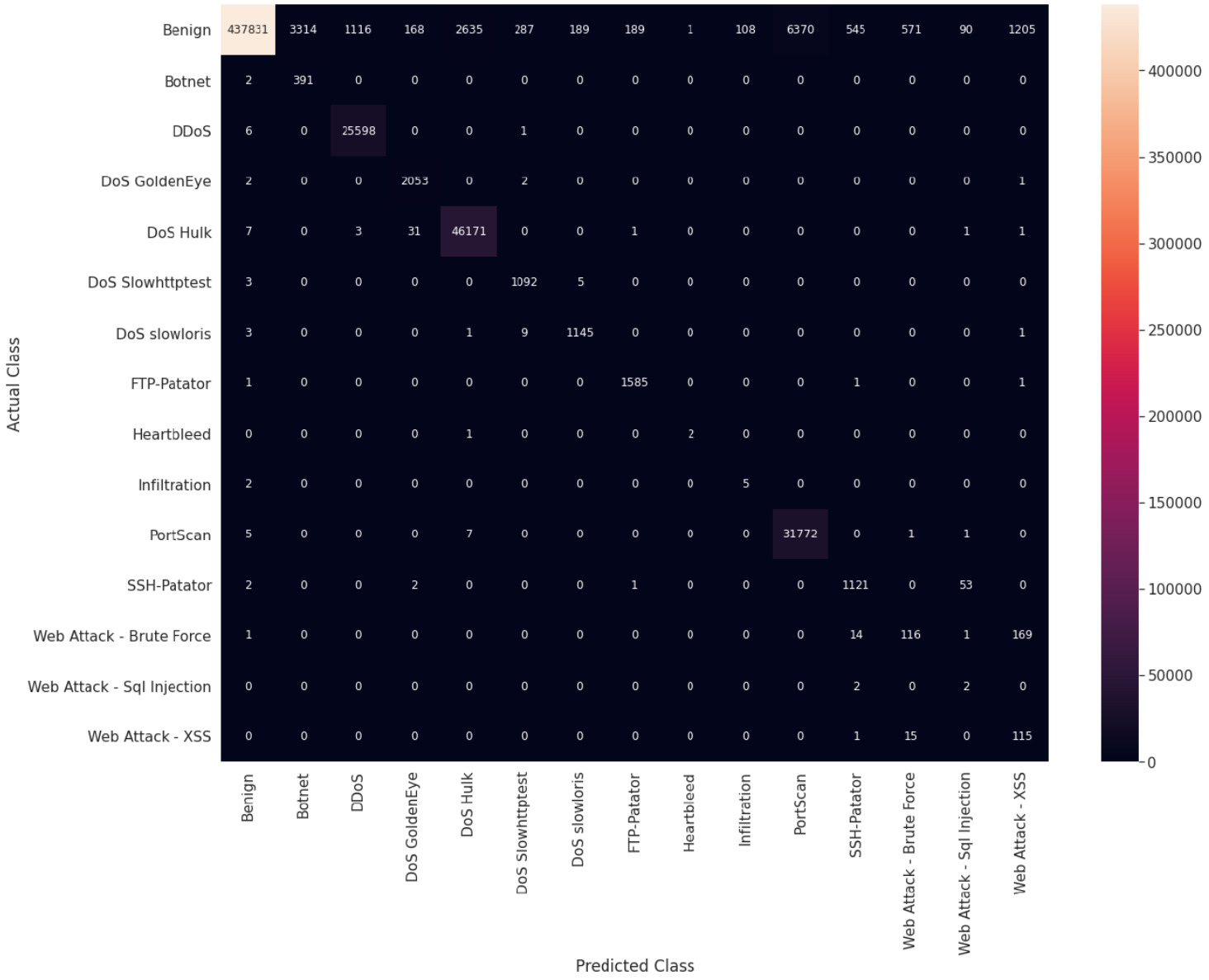

Figure A.15: Long Short-Term Memory Confusion Metrics heat map Fold #3.

Attack	Precision	Recall	F1-Score	Tested Records
Benign	1.00	0.95	0.98	454619
Botnet	0.09	0.99	0.16	393
DDoS	0.95	1.00	0.97	25605
DoS GoldenEye	0.83	1.00	0.90	2058
DoS Hulk	0.94	1.00	0.97	46215
Dos Slowhttptest	0.72	1.00	0.84	1100
DoS Slowloris	0.89	0.99	0.94	1159
FTP Patator	0.53	1.00	0.69	1588
Heartbleed	0.18	1.00	0.31	2
Infiltration	0.00	0.75	0.01	8
PortScan	0.83	1.00	0.90	31786
SSH Patator	0.62	0.80	0.70	1179
Web Attack Brute Force	0.14	0.46	0.21	301
Web Attack SQL Injection	0.01	0.80	0.01	5
Web Attack XSS	0.07	0.75	0.13	131

Table A.16: Long Short-Term Memory Fold #4. The cross validation result is 89.88%.

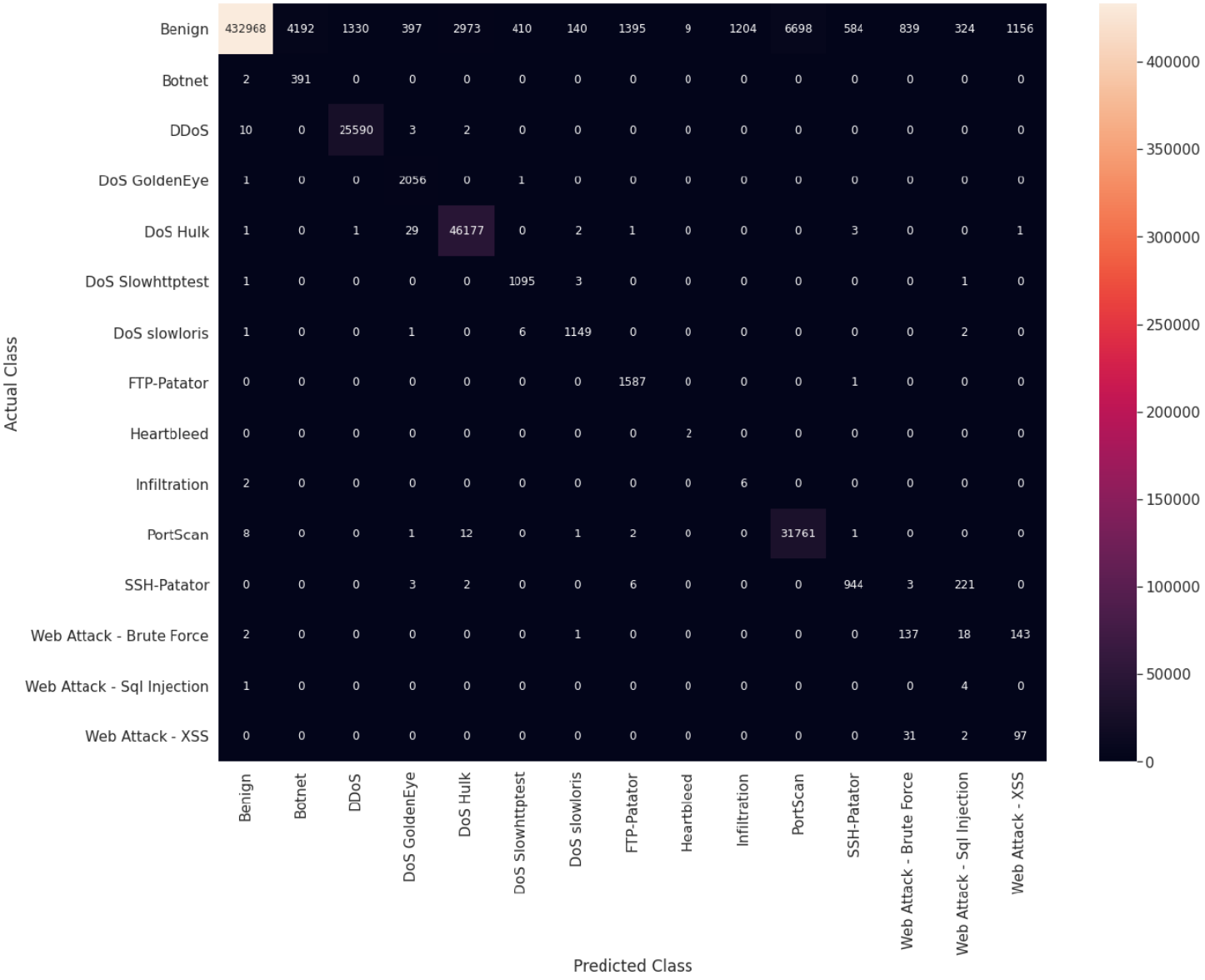

Figure A.16: Long Short-Term Memory Confusion Metrics heat map Fold #4.

A.5 Resources

- Neural Networks (NN), Convolutional Neural Networks (CNN) and Random Forest (RF) approaches are based on:
https://github.com/Jumabek/net_intrusion_detection

- Our models can be found here:
https://github.com/AmirSA92/ML-IDS-Thesis