Msci Computer Science

Anomaly based Intrusion Detection System using Deep Neural Network

by

Georgios Anyfantis

Supervisor: Dr. Asma Adnane

Department of Computer Science
Loughborough University

April 2022

Contents

1 Abstract

Intrusion Detection Systems (IDS) are crucial in detecting possible attacks on a network and informing the administrator about the attack and the attack type. IDS are distinguished into Host-based IDS (HIDS), that check for attacks inside a computer, and Network-based IDS (NIDS), that check for attacks in a network. Additionally, IDS can be Anomaly-based IDS, which utilizes different Machine Learning and Deep Learning techniques to learn attack patterns and Rule-based IDS which utilizes rules that are derived from historical data to detect attacks. Anomaly-based IDS can detect zero-day attacks. As such, there is the need to create a Deep Neural Network (DNN) that can learn to detect and categorize the attacks in order to be used in Anomaly-based IDS. The aim of this paper is to create a DNN that can detect and categorize the attacks that are found in the CIC-IDS-2017 dataset while being as accurate and lightweight as possible. The DNN architectures created will be evaluated from different metrics to gain a better understanding of which architecture performs the best. Pre-processing of the dataset to extract the best performance will be researched to help create an IDS that can perform better with less computational overhead. The experimental data derived by the selected architecture shows a model that performs exceptionally well for the number of attack types it has to detect compared to other published works. Yet, published work tends to perform better as different attack categorization is used. Attacks are bundled to their categories and/or more complex architectures are employed, that is not as lightweight as the proposed one.

2 Introduction

2.1 Intrusion Detection Systems

Intrusion Detection Systems (IDS) are an important tool used in modern-day network security as they allow the detection of malicious intrusions in a network. They are usually deployed to monitor traffic, they can detect traffic irregularities which can indicate a malicious attack is taking place.

IDS can be either network-based, deployed in a network to scan it and detect attacks, by scanning all packets. Or being host-based, checking for intrusions through a particular endpoint of the network.

The two types of intrusion detection systems in terms of how they operate in detecting intrusions are signature-based and anomaly-based.

Signature-based IDS collect real-time data and compares it to historical data associated with a certain type of attack. If the real-time data show a correlation with the historical data of an attack, the system signals that an intrusion has been detected. It is worth noting that this method is prone to not detecting zero-day attacks, that are novel and not seen before due to the use of rules and historical data, which leads to false negatives. Yet, due to its structure, false positives are significantly less than on an anomaly-based IDS.

Anomaly-based systems operate differently, they utilize machine learning in order to detect existing as well as new types of attacks. The use of machine learning allows the networks to adjust on-the-fly when it comes to new attacks, thus allowing them to create new models and recognize new attacks. Yet, due to their nature, they may create false alarms when they see a log pattern not previously encountered as they may falsely classify it, which is called false positives.

It is worth noting the other two distinctions between IDS based on their location. These are the Network and Host-Based IDS.

Network-based IDS (NIDS) is an IDS that is based on a device connected to a network, like a switch, and its job is to detect whether the network is under any kind of attack and send an alert. It usually works by collecting data packets that are passing through the device and checking whether they are malicious or not. This can be done by either using a Signature-based IDS, also known as Rule-based or through the use of an Anomaly-based IDS.

Host-based IDS (HIDS) is an IDS system that runs on hosts. It is used to monitor the internal security of the host, this may be a server connected to the network, about abnormal behaviour, like abnormal system calls and use of specific ports that have previously not been used. The detection occurs by either using an Anomaly-based or a Signature-based system.

It is worth noting that both NIDS and HIDS have to be implemented in a system in order to succeed in advanced system protection as each type covers a specific area of the system and it is not advised to use either one of them on their own as they complement the blind spots of each other. NIDS cannot detect the existence of abnormal host behaviour which may be due to a compromised host and HIDS cannot detect a Denial-of-Service (DoS) attack.

2.2 Aims and Objectives

The aim of this project is to create a Deep Neural Network (DNN) that can detect potential attacks. The DNN must have a high accuracy level, preferably above 90%, and be able to detect and categorize multiple attack types that may occur in a network correctly. Additionally, the DNN should be as lightweight as possible to reduce the detection times, as IDS are time-sensitive.

The DNN will be evaluated in a strictly experimental environment. This means that the DNN will not be implemented in an IDS platform but rather will be run in a custom environment to evaluate its performance. In this environment, we will be able to observe its performance and tune the model in order to work as planned as well as if its performance is optimal.

The DNN will be trained using the CIC-IDS2017 data set [1]. The main reason behind choosing this data set is that it has an official data split and has been used extensively in other studies as a testing and training data set.

2.3 Problem Analysis

Nowadays one of the main problems in cybersecurity is the constant evolution of attacks on a network. Previous types of IDS and techniques used, like Signature Based IDS, rely on written rules and historical data of attacks in order to evaluate if an attack is occurring.

This is an issue as many new attack types go unnoticed resulting in an increased number of network breaches and data leaks. The use of an anomaly-based IDS changes this as the IDS is looking for possible anomalies. Furthermore, when the IDS is paired with a DNN the system has the ability to learn and improve from scanning the network traffic, thus improving with time.

As such, we have chosen to focus on creating a model that can detect possible network attacks and classify them. The ideal candidate of an IDS to run our model would be a network based anomaly-based IDS as our model will be looking to pick up on possible network anomalies.

3 Literature Review

There has been extensive research in the last few years in the area of anomaly-based IDS. Different methods have been used in order to construct these systems. Many rely on Deep Learning or different machine learning technologies while others rely on more traditional classifier methods.

3.1 Data set Analysis and Selection

When configuring and testing the DNNs for the IDSs the data set used for the testing and training is one of the most important aspects in order to achieve higher accuracy and increased performance by the IDS. Many studies currently rely on the KDD Cup 99 data set [2], which is an outdated data set released in 1999 by DARPA. Since then the structure of the internet has changed extensively as well as the type of attacks and the way they work. Since the initial release of the KDD Cup 99 data set, multiple different data sets have been released. Some attack specific like CIRA-CIC-DoHBrw-2020 [3], which focuses on the implementation of Domain Name Server (DNS) over HTTPS and different methods that allow evaluation of the traffic as traditional Firewalls do not work with DoH. While other data sets are more generic like CIC-IDS-2017 which is used for this project [1].

This data set primarily focuses on helping to develop IDS, specifically NIDS as it is the network traffic that a NIDS would have to expect in a real-life application.

The main reason for choosing this data set is that compared to other data sets. it is newer and it has an official data split allowing for easier use of the data for training and testing purposes as well as the fact that it is newly released meaning that the attack types are up to date.

3.2 Analysing the optimal Depth of a DNN

When building a DNN, it is very important to find its most optimal configuration and most importantly the number of hidden layers it has. Shallower DNNs tend to be less accurate than deeper DNNs although this is more of a rule of thumb rather than a scientific theory, as in some cases deeper DNNs tend to underperform or there is no performance improvement. As seen in this paper [4], the authors focus on finding the number of hidden layers that is more accurate. The network is configured with a learning parameter of 0.1 and it has 1 to 5 layers, depending on configuration, for 300 epochs. In the study is found that the best configuration is the one with 3 layers with an accuracy of 93% which is the highest recorded in the series of experiments run for evaluation purposes. Additionally, other methods were used such as Support Vector Machine (SVM) which proved to be not as accurate as a 3-layer

DNN.

The authors also use the ReLu activation function instead of the more traditional sigmoidal and Tanh activation functions as it solves the vanishing gradient problem, which occurs when we use multiple layers with sigmoid or tanh activation functions, as they result in the gradients at higher layers change minimal due to the changes at the previous layers [5]. This results in the weights not being updated properly.

Algorithm	Accuracy	Precision	Recall	f1-score
DNN-1	0.929	0.998	0.915	0.954
DNN-2	0.929	0.998	0.914	0.954
DNN-3	0.930	0.997	0.915	0.955
DNN-4	0.929	0.999	0.913	0.954
DNN-5	0.927	0.998	0.911	0.953
Ada Boost	0.925	0.995	0.911	0.951
Decision Tree	0.928	0.999	0.912	0.953
K-Nearest Neighbour	0.929	0.998	0.913	0.954
Linear Regression	0.848	0.989	0.821	0.897
Navie Bayes	0.929	0.988	0.923	0.955
Random Forest	0.927	0.999	0.910	0.953
SVM*-Linear	0.811	0.994	0.770	0.868
SVM*-rbf	0.811	0.992	0.772	0.868

*Support Vector Machine

Figure 1: The results of different algorithms evaluated using KDD Cup 1999 data set [4]

3.3 Use of LSTM in IDS creation

A proposed method to improve the performance of the IDS using Neural Networks is the use of an LSTM network in order to create a hierarchical spatial-temporal features-based intrusion detection system (HAST-IDS) [6]. This method requires the use of a Deep Convolutional Neural Network (CNN) in order to learn the spatial features of the network traffic and then it uses Long Short Term Memory (LSTM) to learn the temporal features. This method utilizes two different Neural Network (NN) methods in order to improve detection accuracy. Each neural network focuses on a different aspect of the traffic, CNN focuses on learning the spatial features of the traffic from the two-dimensional traffic image, whereas the LSTM focuses on learning the temporal features by the r-packet vectors. This allows the network to automatically learn and adapt without the need for a specifically designed data set.

This method has a limitation due to the use of CNN. The network flows must be reduced to a fixed number as the flows tend to be variable and the number of data packets and their size can affect the temporal feature learning. Through testing, the authors have decided that the best number of packets is 6 with packet size being 100 bytes. The flow size used in this paper is 600 bytes. It is worth noting that these settings have been chosen in respect of optimising the performance of the HAST-IDS. A benefit of this method is the fact that it is quite accurate and has a fast training time compared to other methods that have used

the same data sets while also having a high accuracy rate combined with a low False Alarm Rate. But the main advantage of this method is the fact that it uses raw data, something not seen in the other methods that require the data to be pre-processed, k-fold-cross-validation has been used for evaluation. Normalization is not mentioned to have been performed in the data before training.

Method	Training time	Testing time
MHCVF[44]	240 min	35 min
PLSSVM[37]	104 min	46 min
Multiple-Level Hybrid Classifier (MLHC)[47]	1,443 min	4 min
HAST-IDS	58 min	1.7 min

Figure 2: Training times of HAST-IDS compared to other published methods using the data set DARPA1998 [6]

Method	DoS (%)		Probe (%)		R2L (%)		U2R (%)		Test Dataset Size	EM_{DF}	EM
	DR	FAR	DR	FAR	DR	FAR	DR	FAR			
PLSSVM[37]	78.69	0.73	86.46	13.87	84.85	0.53	30.7	0.47	311,029	8.49E+01	1.07E+03
Multi-Classifier[38]	97.3	0.4	88.7	0.4	9.6	0.4	29.8	0.1	311,029	1.97E+02	2.49E+03
Random Forest[39]	98.91	3.15	55.12	0.45	66.67	5.18	100	0.13	77,287	2.34E+02	2.63E+03
Bayes Net[40]	94.6	0.2	83.8	0.13	5.2	0.6	30.3	0.3	15,437	3.07E+02	2.96E+03
JRip[40]	97.4	0.3	83.8	0.1	0.1	0.4	12.8	0.1	15,437	3.23E+02	3.11E+03
SVM[41]	76.7	0.09	81.2	0.36	11.2	0.08	21.4	0.08	10,000	3.71E+02	3.42E+03
Naive Bayes[42]	99.69	0.04	99.11	0.45	99.11	8.02	64	0.14	311,029	7.95E+02	1.01E+04
ID3[43]	99.9	0.03	99.7	0.55	93.5	0.98	49.1	0.15	311,029	9.84E+02	1.24E+04
EID3[43]	99.9	0.03	99.8	0.39	99.7	0.22	99.8	0.12	311,029	1.22E+03	1.54E+04
MARK-ELM[29]	99.96	0.02	97.42	0.03	94.94	0.05	62.87	0.01	72,793	4.11E+03	4.60E+04
HAST-IDS	99.10	0.02	83.35	0.01	74.19	0.02	64.25	0.02	1,099,731	5.05E+03	7.03E+04

Figure 3: Testing results of HAST-IDS compared to other published methods and classical classification algorithms using the data set DARPA1998 [6]

3.4 Use of Fuzzy Enhanced Support Vector Decision Function in IDS

A different proposed method for building an efficient and successful IDS is the use of the Fuzzy Enhanced Support Vector Decision Function (FESVDF). It creates an IDS model that utilizes a select number of features that are deemed to be the most efficient in terms of detecting an attack [7]. In this paper, the team uses FESVDF in order to select the most important features needed for the IDS to detect an attack thus resulting in a lightweight IDS, which will focus only on the most important features as a detection mechanism. This method results in an IDS with less overhead and thus it is faster in detecting an attack. The testing is conducted using a Neural Network and a Support Vector Machine classifier as benchmarks in order to evaluate the efficiency of the FESVDF.

The authors are creating multiple IDS that are located in different layers of the network and specialise in detecting specific attack types. The feature selection is implemented in order to select the appropriate types of features for each IDS. The research paper focuses on the introduction of four IDS types based on the TCP/IP model. These are Data Link layer IDS (LIDS), Network layer IDS (NIDS), Transport layer IDS (TIDS), and Application layer IDS (AIDS). This is different compared to other studies that just focus on creating one standalone IDS and not multiple components.

The evaluation of the system worked by using for the Neural Network (NN) and Support Vector Machine (SVM) classifier all the features and then using layer-specific features for each layer IDS. The results indicate that the use of FESVDF for feature extraction has resulted in the IDS performing better than any an IDS created to deal with all parameters. But these features are specific for each layer IDS and as such cannot be used to create different IDS types for different layers without compromising their accuracy.

Classifier Method	IDS Type	No. of Features	CR (%)	Training Time	Testing Time
Support Vector Machine (SVM)	All Layers IDS with all features	41	99.46	3.039	0.052
	AIDS	5	99.62	2.594	0.022
	TIDS	4	99.75	1.586	0.020
	NIDS	4	99.73	2.328	0.020
Neural Network (NN)	All Layers IDS with all features	41	99.65	911.680	0.075
	AIDS	5	99.73	162.734	0.034
	TIDS	4	99.84	139.296	0.031
	NIDS	4	99.77	144.281	0.032

Figure 4: Results table using DARPA KDD-99 [7]

3.5 Use of Genetic Programming for a Signature-based IDS

As mentioned in the introduction section of this paper there are two widely used IDS types, anomaly and signature-based IDS. Many papers focus on one of these types but this paper combines both of them in order to create a new IDS [8]. The authors here have primarily created a signature-based IDS, and rule-based IDS, but the rule generation has been achieved through using Genetic Network Programming (GNP). By using GNP the researchers have been able to create rules that can detect patterns and also detect attacks that have not been seen previously, something usually associated with an anomaly-based IDS.

The idea for the combination of both systems stems from the problem that anomaly-based IDS can detect zero-day attacks but suffer from a high false-positive rate whereas a rule-based system does not suffer from a high false-positive rate but cannot detect zero-day attacks. The proposed system is able to mitigate the disadvantages of both systems while combining their advantages. The GNP is trained using the KDD Cup99 data set and it creates rules. These rules are then used in the IDS in order to detect known and unknown attacks. The new system results in an accuracy of 90.26% which is lower than other implementations that we have seen, its detection rate is 96.82%. As seen in the following figure which was taken from the paper currently discussed [8], the conventional rule-based IDS and an Anomaly Based IDS result in a lower Detection Rate than the newly proposed system. When it comes to Positive False Rate, the largest percentage is seen in the Anomaly Based IDS, followed by the Signature Based IDS and finally by the newly proposed system. A similar trend is seen in the Negative False Rate, although in this case, the Anomaly Based IDS has virtually 0% in this rate. It is worth noting that these data were taken from this research [8] and may be outdated.

It is worth mentioning that the results of this implementation are worse than the results seen in other studies. Thus this method is not optimal for an IDS. This may be due to the fact that the study was conducted 10 years ago, yet it is an interesting implementation although the results indicate that it is a worse method than the ones seen in the previous sections. That is mainly due to the fact that it combines two different IDS types together.

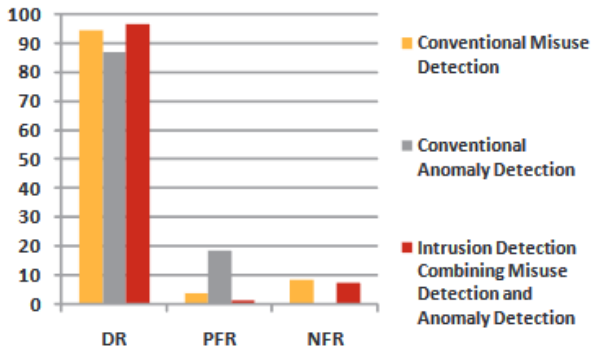

Figure 5: Results graph from [8]

3.6 SHIA Framework

Most of the research papers have been focused on either evaluating algorithms and implementation in either HIDS or NIDS, only *Lightweight ids based on features selection and ids classification scheme* has focused on different types of IDS and that is because the proposed method requires a distributed IDS.

This paper [9] is promising, as it is an in-depth evaluation of multiple machine learning methods and DNNs on what constitutes the best use in creating a hybrid IDS implementing the abilities of a NIDS and a HIDS. This is the new proposed framework, named Scale-Hybrid-Alertnet (SHIA) which identifies malicious events both in the host and network-level and provides alerts to the network administrator. The evaluation is done using the KDD Cup 99 data set and other data sets, including NSL-KDD, UNSW-NB15, Kyoto, WSN-DS, and CIC-IDS-2017.

The main focus of this paper is the creation of a hybrid IDS that is also scalable and distributed. An in-depth evaluation of current classification methods is conducted in order to choose the best one for the method in terms of accuracy and also to showcase the improvements succeeded by using DNN instead of classical classification algorithms.

Additionally, the framework uses big data and relies on distributed deep learning models DNNs to learn possible attacks which allow it to outperform classical machine learning classifiers that have been previously employed in NIDS and HIDS. It is the only hybrid framework seen up until now which has impressive results and thus constitutes an important development when it comes to implementation in systems. It is also worth noting that this framework uses a normal server and does not require specialised hardware to run. The data collected and used by the framework is saved on a NoSQL database.

3.7 Use of Blockchain for a Distributed IDS

In terms of distributed IDS, only *Lightweight ids based on features selection and ids classification scheme* and the SHIA framework [9] have focused on a Distributed IDS. But the Lightweight IDS approach is a collection of individual IDS running in different layers of the TCP protocol without a centralised server. Thus they do not constitute a unified system, whereas the proposed DIDS system [10] focuses on using Blockchain as a way to use a centralised platform where the attack can be reported no matter in which part of the network they occurred, the proposed system is also scalable.

Blockchain is used in this proposed system as a way to store the incidents reported by each IDS. Blockchain was chosen as it is a safe way to store data. In order to confirm that the log reported in the Blockchain is secure, all the IDS in the network run a consensus protocol. This is done as a DIDS has the danger of being compromised if the intruder has breached the network and has altered an IDS or more, thus the authenticity of the records may be compromised. By using a Blockchain and a consensus protocol we can avoid that as the log must be authenticated by the majority of the deployed IDS.

Regarding the processing of the logs, the authors use a cloud-based platform to perform the analysis of the logs. This is done as cloud platforms are a scalable solution but also due to the resources needed to create such a system for large networks where a cloud solution provides the required resources, scalability of increasing or decreasing the power of the centralised server as well as big data analytical tools that can bee used to analyse the logs. It is worth noting that this paper focuses on a rule-based NIDS where the configuration and the tools used are not known.

In terms of application, this design is very similar to the SHIA framework as both approaches are distributed and rely on Big Data Technologies to work. Yet, in the case of the Blockchain method, we see a more general approach in terms of handling and processing the logs whereas in the case of SHIA we see a complete evaluation of a detection system. SHIA does not rely on cloud architecture and uses both HIDS and NIDS whereas the proposed approach heavily relies on the cloud and only uses NIDS.

The main benefit of using the Blockchain method is the safety of using the consensus method to update the Blockchain with the logs, as this ensures the integrity of the data something that is not done in SHIA. Yet, the use of Blockchain is still susceptible to the 51% attack [11]. A 51% attack is when a group of miners, that controls more than 50% of the processing power of the network, block the Blockchain from having new blocks being validated, as a 50% consensus is needed or the blocks to be altered or added in the blockchain, thus having the ability to tamper with the data stored in it. This attack remains theoretical as no one has attempted it and in the case of the DIDS, it would require the attacker to be

able to compromise more than 50% of the running IDS which is very hard to be achieved given the number of IDS needed to be misconfigured for this attack to work.

3.8 Multi-Agent Model for a NIDS

Previously, we have discussed the use of Blockchain in DIDS as well as the use of SHIA and the implementation of FESVDF. All these methods focus on the creation of DIDS. Yet, each implementation is drastically different. A different implementation is proposed [12] focusing on creating a framework that requires multiple agents to run. The proposed framework uses a set of agents to divide each part of the regulation, detection and data processing process of the system in the network.

The framework is broken into a capturing agent that intercepts the packets using a packet sniffing protocol, and a Filtering and Load Balancer agent that checks if the packet has been categorised or not. If it has not, the data is stored in the uncategorised database and it is processed by the decision-maker agent that analyses and categorises the packets in the uncategorised database. Finally, the framework has the Hadoop DFS agent that stores and processes the data for their network characteristics and Agent Manager that generates the systems reports and alerts. It also relies on big data to perform the data processing, which is used to determine if a data packet is an intrusion or not and save the data in the intrusion database.

This method is quite similar to what we have seen in the case of SHIA. Both frameworks rely on big data to process the data but the new framework has a single agent that classifies the data. SHIA uses distributed deep learning and each IDS instance functions as a classification module whereas the new framework only has one agent for classification and the filtering agent that acts upon the known data for reporting whether the characteristics of a data packet match the characteristics of a malicious attack.

This difference may allow for a faster detection time in terms of detecting malicious packets since there is only one agent that is used for classification in collaboration with HDFS and as such we can generate the profile of the malicious packets. Furthermore, this framework may have an advantage over SHIA since it can work as a rule-based IDS since the filtering agent just cross-checks the captured packet with characteristics in the intrusion DB and as such, we can avoid the high number of false positives associated with anomaly-based IDS [8].

Yet, it is important to note that this paper outlines the framework and the authors have not identified the packet interception methods, the machine learning algorithms to detect zero-day attacks and distinguish between malicious and normal traffic as well as how the NIDS agents will work. And as such these advantages are hypothetical and unproven.

3.9 Conclusion of Literature Review

As seen through this Literature Review the use of DNNs and Deep Learning is encouraged as well as seen quite effective compared to more traditional classification methods [9], [7] especially in the case of anomaly-based IDS that have to be able to differentiate between different types of attacks. Furthermore, it should be noted that extensive research has been conducted on different types of anomaly-based IDS implementation.

Notably, we have seen the use of simple DNNs but also enhanced procedures through the FESVDF [7] which were used to isolate important features from the data set that can be used to create different IDS that run on different layers of the network, indicating the importance of isolating the most important features when a developing a NIDS. Another method is seen, which was quite effective, has been the use of LSTM [6]. This method focuses on separating the spatial and the temporal characteristics of the network traffic and assigning them in different networks, in this case, spatial to CNN and temporal to LSTM. This approach is similar to the FESVDF approach in the sense that each DNN is specialized in one aspect of the network traffic.

But seeing the performance metrics of both methods one can deduce that the performance of the FESVDF can be seen as a better solution, although this is debatable due to the evaluation stage using different data sets and metrics. This can be seen as HAST-IDS has a reported accuracy of 99.69% [6]. Looking at *Figure 4* the results of the implementation of FESVDF, the main metric used in CR which is the overall accuracy of the network, it can be observed that FESVDF is performing noticeably better than HAST-IDS. Indicating that this methodology is better than the one proposed by the authors of HAST-IDS.

It is worth noting that we can also see the evaluation of more traditional classification methods both in *Figures 1 and 4*. In *Figure 1* we can see that the DNN especially DNN-3, which means DNN with 3 hidden layers, is outperforming the other methods. The only method that is seen to be similar in terms of accuracy is the K-nearest neighbour and Naive Bayes, from which only the K-nearest neighbour has improved precision.

In *Figure 4* we can see the comparison of SVM with the use of NN in order to create a NIDS for all features as well as a layers IDS. It is noticeable the performance improvement when using a NN. The only significant difference is the training time which increases substantially but the testing time is decreased which shows that NNs can process the data better. Thus proving that the use of DNNs and NNs is preferred.

Finally, we have also seen the use of genetic programming for Signature-based IDS which was meant to be promising by combining the advantages of an anomaly-based and a signature-based IDS. Yet, this resulted in almost a 0% FPR, which was one of the main advantages of a signature-based IDS, but with an accuracy rate of 90.26%, the performance

is significantly less than other implementations. Which makes it a worse choice than using a DNN in an anomaly-based IDS.

In terms of creating functional IDS frameworks we have mainly seen an implementation of NIDS, we have seen the SHIA framework, the use of agents, a blockchain implementation as well as layer-specific NIDS. Each framework has its advantages and disadvantages. Of all the frameworks the two more comprehensive are the SHIA and FESVDF. SHIA is a better framework as it covers both network and host-based IDS whereas the FESVDF only covers the network, which leaves it vulnerable to inside-the-network attacks as it will not detect malicious behaviour in the hosts.

The other two frameworks are not as comprehensive as FESVDF and SHIA are, yet they introduce interesting improvements to the design. Blockchain introduces extra security in terms of data integrity, logs of attacks need to be verified by all IDS in order to be added to the blockchain, thus adding an extra layer of security since more than 50% of the IDS in the system must be compromised whereas in SHIA only one IDS needs to be compromised for the system to be flooded with false alarms or suspicious activity in that network to be hidden.

The implementation of a multi-agent NIDS is quite similar to what we have seen in SHIA and FESVDF implementations but its main advantages are its modularity and overall design. It uses an agent to categorize uncategorised network traffic and one agent to filter uncategorised and categorised traffic, in the categorised traffic it can distinguish between malicious or benign data connections. Furthermore, the overall traffic as presented by the authors is a combination of a rule-based and anomaly-based IDS, although no direct reference is made in the paper but the filtering agent uses a database to reference the incoming data packets. This may result in mitigating some of the disadvantages of both anomaly-based and signature-based IDS but it is theoretical as this framework has not been realised. It is worth noting that the multi-agent architecture only focuses on the network side and not on the host side of the overall system.

Concluding the Literature Review, we can see that the best method to construct anomaly-based IDS is through the use of DNNs as they have better accuracy than traditional classification techniques. DNNs can be seen outperforming other deep learning techniques as well which is quite encouraging in using them in anomaly-based IDS. Furthermore, different frameworks have been identified with the best full implementation being SHIA and the best theoretical implementation being the multi-agent IDS architecture. Yet, it is worth noting that none of these architectures will be used in this paper as we will be strictly focusing on exploring and finding the correct configuration for an accurate DNN.

4 Use of Artificial Neural Networks

4.1 Introduction to Artificial Neural Networks

[13], Artificial Neural Networks (ANN) or ANNs are biologically inspired computational networks. The most common type of ANNs are Multi-Layered Perceptons (MLP) which is a supervised form of the learning algorithm. It can be successfully applied to solve a wide range of problems. They are comprised of three main parts; the input layer, the hidden layers and the output layer.

4.2 Composition of Artificial Neural Networks

As mentioned in the previous section, ANNs are biologically inspired computational networks. They mimic the way neurons in the brain work by being comprised of multiple layers and nodes called neurons. Neurons are where the computation takes place, they take as input one or multiple values and give as output usually one value. They are usually positioned in layers or vectors and their input can be from the original data or from a previous layer, the input data can be the whole output of the layer or just a subgroup of the outputs. Their output can be used as input in another layer or as a normal output.

The neurons perform the data processing by using weights $W_{i,j}$. Weights are positioned in each neuron and are individual for each input and are used as a way to create pathways that are preferred, similar to the neural pathways found in a biological brain. The way inputs for each neuron in ANNs is different. Each input is multiplied by its respective weight and then all inputs are some together. Then the sum is inputted in Tanh activation function, the most famous ones are the sigmoid and *Tanh* activation functions, which return the output. The output is then sent to the next layer for further processing.

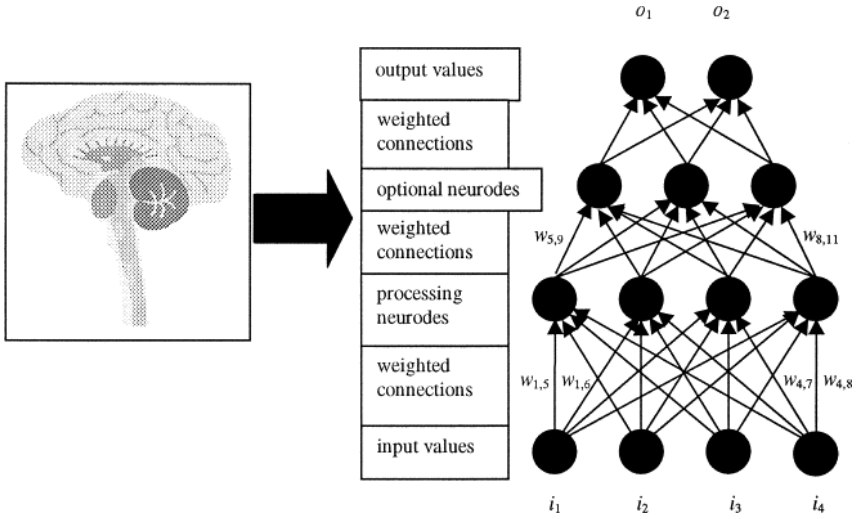

Figure 6: ANN Composition from [14]

Additionally, the weights in each neuron need to be updated in order to make the network better at producing the needed result. The most famous method is Backpropagation. Backpropagation [15] is a technique used to update the network weights, it is a non-linear gradient descent algorithm. It is based on the assumption that the weights in the neurons contribute to the error of the ANN.

Backpropagation begins by calculating the difference between the actual value and the predicted value in the output layer. Once the difference is calculated it is then inputted in the inverse of the activation function and the resulted value is used to recalculate the weights. This is done with the following function $W *_{i\ ,j} = W_{i,j} + \rho\delta_j u_i$. W is the weight, the asterisk denotes the new weight, ρ is the learning parameter, δ is the difference from the output layer and U is the output of that neuron.

When it comes for the non-output layers δ is calculated using the delta from the previous layer using the following function $\delta_i = w_{i,j}\delta_{i+1}f'(S_4)$. Please note that if multiple weights exist then all weights need to be calculated to derive δ.

An ANN is composed of three main parts or layers. In the input layer, where the data is first inputted into the network layer, the hidden layers are used to extract patterns and create pathways that can be used to successfully predict the output and the output layer that releases the output and performs the final processing.

4.3 Deep Neural Networks

Previously, we have discussed about the ANN and its composition. In this section, we will discuss a more specific approach to ANN, the Deep Neural Network (DNN).

It is worth noting that ANNs can be shallow, having just one hidden layer, or being deep, having more than one hidden layer. DNNs are the latter category. There are multiple different subcategories of DNNs like Convolutional Neural Networks, Radial Basis Function Neural Networks etc. But in this paper, we will be focusing on Feed-Forward DNNs (FF-DNN). FF-DNN [16] also known as Multi-Layered Perceptons (MLP) are DNNs with more than one hidden layer. They work very well with classification but also with predicting values, something crucial for the development of a reliable NIDS.

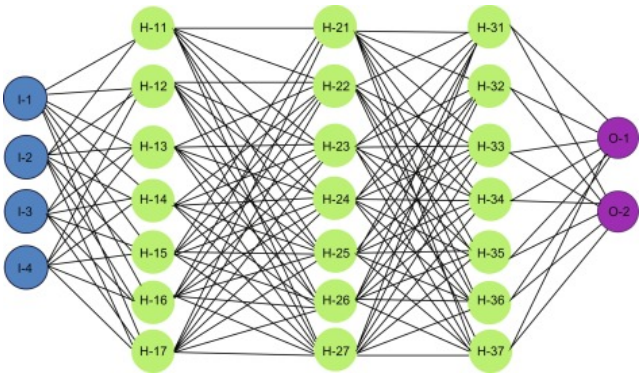

Figure 7: FF-DNN Composition from [16]

4.4 Applications of ANN

As it has been mentioned previously, ANNs and in particular DNNs have multiple applications in particular when it comes to predicting values and classifying values [16]. Some of the most notable applications other than those used as the basis for an anomaly-based IDS are in hydrological systems [17] but are also used for ecological modelling [13] and stream-flow forecasting [18].

Additionally, they are widely used in anomaly-based IDS as mentioned in the Literature Review chapter but also in other cases that have been mentioned before making a really versatile tool for classification and prediction. One of the main areas of its success is that it can be used to understand the relationship between variables that is non-linear[19], something that simple linear regression cannot do.

5 Data set Pre-processing

5.1 Introduction

In order to have a successful implementation using a DNN, data pre-processing needs to take place. Data pre-processing is the act of taking a raw data set and transforming it into a data set that will yield the best results for a Neural Network or a Machine Learning algorithm. The main focus is to extract the most informative subset of features and yield the best result for the classifier [20].

Data pre-processing has multiple steps that need to be undertaken in order to produce a correct data set. The most notable one is feature extraction which is used to extract meaningful data from a data set. This is succeeded by extracting the most informative and compact set of features in order to have the best possible classifier [21]. The extracted data allows our algorithm to be more successful as fewer parameters need to be considered but also noise is removed from the data set which could make detecting a pattern and having an efficient classifier work.

5.2 CIC-IDS-2017 data set main points

The CIC-IDS-2017 data set is a modern data set built to facilitate the creation of modern IDS systems. It accounts for modern attack types which are not covered by previous data sets. Thus it has different features that are implemented in it as seen in figure 8.

It is worth noting that not all of the present features will be present in the final data sets that will be used. This is due to the fact that some of the features may not be statistically relevant enough to be considered for use in the DNN as they may hinder as they act as noise to the model. Also, due to the extra complexity needed for the data to be used. This will be established later on when using the feature extraction algorithm.

The main choice for the data used is the available CSV files that can be obtained from the Canadian Institute for Cybersecurity are in a zip file called MachineLearningCSV. The data there is in different CSV files. The main files are Friday-WorkingHours-Afternoon-DDos.pcap_SCX, Friday-WorkingHours-Afternoon-PortScan.pcap_SCX, Friday-WorkingHours-Morning.pcap_SCX, Monday-WorkingHours.pcap_SCX, Tuesday-WorkingHours.pcap_SCX, Thursday-WorkingHours-Afternoon-Infiltration.pcap_SCX, Thursday-WorkingHours-Morning-WebAttacks.pcap_SCX and Wednesday-WorkingHours.pcap_SCX.

Attribute Name	Attribute Name
FlowID	BwdPackets/s
SourceIP	MinPacketLength
SourcePort	MaxPacketLength
DestinationIP	PacketLengthMean
DestinationPort	PacketLengthStd
Protocol	PacketLengthVariance
Timestamp	FINFlagCount
FlowDuration	SYNFlagCount
TotalFwdPackets	RSTFlagCount
TotalBackwardPackets	PSHFlagCount
TotalLengthofFwdPackets	ACKFlagCount
TotalLengthofBwdPackets	URGFlagCount
FwdPacketLengthMax	CWEFlagCount
FwdPacketLengthMin	ECEFlagCount
FwdPacketLengthMean	Down/UpRatio
FwdPacketLengthStd	AveragePacketSize
BwdPacketLengthMax	AvgFwdSegmentSize
BwdPacketLengthMin	AvgBwdSegmentSize
BwdPacketLengthMean	FwdHeaderLength
BwdPacketLengthStd	FwdAvgBytes/Bulk
FlowBytes/s	FwdAvgPackets/Bulk
FlowPackets/s	FwdAvgBulkRate
FlowIATMean	BwdAvgBytes/Bulk
FlowIATStd	BwdAvgPackets/Bulk
FlowIATMax	BwdAvgBulkRate
FlowIATMin	SubflowFwdPackets
FwdIATTotal	SubflowFwdBytes
FwdIATMean	SubflowBwdPackets
FwdIATStd	SubflowBwdBytes
FwdIATMax	InitWinbytesforward
FwdIATMin	InitWinbytesbackward
BwdIATTotal	actdatapktfwd
BwdIATMean	minsegsizeforward
BwdIATStd	ActiveMean
BwdIATMax	ActiveStd
BwdIATMin	ActiveMax
FwdPSHFlags	ActiveMin
BwdPSHFlags	IdleMean
FwdURGFlags	IdleStd
BwdURGFlags	IdleMax
BwdHeaderLength	IdleMin
FwdPackets/s	Label

Figure 8: Attributes present in the CIC-IDS-2017 data set as taken from [22]

Some of these files do have only the attack types specified in their file names like DDoS or PortScan while other files have a multitude of attacks. Some attacks are mixed in the other files [23] thus making it harder to create three datasets, one for testing, one for training and one for validation, correctly [24]. Furthermore, a lot of data points have incomplete data, Not a Number (NaN) or 'Infinity'. Some studies suggest the removal of those as it is easier and reduces the size of the dataset [25].

Moreover, these datasets are considered flawed in terms of the class imbalance that they have [26]. This can be addressed by more data pre-processing by removing not statistically significant features but also through sampling methods that can address that. An example is Random Under Sampling (RUS) which is said to be very good [23]. This sampling works by removing the data row of the most common class and it does not replace it. It is worth noting that the use of sampling results in better results [23].

5.3 Selection of Feature Extraction Algorithm

Feature extraction as mentioned before is crucial for the algorithm to run correctly and efficiently. The main goal of the algorithms is to extract the features that affect the output the most. Thus, extracting the most information. Furthermore, it should be noted that feature extraction helps in data visualisation, understanding the data set better but also improving storage usage, training and utilization time to enhance the prediction performance of the DNN [27].

There are multiple techniques, some of them being Principal Component Analysis (PCA), Recursive Feature Extraction (RFE) and ANOVA. PCA is a technique that projects the data to lower dimensions with the largest variance [28]. The eigenvectors and eigenvalues of the covariance matrix of the original feature space are calculated and the n% is selected which forms the transformation matrix that will be applied to the features.

The RFE is a method [28] that utilizes an SVM to calculate the features that are not relevant in predicting the target variant and disregards them. The algorithm trains an SVM on the features and calculates their performance. It then removes the least important features until the desired number of features is reached.

ANOVA is a technique that ranks features by calculating the ratio of the variance between and within groups. Groups, denoting different variables, are used as input to the clustering algorithm used [27]. The higher the calculated ratio, the more important the features are. To calculate the ratio, known as the F score, we use the following calculation:

$$F(x) = \frac{s_B^2(x)}{s_W^2(x)}$$

In this calculation $s_B^2(x)$ and $s_W^2(x)$ denote the sample variances between the groups

(Mean Square Between) and within the groups (Mean Square Within). They can be calculated using the following the equations:

$$s_B^2(x) = \sum_{i=1}^{K} n_i \frac{((\sum_{j=1}^{n_i} f_{ij}(x)/n_i) - (\sum_{i=1}^{K}\sum_{j=1}^{n_i} \frac{f_{ij}(x)}{\sum_{i=1}^{K} n_i}))^2}{df_B}$$

$$s_W^2(x) = \sum_{i=1}^{K}\sum_{j=1}^{n_i} \frac{(f_{ij}(x) - (\frac{\sum_{i=1}^{K}\sum_{j=1}^{n_i} f_{ij}(x)}{\sum_{i=1}^{K} n_i}))^2}{df_W}$$

The use of df denotes the degrees of freedom in the calculations. For MSB, the degrees of freedom are $df_B = K - 1$ and for MSW it is $df_W = N - K$. K denotes the number of groups and N denotes the number of samples. The frequency of the textitxthe feature is denoted as $f_{ij}(x)$. Finally, the number of samples in the group is denoted by n_i.

In this paper the chosen method for feature extraction is ANOVA. In regards to choosing ANOVA as the feature extraction technique for the refinement of the data set, the decision was made as it has been adopted for different uses [29] but also as it has really good performance when compared to a Recursive Feature Extraction algorithm, which is considered better but also more computationally heavy [30]. Multiple feature extraction techniques exist, like Principal Component Analysis, and Recursive Feature extraction but the main idea of choosing this technique is that it is fast and has a really good performance like the Recursive Feature Extraction [30] but it is less computational heavy. Additionally, when evaluated, the algorithm seems to be performing better compared to other methods [28]. Lastly, this technique has been used in different applications of DNNs [29] making it an interesting approach.

Furthermore, ANOVA is also preferred as it is an efficient way to reduce dimensionality, which severely will affect the performance of the IDS and the model as the computational complexity is proportional to the number of features that are used [31]. Thus, making it an excellent choice in reducing features and improving the performance of the model [32]. It is also preferred when implemented as a filter method [33, 34] in order to select important features. Thus, ANOVA is preferred as it has good performance and is faster since it is a filter algorithm, filtering using a statistical filter. Additionally, ANOVA is as common in research as Principal Component Analysis (PCA), not many articles employ it as a feature selection algorithm. Making its use an interesting investigation to what degree it can help improve the performance of the network.

To apply ANOVA for feature extraction, firstly the constant and quasi-constant features need to be removed, as these features tend to provide little to no information about the variable being predicted, thus allowing ANOVA to perform better. Then, data standardization

will be performed to avoid a certain value to dominate the classifier. Finally, ANOVA will be applied. Data standardization has been done using the StandardScaler [35] as it is one of the most widely used in the industry while also making the data distribution better. As seen in the graph below, there are quite a few features that are heavily correlated with the output. The main thing is to evaluate the best number of features needed to achieve the best possible result.

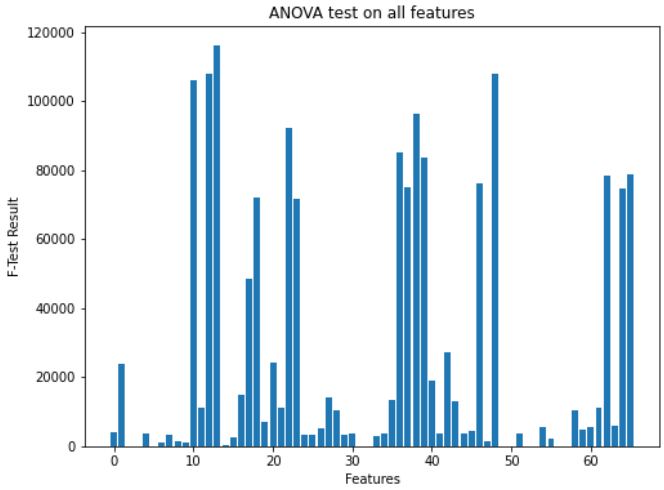

Figure 9: ANOVA results of all the features in the dataset

To evaluate the optimal number of features needed the different number of extracted features was evaluated using a tree classifier. The tree classifier uses the Gini function, it is based on the DecisionTreeClassifier by Sklearn [35]. The classifier was used with default values. Having that we can work on evaluating the number of features needed. The evaluation begins with a number of 10 features and concludes with a maximum of 45 features. As a baseline, a tree is also used with all the features to see the performance that has to be met or even surpassed.

The goal of this evaluation is to find the perfect balance between feature number and performance. As such the use of all the futures in a tree algorithm results in an accuracy of 99.82%. This is the benchmark accuracy that the feature subset needs to get close to or even surpass. Initially, a subset of 10 features is used. The results from the evaluation have a significant deviation from the benchmark accuracy as seen in (Table 1). As such further evaluation needs to be done, 3% may not look like a lot but it is still a significant number that can affect the performance of the DNN.

When increasing the number of features to 15 the accuracy increases compared to the

Number of Features	Accuracy
All Features	99.82%
10	96.13%
15	97.48%
20	97.52%
25	97.53%
30	97.78%
35	99.72%
36	99.79%
38	99.86%
40	99.86%
42	99.83%

Table 1: Accuracy of DecisionTreeClassifier with different number of features

performance in 10 features as seen in (Table 1) but it is still significantly smaller than the output of using all the features. Increasing the number of features to 20, the performance has a tiny increase and indicates there is some room for improvement. At 25 features the accuracy improves only a tiny percentage as seen in (Table 1) which indicates a plateau. Still, a larger feature selection needs to be implemented to ensure that better performance is observed. At 30 selected features the performance is increased and at 35 features there is a significant increase as seen in (Table 1) which is closer to the total number of features selected. This increase in performance indicates that the number of features seems to be reaching its optimal value. At 36 features, the accuracy slightly increases. The same can be seen when the number of features is 38, where the accuracy is better than when all features are used. At 40 features, the accuracy remains the same and at 42 features the accuracy decreases making the best choice of features would be either 36 or 38.

Having completed the tests, the number of features that will be used is 36. The number of features was chosen to be 36 as the performance difference to 38 is minuscule. Additionally, the smaller number results in less complexity for the model.

5.4 Data Normalisation Algorithm

Data Normalisation is one of the most important parts of data pre-processing. Data normalisation is important as it allows the model to be able to back-propagate easier and better. Also, the model can make better predictions as the input is modified to remove skewness to large values and to give correct results by changing the data distribution [36]. Data normalisation involves the scaling of features to fit in the same numerical scale in order to

have the same importance [37]. It is worth noting that data normalisation contributes to the consistency of the data set.

For CIC-IDS-2017, data normalisation is performed using the Power Transforms method. Power transforms is a data normalization method that normalizes data using logarithms. It has two main implementations, the Box-Cox implementation which works for only positive values and the Yeo-Johnson methodology which works for both positive and negative values. In this application, the Yeo-Johnson methodology will be used. The reason behind the choice of this algorithm is that it can handle both negative and positive values but it also addresses the skewness and the kurtosis of the data [38]. It is worth noting that this method seems to be closer to the normal distribution than the Box-Cox method.

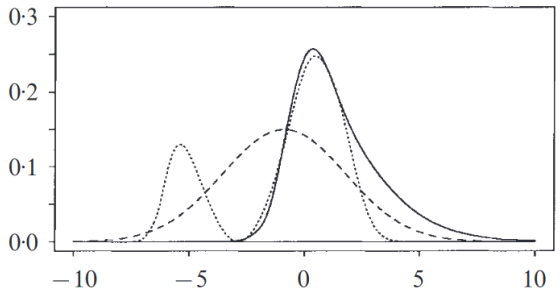

Figure 10: Resulting data distribution of applying Box-Cox to the target data [38]: Dashed line is the target normal density and dot is the Box-Cox transformation

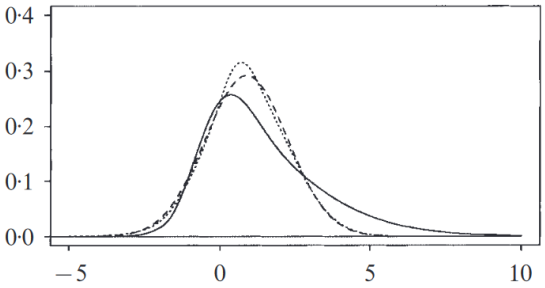

Figure 11: Resulting data distribution of applying Yeo-Johnson to the target data [38]: Dashed line is the target normal density and dot is the Yeo-Johnson transformation

The following equations show how Power Transforms using the Yeo-Johnson method works:

$$\psi(\lambda, y) = \begin{cases} ((y+1)^{\lambda} - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y+1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y+1)^{2-\lambda} - 1)]/(2-\lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y+1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

Where λ denotes the possible type of Box-Cox transformation that will occur, some of the most widely used transformations used are $\lambda = -1, 0, 0.5$. It is worth noting that if y is a strictly positive value Yeo-Johnson transformation is identical to Box-Cox.

In addition, the reason that Power Transforms was chosen compared to other scaling algorithms, like MinMaxScaler and StandardScaler was that Power Transforms ensures that data are closer to a Gaussian Distribution whereas the MinMaxScaler maintains the original shape of the data and doesn't remove data outliers or helps with the data skewness, making it a not so good solution as not all data sets have a good distribution on their own. As for StandardScaler, the main goal of the algorithm is that the variance and standard deviation equal 1 but still do not account for data skewness. Furthermore, Power Transform is considered one of the best ways to normalise traffic data as it transforms different distributions and scales to as close as possible to a Gaussian distribution [39]. An important detail is a fact that Power Transforms is not widely used for research purposes as most papers rely on StandardScaler or in Z-score which makes it an interesting comparison on how much it can improve the performance of the model.

5.5 Addressing the Dataset imbalance

As previously mentioned, the CIC-IDS-2017 dataset is considered imbalanced due to the high discrepancy in the class distribution. This is very important, as this dataset does not have an official data split that could address that discrepancy. This is crucial, in order to have an efficient network the dataset used for learning must be balanced in order to represent all classes correctly.

It is worth noting, that most papers do not address the data split used. This means that they do not mention how data was split or only the percentage split but no details on what was taken into consideration. As such we do not have an official consensus on how to perform this procedure. Yet there is some guidance on what some papers do to address this problem, although their methodology may not help address the specific needs of an IDS.

There are two main ways to use sampling in order to achieve a balanced dataset. One is oversampling, where random data points from the minority clashes are duplicated and added to the dataset. Another method is undersampling, where a random amount of the data points of the dominant class are removed [40]. Both methods are easy to be used but

have their drawbacks. Undersampling may lead to the loss of crucial information that is needed to detect attacks whereas oversampling may lead to overfitting.

An implementation of the oversampling methodology is random oversampling, it is considered a simple oversampling technique where random minority class instances are selected and duplicated and added to the main dataset [40]. This method is though susceptible to overfitting. Synthetic Minority Oversampling Technique (SMOTE) solves the issue of overfitting. This technique belongs to the overfitting class of solutions and works by creating new minority class instances by linear interpolation. This of course has its disadvantages as they tend to lie inside the covex hull of the majority class-leading to over-generalisation [41]. There have been improvements to the algorithm but they tend to increase the computational cost. This method has been used to enhance AdaBoost with Enhanced Feature Selection algorithm thus resulting in better results than using a simple test train split [42].

Another solution, that is based on undersampling is the use of Inverse Random Undersampling (IRUS) [43]. IRUS works by creating multiple subsets that benefit the minority subclass instead of the majority class, the amount of minority examples is P^2 where P stands for the probability of selecting the majority class. The amount of Majority examples in the datasets is $\frac{1}{P^2}$. This implementation leads to an interesting collection of different detectors that can then be fused to create a better classifier, something not easily achieved through using one detector. It is worth noting that this application has only been tested using C45 decision tree and not in a DNN [43].

Additionally, a way to deal with the imbalanced dataset has been the use of a new loss function that helps to deal with the imbalance without modifying the dataset. An example has been found in [40]. Here the authors propose a cost-sensitive loss function that is applied in a Convolutional Neural Network during the demonstration stage. The approach can learn both features representations of both minority and majority classes. This approach is very promising as it yields better results throughout testing thus showing the improved performance of this method.

In our case, we can implement all procedures but because Keras cannot truly implement a cost-sensitive loss function, thus we cannot fully evaluate this proposition. As for the rest of the possible techniques seen up to this point, we believe that IRUS should not be considered as it performs worse than SMOTE on average [43]. Furthermore, we believe that Random Oversampling (ROS) should not be considered as it comes with significant drawbacks meaning that SMOTE is currently the best sampling procedure to be used [40]. As for not implementing a sampling algorithm, we could try that although research has shown that the training algorithm does not perform optimally [23]. Yet, the ratio of the majority to each minority class should be considered. As such, to account for the significantly larger majority class undersampling will be performed to increase the ratio. The selected

method will be Random Undersampling (RUS) due to the ease of implementation. Sampling was provided by Imbalanced learn library [44].

Having selected the preferred number of features Oversampling and Undersampling can be performed. These techniques will be used to artificially increase the number of the minority classes while also decreasing the number of the majority class, ensuring that the ratio of classes is favourable in helping the model perform. SMOTE will be applied to all classes except the majority class and RUS will be applied to the majority class. The ratio achieved of minority classes to the majority class is 1:5.09.

5.6 Metrics implemented in the network evaluation

A very important aspect of evaluating the accuracy and how successful a network is has to do with the metrics used. In order to properly evaluate, we need to decide what areas are of concern, what are the important aspects that the network has to fulfil and how structuring and testing the model works. As such some of the most popular metrics will be considered and a decision will be made on which ones should be included and which ones should not.

It is worth noting that the main metrics interesting in our case have to do with classification since the main goal of the model is to detect and classify attacks, and DNN metrics since we need to be concerned with the overall performance of our system.

Some interesting notation is the following; True Positive (TP) is a prediction that its value was true and was classified as true. True Negative (TN) is a prediction that its value was negative and was classified as negative. False Positive (FP) is a prediction that was classified as positive but its value was negative. False Negative (FN) is a value that was classified as negative but its value was positive.

To begin with, some of the most popular metrics are Accuracy and Precision [45]. Accuracy is the number of correct predictions divided by the number of total predictions. This is one of the main metrics needed as we need to evaluate the efficiency of our model. The higher the accuracy, the better the model performs. In this case, we want one of the highest possible values as the proposed model needs to be deployed later on to protect safety-critical infrastructure. Precision is a metric that calculates the percentage of the correct positives from the total number of positive instances of a whole class [46]. It shows whether or not the classifier works good enough for that class.

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

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

Another important metric is confusion matrices. Although not truly a performance metric, they show the distribution of the predictions and whether they are correct or not.

This is extremely helpful as it can show in which attacks the network works better and in which attacks it underperforms. Thus, indicating future areas of improvement.

Recall is another crucial metric that will be used. Recall is the fraction of true positive classifications and the sum of true positive classifications and false-negative classifications. Recall is used to show how many of the correct positive values were found [45]. This is very helpful as it is an indication of how good the algorithm is in finding the correct results.

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

F1-score is used as the weighted mean of recall and precision [46]. It helps to combine both of these metrics and have a more general idea on how the balance of these metrics is on the model. The F-score is defined by the following formula;

$$F1 - score = \frac{2 * Accuracy * Recall}{Accuracy + Recall}$$

The values are in the range of 0 to 1. Where 0 is the worse possible accuracy and recall and 1 is the perfect balance of these metrics.

Specificity will also be included as an extra metric. It is the percentage of true negatives values divided by the sum of the true negatives and false positives. This metric is important as it shows us how well the model can avoid misclassification and specifically classify the result as it should.

$$Specificity = \frac{TN}{TN + FP}$$

Receiver Operating Characteristic (ROC) curve is a metric that shows the performance of a binary classifier with values in its threshold [46]. It shows the values of the true positive rate vs the true negative rate. The main idea is that we can see the overall performance of the model and pick a good cutoff threshold. Yet, due to the fact that we are attempting to use a multi-class classifier this approach thus, this metric is not useful. The same situation is for the Area Under the Curve (AUC) metric, it calculates the area under the ROC curve and its value denotes the possibility that the classifier will classify a random positive example higher than a negative one [46].

Finally, Mean Square Error and Mean Absolute Error will not be used as these two metrics are used in regression to calculate the error of the predicted value from the actual one. In our case, we do not care about that. We only care about whether the classification has been successful or not.

5.7 Preparation of data in the Experimental Environment

It is worth noting that the files used come from the *MachineLearningCSV* folder. The files in these folders lack 5 main features compared to the original dataset; Timestamp, Source IP,

Destination IP, Flow ID and Protocol. This is done for the model to be able to generalise better, as being trained using these data features would make the model biased against certain IP addresses and could compromise performance.

Furthermore, since the IP addresses are one of the most important aspects of detecting an attack, a fall in the performance of the model is expected after excluding them. Yet, the exclusion is necessary as otherwise the model would be biased against similar IP addresses. That would limit the performance when applied in the real world since some country IP prefixes could be detected as attacks.

Finally, relying on Flow ID, Protocol and Timestamp can also cause problems in the detection of attacks in a real network. This is as Flow IDs are unique to the packets and tend to change depending on the network, also protocol may change as not all attacks of the same family will use the same Protocol thus complicating things even more. As for timestamps, it is better not to use them as they will always be different from the ones in the dataset. Thus, the pattern extracted would be time-specific relating to the past and could not truly be applied in the proposed model.

The first thing needed is to load the data files into the environment. In order to load the files and successfully manipulate the data, we are using the Pandas library [47, 48]. The Pandas library allows to easily filter and remove incomplete data, this data may be instances of Not a Number or Infinity. These values are usually attributed to errors in the data information extraction from the packets. By removing we are talking about the complete removal of the rows affected. This is done in order not to affect the classification by substituting the values with the column's mean which could not be the associated value for that class. Additionally, to ensure less noise exists in the data features that are constant and quasi-constant are removed. As the removed features would provide little to no useful data to the DNN for the classification.

Following the removal of unwanted data, the next step is to prepare the data for splitting. The dataset is initially split into 3 new datasets, a training, a testing and an evaluation dataset. Splitting is performed using the train_test_split method from Sci-kit learn library [35]. The distribution of the original dataset is 64% in the training dataset, 20% in the testing and 16% in the validation dataset.

The reason that the data is split before applying sampling and ANOVA is that sampling changes the data distribution to make picking patterns in the data easier. This may lead to the model overfitting and thus underperforming. To counteract that, sampling is not performed in the testing and validation dataset as they represent how the data would originally be distributed. Thus, making it more of an accurate representation of how the model would behave. ANOVA is performed before sampling in order to decrease the computational load of sampling the data and to avoid information leakage.

ANOVA will be applied with different numbers of features. A Tree Classifier [35] will be used to evaluate the increase in performance of using different features. Having completed feature selection, the optimal number of features for the new datasets is selected. The goal is to keep the number of features as small as possible to ensure that the model will be fast and accurate when it comes to detecting and classifying possible attacks.

Once the features have been selected sampling can commence. SMOTE and RUS are performed. The algorithms will be implemented using the Imbalanced learn library [44]. This allows for the reproducibility of the datasets as well as less time needed to perform sampling.

After, the data sets have been created, data normalization will be taken place. Due to the nature of PowerTransforms, the algorithm may struggle with large values as they will increase in size exponentially. To deal with this issue a Normalizer was used to reduce the scale of the data in order to be handled correctly by the PowerTransforms function [35].

6 Creation and Testing of the Model

6.1 Introduction

The activation function that will be used is ReLU to avoid the vanishing gradient problem. Soft-Max will also be used in the final layer as it allows for a good classification. Finally, different sized Dense Networks are evaluated. Callback functions are used to avoid overfitting by stopping the training of the network if overfitting is detected. They also help by improving the training of the model in case of learning plateaus by altering the model's learning rate. The libraries used to implement the architectures and evaluate them are TensorFlow and Keras for the DNN [49] and Scikit-Learn for the metrics and data pre-processing [35]. The sensitivity metric is provided by AI Fairness 360 library [50]. Sampling was provided by the Imbalanced learn library [44]. The set-up used for data pre-processing, development and evaluation has been Google Colab's Free Jupyter Notebook environment. The environment has 12.69 GBs of RAM and a Disk Size of 107.72 GBs.

Different architectures are implemented, to discover the best combination of unit size in each dense layer and the optimal number of layers used to extract the best performance possible. Additionally, different architectures may behave differently when detecting different attack types, leading to cases where performance varies from attack type to attack type. This should be taken into consideration as it may result in worse performance overall. Thus, testing different architectures could lead to a better performance in all classes instead of the majority classes performing better and the minority ones worse.

6.2 Initial Network architecture

Four hidden layers will initially be implemented with an input size of 36. The number of units in the first hidden layer will be 126. The second layer will contain 64 units with the third layer containing 32 units. Finally, the output layer will have 16 units. Dropout layers with a value of 0.2 are used before each hidden layer, except the first one, and before the output layer. The activation function will be ReLU in all layers except for the output layer which will use Soft-Max. The optimizer used will be Adams with a batch size of 32. The architecture can be seen in Figure 13 in the Appendix.

Architecture Name	Accuracy	Loss Value	Epoch Stop
Initial Architecture	97.87%	0.0539	16

Table 2: Training information of the Initial Architecture

The reported Accuracy is on par with the results seen during the training of the model

Architecture Name	Accuracy	Specificity
Initial Architecture	97.6%	0.9825

Table 3: Accuracy and specificity of the Initial Architecture

Architecture Name	Recall Macro	Recall Micro	Recall Weighted
Initial Architecture	0.86367	0.976	0.976

Table 4: Recall metrics of the Initial architecture

Architecture Name	Precision Macro	Precision Micro	Precision Weighted
Initial Architecture	0.6491	0.976	0.9943

Table 5: Precision metrics of the Initial architecture

Architecture Name	F1-Score Macro	F1-Score Micro	F1-Score Weighted
Initial Architecture	0.672	0.9976	0.9844

Table 6: F1-Score metrics of the Initial architecture

(Table 3 and Table 2). The Recall score was calculated using different modes, one was macro which calculates the global Recall score, meaning that it calculates the overall performance of the model. Micro score calculates the Recall score for each class and outputs the average of all the scores. And weighted, that is the same as the micro score but the averages are weighted based on the class imbalance in the dataset. The Recall results indicate that the network tends to perform exceptionally well for most of the classes (Table 4), indicating that the data pre-processing has helped the network to deal with imbalance, yet overall it is struggling to find the correct results, this is known to be a common issue for DNNs used in NIDS. Precision is calculated using the same modes as Recall, having macro, micro and weighted values. Precision indicates that the model performs exceptionally in regard to individual classes (Micro and Weighted). But overall (Macro) is not performing as good, it is that it is under-performing in one or more classes as seen in (Table 5). The Specificity score indicates that overall the model successfully avoids misclassification, something which is crucial in the performance of the model overall (Table 3). The F1-Score is used to indicate the balance between Accuracy and Recall of the model, 1 means perfect balance and 0 means worse possible accuracy and recall. In this model, the result has been evaluated using macro, micro and weighted values that are similar to previous scores. The scores in (Table 6) indicate that overall (Macro) the balance is not the best, maybe caused by the bad recall score, but for individual classes (Weighted) it is the best possible thus having an excellent balance.

As can be seen in the confusion matrix in (Appendix/Figure 17), the Benign class suffers from false negatives which indicate that benign packets are perceived as attacks. That is a well-documented problem of anomaly-based IDS [8]. It should be noted that most classes report false positives which fall under the general notion of DNNs but also some classes seem to have false negatives that may result in attacks happening and not being detected. These attacks are DoS Hulk and Web Attack Brute Force. In the DoS Hulk, we have fewer false negatives than accurate and false-negative cases thus meaning that it is safer. Yet, in the case of Brute Force Web Attacks, this is not the case meaning that the model's weak point is this particular attack.

6.3 Second DNN Architecture

Having completed the evaluation for the initial architecture, it is time to see another architecture that may be able to perform even better. The new architecture will be a different version of the existing one. The number of units will be altered to allow for more trainable parameters. Four hidden layers will be implemented with an input size of 36. The first hidden layer will have 252 units. The second layer will have 128 units with the third layer having 64 units. Finally, the output layer will have 32 units. Dropout layers with a value of 0.2 will be implemented in all hidden layers and output layer, except the first hidden layer. The activation function will be ReLU except for the output layer which will be Soft-Max. The optimizer used will be Adams with a batch size of 32. The initial training will be conducted using the training and testing dataset and will run for 25 epochs. Callbacks have been employed to reach optimal settings. During training, the accuracy rate reached is less than that of the previous architecture while the loss function is higher (Table 7). Indicating that the increase in trainable parameters does not help. The increase is to 52704 parameters from 15600 parameters. The architecture can be seen in Figure 14 in the Appendix.

Architecture Name	Accuracy	Loss Value	Epoch Stop
Initial Architecture	97.87%	0.0539	16
Second Architecture	97.37%	0.0701	24

Table 7: Training information of architectures up until now

Architecture Name	Accuracy	Specificity
Initial Architecture	97.6%	0.9825
Second Architecture	97.34%	0.9823

Table 8: Accuracy and specificity of all architectures up until now

Architecture Name	Recall Macro	Recall Micro	Recall Weighted
Initial Architecture	0.86367	0.976	0.976
Second Architecture	0.9061	0.97355	0.97355

Table 9: Recall metrics of architectures up until now

Architecture Name	Precision Macro	Precision Micro	Precision Weighted
Initial Architecture	0.6491	0.976	0.9943
Second Architecture	0.6431	0.9735	0.9928

Table 10: Precision metrics of architectures up until now

Architecture Name	F1-Score Macro	F1-Score Micro	F1-Score Weighted
Initial Architecture	0.672	0.9976	0.9844
Second Architecture	0.6653	0.9735	0.9923

Table 11: F1-Score metrics of architectures up until now

Having finished the training of this model, it is time to evaluate the model's performance using the metrics that were previously employed. The metrics are identical to the ones used in the previous model to help in compatibility but in proper evaluation. The model is less accurate (Table 8) than the previous model meaning that the increased size does not help it perform any better.

In terms of the Recall score, there is an increase indicating that the model performs a bit worse than the previous model in terms of individual classes (Micro and Weighted) while its overall performance (Macro) has slightly increased. Thus, the model can detect better the correct classes as seen in (Table 9). In terms of Precision, the model performs exceptionally well in the individual class scores (Micro and Weighted) in (Table 10) but struggles with its global score (Macro). It is worth noting that the model is underperforming compared to the previous model thus there is no overall performance gain. The Specificity is similar to the previous model (Table 8) indicating that the model avoids misclassification but it is still underperforming compared to the previous model. The F1 score in (Table 11) indicates that overall (Macro) the balance is not the best, maybe caused by the bad recall score, but the weighted score is better thus indicating a better balance when accounting for the class imbalance. Still, the result scores are worse than what was observed in the previous model. Meaning that no improvements have been made.

In terms of the confusion matrix, the same trends observed in the previous models can be seen in this model. This can be seen in (Appendix/Figure 18) in the Benign class where we have a large number of false negatives which indicate that benign packets are perceived

as attacks. Yet, what makes it interesting is the fact that the false negatives have increased compared to the previous model as well as an observed decrease in the number of correctly classified packets. Indicating that the model is severely underperforming. This trend can be seen in other classes as well, where an increase in false positives is observed. In the case of DoS Hulk and Brute Force Web Attack, a slight improvement can be observed in terms of the false negatives, but the number of false positives has increased causing the positive effects to cancel out. Thus the new model does not really improve on the performance of the previous model but rather results in diminishing performance that does not justify the increased training time and the time needed to conduct a prediction.

6.4 Third DNN Architecture

Having completed the tests on the previous model, it has been observed that having more trainable parameters does not necessarily improve the performance of the network. As such a new modification to the new architecture has been proposed. Four hidden layers will be implemented with an input size of 36. The number of units in the first hidden layer will be 256. The second layer will have 64 units with the third layer having 32 units. Finally, the output layer will have 16 units. Dropout layers with a value of 0.2 will be implemented. The activation function will be ReLU except for the output layer that will be using Soft-Max. The optimizer used will be Adams with a batch size of 32. The model trained for 6 epochs before being stopped by the implemented callbacks as there was no improvement in the performance from the first epoch. The accuracy achieved with the validation dataset was 80.32% which is significantly lower than both previously tested models. As such, the model will not be further evaluated as its accuracy is lower and there seems to be no chance of improvement.

6.5 Fourth DNN Architecture

The new architecture will contain more layers, it will have 5 hidden layers. The first hidden layer has 256 units. The second hidden layer has 128 units. The third one has 64 units and the output layer has 32 units. Dropout layers with a value of 0.2 will be implemented. The activation function will be ReLU except the output layer which will be Soft-Max. The optimizer used will be Adams with a batch size of 32. The architecture can be seen in Figure 14 in the Appendix. The accuracy is slightly decreased compared to the first model but it has improved compared to the second architecture (Table 12). The same can be said about the loss function which is significantly less than what has been seen.

The accuracy has slightly decreased (Table 13) compared to the first model but increased compared to the second model indicating that it is an improvement over the second architec-

Architecture Name	Accuracy	Loss Value	Epoch Stop
Initial Architecture	97.87%	0.0539	16
Second Architecture	97.37%	0.0701	24
Fourth Architecture	97.52%	0.0526	25

Table 12: Training information of architectures up until now

Architecture Name	Accuracy	Specificity
Initial Architecture	97.6%	0.9825
Second Architecture	97.34%	0.9823
Fourth Architecture	97.499%	0.9821

Table 13: Accuracy and specificity of all architectures up until now

Architecture Name	Recall Macro	Recall Micro	Recall Weighted
Initial Architecture	0.86367	0.976	0.976
Second Architecture	0.9061	0.97355	0.97355
Fourth Architecture	0.9092	0.97992	0.97992

Table 14: Recall metrics of architectures up until now

Architecture Name	Precision Macro	Precision Micro	Precision Weighted
Initial Architecture	0.6491	0.976	0.9943
Second Architecture	0.6431	0.9735	0.9928
Fourth Architecture	0.661	0.974	0.9939

Table 15: Precision metrics of architectures up until now

Architecture Name	F1-Score Macro	F1-Score Micro	F1-Score Weighted
Initial Architecture	0.672	0.9976	0.9844
Second Architecture	0.6653	0.9735	0.9923
Fourth Architecture	0.681	0.975	0.994

Table 16: F1-Score metrics of architectures up until now

ture. Recall scores have increased (Table 14), especially in the macro score indicating that the model performs better overall than the previous models. Thus, a slight improvement has been observed. The model can detect better the correct class. The Precision macro score has significantly improved (Table 15) compared to the previous models, indicating a better accuracy and overall performance. The Specificity of the model is less than that of the previous models, indicating that the model may suffer from misclassification (Table 13).

The F1-Score values (Table 16) are quite close to the one in the first model as well as in the second model indicating that the overall balance is not the best possible but it is better than all the previous models, the only exception is the micro and weighted scores where the initial architecture is performing better.

The confusion matrix (Appendix/Figure 19) will be used to further evaluate the model. In the benign class, where the majority of misclassification occurs, an improvement can be seen as the number of misclassifications is less than that of the second model but it is still larger than that of the initial model. Indicating that the model still suffers from false positives and the problem has not been solved. A trend of improvement can be seen throughout the confusion array regarding false positives and false negatives. But certain attacks seem to perform worse than the second model or slightly better, thus no improvement has been made. This affects the number of false negatives which is extremely important as it may lead to undetected attacks or false corrective action. Overall, the improvement is not better than the initial architecture and it cannot be judged as a better implementation. In the case of DoS Hulk, the model is performing worse than the second architecture but it is performing significantly better than the initial architecture. In the case of Brute Force Web Attack, the performance is the same as all other tested architectures indicating that there has not been enough improvement.

6.6 Fifth DNN Architecture

Having seen the improvements of adding extra layers, the next architecture will be the same architecture as the previous one with the addition of a new layer before the previous first hidden layer. The new layer will have 512 nodes. The same settings will be used as before. The architecture can be seen in Figure 15 in the Appendix. The performance of the architecture (Table 17) is not as good as the initial model and the other advanced architectures, indicating that increasing the size of the network is not improving the performance.

Architecture Name	Accuracy	Loss Value	Epoch Stop
Initial Architecture	97.87%	0.0539	16
Second Architecture	97.37%	0.0701	24
Fourth Architecture	97.52%	0.0526	25
Fifth Architecture	97.37%	0.06	25

Table 17: Training information of architectures up until now

Looking at the metrics, the accuracy is what was expected by the network as seen in (Table 18) and is on par with the second architecture, being lower than the initial and fourth architecture. Recall scores are lower than what has been achieved from the previous

Architecture Name	Accuracy	Specificity
Initial Architecture	97.6%	0.9825
Second Architecture	97.34%	0.9823
Fourth Architecture	97.499%	0.9821
Fifth Architecture	97.34%	0.9816

Table 18: Accuracy and specificity of all architectures up until now

Architecture Name	Recall Macro	Recall Micro	Recall Weighted
Initial Architecture	0.86367	0.976	0.976
Second Architecture	0.9061	0.97355	0.97355
Fourth Architecture	0.9092	0.97992	0.97992
Fifth Architecture	0.9088	0.9734	0.9734

Table 19: Recall metrics of architectures up until now

Architecture Name	Precision Macro	Precision Micro	Precision Weighted
Initial Architecture	0.6491	0.976	0.9943
Second Architecture	0.6431	0.9735	0.9928
Fourth Architecture	0.661	0.974	0.9939
Fifth Architecture	0.6289	0.9734	0.9933

Table 20: Precision metrics of architectures up until now

Architecture Name	F1-Score Macro	F1-Score Micro	F1-Score Weighted
Initial Architecture	0.672	0.9976	0.9844
Second Architecture	0.6653	0.9735	0.9923
Fourth Architecture	0.681	0.975	0.994
Fifth Architecture	0.6588	0.973	0.993

Table 21: F1-Score metrics of architectures up until now

architectures with the exception of the initial architecture and only on the Macro scores (Table 19). This indicates that the architecture seems to be performing better overall (Macro) than the initial architecture. But not as good when compared to the second and fourth architectures. Thus, no real improvement has been made. For Precision, the Micro and Weighted scores in (Table 20) have the same performance as the second architecture but worse compared to the initial and fourth one. The Macro Precision score is better than all architectures with the exception of the fourth architecture. The Specificity (Table 18) is the worst achieved up until this point, thus no improvement has been achieved. The F1-Scores

(Table 21) are worse than all the other architectures, with the exception of the weighted score for the second architecture.

The trend regarding the decrease in performance can also be seen in the confusion matrix (Appendix/Figure 20). The misclassification of the benign class is stronger, showcasing that the model does not learn the characteristics properly. Additionally, it struggles to correctly classify the examples which result in diminishing performance. This can be seen in most of the classes. Surprisingly, the two classes that have been observed to struggle the most, DoS Hulk and Brute Force Web Attack, continue to underperform. It is worth noting that the performance is slightly better than the one observed in the initial architecture, as false positives have decreased while the number of false negatives has increased, but is still worse than the fourth architecture. This highlights that no more layers need to be added.

6.7 Initial architecture with reduced layers, removal of final layer

Having determined that increasing the number of layers will not help improve the performance of the network, it is time to evaluate whether a reduction in layer size would improve the performance. As such the initial architecture will be used with the final layer being removed (Appendix/Figure 16). The model will be trained for 25 epochs and callbacks will be used to improve training and avoid overfitting. The results obtained from training in (Table 22) are promising as the loss function is one of the lowest compared to other architectures tested and the accuracy is quite high, the second highest achieved, meaning it is a promising architecture.

Architecture Name	Accuracy	Loss Value	Epoch Stop
Initial Architecture	97.87%	0.0539	16
Second Architecture	97.37%	0.0701	24
Fourth Architecture	97.52%	0.0526	25
Fifth Architecture	97.37%	0.06	25
Improved Initial Architecture	97.62%	0.0527	25

Table 22: Training information of architectures up until now

Accuracy is exceptionally high, showcasing the increase in improvement of the performance (Table 23). For Recall, the scores are what are to be expected from a smaller-sized network but the performance is better than that of the original network. Additionally, these scores are better than the ones in the previous architecture and the second architecture, with the exception of the Macro score, but worse than the scores of the fourth architecture (Table 24). For Precision, the metrics (Table 25) indicate an increase in performance compared to the initial and second architecture but the performance remains unmatched compared to

Architecture Name	Accuracy	Specificity
Initial Architecture	97.6%	0.9825
Second Architecture	97.34%	0.9823
Fourth Architecture	97.499%	0.9821
Fifth Architecture	97.34%	0.9816
Improved Initial Architecture	97.62%	0.9842

Table 23: Accuracy and specificity of all architectures up until now

Architecture Name	Recall Macro	Recall Micro	Recall Weighted
Initial Architecture	0.86367	0.976	0.976
Second Architecture	0.9061	0.97355	0.97355
Fourth Architecture	0.9092	0.97992	0.97992
Fifth Architecture	0.9088	0.9734	0.9734
Improved Initial Architecture	0.8858	0.9762	0.9762

Table 24: Recall metrics of architectures up until now

Architecture Name	Precision Macro	Precision Micro	Precision Weighted
Initial Architecture	0.6491	0.976	0.9943
Second Architecture	0.6431	0.9735	0.9928
Fourth Architecture	0.661	0.974	0.9939
Fifth Architecture	0.6289	0.9734	0.9933
Improved Initial Architecture	0.6529	0.9762	0.9935

Table 25: Precision metrics of architectures up until now

Architecture Name	F1-Score Macro	F1-Score Micro	F1-Score Weighted
Initial Architecture	0.672	0.9976	0.9844
Second Architecture	0.6653	0.9735	0.9923
Fourth Architecture	0.681	0.975	0.994
Fifth Architecture	0.6588	0.973	0.993
Improved Initial Architecture	0.6716	0.9762	0.9840

Table 26: F1-Score metrics of architectures up until now

the fourth architecture where the network is more precise, with the exception of the micro score. Specificity (Table 22), is better than all other architectures indicating that there may be an improvement in terms of avoiding misclassification. F1 scores (Table 26) indicate an important improvement in terms of the balance between Recall and Precision but the Macro

score was less than what has been achieved in the fourth architecture. Still, they are the third highest achieved at this point, the second-highest in the initial architecture and the second is the fourth one.

Finally, the confusion matrix (Appendix/Figure 21) indicates that the number of false positives has fallen dramatically across classes. If not, they have not increased. Yet, one of the main issues observed is that the number of false negatives has increased significantly. But, the number of correctly classified cases has also increased, counteracting that issue and leading to a better Sensitivity, Accuracy and Precision scores although Recall scores are not that good. In the case of Brute Force Web Attack and DoS Hulk, where most architectures struggle, it can be observed there is an increase in the detection as well as a decrease in false positives and false negatives have been achieved. This is surprisingly good, showcasing the improvement made by reducing the layers of the architecture.

6.8 Initial architecture with reduced layers, removal of first hidden layer

In the previous section, an increase in performance has been achieved by reducing the size of the network. This change in architecture, did not significantly change the number of parameters in the network. This architecture will be the same as the initial architecture but the first hidden layer will be removed. In this case, a significant decrease in trainable parameters has been observed, from 15704 to 5600 as the first hidden layer is removed. This reduction may help to determine the optimal range of trainable parameters. The model was trained for 25 epochs and callbacks were used to avoid overfitting and alter the learning rate in the case that the network's performance was plateauing. The training was terminated at epoch 17 with the best performance being achieved at epoch 12.

The accuracy calculated was 97.2% and the loss value was 0.0646. These figures, although impressive for the reduction of the trainable parameters are still less optimal than what can be achieved by using the previous architectures (Table 22). The accuracy is one of the lowest achieved and significantly worse than the one in the previous architecture, which up until now has been the best one. The accuracy is also the lowest achieved up until this point, with the exception of the third architecture.

It is worth mentioning that the results are still better in the case of loss function from some of the more complex architectures that have been tested but the accuracy is one of the lowest observed indicating that the gains in the loss function are outweighed by the lack of accuracy. Thus, there will be no evaluation of this architecture as it does not seem to provide any further improvements than architectures that have already been evaluated.

6.9 Hyper-parameter configuration choice

All the tests regarding the evaluation of different layers have been conducted using ReLU as the activation function. The reason behind choosing ReLU as the activation function is that it is a parameter-free, non-saturation activation function that is known to significantly reduce training times in deep neural networks [51]. The activation works by mapping the negative side to 0 while applying identity mapping on the positive side. Additionally, ReLU does not suffer from the vanishing gradient problem because the derivatives for the positive side are always a constant value [52]. Furthermore, looking at the function of ReLU, $f(y) = Max(0, x)$, it can be seen that it is a simple function that requires less computational power than Tanh and Sigmoidal activation functions and thus an increase in performance can achieve.

Dropout has been used in the architectures as a way to accelerate the performance achieved by the networks in learning and modifying their weights. It also protects against overfitting [53]. Dropout works by randomly dropping neurons from the network and severing their connection. This allows for the network to learn robust features from the dataset and not learn to rely on other neurons for information. It reduces overfitting and performs better than other regularization techniques [53].

As it can be seen in the architectures created in this section, the activation function for the output layer is Soft-Max. The reason for this choice is that Soft-Max turns the signal received from the previous layer into probabilities about which class the inputs belong to it [54]. Using the probability attributed to each class, we can classify the output [55]. In the case of this paper, the application requires multi-class classification as more than one neuron is required per class. Thus, sigmoidal is not fit for this use, as it is only used in binary classifiers since it can only produce outputs in the range of 1 to 0. Soft-Max on the other hand is multi-dimensional, meaning that it can handle the multi-class nature of the task and it can produce probabilities for each class.

In every model, an optimization function is needed to help perform back-propagation and optimize the network. The choice of the optimization method is made upon the advantages that each optimizer has in the back-propagation of a network. Adam is an efficient stochastic optimization algorithm that only requires first-order gradients and has little memory requirements [56]. The Adam optimizer requires minimal if at all configurations [57], making it the best choice as there is little to no tuning that is needed to be performed. Additionally, Adam has the combined advantages of AdaGrad, which works well with sparse gradients, and RMSProp, which works well in on-line and non-stationary settings. Thus, its advantages are that the magnitudes of parameter updates are not affected by re-scaling of the gradient, Adam works with sparse gradients, and it naturally performs a form of step

size annealing [57].

6.10 Decision on Architecture used

In the previous subsections different architectures have been created, trained and tested to evaluate which one is the most effective to be used for an anomaly-based NIDS. They were evaluated using different metrics that measure different areas of the architecture's performance. As such, there is a better understanding on how the architectures perform. Additionally, due to the time sensitive nature of the application the network complexity should be evaluated as well to ensure that the best and fastest network can will be used.

Initially, the architectures with a reduced number of layers will be evaluated and then the architectures with a larger number of layers will be evaluated. The evaluation will conclude with the comparison of the best small and the best large architectures. Finally, one architecture will be chosen as the proposed architecture.

There are four small architectures that have been created, the initial baseline architecture, the reduced initial baseline architecture with removed output layer, the reduced initial baseline architecture with removed input layer and the third architecture. The reduced baseline architecture with removed input layer and the third architecture are two architectures that will not be considered due to their reduced performance and large loss values. This was decided in the previously as the performance did not increase at all or the performance gain was minimal at best.

The remaining two architectures, are to be evaluated on which one is more suitable for the job. The initial architecture has the same Accuracy as the reduced-sized one but it has a larger loss function meaning it does not perform that well (Table 22). In regards to Recall, (Table 24) the reduced architecture continues to outperform the initial architecture, indicating a better architecture that can detect true positives, which is crucial in this application as it is needed to detect the correct attacks to provide better protection.

In regards to Precision, the initial architecture performs better in all areas except the macro (Table 25), although the improvements are minuscule. The Precision macro score of the initial architecture is worse than the reduced sized architecture. That indicates that, although the baseline architecture performs better for individual classes, its overall performance (Macro), where classes are not individually considered, is not better than that of the reduced-sized one. Specificity indicates how well the architecture avoids misclassification, the initial architecture has again lower specificity than the reduced sized one (Table 23). F1-score is used to indicate the balance between the accuracy and recall, the initial architecture again performs worse than that of the reduced size one (Table 26). Finally, looking at the respective confusion matrices (Appendix Figure 17 and Figure 18) it can be seen that the initial architecture performs as equally well as the reduced one, although in certain

cases it under-performs. Yet, the reduced size architecture tends to produce more false positives which are equally as bad but due to the nature of the application, they are preferred compared to false negatives. Additionally, the reduced size makes it for a faster network which results in a better performance overall. As such the preferred architecture is the reduced one.

In regards to the larger architectures we have the following, an architecture where the units in each layer were doubled (Second Architecture), an architecture with an additional layer of 16 units (Fourth Architecture) and one more which has a new first hidden layer of 512 units (Fifth Architecture).

Accuracy is one of the most important metrics when it comes to evaluating architecture. Using the Accuracy metric (Table 23), the best performing architecture is the fourth architecture. Regarding the loss function, the better performing architecture is again the fourth architecture (Table 22).

Recall, as mentioned previously, is used to show how many of the correct positive values were found [45]. As such it is an important value to help decide the best architecture. Accounting for Recall scores in (Table 24), the best performance can be achieved by the fourth architecture, the second best by the second one and the worse by the fifth one. The architecture that performs the best regarding Precision is again the fourth architecture followed by the second architecture as seen in (Table 25). The same trend can be observed regarding the F1-Score (Table 26). For specificity, using (Table 23) for reference the best architecture is the second one which seems to be different compared to the previous trends. This seems to indicate that the other architectures tend to produce more false positives, which although is to be expected, still suffers from this problem which may lead to misclassification and lead the network administrator to take the wrong corrective action.

The confusion matrices, which can be found in the Appendix, will help outline the overall performance of the architectures in each class as well as to the degree the architectures suffer from false positive and false negative values. It is worth noting, as it has been mentioned before, that the fourth architecture was not performing as good as the second one, and although the number of false negatives has fallen, the number of false positives has increased. This is to be expected, but the increase in true negatives and positives counteracts this problem and increases the accuracy as it is to be expected. The second architecture, suffers from the same problems although false positives are less while false negatives are more compared to the fourth architecture. Less true positive values are predicted, which results in worse Accuracy and Recall scores. Finally, the fifth architecture seems to be the worst performing one as the number of false positives and negatives increases whereas the number of true positives and true negatives decreases. This can be seen in some classes indicating that no performance gain has been achieved. Having considered all the confusion matrices

the best result overall is achieved using the fourth architecture and this will be chosen.

Having selected the two best architectures, one being the simpler architecture and the other being the more complex architecture it is time to evaluate which one is the best overall. In terms of accuracy, the best one is the smaller model (Table 23). Yet, the smaller model tends not to be performing in the other metrics, with the exception of Specificity, as good as the fourth architecture. This can be seen in Recall, Precision and F1-Score (Table 24, 25 and 26). The metrics indicate that the smaller model struggles to detect the correct number of false positives, which is an issue as it may cause the model not to correctly classify attacks.

Additionally, the balance of recall and precision is better in the case of the Fourth Architecture than that of the Improved initial architecture (Table 26). Yet, as it can be seen with Specificity the fourth architecture tends to have a larger number of false negatives than the improved initial architecture has and as such tends to misclassify possible attacks. Moreover, the differences in metrics between the architectures are tiny, at most the difference is 0.0234. Thus the improvements could be seen as negligible.

Taking into consideration the size of the networks, the preferred architecture would be the improved initial architecture. This is due to its exceptional performance but also because of its smaller size, as it has 15000 trainable parameters compared to 53232 of the fourth architecture. Thus, the worse performance that has been observed in some of the metrics is offset by the network size.

7 Comparison with previous work and conclusion

7.1 Introduction

In the previous section, the Improved Initial Architecture without a final layer was chosen from all the evaluated architectures. Yet, these architectures have not been evaluated with architectures that have been published. In this chapter, the proposed architecture will be evaluated against some published architecture and will be examined using the previously chosen metrics. It is worth noting, that most of the architectures in the literature review have not used the dataset implemented for training the proposed architecture and as such a correct comparison cannot be conducted. Yet, for reference, they will be evaluated as well as other published architectures that have used the dataset implemented in the proposed architecture. The Macro scores of the proposed architecture will be used for evaluation as it allows for correct evaluation since they affect the performance of the networks in real-life more.

To calculate the metrics used for correct comparison, some of the metrics need to be transformed into the metrics used in this thesis. Some papers used the Positive False Rate also known as False Alert Rate can be calculated as $1 - Specificity$ looking at the metrics section of this thesis with the proposed papers [9]. Negative False Rate was also used and can be calculated as $1 - Recall$ [9]. Detection Rate was also used, which by looking at the papers it can be deduced to be Recall [9]. These metrics are also used in the other papers in this section. For Positive False Rate these are AI-SIEM [58], SGM-CNN and and SGM-MLP [59], GNP IDS [8]. For Detection Rate the papers are SGM-CNN [59], GNP IDS [8]. Detection rate is used by SGM-CNN and SGM-MLP [59], GNP IDS [8].

7.2 Summary of Important metrics for the proposed architecture and Previous Work done

Name	Accuracy	Recall	Precision	F1-Score	Specificity	Dataset
Proposed Architecture	97.62%	0.8858	0.6529	0.6716	0.9842	CIC-IDS-2017
Shallow DNN [4]	93%	0.915	0.997	0.955	–	KDD Cup 99
HAST-IDS [6]	99.89%	0.9696	–	–	0.98	CIC-IDS-2012
FESVDF [7]	99.65%	–	–	–	–	KDD Cup 99
GNP IDS [8]	90.26%	0.925	–	–	0.9866	KDD Cup 99
SHIA [9]	96.2%	0.962	0.972	0.965	–	CIC-IDS-2017
AI-SIEM [58]	98.97%	0.982	–	0.65	0.992	CIC-IDS-2017
SGM-CNN [59]	99.85%	0.9985	0.9988	0.9986	0.9869	CIC-IDS-2017
SGM-MLP [59]	–	0.9963	0.9976	0.9968	–	CIC-IDS-2017
DMLP IDS [60]	91%	–	–	–	–	CIC-IDS-2017

Table 27: Performance of the respective architectures

7.3 Comparison to work using different datasets

It should be noted that most of the work done in regards to using DNNs and different Neural Networks architectures in IDS tends to be in older datasets that have official data splits and are widely used but outdated. Thus, in this section, the performance of these architectures will be evaluated against the proposed architecture.

The research conducted regarding the optimal architecture of DNNs [4] yields (Table 27) exceptional results, better than the proposed architecture, except in accuracy. But it should be noted that any information regarding data pre-processing and unit size of the layers is not provided. Thus, it cannot truly be evaluated with the proposed architecture due to these limitations.

HAST-IDS [6] is a unique approach that mixes CNNs and LSTMs to produce an exceptional detection system. It outperforms the proposed architecture (Table 27). Yet, one of the more complex architectures evaluated in the previous section could perform even better. Since the proposed architecture was chosen while taking into consideration simplicity that leads to less computational overhead and faster detection times.

FESVDF [7] is an approach to producing a Lightweight IDS that is modular and is concerned with specific attack types. The model performs better (Table 27) due to the specific type of system compared to the monolithic nature of the proposed architecture. Yet, more Neural Networks running results in more computational overhead which may

result in slower detection.

Finally, GNP for rule generation [8] was a novel idea of combining both the advantages of rule-based and anomaly-based IDS without their disadvantages. Their performance is better (Table 27) compared to the proposed architecture. Although, the use of an alternative architecture that was discussed in the previous chapter would surpass the performance of GNP.

7.4 AI-SIEM

AI-SIEM [58] is Security Information and Event Management System (SIEM) that has been enhanced by Artificial Intelligence. It converts collected security events to individual profiles. The main focus is to discriminate between True Positive and False Positive occurrences, allowing for more accurate detection of different attacks. This is achieved by using multiple artificial neural networks. The networks were trained on over 1000 epochs.

Regarding the results that can be observed in Table 27, which are the average from all of the neural architectures implemented. The performance achieved is better than that of the proposed architecture. Additionally, Recall and Specificity are significantly better than the scores achieved by the proposed architecture. Yet, the F1-Score is lower indicating the balance between recall and precision is not that good.

It should be noted, that the increase in performance observed can be attributed to the combination of data pre-processing. As well as the combination of multiple artificial neural network architectures that have been employed to detect the attacks. Additionally, the architectures employed in AI-SIEM are completely different from the proposed architecture and more complex in terms of trainable parameters. The architectures used are LSTM, CNN as well as a Fully-Connected Neural Network (FCNN). FCNN is similar to FF-NN but all neurons are connected to all the previous neurons, with each connection having different weights [58]. Another important difference could be that the proposed architecture trained only for 25 epochs compared to 1000 epochs used for AI-SIEM, indicating that the amount of training affects the performance. Yet, no information is given about the loss function obtained at the end of training.

7.5 Comparison with SHIA Framework

The SHIA framework is an IDS that combines both a host-based and a network-based IDS [9]. This is quite important as it is a complete suite for detecting possible attacks and protecting the system. For this comparison, the focus will be on the NIDS.

The proposed DNN architecture is an FCNN that utilizes batch normalization and dropout. There is no mention of any data pre-processing of the data other than that the

input layer has 77 neurons. Whereas in the proposed architecture the input layer has 36 indicating that more features are used by the network (Appendix/Figure 16). Information about the configuration of the individual layers is provided. The architecture utilizes a larger number of units than the proposed architecture which may lead to increased performance. Dropout is significantly lower than what is employed by the proposed architecture, 0.01 versus 0.2. It is also worth noting that the training of the model was done KDD Cup 99 but testing was conducted using CIC-IDS-2017. The best performance is observed when using the three hidden layers from the SHIA DNN configuration.

The performance of the proposed architecture (Table 27) is less than that of the SHIA framework. It should be noted that the SHIA framework groups certain attacks together and utilizes more features than the proposed architecture. This grouping allows for architecture to have to classify fewer classes meaning that it is easier for the model to extract the patterns for each respective attack. Furthermore, more features are used which could impact the performance of the model. Additionally, the model trained for 300 epochs whereas the proposed architecture model trained for 25 epochs.

In conclusion, better performance is observed in the case of SHIA. But, this can be attributed to the reduced number of classes, larger number of nodes employed, the more complex architecture used and the larger number of features used. It should be noted that no information is given about callbacks that were used to achieve optimal results.

7.6 SGM-CNN

SGM-CNN is using a CNN architecture employing a different type of sampling method [59]. SGM is a sampling method that combines SMOTE with under-sampling based on clustering using Gaussian Mixture Model (GMM). This sampling technique allows for the CNN to extract the best features possible and thus achieve better performance. For training, the batch size is set to 256 and 100 epochs are used.

Looking at the results (Table 27), we can see that the performance achieved is significantly better than that of the proposed architecture. Indicating that the CNN performs better than the simple model employed in the proposed architecture. Furthermore, in the paper, there is a comparison between an MLP using SMOTE with RUS with the SGM-CNN. The performance difference between the CNN and the MLP is tiny indicating that the performance gains may be solely due to the sampling techniques employed. Yet, more interestingly, the configuration of the MLP with the proposed architecture is identical to the proposed architecture although a different optimizer is employed in the MLP, Nadam opposed to Adam.

The MLP performs significantly better than the proposed architecture but this can be attributed to more training and different optimizer used as well as due to the reduced feature

dataset employed by the proposed architecture. Since both SGM-CNN and SGM-MLP use 77 features from CIC-IDS-2017 as opposed to 36 used by the proposed architecture.

Yet, overall it could be argued that the use of the proposed architecture is better compared to the CNN since the model used is significantly less computational heavy and thus it can detect attacks faster. Also, due to the number of features used by the CNN and MLP, it can be argued that the resulting models have way more trainable parameters than the proposed architecture, especially the MLP. Leading to more computational overheads. Additionally, more epochs were used to train the SGM-CNN and the MLP than the proposed model, although there is no mention of the loss value during training. Indicating that if the proposed architecture trained for more epochs the results may have been closer to the SGM-CNN and MLP. But also that the loss function may be higher in these models.

7.7 DMLP IDS

An architecture that is closer to the structure of the proposed architecture is the Deep Multi-Layer Percepton (DMLP) IDS [60]. DMLP uses RFE to select the best features for the classifier to use. That way, the classifier can only focus on the important features and extract the most important information, resulting in similar or even better results than of using all the features. This approach is different from the one used in this thesis, where ANOVA has been employed to select the best features. The difference in design, as well as the use of similarity of both architectures, will play an important role in deciding whether DMLP is better than the proposed architecture.

Looking at the results in Table 27, it can be seen that the only available metrics are accuracy, the loss function when the reduced dataset is used and the ROC curve for the full dataset. Yet, it is not clear how to deduce the exact false positive rate, as no values exist except the ROC curve figure. Thus, Specificity cannot be deduced.

By just using the accuracy, the proposed architecture is performing significantly better, although it should be noted that the performance is from using the whole dataset. The performance of the reduced dataset is 89% with a loss value being 0.18. The dataset is reduced to 10 features which is a reduction of 95% [60]. It is impressive that by just reducing the dataset by that much, the performance loss is only 2%.

By looking at the results, we can deduce that the performance is lower compared to the proposed architecture (Tables 27 and 22). It should also be noted, that only three hidden layers are used and that the system is only used to detect incoming DDoS attacks or benign data. Due to the DMLP being a binary classifier, better performance than that of the proposed architecture was expected. Thus, it can be deduced that the DMLP approach [60] needs refinement in order to improve its performance. Furthermore, no data pre-processing, like normalization, splitting procedure etc. other than RFE has been mentioned in the paper

meaning the bad performance can be due to bad pre-processing.

7.8 Conclusion

Many architectures have been shown in regard to how they have performed with their respective metrics. It is worth noting that the proposed architecture uses a reduced dataset to improve overall performance and have faster detection without a significant decrease in performance. Additionally, attacks have not been grouped to make attack detection easier.

Looking at the architectures that have used different datasets. It can be seen that the performance tends to be similar if not better than the proposed architecture. This can be attributed to using different types of neural network architectures as well as splitting the IDS into different modules to deal with domain-specific attacks [7]. Finally, impressive performance can be seen in the of GNP for rule generation as it is a novel combination of two technologies. Yet, the performance is not as good as compared to the fourth architecture. Another reason these architectures may tend to perform better may be the grouping of attacks as well as the use of outdated attack definitions that may yield better performance.

The SHIA framework [9] has been one of the most promising architectures that provided a whole suite for system protection. The NIDS, which is the area of interest has exceptional performance in its metrics. When compared to the proposed architecture it is performing better This can be attributed to the larger number of units used, the use of batch normalization and that SHIA is an FCNN which is a more complex Network. It should be noted that this approach categorizes the attacks into different classes make the classification of data significantly easier. Furthermore, more features are used for the classification which makes the extraction of patterns easier. As such, SHIA is a far more complex architecture than the one proposed, making the decision to use it debatable as it may impact the detection of attacks in a real environment.

Regarding AI-SIEM [58], the performance observed (Table 27) is better than that of the proposed architecture. This is to be expected since more complex architectures are employed and used to detect the attacks resulting in better performance overall. Yet, the performance difference, with the exception of Recall, is not that big indicating that the lightweight nature of the proposed architecture is still good for use.

SGM-CNN is a CNN that utilizes a specific sampling technique to achieve better results as the model learns the patterns in the data. SGM can be seen to perform better than the sampling technique employed in this thesis. This can be seen in the paper [59] as there is a performance difference, although tiny. The performance of the CNN is better than the proposed architecture, indicating that the use of CNN could be better in correctly detecting and classifying attacks. Yet, it comes at the cost of more computational power as it is a more complex model that does not use feature selection like the proposed architecture.

In the case of SGM-MLP [59], it is clear that more training, the use of Nadam and the use of more features can increase the performance of the network. This is significant as it shows that different optimizers should be researched. It also indicates that Adam may not be the best optimizer used.

Finally, the DMLP is an approach to detect possible DDoS attacks. It is a DNN with three hidden layers. Due to it being a binary classifier, better performance than the one achieved as expected. This indicates that the configuration used by the authors is not as good as the proposed model. Which performs much better even with the reduced features and its multi-classification nature.

In conclusion, the proposed architecture has really good accuracy when compared to the models evaluated. There are certain architectures that outperform the proposed architecture but they mainly rely on more complex architectures, a combination of different Neural Networks and performing no feature reduction to achieve their goal. All of these result in higher computational overhead which may impact the performance of these models when used in real traffic.

Additionally, the low number of epochs used for training and the nature of the system not having many trainable parameters makes it a preferable change to the other architectures that use more nodes and/or employ more complex techniques like CNNs and LSTMs. This is the main advantage of this architecture, as it is a single module that can take detect multiple attack types and is relatively lightweight.

Yet, there is work that can be done in regards to exploring more areas and pushing the understanding of how DNNs can be utilised in IDS. One of them should be on whether or not clustering attack types offer any advantages in real-life scenarios. Or if they just make performing a corrective action harder. Additionally, research should be done to explore how DNNs could be applied in a real-life environment and what the requirements for such a system should be.

Moreover, modifications to the architecture should be explored to see how the performance could be increased. This can be researching different optimizers and how they work, like Nadam [59], or what is the optimal number of units per layer. Although, it is usually hard to perform these modifications without adding any additional computational overhead and there are no guaranteed results. Different sampling methods should also be used to evaluate if any performance can be gained [59].

Additionally, research should be conducted on how different types of Neural Networks can be used to correctly identify the attacks and how such a system would look like in real-life, a possible extension to AI-SIEM but with more emphasis on less computationally heavy networks and maybe different networks detecting different attack types.

Finally, models should also be evaluated in real networks to observe how they perform.

This can be the speed of detecting an attack as well as what is the optimal rate of packet collection and processing from the network traffic. This would be interesting research as most papers mention the time needed to detect an attack but this results are obtained in an experimental environment.

8 References

[1] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *ICISSP 2018*, 01 2018, pp. 108–116.

[2] K. Siddique, Z. Akhtar, F. Aslam Khan, and Y. Kim, "Kdd cup 99 data sets: A perspective on the role of data sets in network intrusion detection research," *Computer*, vol. 52, no. 2, pp. 41–51, 2019.

[3] M. MontazeriShatoori, L. Davidson, G. Kaur, and A. Habibi Lashkari, "Detection of doh tunnels using time-series classification of encrypted traffic," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2020, pp. 63–70.

[4] R. K. Vigneswaran, R. Vinayakumar, K. Soman, and P. Poornachandran, "Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2018, pp. 1–6.

[5] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30. Citeseer, 2013, p. 3.

[6] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2018.

[7] S. Zaman and F. Karray, "Lightweight ids based on features selection and ids classification scheme," in *2009 International Conference on Computational Science and Engineering*, vol. 3, 2009, pp. 365–370.

[8] Y. Gong, S. Mabu, C. Chen, Y. Wang, and K. Hirasawa, "Intrusion detection system combining misuse detection and anomaly detection using genetic network programming," in *2009 ICCAS-SICE*, 2009, pp. 3463–3467.

[9] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[10] M. Kumar and A. K. Singh, "Distributed intrusion detection system using blockchain and cloud computing infrastructure," in *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, 2020, pp. 248–252.

[11] X. Zhan, H. Yuan, and X. Wang, "Research on block chain network intrusion detection system," in *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, 2019, pp. 191–196.

[12] S. OUIAZZANE, M. ADDOU, and F. BARRAMOU, "A multi-agent model for network intrusion detection," in *2019 1st International Conference on Smart Systems and Data Science (ICSSD)*, 2019, pp. 1–5.

[13] Y.-S. Park and S. Lek, "Chapter 7 - artificial neural networks: Multilayer perceptron for ecological modeling," in *Ecological Model Types*, ser. Developments in Environmental Modelling, S. E. Jørgensen, Ed. Elsevier, 2016, vol. 28, pp. 123–140. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444636232000074

[14] S. Walczak and N. Cerpa, "Artificial neural networks," in *Encyclopedia of Physical Science and Technology (Third Edition)*, third edition ed., R. A. Meyers, Ed. New York: Academic Press, 2003, pp. 631–645. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0122274105008371

[15] B. Lavine and T. Blank, "3.18 - feed-forward neural networks," in *Comprehensive Chemometrics*, S. D. Brown, R. Tauler, and B. Walczak, Eds. Oxford: Elsevier, 2009, pp. 571–586. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444527011000260

[16] H. S. Das and P. Roy, "Chapter 5 - a deep dive into deep learning techniques for solving spoken language identification problems," in *Intelligent Speech Signal Processing*, N. Dey, Ed. Academic Press, 2019, pp. 81–100. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128181300000052

[17] A. Malekian and N. Chitsaz, "Chapter 4 - concepts, procedures, and applications of artificial neural network models in streamflow forecasting," in *Advances in Streamflow Forecasting*, P. Sharma and D. Machiwal, Eds. Elsevier, 2021, pp. 115–147. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128206737000032

[18] M. M. Hussain, S. H. Bari, I. Mahmud, and M. I. H. Siddiquee, "Chapter 5 - application of different artificial neural network for streamflow forecasting," in *Advances in Streamflow Forecasting*, P. Sharma and D. Machiwal, Eds. Elsevier, 2021, pp. 149–170. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128206737000068

[19] N.-T. Vu and K.-U. Do, "Chapter 27 - prediction of ammonium removal by biochar produced from agricultural wastes using artificial neural networks: Prospects and bottlenecks," in *Soft Computing Techniques in Solid Waste and Wastewater Management*, R. R. Karri, G. Ravindran, and M. H. Dehghani, Eds. Elsevier, 2021, pp. 455–467. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128244630000124

[20] A. Subasi, "Chapter 2 - data preprocessing," in *Practical Machine Learning for Data Analysis Using Python*, A. Subasi, Ed. Academic Press, 2020, pp. 27–89. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128213797000023

[21] ——, "Chapter 4 - feature extraction and dimension reduction," in *Practical Guide for Biomedical Signals Analysis Using Machine Learning Techniques*, A. Subasi, Ed. Academic Press, 2019, pp. 193–275. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128174449000040

[22] S. Wankhede and D. Kshirsagar, "Dos attack detection using machine learning and neural network," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1–5.

[23] R. Zuech, J. Hancock, and T. M. Khoshgoftaar, "Detecting web attacks in severely imbalanced network traffic data," in *2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI)*, 2021, pp. 267–273.

[24] Z. Pelletier and M. Abualkibash, "Evaluating the cic ids-2017 dataset using machine learning methods and creating multiple predictive models in the statistical computing language r," *nternational Research Journal of Advanced Engineering and Science*, vol. 5, no. 2, pp. 187–191, 2020.

[25] O. Faker and E. Dogdu, "Intrusion detection using big data and deep learning techniques," in *Proceedings of the 2019 ACM Southeast conference*, 2019, pp. 86–93.

[26] A. Thakkar and R. Lohiya, "A review of the advancement in intrusion detection datasets," *Procedia Computer Science*, vol. 167, pp. 636–645, 2020, international Conference on Computational Intelligence and Data Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920307961

[27] H. Nasiri and S. A. Alavi, *A novel framework based on deep learning and ANOVA feature selection method for diagnosis of COVID-19 cases from chest X-ray Images*, 10 2021.

[28] J. Zang, Y. Huang, L. Kong, B. Lei, P. Ke, H. Li, J. Zhou, D. Xiong, G. Li, J. Chen, X. Li, Z. Xiang, Y. Ning, F. Wu, and K. Wu, "Effects of brain atlases and machine learning methods on the discrimination of schizophrenia patients: A multimodal mri study," *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2021.697168

[29] K. J. Johnson and R. E. Synovec, "Pattern recognition of jet fuels: comprehensive gc×gc with anova-based feature selection and principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 60, no. 1, pp. 225–237, 2002, fourth International Conference on Environ metrics and Chemometrics held in Las Vegas, NV, USA, 18-20 September 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169743901001988

[30] S. Abdulsalam, A. Mohammed, J. Ajao, R. Babatunde, R. Ogundokun, C. Christopher, and M. Arowolo, "Performance evaluation of anova and rfe algorithms for classifying microarray dataset using svm," 12 2020.

[31] P. Ravi Kiran Varma, V. Valli Kumari, and S. Srinivas Kumar, "A survey of feature selection techniques in intrusion detection system: A soft computing perspective," in *Progress in computing, analytics and networking*. Springer, 2018, pp. 785–793.

[32] X. Liu, T. Li, R. Zhang, D. Wu, Y. Liu, and Z. Yang, "A gan and feature selection-based oversampling technique for intrusion detection," *Security and Communication Networks*, vol. 2021, 2021.

[33] S.-Y. Ji, B.-K. Jeong, S. Choi, and D. H. Jeong, "A multi-level intrusion detection method for abnormal network behaviors," *Journal of Network and Computer Applications*, vol. 62, pp. 9–17, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804515002970

[34] V. Bulavas, V. Marcinkevičius, and J. Rumiński, "Study of multi-class classification algorithms' performance on highly imbalanced network intrusion datasets," *Informatica*, vol. 32, no. 3, pp. 441–475, 2021.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] S. Schirra, "Chapter 14 - robustness and precision issues in geometric computation**work on this survey was partially supported by the esprit iv long term research

project no. 21957 (cgal).” in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Amsterdam: North-Holland, 2000, pp. 597–632. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444825377500152

[37] S. H. Javaheri, M. M. Sepehri, and B. Teimourpour, “Chapter 6 - response modeling in direct marketing: A data mining-based approach for target selection,” in *Data Mining Applications with R*, Y. Zhao and Y. Cen, Eds. Boston: Academic Press, 2014, pp. 153–180. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124115118000062

[38] I. Yeo and R. A. Johnson, “A new family of power transformations to improve normality or symmetry,” *Biometrika*, vol. 87, no. 4, pp. 954–959, 12 2000. [Online]. Available: https://doi.org/10.1093/biomet/87.4.954

[39] V. A. Mankov, V. Y. Deart, and I. A. Krasnova, “Evaluation of the effect of preprocessing data on network traffic classifier based on ml methods for qos predication in real-time,” in *Advances in Artificial Systems for Medicine and Education IV*, Z. Hu, S. Petoukhov, and M. He, Eds. Cham: Springer International Publishing, 2021, pp. 55–64.

[40] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, “Training deep neural networks on imbalanced data sets,” in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 4368–4374.

[41] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, “Cost-sensitive learning of deep feature representations from imbalanced data,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 8, pp. 3573–3587, 2018.

[42] A. Yulianto, P. Sukarno, and N. A. Suwastika, “Improving adaboost-based intrusion detection system (ids) performance on cic ids 2017 dataset,” in *Journal of Physics: Conference Series*, vol. 1192, no. 1. IOP Publishing, 2019, p. 012018.

[43] M. A. Tahir, J. Kittler, K. Mikolajczyk, and F. Yan, “A multiple expert approach to the class imbalance problem using inverse random under sampling,” in *Multiple Classifier Systems*, J. A. Benediktsson, J. Kittler, and F. Roli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 82–91.

[44] G. Lemaître, F. Nogueira, and C. K. Aridas, “Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning,” *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

[45] D. Powers, "Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation," *Mach. Learn. Technol.*, vol. 2, 01 2008.

[46] S. Minaee, "20 popular machine learning metrics. part 1: Classification & regression evaluation metrics," 2019. [Online]. Available: https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

[47] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[48] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[49] T. Developers, "Tensorflow," Feb. 2022, Specific TensorFlow versions can be found in the "Versions" list on the right side of this page.¡br¿See the full list of authors ¡a href="htt ps://github.com/tensorflow/tensorflow/graphs/contr ibutors"¿on GitHub¡/a¿. [Online]. Available: https://doi.org/10.5281/zenodo.5949169

[50] R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilovic, S. Nagar, K. N. Ramamurthy, J. T. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang, "AI fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias," *CoRR*, vol. abs/1810.01943, 2018. [Online]. Available: http://arxiv.org/abs/1810.01943

[51] S. Theodoridis, "Chapter 18 - neural networks and deep learning," in *Machine Learning (Second Edition)*, second edition ed., S. Theodoridis, Ed. Academic Press, 2020, pp. 901–1038. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128188033000301

[52] K. Santosh, N. Das, and S. Ghosh, "Chapter 2 - deep learning: a review," in *Deep Learning Models for Medical Imaging*, ser. Primers in Biomedical Imaging Devices and Systems, K. Santosh, N. Das, and S. Ghosh, Eds. Academic Press, 2022, pp. 29–63. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012823504100012X

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[54] C. Zhu, "Chapter 2 - the basics of natural language processing," in *Machine Reading Comprehension*, C. Zhu, Ed. Elsevier, 2021, pp. 27–46. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323901185000023

[55] I. B. Djordjevic, "Chapter 14 - quantum machine learning," in *Quantum Information Processing, Quantum Computing, and Quantum Error Correction (Second Edition)*, second edition ed., I. B. Djordjevic, Ed. Academic Press, 2021, pp. 619–701. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128219829000071

[56] Z. Chang, Y. Zhang, and W. Chen, "Electricity price prediction based on hybrid model of adam optimized lstm neural network and wavelet transform," *Energy*, vol. 187, p. 115804, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544219314768

[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[58] J. Lee, J. Kim, I. Kim, and K. Han, "Cyber threat detection based on artificial neural networks using event profiles," *IEEE Access*, vol. 7, pp. 165 607–165 626, 2019.

[59] H. Zhang, L. Huang, C. Q. Wu, and Z. Li, "An effective convolutional neural network based on smote and gaussian mixture model for intrusion detection in imbalanced dataset," *Computer Networks*, vol. 177, p. 107315, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620300712

[60] S. Ustebay, Z. Turgut, and M. A. Aydin, "Intrusion detection system with recursive feature elimination by using random forest and deep learning classifier," in *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, 2018, pp. 71–76.

9 Appendix

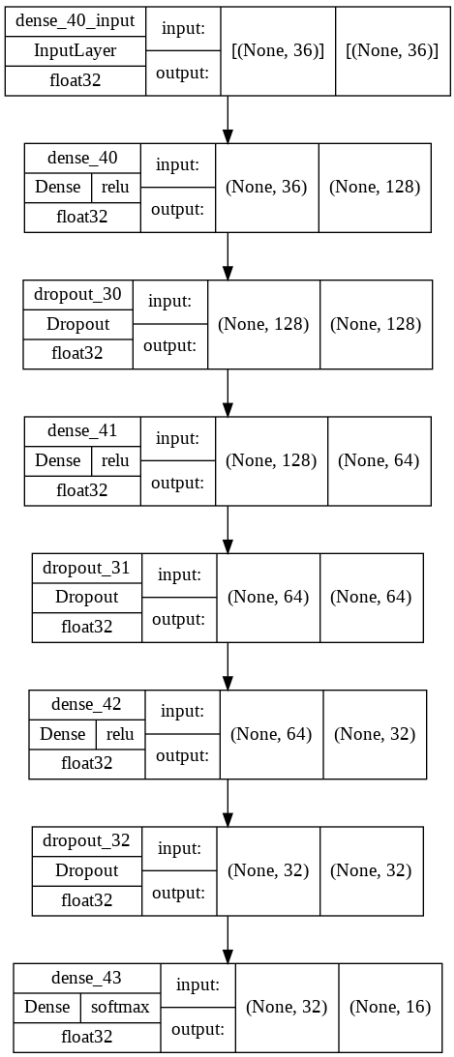

Figure 12: First model architecture used

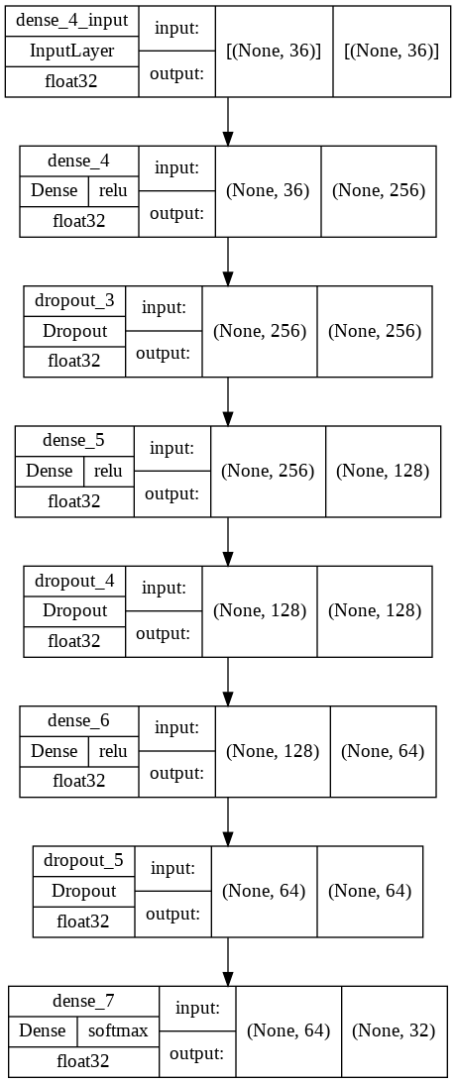

Figure 13: Second model architecture used

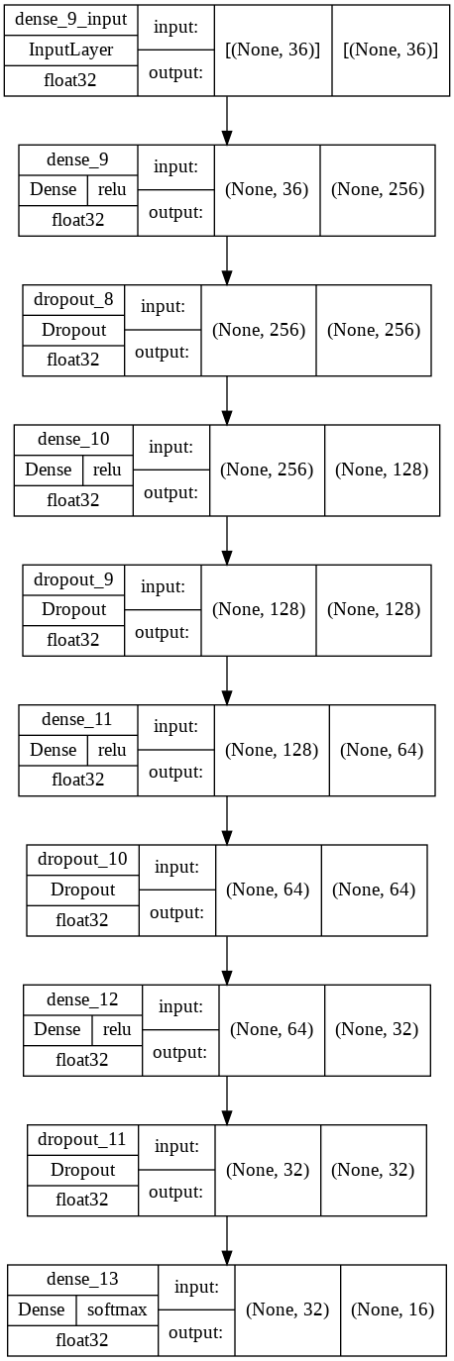

Figure 14: Fourth model architecture used

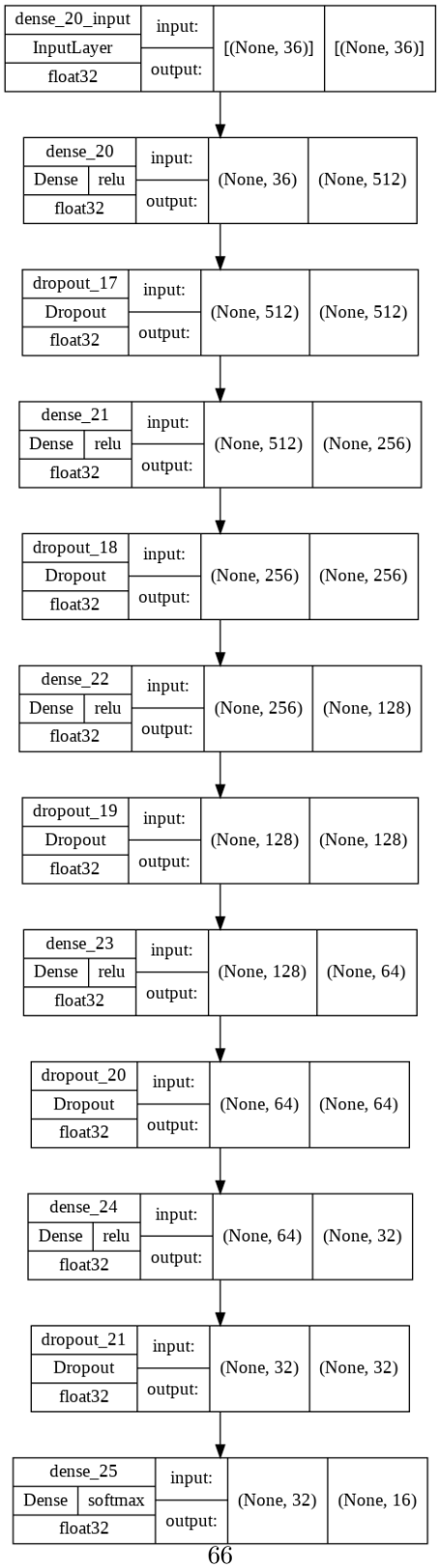

Figure 15: Fifth model architecture used

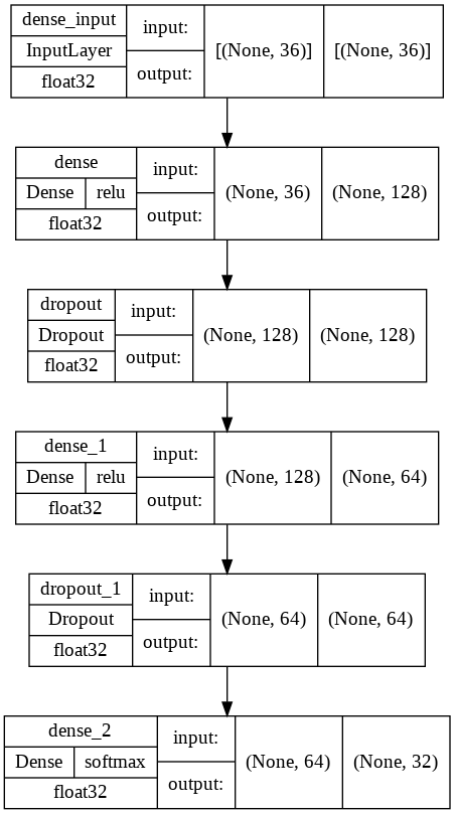

Figure 16: Improved first model architecture used

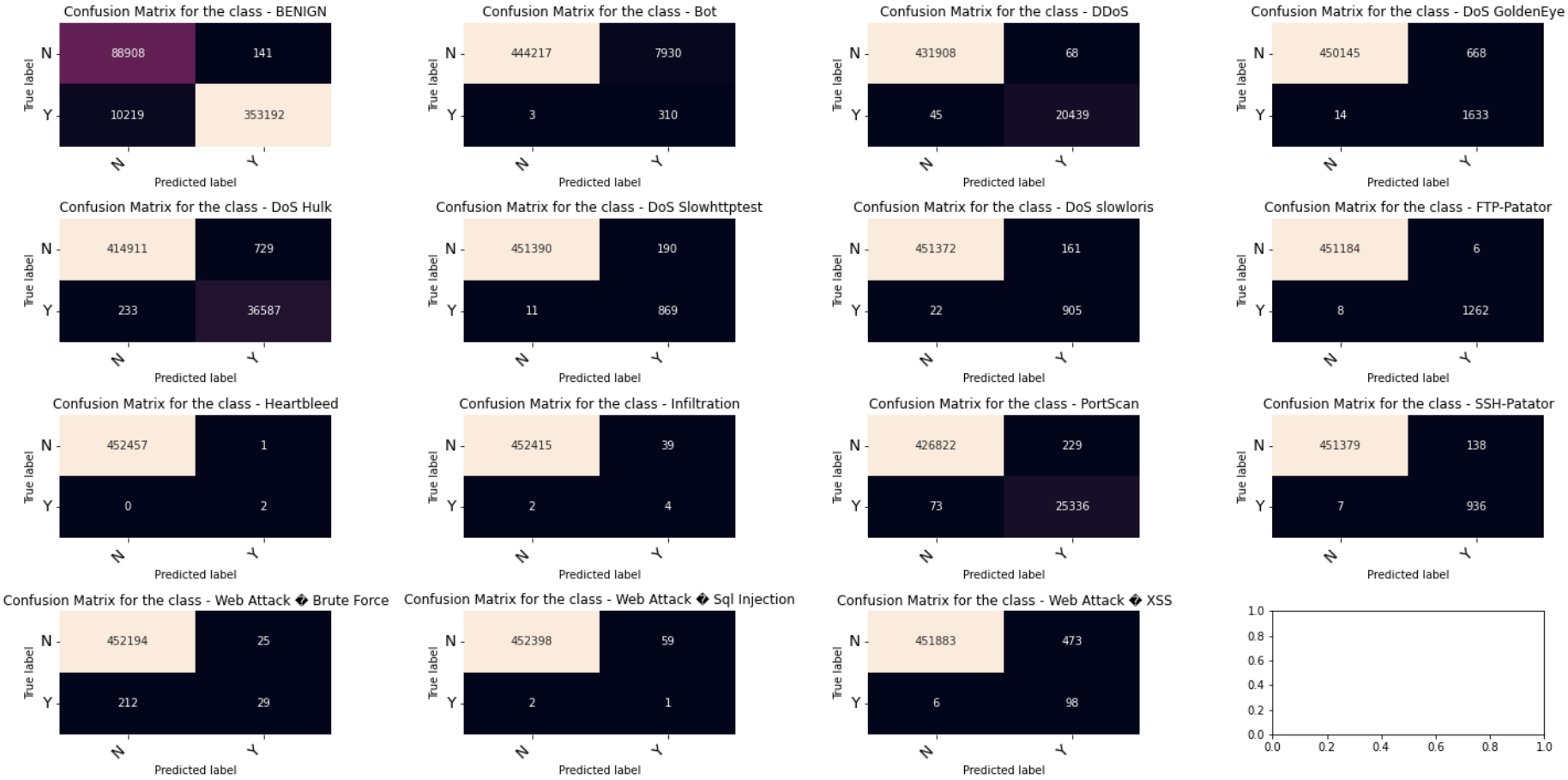

Figure 17: Confusion matrix generated for the first model

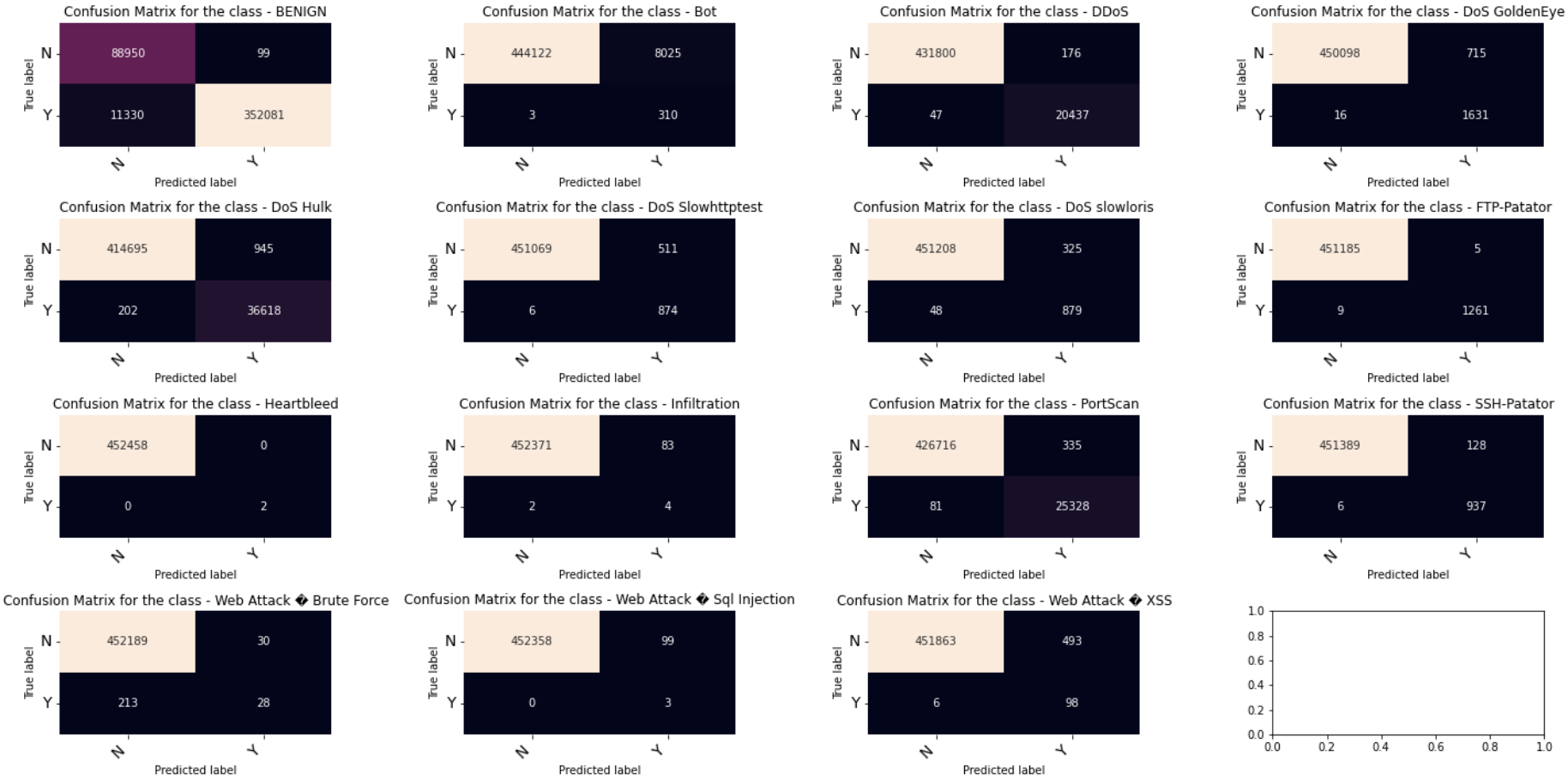

Figure 18: Confusion matrix generated for the second model

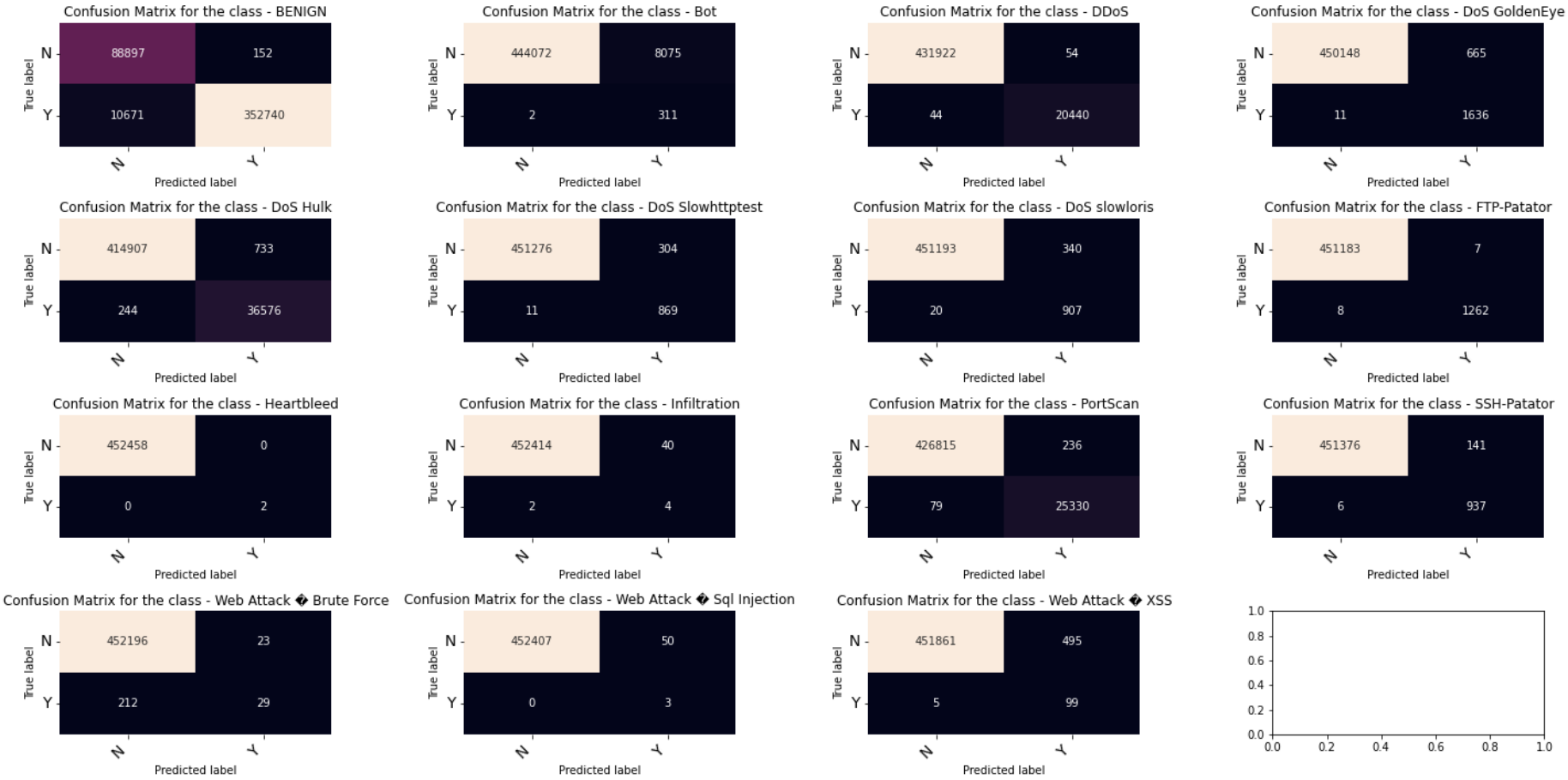

Figure 19: Confusion matrix generated for the fourth model

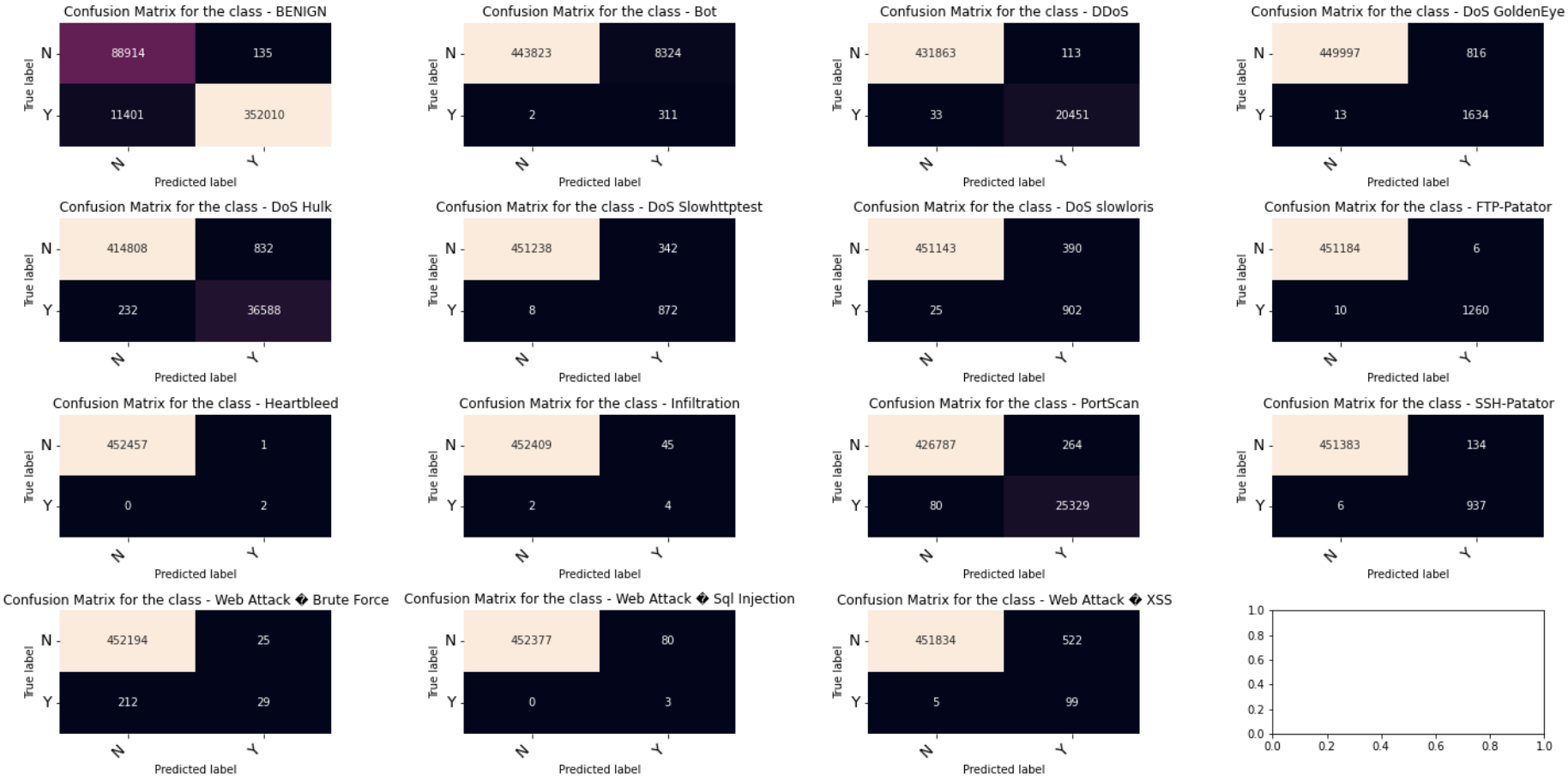

Figure 20: Confusion matrix generated for the fifth model

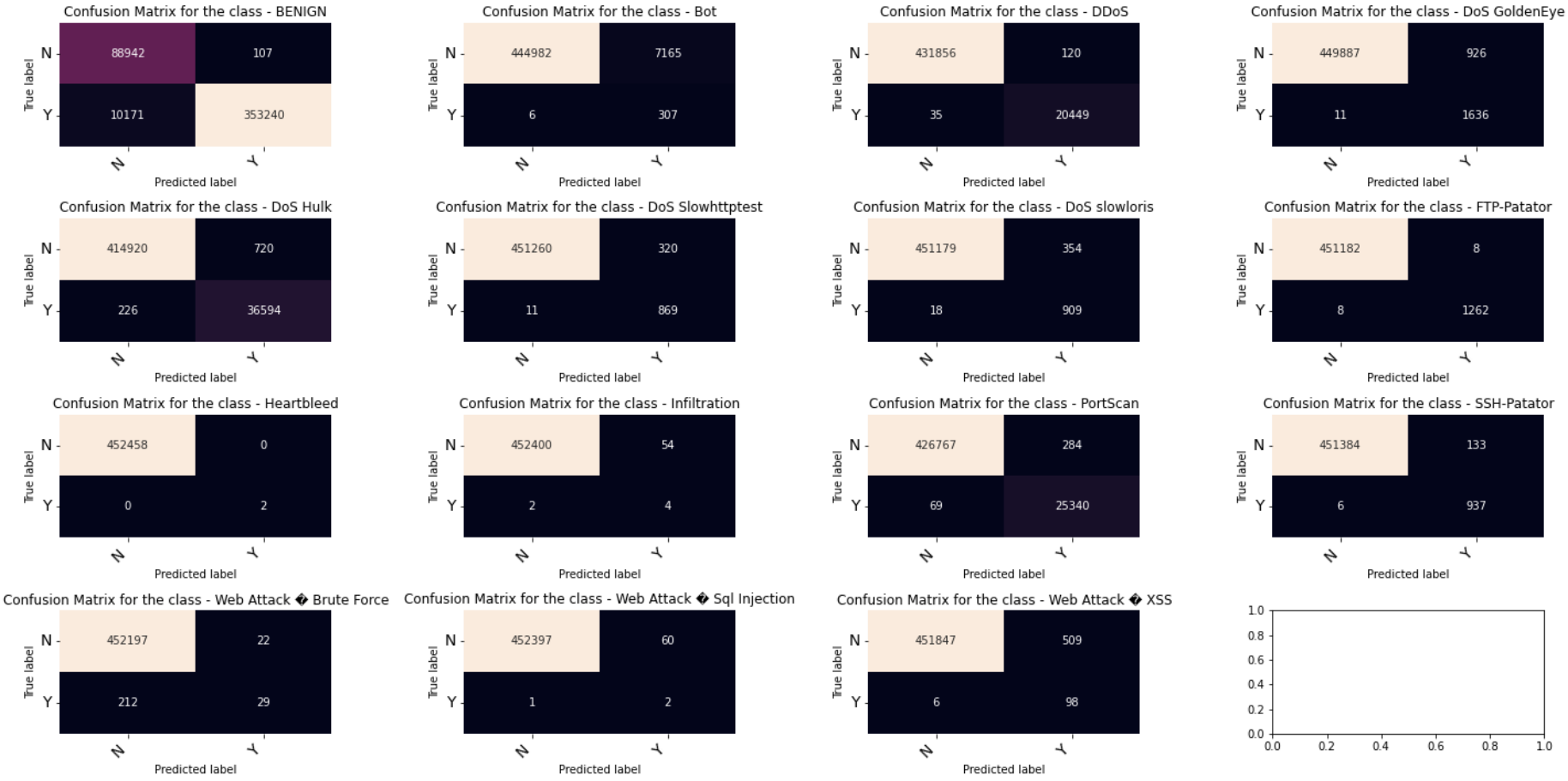

Figure 21: Confusion matrix generated for the reduced Initial architecture