DETECTION OF DDOS ATTACKS BASED ON DENSE NEURAL NETWORKS, AUTOENCODERS AND PEARSON CORRELATION COEFFICIENT

by

Junhong Li

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2020

Table of Contents

List of Tables

List of Figures

Abstract

Distributed Denial of Service (DDoS) is a set of frequent cyber attacks used against public servers. Because DDoS attacks can be launched remotely and reflected by legitimated users on networks, it is hard for victims to detect and prevent them. The objective of this thesis is to explore the detection of DDoS attacks, especially those that have arisen in recent years, by a combination of dense neural networks, autoencoders and Pearson Correlation Coefficient. Three different classification models are designed, trained and tested. In order to gain information about the most recent DDoS attack types, the CICDDoS2019 dataset is selected as the training and testing set. This dataset contains Microsoft SQL Server(MSSQL), Simple Service Discovery Protocol(SSDP), Network Time Protocol(NTP), Trivial File Transfer Protocol(TFTP), Domain Name System(DNS), Lightweight Directory Access Protocol(LDAP), Network Basic Input/Output System(NetBIOS), Simple Network Management Protocol(SNMP), SYN flood, User Datagram Protocol(UDP) flood and UDP-Lag. To imitate the real network environment, the data used in this thesis is raw PCAP files. CIC-FlowMeter, a packet analysis tool, will be used to convert the raw packets into features. Three different deep-learning models are proposed to be used in DDoS detection. The models consist of DNN, Auto Encoder and Pearson Correlation Coefficient, in which the autoencoder works as a feature compressor. The performance of each model on different types of attacks is compared. The thesis also set up a benchmark using traditional machine learning models. The proposed models outperform the traditional machine learning classification models. Furthermore, the F1-score of the proposed models is higher than other approaches.

List of Abbreviations Used

IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
MSSQL	Microsoft SQL
SSDP	Simple Service Discovery Protocol
NTP	Network Time Protocol
TFTP	Trivial File Transfer Protocol
DNS	Domain Name Server
LDAP	Lightweight Directory Access Protocol
NetBIOS	Network Basic Input/Output System
SNMP	Simple Network Management Protocol
UDP	User Datagram Protocol
DNN	Dense Neural Network
AE	Auto Encoder
PCC	Pearson Correlation Coefficient

Acknowledgements

Thanks to Professor Srini Sampalli's significant effort and help in the process of building this thesis. He is a professional, patient and warm-hearted man who makes all of this happen. I also want to thank Professor Nur Zincir-Heywood and Professor Malcolm Heywood to be my readers. I got much help from MyTechLab members, such as Robbie MacGregor, Reetam Taj, Amin Nemati and Darshana Upadhyay, during my experimentation phase. Last but not least, I want to thank my friends, Xuhui Liu, Zhonghui Liu, Yichao Liang and Yang Guo. They didn't leave me alone during the marathon and encouraged me when I was down.

Hope we all stay safe from COVID-19.

Chapter 1

Introduction

Distributed Denial of Service (DDoS) attacks are a class of frequent cyber attacks used against public servers. There are many different types of DDoS attacks that have been widely used in the past decades. Researchers from all over the world have explored many potential ways to detect, prevent and mitigate DDoS attacks. The techniques used to detect DDoS attacks can be mainly divided into two types: rule-based approaches and case-based approaches.

One of the well-known applications that use the rule-based approach is Snort, a packet filter tool developed by Roesh[1]. This tool can filter packets by analyzing many characteristics of each packet that flows on a server or host. For example, it can filter by source IP, destination IP, port number, protocols, packet size and content. Many researchers have enhanced this tool or integrated this tool into their design. Because Snort filters the packets using rules, the false alarm rate is low. In comparison with machine learning strategies, rule-based approaches do not need training, which makes them faster.

Machine learning approaches can be mainly classified into two areas: traditional machine learning and deep learning. Traditional machine learning techniques contain Decision Tree, Random Forest, Logistic Regression, Support Vector Machine and Naive Bayes. These traditional techniques do not have neural networks in the model. In deep learning, training is done by neural network models. There are many types of neural networks used in DDoS detection, such as convolutional neural network, recurrent neural network, dense neural network, autoencoder and hybrid neural network. In recent years, many researchers explored ways to utilize neural networks to detect DDoS attacks. The significant advantage of machine learning approaches is that they can give a higher detection rate than rule-based approaches, especially for unknown attack types[2].

Many researchers have found that rule-based approaches are fast and machine learning approaches can provide higher detection rate[3]–[5]. A couple of hybrid Network Intrusion Detection frameworks have been proposed. The most common approach is using signature-based tools such as Snort to filter the known attacks and feeding the filtered packets into machine learning models to do further detection. By doing this, the framework can easily filter the known attacks using Snort and leave the remaining traffic to machine learning models. Machine learning models focus on detecting the unfiltered attacks in the remaining traffic. In this type of hybrid framework, the signature-based part can relieve the load of machine learning part and detection time can be saved. However, the remaining traffic is determined by machine learning models later to ensure a high detection rate.

The Snort team provides a package that contains Snort software and rules, and the users can install Snort and download the rules from the Snort website[6]. After proper configuration, Snort can filter the live traffic or offline PCAP files. For researchers who have an intrusion dataset in PCAP format, the best practice is to use offline mode.

Machine learning models are the main focus of this thesis. The big challenge of using machine learning models is how to get a higher detection rate and keep the false alarm rate low. Traditional machine learning techniques such as Decision Tree and Random Forest can reach a moderate Detection Rate around 70%, which is shown in Section 3.3. In this thesis, a novel way of combing Dense Neural Networks, autoencoders and Pearson Correlation Coefficient is designed, trained and tested. Pearson Correlation Coefficient can calculate to what extent two features are correlated, and in experimentation, one of the features are dropped to avoid redundant training. Autoencoders work as feature compressor, the uncorrelated features are compressed into a lower dimension, and the major features are kept at the same time. Ideally, the hybrid model can save detection time and improve F1-scores.

Before training the machine learning models, an important step is extracting features from the raw PCAP file into readable CSV files. There are many popular DDoS datasets used by researchers, such as CAIDA UCSD[7] and CICDDoS2019[8]. However, Canadian Institute for Cybersecurity, which published CICDDoS2019, provides an analytical tool for researchers to extract features. The tool is called CICFlowMeter[9]. It can extract 84 features from raw PCAP files. In this thesis, the

CIC-FlowMeter is used to generate CSV files. The CICDDoS2019 dataset is used as training and testing data as well.

CICDDoS2019 is a DDoS attack dataset. It is extracted from CICIDS2018[10]. The dataset has two versions. The first version is CSV files, which contain 12 different DDoS attacks such as SNMP, TFTP and SYN. There are 84 features provided in the CSV files. Another version is raw PCAP files. The PCAP files are captured by Sharafaldin[8] and his colleagues in a two-day attack session. They launch different attacks at different time ranges. The CSV files already have labels, while the PCAP files need to be manually labelled. They also provide the attack schedule on their website[11]. The timestamp from the PCAP files can be used to determine the attack types.

In this thesis, some traditional learning methods are trained and tested as benchmarks. The hyper-parameters and training process are remained default by sklearn. Traditional learning models are trained on a sampled dataset extracted from CICDDoS2019 CSV files. Then, the PCAP files from CICDDoS2019 are converted to CSV files using CIC-FlowMeter. After that, three different deep learning approaches are introduced and tested. The first is a Dense Neural Network. The second is a hybrid neural network, consisting of an autoencoder and a DNN. In the model, the autoencoder works as a feature compressor. In the third model, there is also an autoencoder and a DNN. However, the difference between the third model and the second model is that before feeding the data into the model, the data features are selected by Pearson Correlation Coefficient[12]. These three models are trained and tested on CICDDoS2019 dataset, and the results are compared with the results of Sharafaldin et al.[8] and the benchmarks.

This chapter is followed by another four chapters: In chapter 2, different techniques used to detect DDoS attacks and other intrusions are introduced. The techniques include signature-based detection and machine learning approaches. In chapter 3, all the approaches used in this paper are introduced, including tools, algorithms, datasets, models and metrics. In chapter 4, all results generated using the methodologies from chapter 3 and analysis of the results are discussed in detail. In chapter 5, the conclusions derived from the whole procedure are discussed, and the potential future work is also discussed. At the end of this thesis, references are given.

Chapter 2

Background

As is always a hot topic in the area of network security, DDoS attack and intrusion detection techniques for such attacks are explored by some researchers[13]. The threat of network attacks has risen sharply in the past few decades[13]. DDoS attacks are a significant challenge that most big companies face. For Internet users, DDoS attacks aim to ultimately bring down the accessibility of different Internet-based services to legitimate users[14]. For companies, a DDoS attack uses many computers to launch a significant volume of requests to attack one or more victims[15], and the nature of distribution makes it more difficult to detect them[16]. However, many researchers proposed different approaches to detect, respond and mitigate DDoS attacks. The techniques used to detect the DDoS attacks can be divided into three types:(1) Signature-based or rule-based techniques to examine the headers or content of network packets.[17], [18]. (2) Case-based techniques such as machine learning or statistical models to classify the traffic by examining the headers or behaviour of the packets[19]. (3) Integration of multiple classifiers or techniques to build a hybrid classification model[20].

In this chapter, the three types of detection approaches will be introduced. Moreover, the prior work that has been done by other researchers will be discussed.

2.1 Signature-based Techniques

Signature-based methods supporting real-time intrusion detection have been established as an effective way to identify attacks. Rule-based processes and tools being fed data from a signature database can quickly identify a variety of attempts to gather information about open ports and available services to create buffer overflows to execute CGI attacks[1], [3]. These tools principally rely upon protocol analysis to accomplish network intrusion detection tasks but are capable of doing a more detailed analysis of raw packets to identify attacks of interest in the present such as DoS, brute force, and browser-based attacks.

The designers of network intrusion detection systems face the unique challenge of having to balance the need to efficiently detect intrusions in real-time against the need to be flexible in responding to attacks with unknown or novel characteristics. Frequently, they employ methods that leverage efficient, signature-based approaches to the detection of misuse and conventional attacks, leaving the identification of network traffic patterns not easily categorized by these tools to classification algorithms[21]–[24].

2.1.1 Prior Works

Ficco et al.[25] approach uses a set of components hierarchically organized to accumulate streams of information at different levels (hypervisor, infrastructure, platform and application). The consolidated data is correlated and used to distinguish whether monitored activities are due to malicious behaviours. This strategy assists in identifying compromised virtual components and distributed attacks.

Considering numerous vulnerabilities in networks, Chiba et al.[23] have categorized them into two types of threads; Insider attacks and outsider attacks. An integrated and cooperative NIDS framework is deployed at the frontend on the controller and back end on every processing server to identify both classifications of intrusions. All the NIDS placed on the servers work cooperatively to update their signature database by receiving alerts stored in the central log. This makes it possible for correlation in the central registry and hence, detection of unknown attack is possible. The cognitive module in this design uses Snort to classify an attack by detecting intrusions based on the misuse detection database. Snort tries to determine the nature of the attack and transmits the information to the Alert System, and the packet will be refused. This technique allows the researchers to easily update the misuse database without any alteration of the existing rules.

A protocol-based network intrusion detection system is designed by Patil et al.[4] to detect DoS/DDoS attacks in networks. In this system, Incoming packets are distributed according to the protocol and queued for additional processing. Relevant features will be extracted, and protocol-specific classifiers are applied on each packet to generate alerts and thus update the attack signature database. This approach focuses on detecting DoS/DDoS attack types. The main features that the classifiers focus on are the types of protocols.

Singh et al.[5] have designed a framework using Snort as a rule-based attack detection system and have installed NIDS in the virtual bridge to monitor network traffic and to form low-level intrusion alerts. The correlation section in this design converts these low-level intrusion alerts to high-level intrusions.

Patil et al.[4] have proposed a framework to detect intrusion using Snort, a signature-based tool. The overall architecture adapts a module called correlation unit. The correlation unit is a component that can be deployed over all networks so that all hosts can share the signatures in real-time. Snort itself has a detection engine that can match the packet with rules for any correlation. The Snort signature will be generated only when the major alert factor reaches a pre-set threshold.

Kumar et al.[26] described a signature-based IDS using Snort. In this paper, the rules used in the detection engine is generated by known intrusion signature system. A rule is divided into rule headers and rule options. Cisco has more than 2500 rule bases in its database. Moreover, users can also change the rules based on their needs.

Altwaijry et al.[27] has proposed an automatic tool, WHASG, to generate Snort signatures using a honeypot. In this design, there is a component called rules module. It receives all the information required from the main module. Then it will generate the signature. In this paper, Altwaijry et al. designed a data structure called signature container array. An attack packet will be fed into the module, and a Snort signature will be generated automatically.

There are also some online tools that can help to generate Snort signatures, such as Snorpy. Snorpy is useful when researchers want to create one rule or two. However, it is infeasible to use it to generate signatures in an automatic system.

2.1.2 Snort and its Rules

Snort is a lightweight signature-based network intrusion detection system created by Martin Roesch[1]. It can rapidly filter the TCP/IP traffic based on headers and options.

The Snort rules typically have 24 option fields[28].

- content
- content list
- flags
- ttl
- itype
- icode
- fragbits
- id
- ack
- seq
- logto
- dsize
- offset
- depth
- nocase
- msg
- tos
- ipoption
- icmp id
- icmp seq
- session
- rpc
- resp
- react

Here is an example of Snort rules:

$$alert\ tcp\ any\ any -> \text{ 192.168.1.0/24 111 } (content : \text{"}|00\ 01\ 86\ a5|\text{"}; msg : \text{"}mountd\ access\text{"};)$$

This rule alerts all traffic that has all the following characteristics:

1. it is a TCP traffic
2. could be from any IP and any port
3. the direction is "in"
4. it has the same content as 00 01 86 $a5$

If Snort captures traffic like this, it will print the message "mountd access" to the log file or console.

The default DDoS rules can be found in Caswell et al.'s book[29]. Before training the neural networks, these rules will be added into Snort. The rules will be used as default, but the users can also apply the rules defined by themselves.

Here is an example of DDoS detection rules:

$$alert\ udp\ \$EXTERNAL\_NET\ any -> \$HOME\_NET\ 10498$$

$$(msg : \text{"}...\text{"};\ content : \text{"}pong\text{"};\ classtype : attempted - dos;\ sid : 246;\ rev : 2;)$$

There are 32 more default rules for DDoS detection[28] by default.

2.2 Traditional Machine Learning Techniques

2.2.1 Naive Bayes Classifier

Naive Bayes classifier is a simple probabilistic classifier[30]. In supervised learning, Naive Bayes classifier can be trained rapidly without an enormous amount of computational resources. Modi et al.[31] proposed a NIDS that integrates Naive Bayes classifier and Snort. In this framework, Snort signature-based detection filters the captured packets. The captured packets will be divided into two sets: intrusion packets and non-intrusion packets. The intrusion packets will be logged and denied by the system. Meanwhile, the non-intrusion packets will be preprocessed and fed into the anomaly detection module. The anomaly detection module employs the Naive Bayes classifier to further classify the non-intrusion packets into normal and intrusion packets. Once the packets are classified as intrusions, they will be logged and denied. Only when the packets are labelled as normal can they be allowed to go to the system. The F-1 score tested on the KDD'99 dataset varies from 91.25% to 98.01%. Qin et al.[32] designed a similar framework as Modi et al [1] did. Qin's performance test on the DARPA'99 dataset got a detection rate of 97%, a prediction rate of 97% and a false alarm rate of 0.1%.

2.2.2 Support Vector Machine

Jing et al.[33] have proposed Support Vector Machine(SVM) with a new scaling method in 2019. The necessary steps are: (1) divide the dataset into the training set and testing set; (2) Preprocessing the data (both training set and testing set) with scaling method; (3) Train the SVM model with the training set; (4) Test the model with the testing set; (5) Record the classification result. The paper uses UNSW-NB15 dataset. The dataset has nine distinct attack labels and one normal label. There are two successive parts in the experimentation. One is to train and test the model on binary labels while the other on multi labels. The detection rate of anomaly data is 97%, and the false-positive rate is 27.5%. The detection rate of normal data is 72.5%, and the false-positive rate is 3%. In the multi-class test, the accuracies of ten classes vary from 86.7% to 99.9%, the detection rate ranges from 0% to 95.8%, and the false-positive rate ranges from 0% to 11.85%.

2.2.3 Decision Tree

Hong et al.[34] propose a lightweight NIDS that uses Decision Tree. In this proposal, they use Chi-Square and Enhanced C4.5 in the detection model. The authors down-sample the KDD'99 dataset to 10% of the original size. The Chi-Square module is used to select proper features. A subset of the data with less normal and more attack traffic will be fed into the Chi-Square selector. The selector can automatically determine a proper χ^2 threshold, which can keep the original data fidelity. The C4.5 module will use the features selected by the Chi-Square module to train a multi-class decision tree. The C4.5 model uses information gain as the criteria to determine the split points. Then the authors build a balanced subset and use it to train the Enhanced C4.5 model.

The model is evaluated using the rest of the KDD'99 dataset. The experiment result shows that the True Positive Rate varies from 50.01% (U2R) to 99.99% (Normal). Moreover, the False Alarm Rate varies from 1.48% (DOS) to 28.32%(U2R).

2.2.4 Random Forest

Zhang et al.[35] propose a hybrid detection system using random forest. In their proposal, they employ both misuse detection and anomaly detection. Misuse detection refers that the system alerts when it captures attack traffic. Otherwise, the system will label the traffic as benign traffic and let it in. In this case, uncertain traffic will not be handled. Using only this method will decrease the Detection Rate. Anomaly detection means the opposite way. It will only let the benign traffic in and alert all other traffic. Using only this technique will increase the False Alarm Rate.

Zhang et al. use Random Forest in both misuse and anomaly detection module. During the training of the misuse module, they over-sample the majority and down-sample the minority to avoid data imbalance problems. The training data are well labelled. In the anomaly detection module, they also employ the Random Forest technique. In this procedure, they use unsupervised learning to train the model. They train the model using only benign traffic such as HTTP, FTP and telnet. The Random Forest then will detect the outliers or unusual behaviours. The model mainly detects two types of outliers. One is traffic that significantly deviates from other services of the same type. The other one is traffic that behaves like another type of service. For example, if telnet traffic is clustered into HTTP service, the traffic will be determined as an attack.

Combining the two detection modules, the authors achieved 94,7% of the overall detection rate and 2% of the overall false alarm rate.

Anomaly detection component by Patil et al.[21] identifies and classifies an attack by analyzing the applied network traffic using the Random Forest classifier algorithm. They have performed comparative analysis using several algorithms to detect anomalous behaviour and classify it as an attack. The researchers have worked on two different datasets, i.e., the UNSW-NB15 dataset and the CICIDS-2017 dataset comparing them with false-positive rates and detection accuracy as a measurement. The results indicate that the Random Forest classifier delivers exceptionally higher accuracy and has the least false-positive rate. Random Forest classifier algorithm also helps and performs better in achieving real-time validation, detecting a variety of attacks, fast detection, high handling of network traffic.

Where Patil et al.[21] used a pre-trained Random Forest classifier to accomplish anomaly detection (making the first attempt to identify malicious activity in unknown network traffic patterns), some researchers have proposed adopting an ensemble approach using feature selection, similar to that outlined by Zhou et al.[36]. A hybrid/ensemble technique proposed by Moustafa et al.[2] could yield better results when developing machine learning models for detecting new attacks. The performance of traditional classifiers are assessed in Chapter4 when trained on modern datasets such as CSE-CIC-IDS2018[10], and found that Random Forest-based classifiers (particularly those using boosting, or better still optimized gradient boosting), appeared to out-perform naive approaches or classification via simple Decision Tree. These results, coupled with the arguments and experimental validation provided by Zhou et al.[36], have informed the researchers to use a pre-trained classifier, built using their preferred approach of combining feature selection with an ensemble classifier based on a combination of C4.5 and Random Forest, for anomaly detection. One possible drawback of this strategy is that Random Forest-based approaches can fail to account for environmental factors, and so some method of correcting possible false positives that accounts for this is highly desirable.

Patil et al.[4] proposed a Protocol specific Multi-threaded Network Intrusion Detection System (PM-NIDS). It aims at detecting DoS and DDoS attacks in the cloud system. It works by employing different classifiers, i.e. random forest algorithm, decision tree algorithm and OneR classifier, based on the protocol of the incoming packet. Experiments and results prove that the proposed design delivers high accuracy and low false positives, however detecting a variety of attacks and real-time validation is not fulfilled.

2.3 Deep Learning Techniques

2.3.1 Dense Neural Network

Al-Maksousy et al.[37] proposed a simple system that integrates Dense Neural Network (DNN). The brief steps of the system are: (1) Use a data sniffer (e.g. WireShark) to record the data flowing over the client network. The data sniffer will generate PCAP files that contain all the information of packets. (2) The PCAP files will be forwarded to tcptrace, a utility that can extract over 90 meaningful high-dimensional data. The results will be saved in CSV files. (3) The CSV data will be fed into the first DNN, which is a binary classifier. This DNN will only determine the packets are normal or suspicious. If the packets are labelled as suspicious, they will be directed to the alarm module as well as the second DNN (4) The second DNN is a multi-class classifier. This DNN is trained on malware datasets. The purpose of the second DNN is to label the packets with known malware types. The author tested the system on the KDD'99 dataset. The testing accuracy of suspicious data is 99.7%. However, the testing accuracy of benign data is 99.9%.

2.3.2 Convolutional Neural Network

Convolutional Neural Network is a type of neural network designed for tasks related to images. It has a special layer called convolutional layer. Convolution layer can extract local features from images and pass them to the next layer. Normally, CNN is used in some tasks that need to process images. However, some researchers found CNN performs well in Network Intrusion Detection [38]–[42].

Vinayakumar et al.[38] use a six-layer model. The first three layers are CNN, and the last three layers are CNN with Recurrent Neural Network, Long-Short-Time-Memory and Gated Recurrent Unit. The multi-class test accuracy on KDD'99 minimal features varies from 92.3% to 98.7%. Li et al.[39] proposed another algorithm using CNN to detect intrusions. Their work employs multi-scale CNN, Inception models as well as batch normalization. The Detection Rate on KDD'99 reaches 93.22%, and the False Alarm Rate reaches 2.18%. Khan et al.[40] proposed a simple CNN model that consists of an input layer, two convolutional layers, two max-pooling layers and a fully connected output layer. The overall accuracy tested on KDD'99 reaches 99.23% after training for 800 epochs. Xiao et al.[41] apply feature reduction before feeding the data into CNN. They compared feature reduction performance using Principle Component Analysis and Auto Encoder. After that, the authors implemented a CNN with Batch Normalization and dropout layers. The results show that Auto Encoder got a higher Detection Rate and lower False Alarm Rate when testing on the KDD'99 dataset. The overall multi-class accuracy reaches 94.0%. Yang et al.[42] designed an Improved Convolutional Neural Network. The experiment results on KDD'99 show that AUC reaches 0.9392.

2.3.3 Recurrent Neural Network

RNN is s type of neural network that has a special memory mechanism such as gates or Gated Recurrent Unit(GRU). The gates or GRUs can remember the features that were earlier fed into the RNN for a period of time. After a certain time, the gates or GRUs will forget the earlier features. This mechanism is commonly used when time-series needs to be established. A typical use case of RNN is Natural Language Processing, in which time-series plays an important role. In the area of intrusion detection, some researchers built well-performed models using RNN[43]–[47].

Yin et al.[43] use RNN with forward propagation and weights updates (backpropagation). The authors applied grid search to the parameters. The best test accuracy on KDD'99 is 81.29%. Qureshi et al.[44] rebalanced the KDD'99 dataset before training and testing. The proportion of abnormal data in the training set is rebalanced to 46.5%. The authors have referred to Bajaj et al.[48]'s work about feature reduction and dropped some features in the preprocessing. The highest accuracy they got on the KDD'99 test set is 94.50%. Althubiti et al.[45] use Long-Short-Term-Memory

RNN and ADAM optimizer. LSTM is a special type of RNN which is explicitly designed for tasks that need short-term memory. After reducing the features to five-dimension, the test accuracy on CSIC 2010 HTTP dataset reaches 99.57%. Meng et al.[46] took a further step and integrate kernel PCA and LSTM. Kernel PCA is a type of dimension reduction technique. The main difference between PCA and Kernel PCA is that Kernel PCA generalize PCA from linear to nonlinear dimension reduction. The overall Detection Rate tested on KDD'99 is 99.46%, while the False Alarm Rate is 4.86%. Le et al.[47] compared several gradient descent optimizers with LSTM. Gradient Descent is a classic optimizer used in deep learning. However, there are many variations of Gradient Descent optimizers. The authors also compared Adagrad, Adadelta, RMSprop, Adam, Adamax and Nadam. The Nadam optimizer outperforms other optimizers. The Detection Rate tested on KDD'99 reaches 98.95%, and the False Alarm Rate gets 9.98%.

2.3.4 AutoEncoder

Rezvy et al [49] chose AutoEncoder as the pre-train model to re-configure features and then use DNN to classify the traffic. AutoEncoder works as a feature re-constructor in this system. The system works as below: (1) Pre-process the dataset. The author used down-sampling and over-sampling techniques to avoid the imbalance of data. The dataset is divided into 80% for training and validation and 20% for testing. (2) Pre-Train the AutoEncoder with the training set. The categorical columns are encoded with one-hot. Before feeding the data into the model, the data is normalized using a min-max scaler. The AutoEncoder compressed the feature dimension from 122 to 61, and re-construct it back to 122. (3) The re-constructed 122 features are fed into a three-layer DNN. The DNN is a five-class classifier. It labels the packets with the types of attacks. The author tested the model with the NSL-KDD'99 five-class dataset. The accuracy of the five classes varies from 89.2% to 99.9%. The overall accuracy of the attacks is 99.3%.

2.4 Hybrid Techniques

2.4.1 Rule Based Techniques

Signature-based approaches and rule-based processes (e.g. Snort[1], [3]) are employed to generate alerts at a local level to improve efficiency, handing anomaly detection and the classification of unknown traffic patterns to a pre-trained machine learning model. Processes benefiting from centralization or requiring more computational resources are further passed to a cluster where simple correlations and more involved machine learning tasks can be accomplished to efficiently reduce the likelihood of errors in detection, limiting false positives. Detections coming from the central correlation unit, or generated by either of the classifiers, result in updates to a signature database feeding back into the rule-based tools and improving future outcomes.

Where past researchers have described complex algorithms for estimating system load and scheduling tasks, with the aim of supporting truly distributed systems[22], a more streamlined approach is proposed in this paper. Local, low-power devices associated with routing, logging and other SIEM functions can reliably handle tasks associated with packet capture (excluding IDS-related traffic), filtering/reducing raw data (capturing required header fields), feature extraction; and basic anomaly detection. More computationally costly, resource-intensive tasks cannot be efficiently managed locally. In Chiba et al.'s[23] approach, a local device will act as a controller, sending network flows and other data to a better-provisioned cluster when measures of certainty are low (and additional verifications are needed to reduce errors related to false positives) or computationally costly tasks are anticipated (e.g. when data with unfamiliar characteristics are to be processed and classified). The algorithms driving control and switching can be tuned based upon feedback data generated by the cluster, and a DB supporting rule- and signature-based detection managed by the same.

The processes of their proposed IDS can be grouped conceptually into three related modules:

Cognitive Module: Classic rule-based and signature-based detection strategies are a part of what might be thought of as the Cognitive Module.

Classification Module: The module for classification of potentially anomalous behaviour that relies on simple ML classifiers at the local level, and more complex ML and DL processes on the cluster.

Adaptive Module: The relationship between the local device/controller and the cluster results in a flexible, adaptive solution. The cluster supports correlation and error correction (reducing false-positive alerts originating with the Cognitive and Classification modules when measures of certainty are inadequate) and populates/updates a DB to improve future responses. Switching and scheduling of tasks can result in more efficient use of resources and more efficient classification of unknown/unusual behaviours or patterns.

In this design, network traffic will be captured and analyzed using Tcpdump, a command-line utility tool. It is a resourceful and robust tool that incorporates many options and filters. This tool enables us to capture any network traffic through all active interfaces and can even store it into a file with a ".pcap" extension. Snort, an Intrusion detection system, is used to detect any known attack signatures and anomalies present in the captured traffic. All the captured raw data is sent to Snort without any removal of payload data, which makes it likely for Snort even to look at the payload and examine any indications of malicious content. Snort makes it possible to detect known attacks and signatures as an efficient rule-based approach. Besides that, raw data is sent to Tshark in order to reduce the captured traffic by filtering the data stream to obtain appropriate packet headers. This reduced and filtered dataset is the input for CSEFlowMeter, where required feature extraction takes place. The features are extracted effectively adopting the proposed scheme and finally forwarded to the anomaly detection stage where appropriate machine learning algorithm(s) are employed to build a model and discover any additional information or attack signatures.

The second component is the phase where detection rules for Snort are updated. When a new exploit or an attack becomes identified through any of the adopted classifiers, those particular instance information gets appended to the existing rule base. It enables immediate detection of those particular attack instances subsequently.

Correlation and Error Correction segment is also responsible for identifying resource-intensive machine learning tasks that require much more computational power. This identification helps in switching and reassigning the job to be executed in a much more robust environment using remote resources such as an external cloud server.

The detected anomaly traffic will be handled by the Alert and Mitigation module. After this, the newly detected traffic will be added into Snort and knowledge bases. Moreover, the knowledge bases are deployed on the cloud, so once a new anomaly is detected, all IDS on the cloud will get an immediate update.

Haddad et al.[24] designed a collaborative framework for intrusion detection in cloud computing using Snort as a signature-based detection and Support Vector Machine algorithm to identify anomalous attack patterns. This system was primarily designed in a collaborative way to detect distributed attack patterns and defend against them. The collaborative system additionally helps to keep updating the knowledge base from time to time.

2.4.2 NN-based Hybrid Technique

When attempting to address the potential of false-positive detections from either rule-based processes or the local anomaly detection unit, it is good to pass traffic flows off to a more resource-intensive model using a hybrid Convolutional Neural Network (CNN and DNN). This will significantly increase the accuracy of detection and reduce the likelihood of false positives. Ma et al.[50] introduced a new feature called environmental features. It can analyze the active flows in the flow-sliding window. Combining the environmental features and general statistical features can give us a new perspective to dig more information from the raw flow data. Considering environmental features can also make it possible to detect a series of attacks that have certain behaviour routines. It also helps with analyzing the emergence of flooding attacks. The model also uses CNN, which can learn sequential characteristics of data very quickly by convolutional layers and pooling layers. The experiment results also show that the hybrid NN has excellent performance on multi-type classification.

In Ma et al.'s proposed NIDS, the classification module employs Neural Networks. The NIDS is a hybrid architecture that consists of four parts. The anomaly detection is divided into two parts. The first part is Snort, which is a signature-based intrusion detection system. Snort can detect known intrusion traffic, but it does not perform well on unknown or latest intrusions. To tackle this problem, they introduced a hybrid neural network to do further classification.

The hybrid neural network uses three novel feature sets derived from raw traffic data. Namely, the sequence packet features general statistical features and environmental features of the traffic. Unlike previous IDS, this hybrid neural network can make use of comprehensive features from traffic data. The network not only focuses on single flows from outside but also monitoring sequential flows between two IP addresses. The network introduced a sliding window to detect the correlation in time series between two IP addresses. Moreover, the length of the sliding window can be modified with corresponding requirements.

Chiba et al.[51] use Snort as an Intrusion detection system for misuse detection and the Optimized Back-Propagation Neural network (BPN) algorithm for anomaly detection. The back-propagation algorithm has some drawbacks such as slow detection speed, less detection accuracy and slow convergence speed. However, an optimization module is used along with the BPN to increase the detection rate, maintain high accuracy, achieve low false positives and low false negatives. The authors have proposed a cooperative design. However, no performance outcomes or results are presented in the paper.

Distributed intrusion detection systems over heterogeneous network architectures by [22] applied the Local Outlier Factor (LOF) algorithm from Chandola et al. This algorithm yields promising results, and it is more efficient when applied in data that has regions of varying densities such as network traffic. However, this method is computationally infeasible if there are a large number of data points that require other techniques such as sampling to reduce complexity [52].

Chapter 3

Methodology

In this chapter, the basic procedures of the proposed framework will be introduced.

The CICDDoS2019 dataset[8] will be used in this thesis. The dataset will be explained and discussed in detail in Section 3.2. The dataset has two versions of data. One is the raw packets data (PCAP files) captured from two one-day attack plans. The other is the features derived from the raw data using CIC-FlowMeter[9], [53]. This version is in the format of CSV. It contains manually labelled attack types as well.

Even though the CSV files already have the labels, and it is more convenient to use it as the input files in machine learning, the files were examined in detail, and two problems showed up. The first problem is that some data is missing or infinity, which is not applicable in machine learning models. The second problem is that some columns in the dataset have a few significant figures. This could cause a problem because the precision of the float numbers is low. This problem can be fixed by regenerating the CSV files directly from the raw packets using CIC-FlowMeter.

After regenerating the dataset by CIC-FlowMeter, the next step is training and testing the traditional machine learning models and comparing the results. The purpose of this step is to understand how difficult to classify the attack types of the dataset and set up a benchmark for further experiments. Therefore, four traditional machine learning models will be trained and tested on a proportionally sampled sub-dataset. The four models are (1) Decision tree (2) Random forest (3) Multinomial logistic regression (4) Naive Bayes.

The exploration of traditional machine learning models will be followed by deep learning models, which are the proposed approaches of this thesis. Because three different deep learning approaches are introduced and used in this thesis, and two of which would have a feature reduction or compression module, the feature reduction or compression module will be introduced before training deep learning models. In

Section 3.4, three different feature reduction or compression techniques will be tested and compared. The three techniques are (1) Principal Component Analysis, (2) ANOVA, (3) Autoencoder. The technique that has the best results among all will be used as the proposed feature reduction module.

Once the feature reduction technique is settled, the next step is training deep learning models. In Section 3.6, three deep learning approaches are designed, trained and tested. The three models follow the evolutional style, from the simplest Dense Neural Network to a hybrid neural network with statistical feature selection and feature compressing. The three models are (1) DNN model (2) DNN models with AutoEncoder as the feature compression module (3) The same deep model as (2), but some columns in the input features will be dropped according to the Pearson Correlation Coefficient[12]. The three models will be trained and tested on the regenerated CSV files from the CICDDoS2019 dataset. The results will be compared with traditional machine learning models and the results of the CICDDoS2019 original paper[8].

Furthermore, in the last Section (3.7), the metrics used to evaluate the models will be introduced in detail.

3.1 CIC-FlowMeter

NIDS is designed for collecting, analyzing, classifying and responding to traffic that comes from outside networks. The first step for NIDS is capturing network traffic. Packet capture is accomplished with tcpdump or Wireshark. From which, PCAP files can be generated. Moreover, the PCAP files will be analyzed in the next steps.

CIC-DDoS-2019 dataset[8] has two versions of data. One is CSV files, which are already labelled by University of New Brunswick. The other one is raw PCAP files that were captured in their two-day experiment.

For machine learning techniques, the CSV files can be directly used in training and testing only if the dataset is clean and complete. If the dataset is not clean or complete, proper data cleaning work is necessary. After inspecting the CSV files provided in CIC-DDoS-2019, the data is not clean enough for applying machine learning. There are mainly problems with the CSV files: (1) the classes are highly imbalanced, especially for the benign traffic (2) The significant figures are too few in some columns, which make the data uninterpretable. For example, the column "Fwd IAT Mean" contains 78.3% value of 0.0. This problem causes an information loss which could influence the predicting accuracy. (3) The feature names do not equal to the documentation.

The third problem is not a big issue for analysis. However, the first problem could cause high accuracy and low F-score. The second problem hides useful information from the researchers, which could make the model underfitting.

In order to solve the problems, one of the best feasible ways is to recreate the CSV dataset by using CIC-FlowMeter.

CIC-FlowMeter[9], [53] is a network traffic flow generator and analyser published by Canadian Institute for Cybersecurity. It can generate 84 features from raw PCAP files.

In this paper, all CSV files are regenerated from CIC-FlowMeter. The PCAP files used in this procedure are CIC-DDoS-2019, which will be introduced in Section 3.2.

3.2 Dataset

The dataset used in this paper is CICDDoS 2019[8]. This dataset contains only DDoS attacks and benign traffic. The creators describe it as a realistic cyber defence dataset. This dataset is a joint project of the Canadian Communications Security Establishment (CSE) and The Canadian Institute for Cybersecurity (CIC). CIC-DDoS 2019 is a new, high quality, synthetic dataset, providing both network traffic and log data. In order to generate the dataset, networks of target machines were instantiated via AWS and automated using CIC-BenignGenerator[10]. These machines represented five departments of a target organization, with 420 clients and 30 servers in total. Target machines were instrumented and then systematically attacked using an attack infrastructure of 50 machines, with log data and network traffic data captured and categorized. There is evidence of considerable effort on the part of the dataset's creators to enhance external validity through their choice of architecture, the design of both target and attack networks, and their experimental design.

Benign: In this dataset, there are also benign data. The authors used CIC-BenignGenerator[10] to imitate benign background traffic based on the profiles of abstract behaviour of 25 users. Benign traffic is based on HTTP, HTTPS, FTP, SSH, and email protocols.

Attacks: Two genres of DDoS attacks are captured in this dataset. The first is Reflection-based DDoS, including MSSQL, SSDP, NTP, TFTP, DNS, LDAP, NetBIOS and SNMP. In this type of attack, the real attackers can hide behind the legitimated clients and utilize them in an attack. It makes the victims more challenging to differentiate the users and attackers only by the source. These attacks are based on TCP(MSSQL and SSDP), UDP(NTP and TFTP) or both(DNS, LDAP, NETBIOS and SNMP). The second is Exploitation-based attacks, including SYN flood, UDP flood and UDP-Lag. This type of attack will spoof the source IP address and sent a large number of packets to the victim server. This will cause the victim resources exhausted. The explanations of all types of attacks can be found below.

1. MSSQL: MSSQL stands for Microsoft SQL. The attackers pretend to be the Microsoft SQL Server and send responses to the victims. It abuses Microsoft SQL Server Resolution Protocol and spoofs the MS SQL server's IP address[54].

2. SSDP: Simple Service Discovery Protocol (SSDP). It is a type of reflection DDoS attacks. SSDP DDoS attack sends an amplified traffic stream to the victim's server. It exploits the Universal Plug and Play (UPnP) network protocols. This attack can overwhelm the target's infrastructure and take their web resource offline[55], [56].

3. NTP: Network Time Protocol (NTP) is a type of protocol used to synchronize the clocks through the Internet. NTP amplification uses NTP servers to overwhelm the target with UDP traffic. The attacker typically sends requests to the NTP servers with spoofing the IP address, which belongs to the victims[57].

4. TFTP: TFTP attack is a type of amplification DDoS attack based on the Trivial File Transfer Protocol (TFTP). The amplification factor can reach up to 60. A TFTP server is normally used to store device images and configuration files. TFTP is a stateless protocol and does not have authentication methods, which makes it easier to launch and harder to detect[58].

5. DNS: DNS attack is a type of amplification DDoS attack exploits Domain Name Servers and exhausts the bandwidth of the victims[59]. This attack can overwhelm the victims and make them inaccessible. DNS attacks can be easily launched by bots[60].

6. LDAP: LDAP stands for Lightweight Directory Access Protocol. This is a type of amplification DDOS attack, in which the amplification factor can be up to 55[61]. LDAP is mainly used in corporate networks, this is the reason why it is widely used to attack corporate networks[62].

7. NetBIOS: NetBIOS stands for Network Basic Input/Output System. Its amplification factor is 3.8[61]. This attack is based on UDP.

8. SNMP: SNMP stands for Simple Network Management Protocol. SNMP is a network management protocol used to configure and collect information from network devices. During an SNMP reflection attack, the attackers send a large amount of SNMP queries using a spoofing IP address that belongs to the victim. After that, the SNMP servers will reply to the victim's IP address[63].

9. SYN: A SYN flood attack is also called a half-open attack. It aims to consume all server resources and make a server unavailable. The attacker constantly sends a connection request (SYN) to the victim server but does not reply to the ACK from the victim server. The TCP connection will keep half-open for some time, and all ports become unavailable[64].

10. UDP: A UDP flood uses User Datagram Protocol (UDP) to launch attacks. The attacker sends a large amount of UDP packets to the victim server's port with a spoofing IP address. If no program is running on that port, the victim server will send an ICMP packet to remind the sender. However, the source IP is unreachable, and the victim server will never get a response. By doing this, the victim server ports will be exhausted[61].

11. UDP-Lag: The UDP-Lag attack is a kind of attack that disrupts the connection between the client and the server. This attack is mostly used in online gaming. This attack can make the UDP connection slower than normal. This could be a serious problem when the server requires a short time lag[8].

Attacks	Attack Time	PCAP Files	Num of Files
NTP	10:35 - 10:45	01 - 188	188
DNS	10:52 - 11:05	192 - 196	5
LDAP	11:22 - 11:32	379 - 443	65
MSSQL	11:36 - 11:45	444 - 470	27
NetBIOS	11:50 - 12:00	475 - 486	12
SNMP	12:12 - 12:23	487 - 571	85
SSDP	12:27 - 12:37	572 - 592	21
UDP	12:45 - 13:09	593 - 617	25
UDP-Lag	13:11 - 13:15	Not found	0
WebDDoS	13:18 - 13:29	Not found	0
SYN	13:29 - 13:34	618 - 620	3
TFTP	13:35 - 17:15	621 - 818	198
Benign	other	189 - 191, 197 - 378, 471 - 474	189

Table 3.1: Second Day

In this dataset, the authors provided two types of data for researchers. One is generated CSV files, and the other one is raw PCAP files captured from their experiment. Since the problems of the CSV files are addressed in Section 3.1, the data to be used in this paper would be the PCAP files. The attack schedule for this dataset can be found on the CICCCoS2109 website[11]. According to the schedule, the corresponding PCAP files are listed in Table 3.1. Please note that in some adjacent files, there could be partial attacks and partial benign.

Table 3.2: Features in CICDDoS2019 [8], [11].

Begin of Table	
Feature Name	Description
Flow Duration	Flow duration
Total Fwd Packet	Total packets in the forward direction
Total Bwd packets	Total packets in the backward direction
Total Length of Fwd Packet	The total size of packets in the forward direction
Fwd Packet Length Max	Maximum size of packets in the forward direction

Continuation of Table 3.2	
Feature Name	Description
Fwd Packet Length Min	The minimum size of packets in the forward direction
Fwd Packet Length Mean	The average size of packets in the forward direction
Fwd Packet Length Std	Standard deviation size of packets in the forward direction
Bwd Packet Length Max	Maximum size of packets in the backward direction
Bwd Packet Length Min	The minimum size of packets in the backward direction
Bwd Packet Length Mean	Mean size of packets in the backward direction
Bwd Packet Length Std	Standard deviation size of packets in the backward direction
Flow Bytes/s	flow byte rate that is the number of packets transferred per second
Flow Packets/s	flow packets rate that is the number of packets transferred per second
Flow IAT Mean	The average time between the two flows
Flow IAT Std	Standard deviation time two flows
Flow IAT Max	Maximum time between two flows
Flow IAT Min	Minimum time between two flows
Fwd IAT Total	Total time between two packets sent in the forward direction
Fwd IAT Mean	The mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction

Continuation of Table 3.2

Feature Name	Description
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Bwd IAT Total	Total time between two packets sent in the backward direction
Bwd IAT Mean	The mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Length	Total bytes used for headers in the forward direction
Bwd Header Length	Total bytes used for headers in the forward direction
Fwd Packets/s	Number of forwarding packets per second
Bwd Packets/s	Number of backward packets per second
Packet Length Min	Minimum length of a flow

Continuation of Table 3.2	
Feature Name	Description
Packet Length Max	The maximum length of a flow
Packet Length Mean	Mean length of a flow
Packet Length Std	Standard deviation length of a flow
Packet Length Variance	Minimum inter-arrival time of packet
FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWE Flag Count	Number of packets with CWE
ECE Flag Count	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Average Packet Size	The average size of packets
Fwd Segment Size Avg	Average size observed in the forward direction
Bwd Segment Size Avg	Average size observed in the backward direction
Fwd Bytes/Bulk Avg	The average number of bytes bulk rate in the forward direction
Fwd Packet/Bulk Avg	The average number of packets bulk rate in the forward direction
Fwd Bulk Rate Avg	The average number of bulk rate in the forward direction
Bwd Bytes/Bulk Avg	The average number of bytes bulk rate in the backward direction

Continuation of Table 3.2	
Feature Name	Description
Bwd Packet/Bulk Avg	The average number of packets bulk rate in the backward direction
Bwd Bulk Rate Avg	The average number of bulk rate in the backward direction
Subflow Fwd Packets	The average number of packets in a sub-flow in the forward direction
Subflow Fwd Bytes	The average number of bytes in a sub-flow in the forward direction
Subflow Bwd Packets	The average number of packets in a sub-flow in the backward direction
Subflow Bwd Bytes	The average number of bytes in a sub-flow in the backward direction
FWD Init Win Bytes	Number of bytes sent in the initial window in the forward direction
Bwd Init Win Bytes	The number of bytes sent in the initial window in the backward direction
Fwd Act Data Pkts	The number of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction
Active Mean	The mean time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Active Max	The maximum time a flow was active before becoming idle
Active Min	The minimum time a flow was active before becoming idle
Idle Mean	Meantime a flow was idle before becoming active

Continuation of Table 3.2	
Feature Name	Description
Idle Std	Standard deviation time a flow was idle before becoming active
Idle Max	The maximum time a flow was idle before becoming active
Idle Min	The minimum time a flow was idle before becoming active

The authors also conducted data analysis based on what they generated from CIC-FlowMeter. According to their algorithm, the features weighted most are extracted from the 80 features. Based on their selected features, they applied the ID3 decision tree, sklearn random forest, Naive Bayes and multinomial Logistic Regression. The results of their experiment will be compared in Chapter 4. Moreover, in Table 3.3, the top 5 weighted features selected by them are shown. In Chapter 4, these features will be used in more machine learning models, and the results yield from those models will be used as benchmarks.

3.3 Traditional Machine Learning Approaches Exploration

In this section, multiple popular machine learning techniques will be trained and tested on CIC-DDoS 2019 dataset. According to No Free Lunch (NFL) theorem[65], no algorithm can perform best on every problem. An exploration of machine learning approaches must be done before design the whole framework. The purpose of this section is to choose a technique that can perform better on this particular dataset.

A simple approach to testing the performance of a machine learning technique is shown in Algorithm 1.

In this section, the dataset is down-sampled into a smaller size and balanced. The detail of the sampled dataset is shown in Table 3.4.

Label Encoder from scikit-learn[66] is used in the experiment. The corresponding code-label map is also shown in Table 3.4. The Confusion Matrix in the subsections below refers to this encoding scheme.

Table 3.3: Feature selection for detection [8]

Label	Feature
UDP-Lag	ACK Flag Count, Init Win bytes forward, min seg size forward, Fwd IAT Mean, Fwd IAT Max
TFTP	Fwd IAT Mean, min seg size forward, Fwd IAT Max, Flow IAT Max, Flow IAT Mean
WebDDoS	ACK Flag Count, Init Win bytes forward, Fwd Packet Length Std, Packet Length Std, min seg size forward
DNS	Max Packet Length, Fwd Packet Length Max, Fwd Packet Length Min, Average Packet Size, Min Packet Length
Benign	ACK Flag Count, Flow IAT Min, Init Win bytes forward, Fwd Packet Length Std, Packet Length Std
MSSQL	Fwd Packets/s, Protocol
LDAP	Max Packet Length, Fwd Packet Length Max, Fwd Packet Length Min, Average Packet Size, Min Packet Length
NetBIOS	Fwd Packets/s, min seg size forward, Protocol, Fwd Header Length, Fwd Header Length.1
NTP	Subflow Fwd Bytes, Length of Fwd Packets, Fwd Packet Length Std, min seg size forward, Flow IAT Min
SSDP	Destination Port, Fwd Packet Length Std, Packet Length Std, Protocol, min seg size forward
SNMP	Max Packet Length, Fwd Packet Length Max, Fwd Packet Length Min, Average Packet Size, Min Packet Length
Syn	ACK Flag Count, Init Win bytes forward, min seg size forward, Fwd IAT Total, Flow Duration
UDP	Destination Port, Fwd Packet Length Std, Packet Length Std, min seg size forward, Protocol

Algorithm 1: algorithm used for testing the performance of a machine learning technique on CIC-DDoS 2019 dataset.

```
1 def explore_performance ($df, model$);
  Input  : DataFrame $df$, model $model$
  Output: confusion matrix $cm$
2 $df$ ← select a balanced subset from $df$;
3 scale $df$ using $sklearn.preprocessing.MinMaxScaler$;
4 split $df$ into $trainset$ and $testset$;
5 train model with 10 fold cross validation;
6 plot $loss$ and $acc$ for both training and validating;
7 calculate confusion matrix $cm$;
8 return $cm$;
```

DDoS type	number of rows	Encode Label
SSDP	130531	7
UDP	125386	8
NTP	120264	4
LDAP	108997	2
SNMP	103197	6
DNS	101420	1
MSSQL	90450	3
NetBIOS	81866	5
BENIGN	27375	0

Table 3.4: Detail of Sampled Dataset

Algorithm	Precision	Recall	F1-score
ID3	0.78	0.65	0.69
Random Forest	0.77	0.56	0.62
Naive Bayes	0.41	0.11	0.05
Logistic Regression	0.25	0.02	0.04

Table 3.5: The results from Sharafaldin et al [8]

3.3.1 Results from the dataset creators

In the original paper of CICDDoS2019[8], the authors applied their feature selection algorithm to every attack type. The top five selected features are shown in detail in Table 3.3. They also provide a classification result based on the selected features. The authors have applied ID3 (a type of decision tree), Random Forest, Logistic Regression and Naive Bayes. The results from their paper will be listed in Table 3.5, and the results will be used to make a comparison with the results in the following subsections.

The reason why rerun the experiment is that, first, the dataset is regenerated, the experiment must be reconducted to keep the consistency. Second, the original features (unselected) must be tested and will be used as the benchmark in the following experiments.

3.3.2 Decision Tree

Decision Tree is used by many researchers[67]–[69] to detect DDoS attacks. Decision Tree determines the types of attacks by calculating the information gain at every split point. The information gain is calculated by the difference of information purity before and after splitting. There are mainly two types of formula to calculate the information purity: entropy (Equation 3.1) and Gini Index (Equation 3.1).

$$entropy = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{3.1}$$

$$Gini = 1 - \sum_{i=1}^{n} p(x_i)^2 \tag{3.2}$$

In both equations, $p(x_i)$ refers to the probability of the class out of all classes.

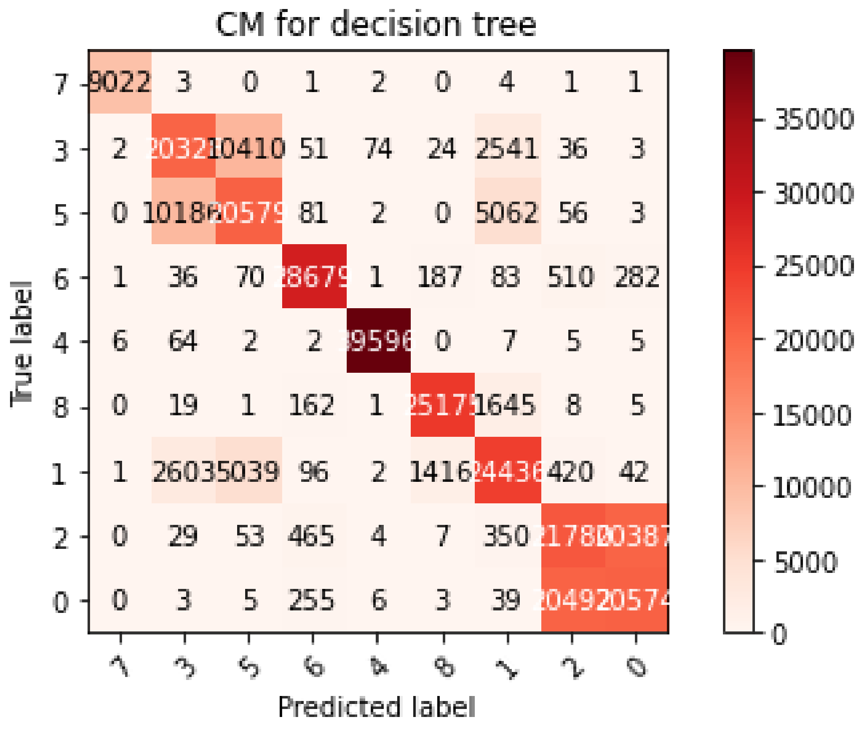


Figure 3.1: Confusion Matrix for Decision Tree

sklearn.tree.DecisionTreeClassifier[70] is used in this experiment. The results for Decision Tree is shown below. And the Confusion Matrix is shown in Figure 3.1.

Accuracy: 0.71600

Precision: 0.75496

Recall: 0.75433

F1-score: 0.75462

3.3.3 Random Forest

Random Forest is an ensemble classifier that consists of multiple decision trees. Each decision tree will take a different feature subset when it train and test. Based on different features, the decision trees may behave differently on the dataset. It is possible that the decision trees in the random forest give a different prediction on the same data. A simple way to tackle this problem is by utilizing a voting mechanism. For example, five out of nine decision trees give label BENIGN to specific traffic, and the forest can determine the traffic is BENIGN.

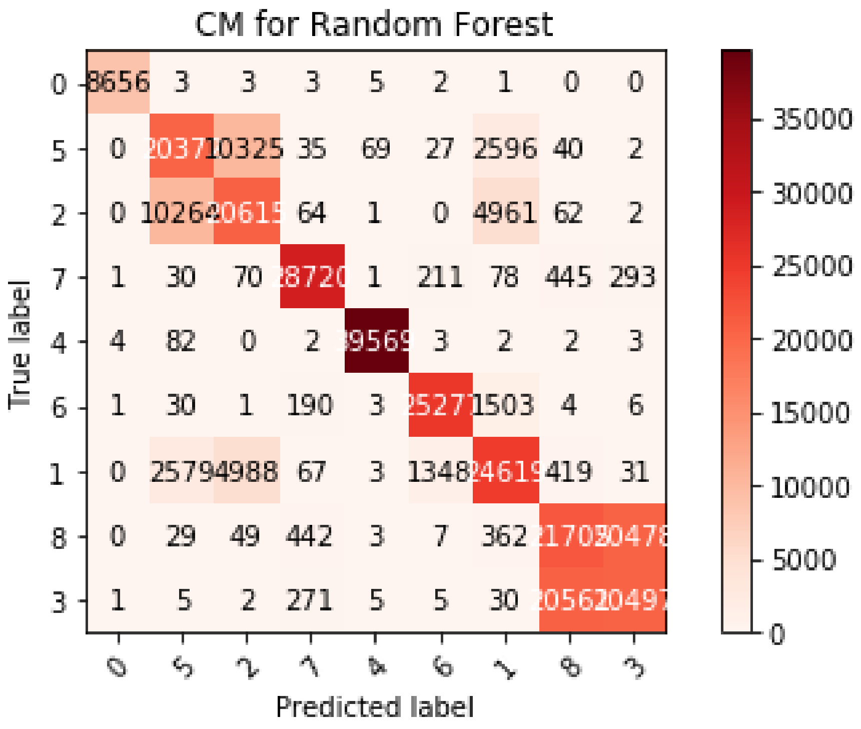


Figure 3.2: Confusion Matrix for Random Forest

Brabec et al.[71] do a comparative analysis of the voting scheme in network intrusion detection. Their works prove that majority voting increases the F-score of intrusion detection. The comparison of experiments from Section 3.3.2 and 3.3.3 also supports the idea.

sklearn.ensemble.RandomForestClassifier[72] is used in this experiment. The hyperparameter n_estimator is set to default(100). The result for Random Forest is shown below. And the Confusion Matrix is shown in Figure 3.2.

Accuracy: 0.73960

Precision: 0.77993

Recall: 0.77781

F1-score: 0.77393

3.3.4 Logistic Regression

Logistic Regression is a type of classifier which uses the approach of regression. Logistic Regression is mostly used in binary classification. However, Logistic Regression can also be modified to adapt to multi-label classification[73]. A simple way to achieve this goal is using binary classification for a single class in the remaining data. This method is called Leave-One-Out[74]. For example, if the dataset has three labels, firstly, binary classification is applied to determine class A and non-A. Then, apply binary classification again on non-A data to determine class B and non-B. At last, the non-B data can be labelled as class C.

Logistic Regression is used by many researchers[74]–[77]. Among the researchers, Subba et al.[75], Kamarudin et al.[76] and Ghosh et al.[77] all train and test on NSL-KDD'99 dataset. Subba and his team got the best result among the three team. The overall binary accuracy reached 98.27% and the multi-class accuracy varies from 92.43% and 99.02%.

sklearn.linear_model.LogisticRegression[78] is employed in this experiment. The multi-class classification results are shown below. The confusion matrx is shown in Figure 3.3.

Accuracy: 0.73960

Precision: 0.77993

Recall: 0.77781

F1-score: 0.77393

3.3.5 Naive Bayes

Naive Bayes is a statistical model that considers prior knowledge. The assumption to make Naive Bayes work is that every event in the causal model is independent. However, Naive Bayes itself does not perform well in some scenarios.

Many researchers combine Naive Bayes with other techniques to improve performance. Varuna rt al.[79] combines k-means clustering and Naive Bayes classifier to detect network intrusion. Almansobe et al.[80] addressed the challenges for intrusion detection system using naive Bayes and PCA algorithm. Li et al.[81] used Naive Bayes with AdaBoost to Enhance Network Anomaly Intrusion Detection. Halimaa et al.[82] explore a way to use Naive Bayes with SVM to improve the performance. Veetil et al.[83] introduced Hadoop, a big data analysis tool, to the framework and worked with Naive Bayes.

Meanwhile, there are also some researchers work on improving Naive Bayes itself instead of co-operating with other classifiers. Panda et al.[84] applied discriminative

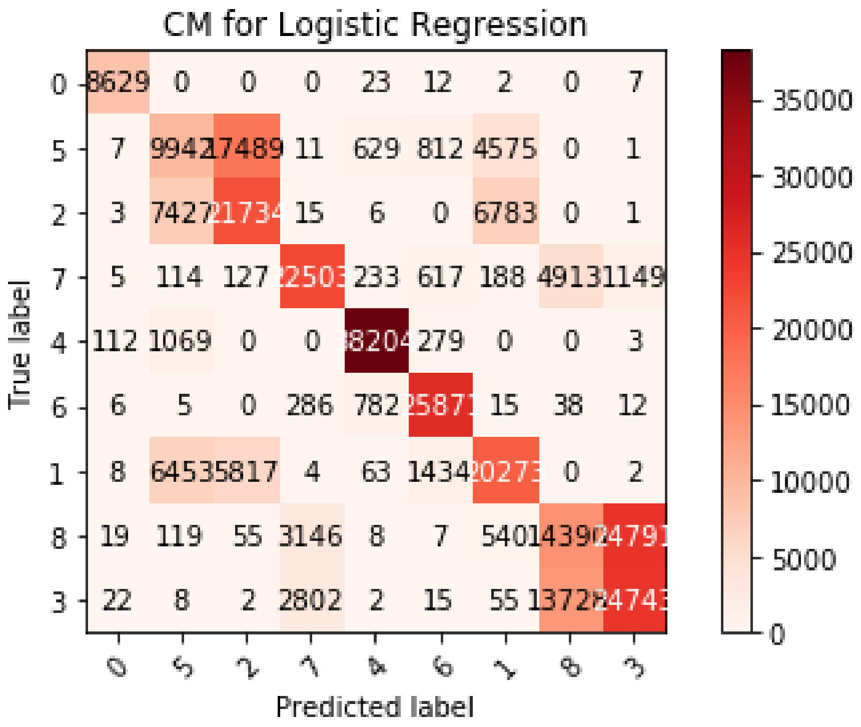


Figure 3.3: Confusion Matrix for Logistic Regression

multinomial Naïve Bayes with various filtering analysis to build a NIDS. Moreover, Bhosale et al.[85] made some modifications to Naive Bayes and applied feature selection to the dataset.

sklearn.naive_bayes.MultinomialNB[86] is used in this experiment. The result can be found below. And the corresponding confusion matrix is shown in Figure 3.4.

Accuracy: 0.54138

Precision: 0.67635

Recall: 0.54744

F1-score: 0.49248

3.4 Feature Compressing

After dropping all unnecessary features, there are 49 features left. In order to find the most important features from the remaining ones, there are mainly two ways. The first way is using feature selection techniques such as Principle Component Analysis[87] and ANOVA[88]. These two techniques are statistical models. Dey et al.[88] conclude

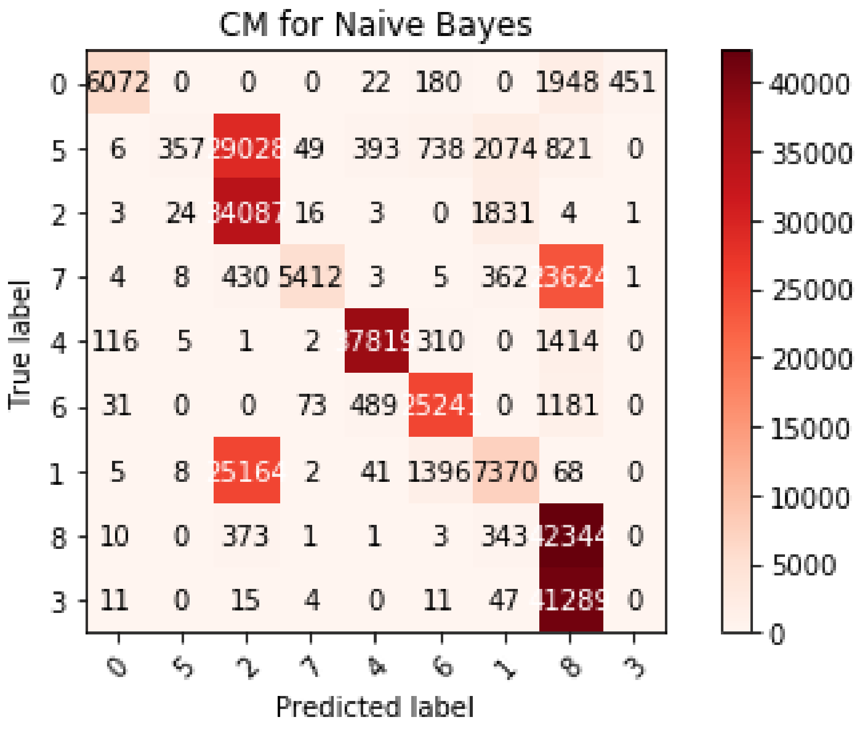


Figure 3.4: Confusion Matrix for Naive Bayes

from their experiment that ANOVA achieved the lowest False Alarm Rate when tested on the NSL-KDD'99 dataset. These two techniques can only select the features from the original columns, which means they can not derive component features that do not exist in the original features. The second way is using a neural network to compress the original features into a lower dimension, and reconstruct the compressed features back to the original ones. The performance of the compressing can be evaluated by calculating the loss of the original features and the reconstructed features. This method employs Auto Encoder[49], shown in Figure 3.5. Auto Encoder is a neural network in essence, which implies it can operate non-linear feature selection.

To make the best of non-linear feature selection, Auto Encoder is chosen in this paper. The experiments and results of comparing PCA, ANOVA and Auto Encoder can be found in Section4.4.

The process of training an autoencoder is shown in Algorithm 2.

Algorithm 2: algorithm used for training the auto encoder and return the loss between the original features and the compressed features.

```
1  def evaluate ($df, model$);
   Input  : DataFrame $df$, model $model$ and number of epochs $epoch$
   Output: $loss$
2  $df \leftarrow df$ drop the rejected, categorical columns and "Label";
3  $df \leftarrow df$ drop the $Nan$ and $Inf$ rows;
4  scale $df$ using $sklearn.preprocessing.MinMaxScaler$;
5  while $epoch > 0$ do
6      train $model$ using $df$ as both the train and validation set;
7      $loss \leftarrow$ the binary crossentropy between the original features and the
         reconstructed features;
8      print $loss$;
9      $epoch \leftarrow epoch - 1$ ;
10 return;
```

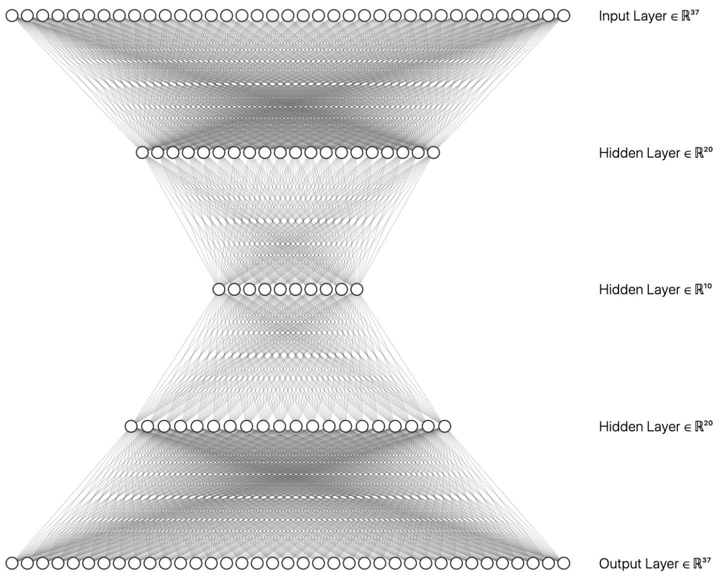

Input Layer ∈ $\mathbb{R}^{37}$
Hidden Layer ∈ $\mathbb{R}^{20}$
Hidden Layer ∈ $\mathbb{R}^{10}$
Hidden Layer ∈ $\mathbb{R}^{20}$
Output Layer ∈ $\mathbb{R}^{37}$

Figure 3.5: Auto Encoder representation

3.5 Pandas-profiling

Sharafaldin et al.[8] has selected the top five important features for each attack type. The idea behind the selection is similar to Principal Component Analysis(PCA). The purpose of PCA is to select a set of uncorrelated features from all columns. However, in this experiment, the opposite way is going to be conducted. Which means, the highly correlated features will be dropped in the procedure of training.

The metrics used to evaluate the correlation between the features is the Pearson correlation coefficient. Pearson correlation coefficient is developed by Karl Pearson[12]. It measures the linear correlation between the two features. The formula of the Pearson correlation coefficient is shown in Equation 3.3. In Equation 3.3, $E[X]$ represents the expectation of variable X.

$$\rho_{X,Y} = \frac{E[XY] - E[X]E[Y]}{\sqrt{E[X^2] - [E[X]]^2}\sqrt{E[Y^2] - [E[Y]]^2} \tag{3.3}$$

The coefficient is between -1 to 1. 1 means 100% positive linear correlated while -1 means 100% negative linear correlated. Moreover, 0 means the two variables are not correlated. In this experiment, the correlation is considered high if the Pearson correlation coefficient is larger than 0.9 or less than -0.9.

The tool used to calculate the Pearson correlation coefficient is Pandas-profiling. Pandas-profiling[89] is a python-based statistical tool that does exploratory data analysis. It will generate an HTML file that contains:

- Type inference: detect the types of columns in a dataframe.
- Essentials: type, unique values, missing values
- Quantile statistics such as minimum value, Q1, median, Q3, maximum, range, interquartile range
- Descriptive statistics such as mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness
- Most frequent values
- Histogram

- Correlations highlighting of highly correlated variables, Spearman, Pearson and Kendall matrices

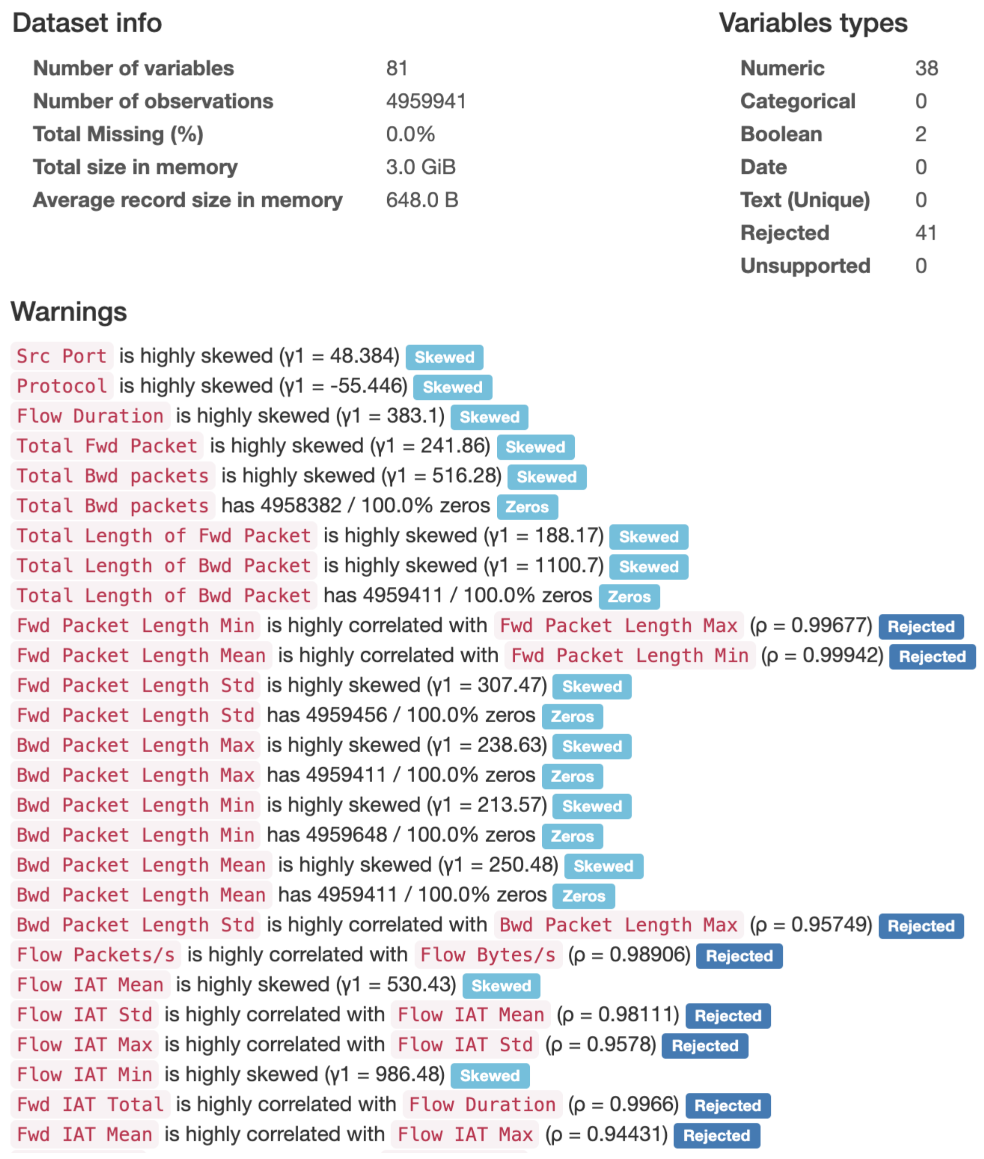
- Missing values matrix, count, heatmap and dendrogram of missing values

- Text analysis learn about categories (Uppercase, Space), scripts (Latin, Cyrillic) and blocks (ASCII) of text data.

Among all the statistics, correlations are the most important for the analysis of the dataset. This tool will give alerts when there are two columns that have the Pearson Correlation Coefficient higher than 0.9 or less than -0.9. In this case, these two columns should not present in the learning process at the same time. An example of Pandas-profiling report for SNMP is shown in Figure 3.6.

An example of Pearson correlation coefficient for SNMP is shown in Figure 3.7.

The Pearson correlation coefficient for each attack type is calculated, respectively. The numbers of highly correlated features vary from 41 to 47. There are two approaches to drop the features. (1) Drop the corresponding features for each attack type. Use the remained features to train and test. (2) Drop only the intersection of the features derived from each attack type. Keep the other features and then train and test. There are two problems if the first approach is adopted: (1) When the test on a mixed dataset which contains more than one attack type, it is hard to decide which columns are dropped. (2) The features that are highly correlated in one attack type and not correlated in another attack type may be an important feature to discriminate the types. However, there are also two issues if the second approach is adopted. (1) The number of remaining features is more than the number of the first approach. The training and prediction time may increase. (2) The respective accuracy for each attack type may decrease because there are highly correlated features in the dataset. However, the ultimate purpose of the proposed framework is to discriminate attacks from benign background traffic. One of the best practices is to keep the input features of all the models identical since the types of traffic are not known. Considering all the conditions mentioned above, the second approach will be used in the experiment. Other than the highly correlated features, the features which have constant values such as 0 will be dropped simultaneously.



Dataset info

Number of variables	81
Number of observations	4959941
Total Missing (%)	0.0%
Total size in memory	3.0 GiB
Average record size in memory	648.0 B

Variables types

Numeric	38
Categorical	0
Boolean	2
Date	0
Text (Unique)	0
Rejected	41
Unsupported	0

Warnings

`Src Port` is highly skewed (γ1 = 48.384) Skewed
`Protocol` is highly skewed (γ1 = -55.446) Skewed
`Flow Duration` is highly skewed (γ1 = 383.1) Skewed
`Total Fwd Packet` is highly skewed (γ1 = 241.86) Skewed
`Total Bwd packets` is highly skewed (γ1 = 516.28) Skewed
`Total Bwd packets` has 4958382 / 100.0% zeros Zeros
`Total Length of Fwd Packet` is highly skewed (γ1 = 188.17) Skewed
`Total Length of Bwd Packet` is highly skewed (γ1 = 1100.7) Skewed
`Total Length of Bwd Packet` has 4959411 / 100.0% zeros Zeros
`Fwd Packet Length Min` is highly correlated with `Fwd Packet Length Max` (ρ = 0.99677) Rejected
`Fwd Packet Length Mean` is highly correlated with `Fwd Packet Length Min` (ρ = 0.99942) Rejected
`Fwd Packet Length Std` is highly skewed (γ1 = 307.47) Skewed
`Fwd Packet Length Std` has 4959456 / 100.0% zeros Zeros
`Bwd Packet Length Max` is highly skewed (γ1 = 238.63) Skewed
`Bwd Packet Length Max` has 4959411 / 100.0% zeros Zeros
`Bwd Packet Length Min` is highly skewed (γ1 = 213.57) Skewed
`Bwd Packet Length Min` has 4959648 / 100.0% zeros Zeros
`Bwd Packet Length Mean` is highly skewed (γ1 = 250.48) Skewed
`Bwd Packet Length Mean` has 4959411 / 100.0% zeros Zeros
`Bwd Packet Length Std` is highly correlated with `Bwd Packet Length Max` (ρ = 0.95749) Rejected
`Flow Packets/s` is highly correlated with `Flow Bytes/s` (ρ = 0.98906) Rejected
`Flow IAT Mean` is highly skewed (γ1 = 530.43) Skewed
`Flow IAT Std` is highly correlated with `Flow IAT Mean` (ρ = 0.98111) Rejected
`Flow IAT Max` is highly correlated with `Flow IAT Std` (ρ = 0.9578) Rejected
`Flow IAT Min` is highly skewed (γ1 = 986.48) Skewed
`Fwd IAT Total` is highly correlated with `Flow Duration` (ρ = 0.9966) Rejected
`Fwd IAT Mean` is highly correlated with `Flow IAT Max` (ρ = 0.94431) Rejected

Figure 3.6: Snapshot of Pandas-profiling

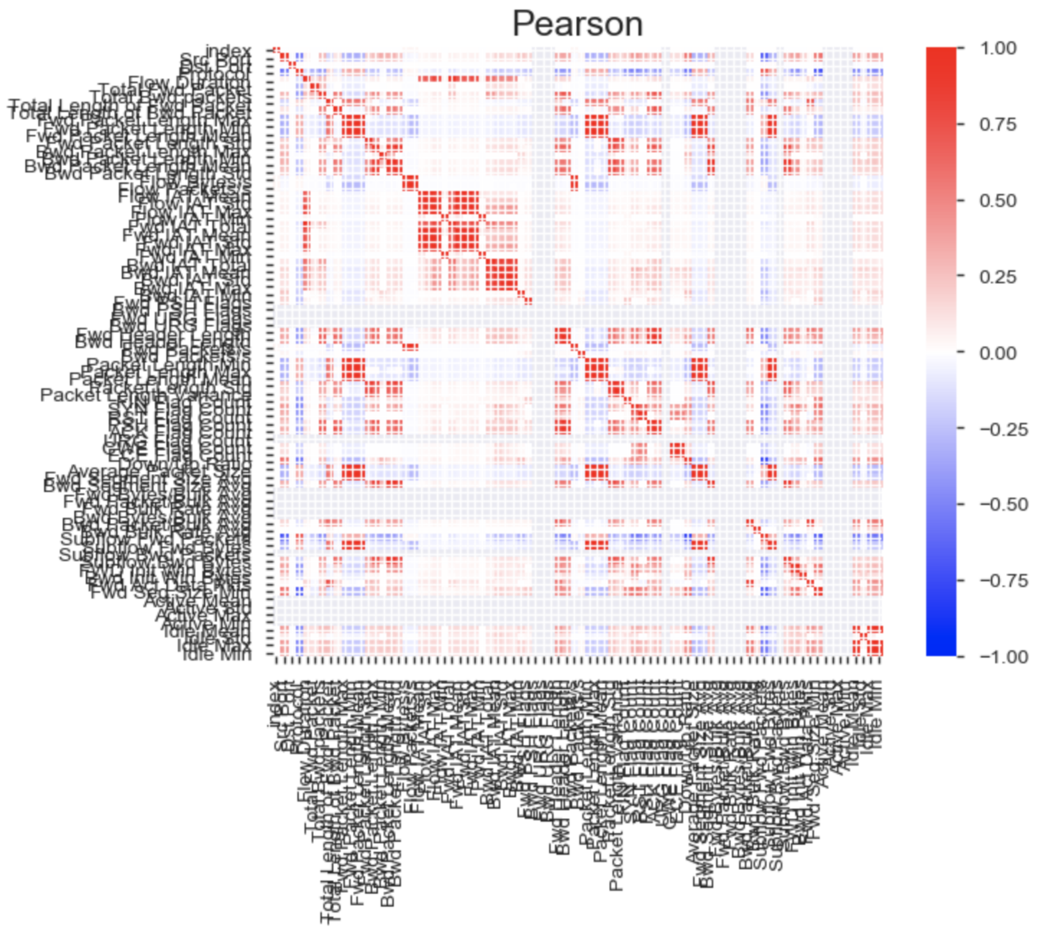


Figure 3.7: Pearson correlation coefficient

'Active Max'	'Subflow Bwd Packets'	'Bwd IAT Max'
'Active Min'	'Idle Min'	'Fwd Bulk Rate Avg'
'Bwd PSH Flags'	'Packet Length Min'	'Bwd URG Flags'
'Idle Max'	'Fwd IAT Total'	'Bwd Bytes/Bulk Avg'
'Bwd Segment Size Avg'	'Fwd Packet/Bulk Avg'	'Fwd URG Flags'
'URG Flag Count'	'Active Mean'	'ECE Flag Count'
'Fwd Bytes/Bulk Avg'	'ACK Flag Count'	'Fwd Packets/s'
'Bwd Packet Length Std'	'Subflow Bwd Bytes'	'Fwd IAT Std'
'Bwd IAT Std'	'Fwd IAT Mean'	'Active Std'
'Packet Length Mean'	'Fwd IAT Max'	'Fwd Segment Size Avg'

Table 3.6: Features to be dropped

According to the result of Pandas-profiling, the following columns should be dropped during the learning procedure:

The following columns should also be dropped because they are categorical and sparse:

- 'Flow ID'
- 'Source IP'
- 'Destination IP'
- 'Timestamp'

3.6 Classifiers

In this section, three different approaches are designed and tested. The first approach is using DNN to do the classification. The second approach is using autoencoder to reduce the feature dimensions before classifying with DNN. The third approach is firstly dropping some features based on the Pearson correlation coefficient. Secondly, use autoencoder to make further feature reduction. Lastly, classify the data with DNN. Three approaches will be explained in detail in the following subsections.

The reason why not using a single multiclass DNN is here. Firstly, in real networks, no one knows what type of traffic the netflow is. The job of detection does is to discriminate attacks from benign traffic. That is the reason why multi classifiers are not required. Secondly, a multi class test has been conducted, but the results are not good.

Note: In this thesis, DNN refers to Dense Neural Network.

3.6.1 DNN Classifier

DNN stands for Dense Neural Network. DNN is a type of neural network that consists of only dense layers. As is known, a neural network is built with multiple layers. Each layer is connected to the next layer, except for input and output layers. In Dense Neural Network, each neuron on each layer is connected to each neuron on the next layer, as shown in Figure 3.8. The number of neurons on the input layer should be equal to the dimension of features. The number of neurons on the output layer depends on how many distinct labels does the dataset has. For example, if there is only one neuron on the output layer, the DNN can be used as a binary classifier, as 0 represents false, and 1 represents true. If the DNN is used to do multi-label classification, the number of neurons may vary. Usually, researchers use the one-hot encoder to encode the labels. Using the one-hot encoder, the number of neurons on the output layer should be equal to the number of distinct labels.

3.6.2 AutoEncoder

AutoEncoder is also a type of dense neural network. However, it is not designed to do classification. In an autoencoder, the input and the output should be identical. An autoencoder consists of two main parts. The first part is called encoder. The primary purpose of the encoder is to compress the original features into a lower dimension. For example, in Figure3.5, the encoder part consists of the first three layers. The first layer is the input layer, which has the same number of neurons as the input feature dimension. The second part is called decoder. The purpose of the decoder is to reconstruct the compressed features back to the original features. As is shown in Figure3.5, from the third layer to the fifth layer, it represents the decoder part. The last layer is the output layer, and the number of neurons on the output layer is the same as the original feature dimension. In the training process of the autoencoder, the training input and the training output should be the same. If the autoencoder can successfully reconstruct the features back to the original ones, the compressed features can be used to classify since the compressed features can represent all the original features.

After autoencoder, the compressed features will be used as input features of DNN in order to classify the traffic data.

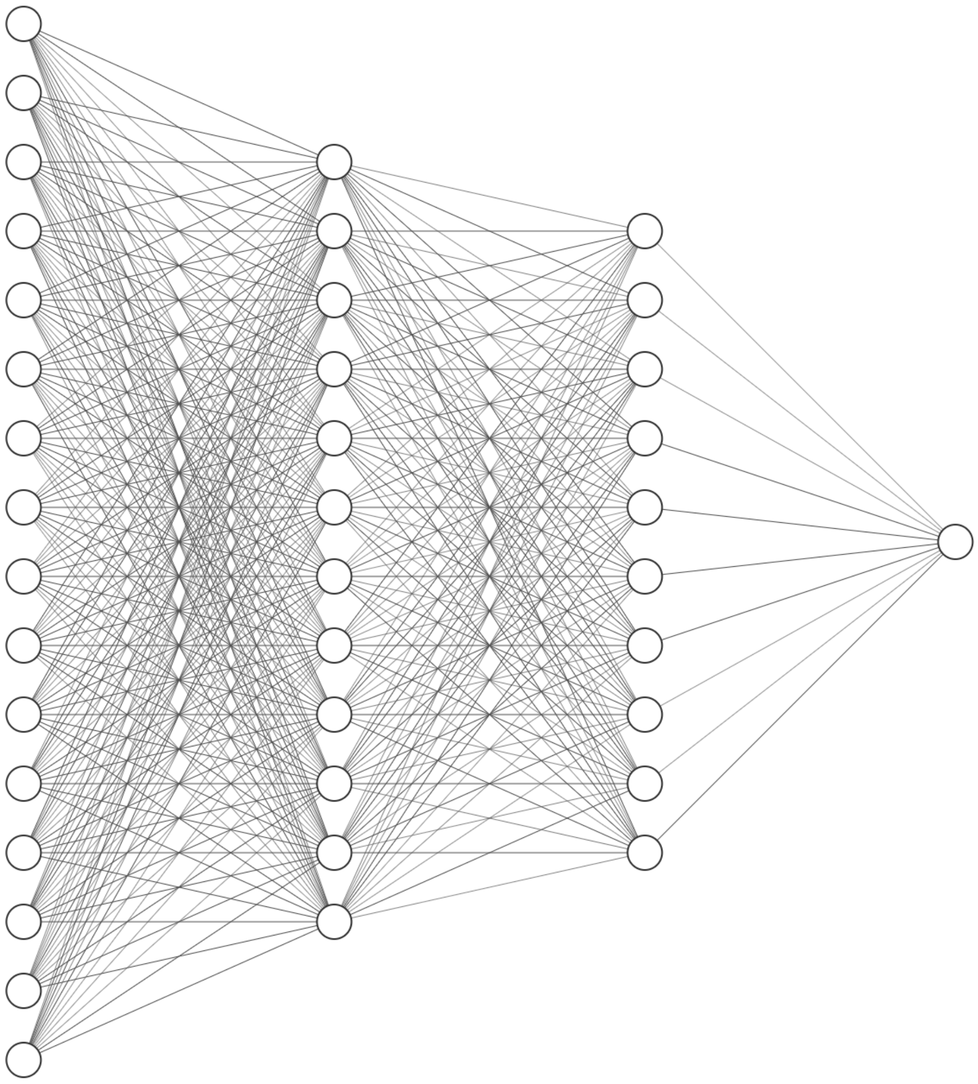


Figure 3.8: Illustration of DNN

3.6.3 Pearson Correlation Coefficient

Pearson Correlation Coefficient is explained in Section 3.5. The main procedure is: (1) use Pandas-Profiling to calculate the Pearson Correlation Coefficient. (2) Drop one of the highly correlated features. (3) Feed the remaining features into the model that is identical to the one described in Subsection 3.6.2.

3.7 Metrics

In this section, all metrics that are used to evaluated models are defined below.

True Positive (TP) Attack records correctly classified as attack records.

True Positive is expected to be high in the result.

True Negative (TN) Benign records correctly classified as benign records.

True Negative is expected to be high in the result.

False Positive (FP) Normal records incorrectly classified as attack records.

False Positive is expected to be low in the result.

False Negative (FN) Attack records incorrectly classified as benign records.

False Negative is expected to be low in the result.

False Positive Rate(FPR) The proportion of incorrectly classified normal records among all normal records.

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

False Positive Rate is expected to be low in the result.

Accuracy The overall success rate of an IDS, showing the percentage of correct classifications.

In the result, Accuracy is expected to be high.

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

Precision The proportion of correct positive attack classifications.

Precision is expected to be high in the result.

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

Recall (TPR) The proportion of correctly detected positive values.

Recall is expected to be high in the result.

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

F-Measure The F-Measure of a model is the harmonic mean of precision and recall.

F-Measure is expected to be high in the result.

$$F - Measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Confusion Matrix A $N \times N$ Matrix that helps summarize how successful the model was in predicting attacks, where N is the number of unique labels. Higher values across the primary diagonal indicate better results.

Learning Curve A learning curve shows the effect of increasing the size of the training data on the score of the model. It can be used to infer overfitting and bias in the training data.

ROC Curve A Receiver-Operating Characteristic (ROC) Curve shows the relation between the TPR and FPR at varying classification thresholds. A steeper curve towards the y-axis represents better detection by the model.

Area Under the Curve (AUC) The Curve refers to the ROC Curve. The AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one[90].

Chapter 4

Experiment

4.1 Hardware configuration

Operating System: Windows 10 Education 64-bit

Version: 1909

OS build: 18363.657

Processor: Intel(R) Core(TM) i7-9700KF CPU @ 3.60GHz 3.60GHz

Installed RAM: 128 GB

GPU: NVIDIA Quadro RTX 8000

Bios Information: Version 90.2.30.0.1

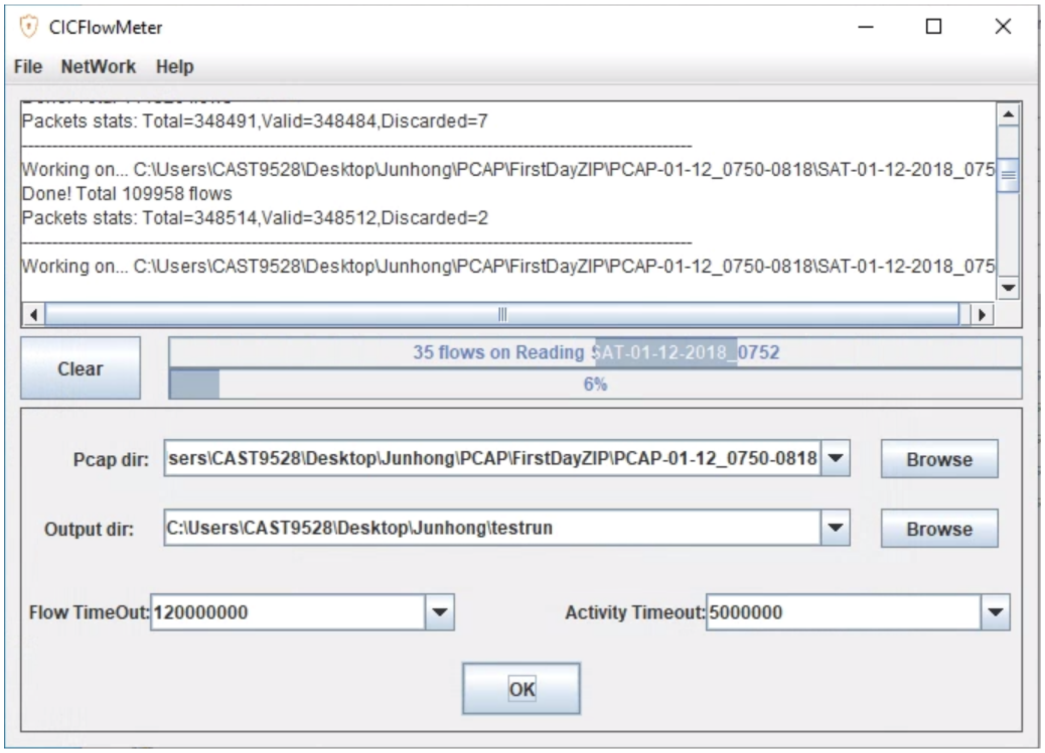
Graphics Memory: 114644 MB

4.2 Generate New CSVs

As mentioned in Section 3.1, the CSV files provided by CICDDoS2019 is not suitable for this experiment. The best way to utilize the dataset is to directly generate the CSV files from the raw PCAP files using CIC-FlowMeter. The source code of CIC-FlowMeter can be found from their GitHub Repository[91].

The procedure for generating CSV files is easy. After following the setup guidelines on the GitHub page, the Graphic User Interface will show up on the screen. The CIC-FlowMeter offers two modes to convert the PCAP files: realtime and offline. For this experiment, the PCAP files are already provided by the CICDDoS2019 dataset so that we can choose the offline mode.

After a simple configuration of the PCAP directory and output directory, the CIC-FlowMeter can run and generate the CSV files consequently. The snapshot of running the CIC-FlowMeter is shown in Figure 4.1.



```
Packets stats: Total=348491,Valid=348484,Discarded=7
--------------------------------------------------------------------------------
Working on... C:\Users\CAST9528\Desktop\Junhong\PCAP\FirstDayZIP\PCAP-01-12_0750-0818\SAT-01-12-2018_075
Done! Total 109958 flows
Packets stats: Total=348514,Valid=348512,Discarded=2
--------------------------------------------------------------------------------
Working on... C:\Users\CAST9528\Desktop\Junhong\PCAP\FirstDayZIP\PCAP-01-12_0750-0818\SAT-01-12-2018_075
```

35 flows on Reading SAT-01-12-2018_0752

6%

Pcap dir: sers\CAST9528\Desktop\Junhong\PCAP\FirstDayZIP\PCAP-01-12_0750-0818

Output dir: C:\Users\CAST9528\Desktop\Junhong\testrun

Flow TimeOut: 120000000

Activity Timeout: 5000000

Figure 4.1: Running CIC-FlowMeter

4.3 Data preprocessing

4.3.1 Rebalance Data

The CSV data provided by CIC-DDoS-2019 contains only 27375 rows of benign data. In contrast, the number of attacks are more than 40 million, which is around 1500 times more than benign data. The dataset is highly imbalanced. To tackle this problem, there are mainly two methods: down-sampling the majority and over-sampling the minority. In Section 3.3, to explore the performance of different types of machine learning techniques, down-sampling is employed to rebalance the dataset. In Section 3.3, all 27375 benign data is used in the dataset, while all other attacks are down-sampled to from 81866 to 130531. The size of this re-sampled dataset is ideal for exploring the performance of machine learning techniques, however, it is not enough to train a better performing neural network. In this section, up-sampling is employed to solve this problem.

The CSV data provided by CIC-DDoS-2019 is generated by CIC-FlowMeter[9], [53]. And the PCAP files that are used to generate CSV data are also provided by CIC-DDoS-2109. In the dataset, not only the attacks traffic but also the benign traffic are logged in PCAP files. The details of the PCAP files is shown in Table 3.1. As stated in Table 3.1, there are 189 PCAP files containing only benign data. From which, we can generate 5662323 benign traffic.

There is a minor issue about CIC-FlowMeter. The CIC-FlowMeter is updated to version 4, which is stabler than version 3. There are mainly three differences between the two versions is:(1) the names of the features are changed; (2) three columns are ignored: Fwd Header Length.1, SimillarHTTP, Inbound; (3) Timestamp format is changed. In order to keep the consistency, all attack PCAP files are reprocessed using CIC-FlowMeter V4.

One big issue using the changed Timestamp format is that it is incompatible with python DateTime format. In order to get rid of this issue, the time format used in CIC-FlowMeter is changed to standard 24-hour format: dd/MM/yyyy HH:mm:ss. The code is located in src/main/java/cic/cs/unb/ca/flow/jnetpcap/DateFormatter.java.

After converting the PCAP files into CSV files, the "Label" column is left as

Attack	Size	Proportion
PortMap	2312	0.0116%
NetBIOS	3844874	19.339%
LDAP	1917733	9.646%
MSSQL	5781928	29.082%
UDP	3855870	19.394%
UDP-Lag	3479	0.0175%
SYN	4444750	22.356%
benign	30787	0.1549%
Total	19881733	100%

Table 4.1: Regenerated First Day Dataset Details

Attack	Size	Proportion
NTP	1195447	2.388%
DNS	16599	0.0332%
LDAP	2207225	4.409%
MSSQL	3987736	9.965%
NetBIOS	4273829	8.537%
SNMP	4959941	9.907%
SSDP	2630865	5.255%
UDP	3113814	6.220%
SYN	2050115	4.095%
TFTP	19957711	39.865%
benign	5662323	11.310%
Total	50063090	100%

Table 4.2: Regenerated Second Day Dataset Details

"Need Manual Label". Which means, the rows must be labelled by the timestamp. The time-class reference table can be found in Table 3.1.

The regenerated dataset details are shown in Table 4.2 and 4.1.

4.3.2 Data Cleaning

As mentioned in Section 3.5, the sparse categorical columns and highly correlated columns should be dropped in the process. Other than that, the Nan value and Inf value should also be dropped. This operation is conducted by the Listing 4.1.

Listing 4.1: Data Cleaning

```
import pandas as pd
```

```
data_path = 'final_complete.csv'
data = pd.read_csv(data_path)
rejected = [...]
cat_columns = [...]
data = data.drop(columns=rejected)
data = data.drop(columns=cat_columns)
data.replace([np.inf, -np.inf], np.nan)
data = data.dropna()
data = data.reindex(sorted(data.columns), axis=1)
```

4.3.3 Data Normalization

Data normalization is usually required when researchers apply deep learning techniques to data that have different scales on attributes. Wang et al.[92] did research comparing the model performance with and without attribute normalization. The result shows that with statistical attribute normalization, discriminative models such as SVM and KNN becomes more effective than without statistical attribute normalization. In this part, two scalers are tested and compared. One is sklearn.preprocessing.StandardScaler. The other one is sklearn.preprocessing.MinMaxScaler.

The StandardScaler mapping function is shown in Equation 4.1.

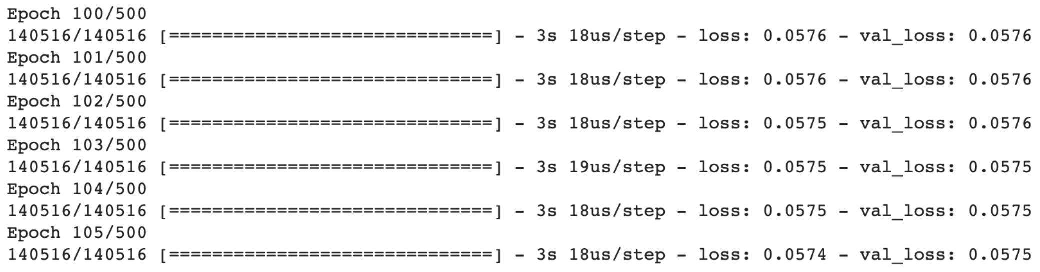
$$f'_{:,i} = \frac{f_{:,i} - mean(f_{:,i})}{std(f_{:,i})}. \tag{4.1}$$

The MinMaxScaler mapping function is shown in Equation 4.2.

$$f'_{:,i} = \frac{f_{:,i} - min(f_{:,i})}{max(f_{:,i}) - min(f_{:,i})}. \tag{4.2}$$

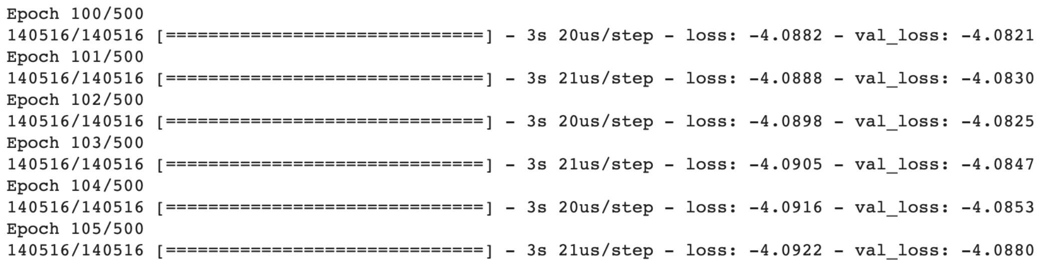
According to sklearn manual[93], [94], MinMaxScaler is very sensitive to the presence of outliers. Furthermore, StandardScaler is more useful when handling negative values.

In order to check which scaler performs better on this dataset, a small experiment is conducted. The subset, details shown in Table 3.4, is used to test the scalers. The



```
Epoch 100/500
140516/140516 [==============================] - 3s 18us/step - loss: 0.0576 - val_loss: 0.0576
Epoch 101/500
140516/140516 [==============================] - 3s 18us/step - loss: 0.0576 - val_loss: 0.0576
Epoch 102/500
140516/140516 [==============================] - 3s 18us/step - loss: 0.0575 - val_loss: 0.0576
Epoch 103/500
140516/140516 [==============================] - 3s 19us/step - loss: 0.0575 - val_loss: 0.0575
Epoch 104/500
140516/140516 [==============================] - 3s 18us/step - loss: 0.0575 - val_loss: 0.0575
Epoch 105/500
140516/140516 [==============================] - 3s 18us/step - loss: 0.0574 - val_loss: 0.0575
```

Figure 4.2: MinMaxScaler Loss



```
Epoch 100/500
140516/140516 [==============================] - 3s 20us/step - loss: -4.0882 - val_loss: -4.0821
Epoch 101/500
140516/140516 [==============================] - 3s 21us/step - loss: -4.0888 - val_loss: -4.0830
Epoch 102/500
140516/140516 [==============================] - 3s 20us/step - loss: -4.0898 - val_loss: -4.0825
Epoch 103/500
140516/140516 [==============================] - 3s 21us/step - loss: -4.0905 - val_loss: -4.0847
Epoch 104/500
140516/140516 [==============================] - 3s 20us/step - loss: -4.0916 - val_loss: -4.0853
Epoch 105/500
140516/140516 [==============================] - 3s 21us/step - loss: -4.0922 - val_loss: -4.0880
```

Figure 4.3: MinMaxScaler Loss

MinMaxScaler and StandardScaler testing results are respectively shown in Figure 4.2 and 4.3.

As is shown in Figure 4.3, the loss remains negative. This problem is caused by the mechanism of StandardScaler. The StandardScaler always tries to scale all data to a normal distribution with mean equals zero, and the standard deviation equals one. This would make 50% of the data fall into the negative range. In this case, MinMaxScaler does a better job since the loss is always positive.

4.3.4 Data Split

During the training and testing process, the balance of data is an important factor that could influence the final testing results. Since the dataset has around 5 million rows of benign data, the best practice is to extract a proper proportion of attack data to match the number of benign traffic. However, if the number of attacks is too few, it is also acceptable to down-sample the benign traffic to match the number of attacks. Due to this reason, the number of each attack type used for training and testing is listed in Table 4.3.

Attack type	train set size	validation set size	test set size
SNMP	3962742/3472844	1188822/1041853	1698318/1488362
NetBIOS	3963512/2991795	1189053/897538	1698648/1282198
MSSQL	3961230/2793812	1188369/838144	1697670/1197348
UDP	3962126/2181172	1188638/654352	1698054/934788
SSDP	3939803/1844031	1181941/553209	1688487/790299
LDAP	3962518/1526167	1188755/463850	1698222/662643
SYN	3962527/1436180	1188758/430854	1698226/615506
NTP	3964170/836269	1189251/250880	1698930/358401
DNS	11550/11690	3465/3507	4950/5010
WebDDoS	1496/1540	449/462	641/660
UDP-Lag	3710/3817	1113/1145	1590/1636
TFTP	3962077/3965175	1188623/1189552	1698033/1699361

Table 4.3: Data split on each attack type

	Predicted Benign	Predicted Attack
True Benign	25335	58
True Attack	15	37510

Table 4.4: Training Confusion Matrix of PCA

4.4 Dimension Reduction Techniques

In this section, three different dimension reduction techniques are trained and tested. The experiment is using a subset extracted from CICDDoS2019. In this section, only the UDP flooding attacks are tested. The train set has 25393 benign attacks and 37525 UDP attacks. In the test procedure, 3134 benign data and 3754680 UDP data is used. The primary purpose of this section is to choose the best feature reduction and compressing technique for the CICDDoS2019 dataset.

4.4.1 PCA

PCA stands for Principal Component Analysis. It is a type of linear statistical orthogonal transformation tool. It can give the researchers the most uncorrelated variables among all features. The training confusion matrix is shown in Table 4.4. The testing confusion matrix is shown in Table 4.5. Moreover, the ROC curve is shown in Figure 4.4.

	Predicted Benign	Predicted Attack
True Benign	3128	6
True Attack	101	3754579

Table 4.5: Testing Confusion Matrix of PCA

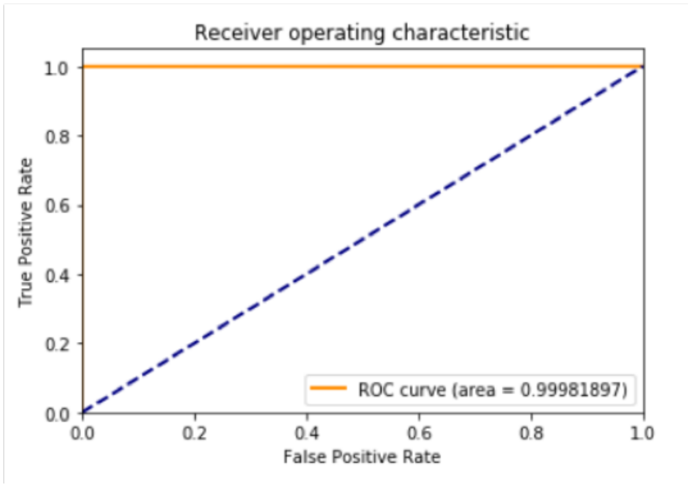


Figure 4.4: ROC of PCA

4.4.2 ANOVA

ANOVA stands for Analysis of variance. It is also a type of statistical model. It can analyze the differences among group means in a sample. The training confusion matrix is shown in Table 4.6. The testing confusion matrix is shown in Table 4.7. Moreover, the ROC curve is shown in Figure 4.5.

4.4.3 Auto Encoder

Auto Encoder is a type of feature compressing model based on Dense Neural Network. It is also a type of non-linear model. It tries to compress the original features into a lower dimension and reconstruct it back to the original features. The training confusion matrix is shown in Table 4.8. The testing confusion matrix is shown in Table 4.9. Moreover, the ROC curve is shown in Figure 4.6.

	Predicted Benign	Predicted Attack
True Benign	25188	0
True Attack	22	37708

Table 4.6: Training Confusion Matrix of ANOVA

	Predicted Benign	Predicted Attack
True Benign	3134	0
True Attack	629	3754051

Table 4.7: Testing Confusion Matrix of ANOVA

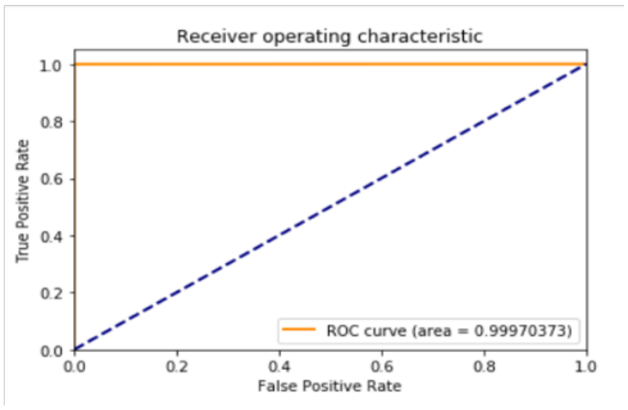


Figure 4.5: ROC of ANOVA

	Predicted Benign	Predicted Attack
True Benign	25372	21
True Attack	15	37510

Table 4.8: Training Confusion Matrix of Auto Encoder

	Predicted Benign	Predicted Attack
True Benign	3125	9
True Attack	101	3754579

Table 4.9: Testing Confusion Matrix of Auto Encoder

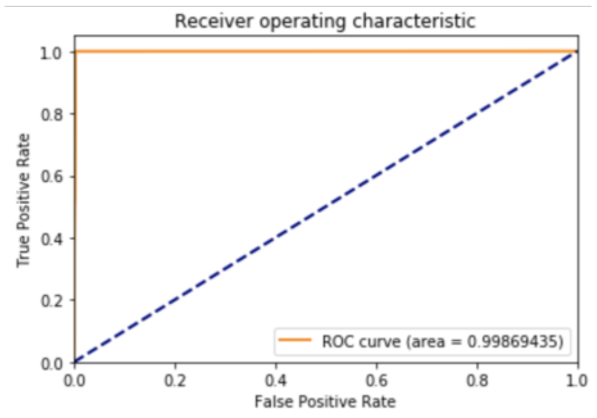


Figure 4.6: ROC of Auto Encoder

After calculating the F1-score for each model, the Auto Encoder outperforms other models. Thus, in the following experiment, Auto Encoder will be used as the feature compressing module.

4.5 Training Auto Encoder

As AutoEncoder captures the non-linear nature of the feature, it shows an excellent fit for the dataset we used for the experimentation. Autoencoder is a neural-network-based feature engineering approach and able to learn the hidden features of the data with an iterative training process. In this process of learning, autoencoder learns the correlation and intermediate relationship between the individual attributes and extract the optimum information from the features. In our experimentation, For the above reasons, autoencoder was finally chosen for extracting representative features.

The python code of building the Auto Encoder is shown in Listing 4.2

Listing 4.2: Auto Enoder code

```python
from keras.layers import Input, Dense
from keras.models import Model

input_data = Input(shape=(37,))
encoded_1 = Dense(20, activation='relu')(input_data)
encoded_2 = Dense(10, activation='relu')(encoded_1)
decoded_1 = Dense(20, activation='sigmoid')(encoded_2)
decoded_2 = Dense(37, activation='sigmoid')(decoded_1)

autoencoder = Model(input_data, decoded_2)
encoder = Model(input_data, encoded_2)

autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

The optimally compressed dimension will be derived from a series of the control experiment. All variables other than the compressed dimension remain the same in every experiment.

dimension	validation loss	dimension	validation loss
20	0.0537	15	0.0537
10	0.0600	5	0.0561

Table 4.10: validation loss of different dimensions

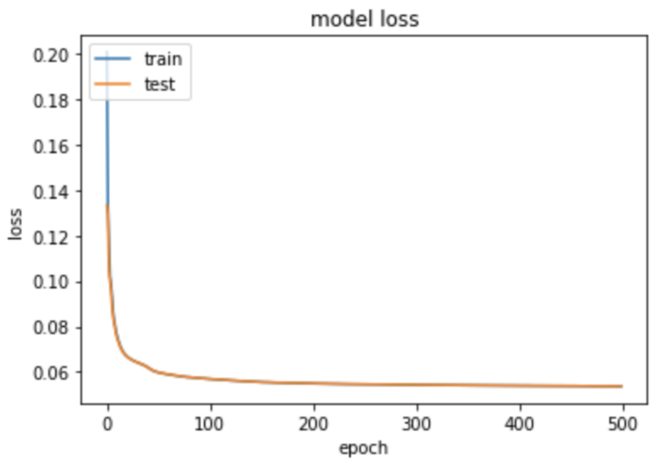


Figure 4.7: Train loss and test loss over epochs

The training details are listed below:

- train set size: 209726
- test set size: 209726
- input features: 37
- number of DDoS: 125386
- number of benign: 84340
- validation split: 0.33
- shuffle: true
- optimizer: adadelta
- loss function: binary_crossentropy
- epochs: 500
- batch_size: 256

The experiment result is shown in Table 4.10

The training loss and test loss over 500 epochs are shown in Figure 4.7.

From this control experiment on Auto Encoder, a conclusion can be drawn. For this dataset, the features can be compressed down to five dimensions with a loss below 0.06. The trained Auto Encoder in this section will be used to compress the data features before feeding the data into the Classification Module.

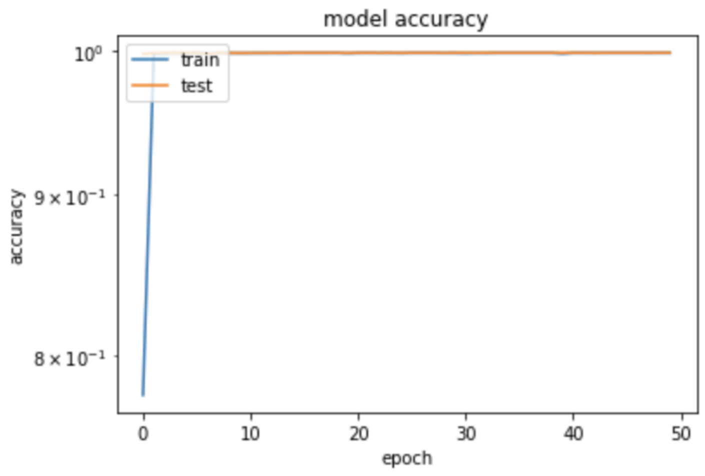


Figure 4.8: DNN accuray over epochs

4.6 DNN Training

4.6.1 Architecture of the Dense Neural Network

A five-layer neural network is constructed, and a broad range of experimentation is executed in regard to selecting the optimal hyperparameters. The first layer of the network has 25 neurons, the second layer has 50 neurons, the third layer has 50 neurons, the fourth layer has 25 neurons, and finally, the output layer has one neuron as this network is configured for binary classification. Tanh and Relu was tested, but they don't give promising results. After testing with several activation function, Sigmoid outperforms others. Sigmoid activation function was used in all the layers.

The data, which is considered as benign according to Snort, was fed into AutoEncoder for dimensionality reduction, then DNN for classification. The DNN classifies those traffic as an attack or not an attack. The classified predictions were used to train and test a decision tree. This process is discussed later in the next section.

The remaining data filtered by Snort has the same features as the original CSVs. It has 83 features, as shown in Table 3.2. Before feeding the data into AutoEncoder, Pandas-profiling was used to analyze the correlations between the features. According to the Pandas-Profiling results, the highly-correlated features were dropped.

4.6.2 Training Details and Results

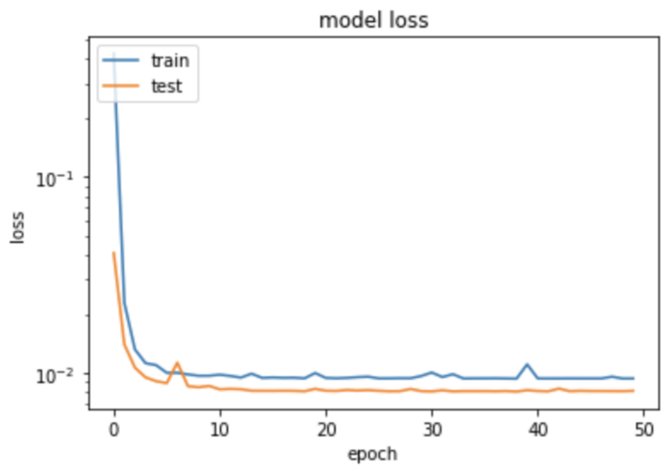


Figure 4.9: DNN loss over epochs

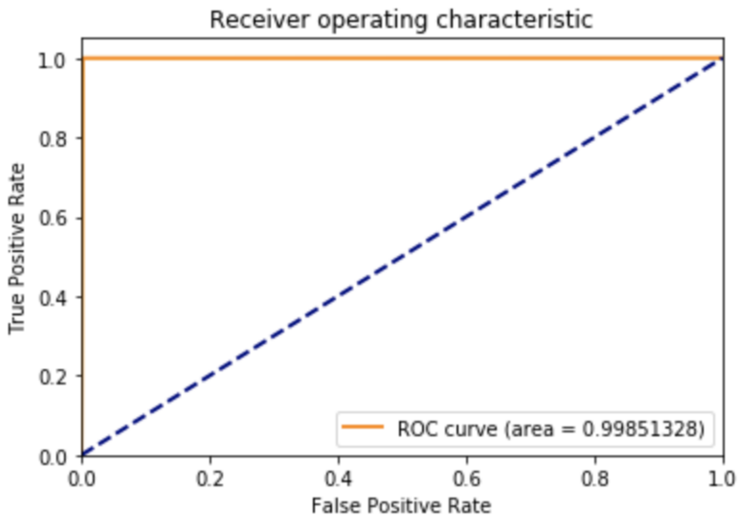


Figure 4.10: ROC of DNN

Attack type	Accuracy	Precision	Recall	F1-Score
SNMP	0.75058	0.75015	0.75119	0.75018
NetBIOS	0.99643	0.99650	0.99621	0.99635
MSSQL	0.94096	0.93752	0.94960	0.94018
UDP	0.99398	0.99345	0.99341	0.99343
SSDP	0.99291	0.99138	0.99229	0.99183
LDAP	0.73057	0.65391	0.56543	0.55831
SYN	0.99984	0.99987	0.99972	0.99979
NTP	0.99719	0.99552	0.99469	0.99502
DNS	0.72671	0.75768	0.72565	0.71763
WebDDoS	0.49270	0.24635	0.50000	0.33007
UDP-Lag	0.93707	0.93704	0.93710	0.93707
TFTP	0.99943	0.99943	0.99943	0.99943

Table 4.11: DNN results

Attack type	Accuracy	Precision	Recall	F1-Score
SNMP	0.88263	0.88192	0.88283	0.88227
NetBIOS	0.99444	0.99413	0.99454	0.99433
MSSQL	0.94087	0.94818	0.94088	0.94128
UDP	0.98747	0.98619	0.98645	0.98632
SSDP	0.98892	0.98959	0.98482	0.98716
LDAP	0.72037	0.62770	0.56909	0.56751
SYN	0.99973	0.99966	0.99962	0.99964
NTP	0.99239	0.98346	0.99237	0.99239
DNS	0.72309	0.76445	0.72189	0.71112
WebDDoS	0.49270	0.24635	0.50000	0.33007
UDP-Lag	0.86578	0.87384	0.86683	0.86527
TFTP	0.99286	0.99293	0.99286	0.99286

Table 4.12: AE-DNN results

Attack type	Accuracy	Precision	Recall	F1-Score
SNMP	0.87664	0.87619	0.87772	0.87643
NetBIOS	0.99409	0.99361	0.99436	0.99398
MSSQL	0.94071	0.93725	0.94931	0.93992
UDP	0.98299	0.97885	0.98445	0.98153
SSDP	0.98255	0.97774	0.98229	0.97997
LDAP	0.72800	0.64595	0.56501	0.55867
SYN	0.99870	0.99770	0.99897	0.99833
NTP	0.98989	0.99148	0.97330	0.98211
DNS	0.68845	0.75553	0.68690	0.66546
WebDDoS	0.93313	0.94177	0.93214	0.93268
UDP-Lag	0.93986	0.94299	0.93929	0.93969
TFTP	0.99717	0.99717	0.99717	0.99717

Table 4.13: PCC-AE-DNN results

Table 4.14: CM

Begin of Table			
Attack and Model	Label	Predicted Benign	Predicted Attack
SNMP, DNN	Benign	1260052	438266
SNMP, DNN	Attack	356549	1131813
SNMP, AE_DNN	Benign	1494303	204015
SNMP, AE_DNN	Attack	170001	1318361
SNMP, PCC_AE_DNN	Benign	1462717	235601
SNMP, PCC_AE_DNN	Attack	157510	1330852
NetBIOS, DNN	Benign	1694820	3828
NetBIOS, DNN	Attack	6825	1275373
NetBIOS, AE_DNN	Benign	1688164	10484
NetBIOS, AE_DNN	Attack	6086	1276112
NetBIOS, PCC_AE_DNN	Benign	1689686	8962
NetBIOS, PCC_AE_DNN	Attack	8990	1273208

Continuation of Table 4.14			
Attack and Model	Label	Predicted Benign	Predicted Attack
MSSQL, DNN	Benign	1527238	170432
MSSQL, DNN	Attack	481	1196867
MSSQL, AE_DNN	Benign	1527161	170509
MSSQL, AE_DNN	Attack	670	1196678
MSSQL, PCC_AE_DNN	Benign	1527115	170555
MSSQL, PCC_AE_DNN	Attack	1096	1196252
UDP, DNN	Benign	1690210	7844
UDP, DNN	Attack	8004	926784
UDP, AE_DNN	Benign	1680987	17067
UDP, AE_DNN	Attack	15928	918860
UDP, PCC_AE_DNN	Benign	1663068	34986
UDP, PCC_AE_DNN	Attack	9809	924979
SSDP, DNN	Benign	1687469	10189
SSDP, DNN	Attack	7448	782851
SSDP, AE_DNN	Benign	1690959	6699
SSDP, AE_DNN	Attack	20868	769431
SSDP, PCC_AE_DNN	Benign	1668808	28850
SSDP, PCC_AE_DNN	Attack	14568	775731
LDAP, DNN	Benign	1599570	98652
LDAP, DNN	Attack	537433	125210
LDAP, AE_DNN	Benign	1552133	146089
LDAP, AE_DNN	Attack	514074	148569
LDAP, PCC_AE_DNN	Benign	1590526	107696
LDAP, PCC_AE_DNN	Attack	534469	128174
SYN, DNN	Benign	1698183	43

Continuation of Table 4.14			
Attack and Model	Label	Predicted Benign	Predicted Attack
SYN, DNN	Attack	331	615175
SYN, AE_DNN	Benign	1697965	261
SYN, AE_DNN	Attack	368	615138
SYN, PCC_AE_DNN	Benign	1695485	2741
SYN, PCC_AE_DNN	Attack	276	615230
NTP, DNN	Benign	1696509	2421
NTP, DNN	Attack	3295	355106
NTP, AE_DNN	Benign	1687845	11086
NTP, AE_DNN	Attack	4564	353837
NTP, PCC_AE_DNN	Benign	1696828	2102
NTP, PCC_AE_DNN	Attack	18693	339708
DNS, DNN	Benign	2726	2224
DNS, DNN	Attack	498	4512
DNS, AE_DNN	Benign	2587	2363
DNS, AE_DNN	Attack	395	4615
DNS, PCC_AE_DNN	Benign	2123	2827
DNS, PCC_AE_DNN	Attack	276	4734
WebDDoS, DNN	Benign	641	0
WebDDoS, DNN	Attack	660	0
WebDDoS, AE_DNN	Benign	641	0
WebDDoS, AE_DNN	Attack	660	0
WebDDoS, PCC_AE_DNN	Benign	554	87
WebDDoS, PCC_AE_DNN	Attack	0	660
UDP-Lag, DNN	Benign	1493	97
UDP-Lag, DNN	Attack	106	1530
UDP-Lag, AE_DNN	Benign	1496	94
UDP-Lag, AE_DNN	Attack	339	1297

Continuation of Table 4.14			
Attack and Model	Label	Predicted Benign	Predicted Attack
UDP-Lag, PCC_AE_DNN	Benign	1430	160
UDP-Lag, PCC_AE_DNN	Attack	34	1602
TFTP, DNN	Benign	1696898	1135
TFTP, DNN	Attack	796	1698565
TFTP, AE_DNN	Benign	1676183	21850
TFTP, AE_DNN	Attack	1794	1697567
TFTP, PCC_AE_DNN	Benign	1691249	6784
TFTP, PCC_AE_DNN	Attack	2844	1696517

4.7 Discussion of the results

As is shown in Section 4.6, three models performs different on different attacks. In this section, the best model for each attack type is selected in the ensembled model. The metrics F1-score is used to select the models to avoid low accuracy when the testing data is highly imbalanced. Taking this into consideration, the selected models are listed in Table 4.15.

From the results, we can conclude that for different attack types, the performance of each model would be different. This result reflects a famous theory: No free lunch.

In comparison with the DNN and DNN with AE, a general conclusion is that the Auto Encoder would bring down the F1-score in most cases. However, for some attack types such as SNMP, the DNN with AE improved the testing F1-score. However, for some attack types such as UDP-Lag, the DNN with AE brought the F1-score from 93,7% to 86,5%.

In comparison with the approaches with and without dropping pandas-profiling-selected columns, the results show in most cases, with dropping the columns, the F1-score is slightly lower than without dropping the columns. This situation may be caused by the loss of information in the dropped columns. However, in some cases

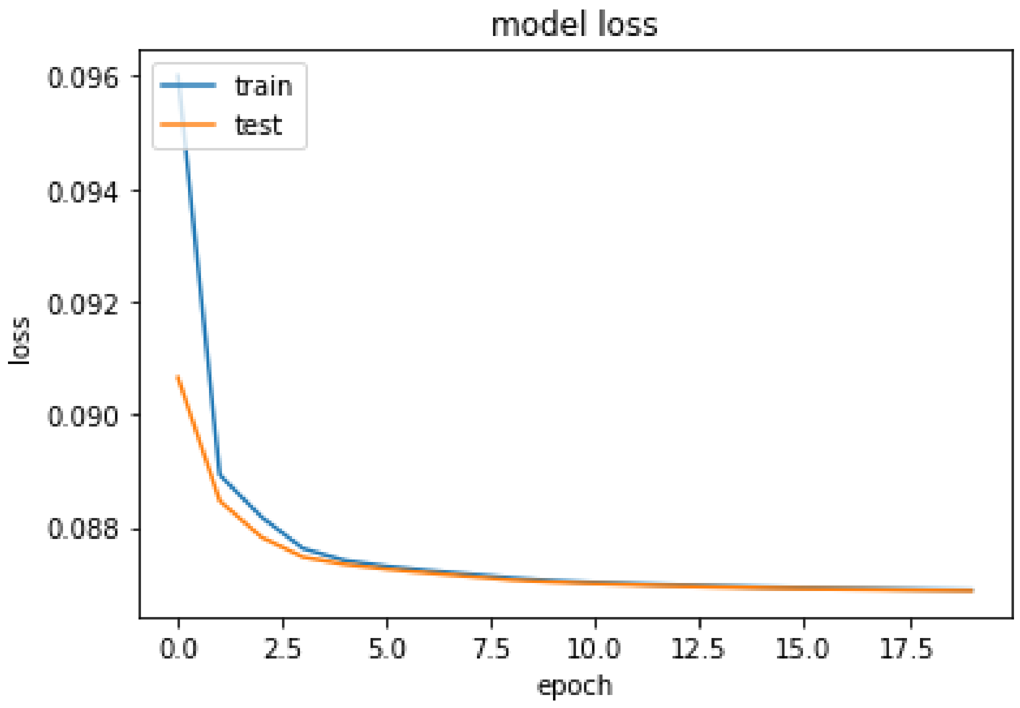


Figure 4.11: SNMP AE loss

Attack Type	Selected Model	F1-score
SNMP	AE_DNN	0.88227
NetBIOS	DNN	0.99635
MSSQL	AE_DNN	0.94128
UDP	DNN	0.99343
SSDP	DNN	0.99183
LDAP	AE_DNN	0.56751
SYN	DNN	0.99979
NTP	DNN	0.99502
DNS	DNN	0.71763
WebDDoS	PCC_AE_DNN	0.93268
UDP-Lag	PCC_AE_DNN	0.93969
TFTP	DNN	0.99943

Table 4.15: Model Selection

such as UDP-Lag, the F1-score is slightly improved with dropping the columns. There is one case that deserves attention, WebDDoS. The DNN and the DNN with AE can not successfully discriminate it from benign attacks, but with dropping the selected columns, the model can precisely tell the attacks from the benign data. The F1-score increases from 33.0% to 93.3%. The reason behind this may be after dropping the columns, the interference factors are gone. Thus, the remaining data has higher purity and becomes less challenging to be determined.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this paper, three different deep learning models are proposed for DDoS detection. The testing results reflect a famous theorem: No Free Lunch. We found that for each DDoS attack type, three models perform differently. For some attack types, using feature reduction or compressing techniques may bring down the F1-score. However, for other attack types, this technique may increase the performance drastically. The F1-score for SNMP increased from 75.0% to 88.2% by applying Auto Encoder. For UDP, the difference is more prominent. Before using Auto Encoder, the F1-score is 0.3%, which means the model misclassified almost all attacks to benign. After applying Auto Encoder, the F1-score is increased to 98.6%. Another example is WebDDoS; both DNN and DNN with Auto Encoder did not give high F1-score. In comparison with the third approach, which integrates pandas-profiling, the F1-score is improved significantly to 93.3%. The results show that the three proposed models outperform each other in some circumstances.

The performance of the proposed models is better than the benchmarks we set in Section 3.3. The proposed models outperform the models in the original paper[8] as well. In the results of Sharafaldin[8], the highest F1-score is 69%. The highest F1-score from benchmarks is 77.393%.

Although DDoS attack remains to be a major type of attacks that happens in the networks. Because DDoS attacks can be launched and reflected by legitimated users on the networks, it is hard for the victims to detect and prevent it. However, many researchers have targeted this problem.

In order to explore the difficulty of detecting DDoS attacks, especially those who have risen in recent years, the CICDDoS2019 dataset is selected as the training and testing set in this paper. To gain the most precise information, we also used CIC-FlowMeter to generate the features directly from raw PCAP files.

5.2 Future work

In the experiment, the models were trained and tested separately. This means the models can only discriminate against an attack from benign traffic. However, in real networks, different types of attacks are mixed with benign traffic. The models trained in this paper can not determine an attack from a mixture like that.

Future researchers should consider, the next step of the research should focus on developing an algorithm to combine the models and build an ensemble classifier. In the ensemble classifier, each model should give a prediction, and the ensemble model should develop a mechanism to make a decision based on the predictions. The algorithm could be a voting system or a weighted mechanism.

Another shortcoming of the research is that the models can not work in real-time. As is known, a server could receive thousands of requests per second. But CIC-FlowMeter can only analyze the packets and output CSV files. If the CIC-FlowMeter can be modified to generate feature tensors and the tensors can be directly fed into the models, a real-time DDoS detection system can be built.

Last but not least, in the dimension reduction part, another important technique is not used. It is independent component analysis. Future researchers should consider using this technique and comparing the testing results.

Although researchers have done a significant amount of work regarding DDoS detection and prevention, the threats and challenges of DDoS will keep showing up in the future. Every small piece of work from each researcher will make a difference in the area. With more novel DDoS attack types rising and more researchers working, DDoS detection techniques will ultimately evolve.

Bibliography

[1] Martin Roesch. “Snort – Lightweight Intrusion Detection for Networks”. en. In: (1999), p. 11.

[2] Nour Moustafa, Jiankun Hu, and Jill Slay. “A holistic review of Network Anomaly Detection Systems: A comprehensive survey”. en. In: *Journal of Network and Computer Applications* 128 (Feb. 2019), pp. 33–55. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2018.12.006. URL: http://www.sciencedirect.com/science/article/pii/S1084804518303886 (visited on 03/06/2020).

[3] Michael Brennan. “Using Snort For a Distributed Intrusion Detection System”. en. In: (2002), p. 12.

[4] Rajendra Patil, Harsha Dudeja, Snehal Gawade, et al. “Protocol Specific Multi-Threaded Network Intrusion Detection System (PM-NIDS) for DoS/DDoS Attack Detection in Cloud”. In: *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. ISSN: null. July 2018, pp. 1–7. DOI: 10.1109/ICCCNT.2018.8494130.

[5] Dinesh Singh, Dhiren Patel, Bhavesh Borisaniya, et al. “Collaborative IDS Framework for Cloud”. en. In: (2016), p. 11.

[6] *Snort - Network Intrusion Detection & Prevention System*. URL: https://www.snort.org/ (visited on 03/16/2020).

[7] CAIDA: Center for Applied Internet Data Analysis. *UCSD Network Telescope Aggregrated DDoS Metadata*. Library Catalog: www.caida.org. URL: https://www.caida.org/data/passive/telescope-ddos.xml (visited on 03/16/2020).

[8] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, et al. “Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy”. In: *2019 International Carnahan Conference on Security Technology (ICCST)*. ISSN: 1071-6572. Oct. 2019, pp. 1–8. DOI: 10.1109/CCST.2019.8888419.

[9] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, et al. "Characterization of Tor Traffic using Time based Features". In: Feb. 2020, pp. 253–262. ISBN: 978-989-758-209-7. URL: `https://www.scitepress.org/PublicationsDetail.aspx?ID=g4gLnPa%2f2OM%3d&t=1` (visited on 02/27/2020).

[10] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:" en. In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116. ISBN: 978-989-758-282-0. DOI: `10.5220/0006639801080116`. URL: `http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006639801080116` (visited on 02/15/2020).

[11] *DDoS 2019 — Datasets — Research — Canadian Institute for Cybersecurity — UNB*. en. Library Catalog: www.unb.ca. URL: `https://www.unb.ca/cic/datasets/ddos-2019.html` (visited on 03/10/2020).

[12] Pearson Karl. *Proceedings of the Royal Society of London*. en. Taylor & Francis, 1895.

[13] Julian Jang-Jaccard and Surya Nepal. "A survey of emerging threats in cybersecurity". en. In: *Journal of Computer and System Sciences*. Special Issue on Dependable and Secure Computing 80.5 (Aug. 2014), pp. 973–993. ISSN: 0022-0000. DOI: `10.1016/j.jcss.2014.02.005`. URL: `http://www.sciencedirect.com/science/article/pii/S0022000014000178` (visited on 03/17/2020).

[14] Jasmeen Chahal, Abhinav Bhandari, and Sunny Behal. "Distributed Denial of Service Attacks: A Threat or Challenge". In: *New Review of Information Networking* 24 (Apr. 2019), pp. 31–103. DOI: `10.1080/13614576.2019.1611468`.

[15] Christos Douligeris and Aikaterini Mitrokotsa. *DDoS attacks and defense mechanisms: classification and state-of-the-art*. 2004.

[16] Srikanth Kandula, Dina Katabi, Matthias Jacob, et al. "Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds". en. In: (), p. 14.

[17] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. "Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems". In: *Acm Comp. Surv* 39.1 (2007).

[18] Saman Taghavi Zargar, James Joshi, and David Tipper. "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks". en. In: *IEEE Communications Surveys & Tutorials* 15.4 (2013), pp. 2046–2069. ISSN: 1553-877X. DOI: `10.1109/SURV.2013.031413.00127`. URL: `http://ieeexplore.ieee.org/document/6489876/` (visited on 03/17/2020).

[19] Thuy T.T. Nguyen and Grenville Armitage. "A survey of techniques for internet traffic classification using machine learning". en. In: *IEEE Communications Surveys & Tutorials* 10.4 (2008), pp. 56–76. ISSN: 1553-877X. DOI: `10.1109/SURV.2008.080406`. URL: `http://ieeexplore.ieee.org/document/4738466/` (visited on 03/17/2020).

[20] Michał Woźniak, Manuel Graña, and Emilio Corchado. "A survey of multiple classifier systems as hybrid systems". en. In: *Information Fusion* 16 (Mar. 2014), pp. 3–17. ISSN: 15662535. DOI: `10.1016/j.inffus.2013.04.006`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S156625351300047X` (visited on 03/17/2020).

[21] Rajendra Patil, Harsha Dudeja, and Chirag Modi. "Designing an efficient security framework for detecting intrusions in virtual network of cloud computing". en. In: *Computers & Security* 85 (Aug. 2019), pp. 402–422. ISSN: 0167-4048. DOI: `10.1016/j.cose.2019.05.016`. URL: `http://www.sciencedirect.com/science/article/pii/S0167404818310629` (visited on 03/06/2020).

[22] José Francisco Colom, David Gil, Higinio Mora, et al. "Scheduling framework for distributed intrusion detection systems over heterogeneous network architectures". en. In: *Journal of Network and Computer Applications* 108 (Apr. 2018), pp. 76–86. ISSN: 1084-8045. DOI: `10.1016/j.jnca.2018.02.004`. URL: `http://www.sciencedirect.com/science/article/pii/S1084804518300419` (visited on 03/05/2020).

[23] Z. Chiba, N. Abghour, K. Moussaid, et al. "A Cooperative and Hybrid Network Intrusion Detection Framework in Cloud Computing Based on Snort

and Optimized Back Propagation Neural Network". en. In: *Procedia Computer Science*. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops 83 (Jan. 2016), pp. 1200–1206. ISSN: 1877-0509. DOI: `10.1016/j.procs.2016.04.249`. URL: `http://www.sciencedirect.com/science/article/pii/S1877050916302824` (visited on 03/05/2020).

[24] Zayed Al Haddad, Mostafa Hanoune, and Abdelaziz Mamouni. "A collaborative framework for intrusion detection (C-NIDS) in Cloud computing". In: *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*. ISSN: null. May 2016, pp. 261–265. DOI: `10.1109/CloudTech.2016.7847708`.

[25] Massimo Ficco, Luca Tasquier, and Rocco Aversa. "Intrusion Detection in Cloud Computing". In: *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. ISSN: null. Oct. 2013, pp. 276–283. DOI: `10.1109/3PGCIC.2013.47`.

[26] Vinod Kumar and Dr Om Prakash Sangwan. "Signature Based Intrusion Detection System Using SNORT". en. In: *International Journal of Computer Applications* (2012), p. 7.

[27] Hesham Altwaijry and Khalid Shahbar. "(WHASG) Automatic SNORT Signatures Generation by using Honeypot". en. In: *Journal of Computers* 8.12 (Dec. 2013), pp. 3280–3286. ISSN: 1796-203X. DOI: `10.4304/jcp.8.12.3280-3286`. URL: `http://ojs.academypublisher.com/index.php/jcp/article/view/11506` (visited on 03/05/2020).

[28] Martin Roesch. *Writing Snort Rules*. 2001. URL: `https://paginas.fe.up.pt/~mgi98020/pgr/writing_snort_rules.htm#includes` (visited on 02/18/2020).

[29] Brian Caswell, James C. Foster, Ryan Russell, et al. *Snort 2.0 Intrusion Detection*. Syngress Publishing, 2003. ISBN: 978-1-931836-74-6.

[30] Keyvan Karami, Saeed Zerehdaran, Ali Javadmanesh, et al. "Characterization of bovine (Bos taurus) imprinted genes from genomic to amino acid attributes by data mining approaches." English. In: *PLoS ONE* 14.6 (June 2019). Publisher: Public Library of Science, e0217813–e0217813. ISSN: 19326203. URL: `https://go.gale.com/ps/i.do?p=AONE&sw=w&issn=19326203&v=2.1&it=r&id=GALE%7CA587945359&sid=googleScholar&linkaccess=abs` (visited on 03/22/2020).

[31] Chirag N. Modi, Dhiren R. Patel, Avi Patel, et al. "Bayesian Classifier and Snort based network intrusion detection system in cloud computing". In: *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)*. ISSN: null. July 2012, pp. 1–7. DOI: `10.1109/ICCCNT.2012.6396086`.

[32] Qin Zhao, Jizhou Sun, and Song Zhang. "A hybrid and hierarchical NIDS paradigm utilizing naive Bayes classifier". In: *Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04CH37513)*. Vol. 1. ISSN: 0840-7789. May 2004, 145–148 Vol.1. DOI: `10.1109/CCECE.2004.1344977`.

[33] Dishan Jing and Hai-Bao Chen. "SVM Based Network Intrusion Detection for the UNSW-NB15 Dataset". In: *2019 IEEE 13th International Conference on ASIC (ASICON)*. ISSN: 2162-7541. Oct. 2019, pp. 1–4. DOI: `10.1109/ASICON47005.2019.8983598`.

[34] Dai Hong and Li Haibo. "A Lightweight Network Intrusion Detection Model Based on Feature Selection". In: *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*. ISSN: null. Nov. 2009, pp. 165–168. DOI: `10.1109/PRDC.2009.34`.

[35] J. Zhang and M. Zulkernine. "A hybrid network intrusion detection technique using random forests". In: *First International Conference on Availability, Reliability and Security (ARES'06)*. ISSN: null. Apr. 2006, 8 pp.–269. DOI: `10.1109/ARES.2006.7`.

[36] Yuyang Zhou, Guang Cheng, Shanqing Jiang, et al. "An Efficient Intrusion Detection System Based on Feature Selection and Ensemble Classifier". In:

arXiv:1904.01352 [cs] (Sept. 2019). arXiv: 1904.01352. URL: http://arxiv.org/abs/1904.01352 (visited on 03/06/2020).

[37] Hassan Hadi Al-Maksousy, Michele C. Weigle, and Cong Wang. "NIDS: Neural Network based Intrusion Detection System". In: *2018 IEEE International Symposium on Technologies for Homeland Security (HST)*. ISSN: null. Oct. 2018, pp. 1–6. DOI: 10.1109/THS.2018.8574174.

[38] R Vinayakumar, K P Soman, and Prabaharan Poornachandran. "Applying convolutional neural network for network intrusion detection". In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. ISSN: null. Sept. 2017, pp. 1222–1228. DOI: 10.1109/ICACCI.2017.8126009.

[39] Li Yong and Zhang Bo. "An Intrusion Detection Model Based on Multi-scale CNN". In: *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. ISSN: null. Mar. 2019, pp. 214–218. DOI: 10.1109/ITNEC.2019.8729261.

[40] Riaz Ullah Khan, Xiaosong Zhang, Mamoun Alazab, et al. "An Improved Convolutional Neural Network Model for Intrusion Detection in Networks". In: *2019 Cybersecurity and Cyberforensics Conference (CCC)*. ISSN: null. May 2019, pp. 74–77. DOI: 10.1109/CCC.2019.000-6.

[41] Yihan Xiao, Cheng Xing, Taining Zhang, et al. "An Intrusion Detection Model Based on Feature Reduction and Convolutional Neural Networks". In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 42210–42219. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2904620.

[42] Hongyu Yang and Fengyan Wang. "Wireless Network Intrusion Detection Based on Improved Convolutional Neural Network". In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 64366–64374. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2917299.

[43] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, et al. "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks". In: *IEEE Access* 5 (2017). Conference Name: IEEE Access, pp. 21954–21961. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2762418.

[44] Ayyaz-Ul-Haq Qureshi, Hadi Larijani, Jawad Ahmad, et al. “A Novel Random Neural Network Based Approach for Intrusion Detection Systems”. In: *2018 10th Computer Science and Electronic Engineering (CEEC)*. ISSN: null. Sept. 2018, pp. 50–55. DOI: `10.1109/CEEC.2018.8674228`.

[45] Sara Althubiti, William Nick, Janelle Mason, et al. “Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection”. In: *SoutheastCon 2018*. ISSN: 1091-0050. Apr. 2018, pp. 1–5. DOI: `10.1109/SECON.2018.8478898`.

[46] Fanzhi Meng, Yunsheng Fu, Fang Lou, et al. “An Effective Network Attack Detection Method Based on Kernel PCA and LSTM-RNN”. In: *2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)*. ISSN: null. Dec. 2017, pp. 568–572. DOI: `10.1109/ICCSEC.2017.8447022`.

[47] Thi-Thu-Huong Le, Jihyun Kim, and Howon Kim. “An Effective Intrusion Detection Classifier Using Long Short-Term Memory with Gradient Descent Optimization”. In: *2017 International Conference on Platform Technology and Service (PlatCon)*. ISSN: null. Feb. 2017, pp. 1–6. DOI: `10.1109/PlatCon.2017.7883684`.

[48] Karan Bajaj and Amit Arora. “Improving the Intrusion Detection using Discriminative Machine Learning Approach and Improve the Time Complexity by Data Mining Feature Selection Methods”. en. In: *International Journal of Computer Applications* 76.1 (Aug. 2013), pp. 5–11. ISSN: 09758887. DOI: `10.5120/13209-0587`. URL: `http://research.ijcaonline.org/volume76/number1/pxc3890587.pdf` (visited on 03/03/2020).

[49] Shahadate Rezvy, Miltos Petridis, Aboubaker Lasebae, et al. “Intrusion Detection and Classification with Autoencoded Deep Neural Network”. en. In: *Innovative Security Solutions for Information Technology and Communications*. Ed. by Jean-Louis Lanet and Cristian Toma. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 142–156. ISBN: 978-3-030-12942-2. DOI: `10.1007/978-3-030-12942-2_12`.

[50] Chencheng Ma, Xuehui Du, and Lifeng Cao. "Analysis of Multi-Types of Flow Features Based on Hybrid Neural Network for Improving Network Anomaly Detection". In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 148363–148380. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2019.2946708`.

[51] Zouhair Chiba, Noureddine Abghour, Khalid Moussaid, et al. "A novel architecture combined with optimal parameters for back propagation neural networks applied to anomaly network intrusion detection". en. In: *Computers & Security* 75 (June 2018), pp. 36–58. ISSN: 0167-4048. DOI: `10.1016/j.cose.2018.01.023`. URL: `http://www.sciencedirect.com/science/article/pii/S0167404818300543` (visited on 02/04/2020).

[52] Varun Chandola, Eric Eilertson, Levent Ertoz, et al. "Minds: Architecture & Design". en. In: *Data Warehousing and Data Mining Techniques for Cyber Security*. Ed. by Anoop Singhal. Advances in Information Security. Boston, MA: Springer US, 2007, pp. 83–107. ISBN: 978-0-387-47653-7. DOI: `10.1007/978-0-387-47653-7_6`. URL: `https://doi.org/10.1007/978-0-387-47653-7_6` (visited on 03/06/2020).

[53] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, et al. "Characterization of Encrypted and VPN Traffic using Time-related Features". In: Feb. 2020, pp. 407–414. ISBN: 978-989-758-167-0. URL: `http://www.scitepress.org/DigitalLibrary/PublicationsDetail.aspx?ID=cazNsYLjzhw=&t=1` (visited on 02/27/2020).

[54] *Attackers Using New MS SQL Reflection Techniques - The Akamai Blog*. URL: `https://blogs.akamai.com/2015/02/plxsert-warns-of-ms-sql-reflection-attacks.html` (visited on 03/11/2020).

[55] *SSDP DDoS Attack*. en-us. Library Catalog: www.cloudflare.com. URL: `https://www.cloudflare.com/learning/ddos/ssdp-ddos-attack/` (visited on 03/11/2020).

[56] *What is an SSDP DDoS attack?* en-US. Library Catalog: ddos-guard.net. URL: `http://ddos-guard.net` (visited on 03/11/2020).

[57] *What is NTP Amplification — DDoS Attack Glossary — Imperva*. en-US. Library Catalog: www.imperva.com Section: DDoS. URL: `https://www.imperva.com/learn/application-security/ntp-amplification/` (visited on 03/11/2020).

[58] Boris Sieklik, Richard Macfarlane, and William Buchanan. "TFTP DDoS amplification attack". In: *Computers & Security* 57 (Oct. 2015). DOI: `10.1016/j.cose.2015.09.006`.

[59] Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis, et al. "Detecting DNS Amplification Attacks". In: Oct. 2007, pp. 185–196. DOI: `10.1007/978-3-540-89173-4_16`.

[60] *DNS Amplification DDoS Attack*. en-us. Library Catalog: www.cloudflare.com. URL: `https://www.cloudflare.com/learning/ddos/dns-amplification-ddos-attack/` (visited on 03/13/2020).

[61] *UDP-Based Amplification Attacks — CISA*. URL: `https://www.us-cert.gov/ncas/alerts/TA14-017A` (visited on 03/13/2020).

[62] *Attackers are now abusing exposed LDAP servers to amplify DDoS attacks*. en. Library Catalog: www.pcworld.com. Oct. 2016. URL: `https://www.pcworld.com/article/3135771/attackers-are-now-abusing-exposed-ldap-servers-to-amplify-ddos-attacks.html` (visited on 03/13/2020).

[63] Imperva Impreva. *What is SNMP Reflection and Amplification — DDoS Attack Glossary — Imperva*. en-US. Library Catalog: www.imperva.com Section: Threats. URL: `https://www.imperva.com/learn/application-security/snmp-reflection/` (visited on 03/13/2020).

[64] *SYN Flood DDoS Attack*. en-us. Library Catalog: www.cloudflare.com. URL: `https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/` (visited on 03/13/2020).

[65] D.H. Wolpert and W.G. Macready. "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 67–82. ISSN: 1941-0026. DOI: `10.1109/4235.585893`.

[66] *sklearn.preprocessing.LabelEncoder — scikit-learn 0.22.1 documentation.* URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html` (visited on 02/24/2020).

[67] Yi-Chi Wu, Huei-Ru Tseng, Wuu Yang, et al. "DDoS Detection and Traceback with Decision Tree and Grey Relational Analysis". In: *2009 Third International Conference on Multimedia and Ubiquitous Engineering.* ISSN: null. June 2009, pp. 306–314. DOI: `10.1109/MUE.2009.60`.

[68] Yixin Chen, Jianing Pei, and Defang Li. "DETPro: A High-Efficiency and Low-Latency System Against DDoS Attacks in SDN Based on Decision Tree". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC).* ISSN: 1550-3607. May 2019, pp. 1–6. DOI: `10.1109/ICC.2019.8761580`.

[69] S. Brahanyaa and L. Jani Anbarasi. "Classification of SNMP Network Dataset for DDoS attack prevention". In: *2018 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC).* ISSN: 2471-7851. Dec. 2018, pp. 1–5. DOI: `10.1109/ICCIC.2018.8782319`.

[70] *sklearn.tree.DecisionTreeClassifier — scikit-learn 0.22.1 documentation.* URL: `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html` (visited on 02/26/2020).

[71] Jan Brabec and Lukas Machlica. "Decision-Forest Voting Scheme for Classification of Rare Classes in Network Intrusion Detection". In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC).* ISSN: 1062-922X. Oct. 2018, pp. 3325–3330. DOI: `10.1109/SMC.2018.00563`.

[72] *3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.22.1 documentation.* URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html` (visited on 02/26/2020).

[73] Hui Zhang, Ping Chen, and Qiang Wang. "Fault Diagnosis Method Based on EEMD and Multi-Class Logistic Regression". In: *2018 3rd International Conference on Smart City and Systems Engineering (ICSCSE).* ISSN: null. Dec. 2018, pp. 859–863. DOI: `10.1109/ICSCSE.2018.00185`.

[74] Rohan Bapat, Abhijith Mandya, Xinyang Liu, et al. “Identifying malicious botnet traffic using logistic regression”. In: *2018 Systems and Information Engineering Design Symposium (SIEDS)*. ISSN: null. Apr. 2018, pp. 266–271. DOI: 10.1109/SIEDS.2018.8374749.

[75] Basant Subba, Santosh Biswas, and Sushanta Karmakar. “Intrusion Detection Systems using Linear Discriminant Analysis and Logistic Regression”. In: *2015 Annual IEEE India Conference (INDICON)*. ISSN: 2325-9418. Dec. 2015, pp. 1–6. DOI: 10.1109/INDICON.2015.7443533.

[76] Muhammad Hilmi Kamarudin, Carsten Maple, Tim Watson, et al. “Packet Header Intrusion Detection with Binary Logistic Regression Approach in Detecting R2L and U2R Attacks”. In: *2015 Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensic (CyberSec)*. ISSN: null. Oct. 2015, pp. 101–106. DOI: 10.1109/CyberSec.2015.28.

[77] Partha Ghosh and Rajarshee Mitra. “Proposed GA-BFSS and logistic regression based intrusion detection system”. In: *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*. ISSN: null. Feb. 2015, pp. 1–6. DOI: 10.1109/C3IT.2015.7060117.

[78] *sklearn.linear_model.LogisticRegression — scikit-learn 0.22.1 documentation.* URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visited on 02/26/2020).

[79] S. Varuna and P. Natesan. “An integration of k-means clustering and naïve bayes classifier for Intrusion Detection”. In: *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*. ISSN: null. Mar. 2015, pp. 1–5. DOI: 10.1109/ICSCN.2015.7219835.

[80] Saqr Mohammed Almansob and Santosh Shivajirao Lomte. “Addressing challenges for intrusion detection system using naive Bayes and PCA algorithm”. In: *2017 2nd International Conference for Convergence in Technology (I2CT)*. ISSN: null. Apr. 2017, pp. 565–568. DOI: 10.1109/I2CT.2017.8226193.

[81] Wei Li and QingXia Li. “Using Naive Bayes with AdaBoost to Enhance Network Anomaly Intrusion Detection”. In: *2010 Third International Conference on Intelligent Networks and Intelligent Systems*. ISSN: null. Nov. 2010, pp. 486–489. DOI: 10.1109/ICINIS.2010.133.

[82] Anish Halimaa A. and K. Sundarakantham. “Machine Learning Based Intrusion Detection System”. In: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. ISSN: null. Apr. 2019, pp. 916–920. DOI: 10.1109/ICOEI.2019.8862784.

[83] Sanjai Veetil and Qigang Gao. “Chapter 18 - Real-time Network Intrusion Detection Using Hadoop-Based Bayesian Classifier”. en. In: *Emerging Trends in ICT Security*. Ed. by Babak Akhgar and Hamid R. Arabnia. Boston: Morgan Kaufmann, Jan. 2014, pp. 281–299. ISBN: 978-0-12-411474-6. DOI: 10.1016/B978-0-12-411474-6.00018-9. URL: http://www.sciencedirect.com/science/article/pii/B9780124114746000189 (visited on 02/07/2020).

[84] Mrutyunjaya Panda, Ajith Abraham, and Manas Ranjan Patra. “Discriminative multinomial Naïve Bayes for network intrusion detection”. In: *2010 Sixth International Conference on Information Assurance and Security*. ISSN: null. Aug. 2010, pp. 5–10. DOI: 10.1109/ISIAS.2010.5604193.

[85] Karuna S. Bhosale, Maria Nenova, and Georgi Iliev. “Modified Naive Bayes Intrusion Detection System (MNBIDS)”. In: *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*. ISSN: null. Dec. 2018, pp. 291–296. DOI: 10.1109/CTEMS.2018.8769248.

[86] *sklearn.naive_bayes.MultinomialNB — scikit-learn 0.22.1 documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html (visited on 02/26/2020).

[87] H. Sun, Y. Zhaung, and H. J. Chao. “A Principal Components Analysis-Based Robust DDoS Defense System”. In: *2008 IEEE International Conference on Communications*. ISSN: 1938-1883. May 2008, pp. 1663–1669. DOI: 10.1109/ICC.2008.321.

[88] Samrat Kumar Dey and Md. Mahbubur Rahman. "Flow Based Anomaly Detection in Software Defined Networking: A Deep Learning Approach With Feature Selection Method". In: *2018 4th International Conference on Electrical Engineering and Information Communication Technology (iCEEiCT)*. ISSN: null. Sept. 2018, pp. 630–635. DOI: `10.1109/CEEICT.2018.8628069`.

[89] *pandas-profiling/pandas-profiling*. original-date: 2016-01-09T23:47:55Z. Feb. 2020. URL: `https://github.com/pandas-profiling/pandas-profiling` (visited on 02/16/2020).

[90] Tom Fawcett. "An introduction to ROC analysis". en. In: *Pattern Recognition Letters* 27.8 (June 2006), pp. 861–874. ISSN: 01678655. DOI: `10.1016/j.patrec.2005.10.010`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S016786550500303X` (visited on 02/26/2020).

[91] AHLashkari. *ahlashkari/CICFlowMeter*. original-date: 2018-02-12T16:57:30Z. Mar. 2020. URL: `https://github.com/ahlashkari/CICFlowMeter` (visited on 03/10/2020).

[92] Wei Wang, Xiangliang Zhang, Sylvain Gombault, et al. "Attribute Normalization in Network Intrusion Detection". In: Dec. 2009, pp. 448–453. DOI: `10.1109/I-SPAN.2009.49`.

[93] *6.3. Preprocessing data — scikit-learn 0.22.2 documentation*. URL: `https://scikit-learn.org/stable/modules/preprocessing.html` (visited on 03/08/2020).

[94] *Compare the effect of different scalers on data with outliers — scikit-learn 0.22.2 documentation*. URL: `https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html` (visited on 03/08/2020).