NETWORK ANOMALY DETECTION

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER FOR
THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2024

Student id: 10783788

Department of Computer Science

Contents

Abstract **7**

Declaration **8**

Copyright **9**

Acknowledgements **10**

1 Introduction **11**

1.1 Motivation 12
1.2 Objectives 12
1.3 Report Structures 13

2 Background **14**

2.1 Network Anomaly Detection 14
2.1.1 Traditional Network Anomaly Detection Systems 14
2.1.2 Application of Machine Learning in Network Anomaly Detection 15
2.2 Machine Learning Algorithm 15
2.2.1 Algorithm Selections 16
2.3 Dataset Introduction 17
2.3.1 CIC_IoT23 18
2.3.2 UGR'16 20

3 Design **22**

3.1 Data Preprocessing 22
3.1.1 CIC_IOT23 23

3.1.2 UGR'16 24
3.1.3 Normalization 28
3.2 Machine Learning Algorithms 29
3.2.1 Random Forest 29
3.2.2 SVM 31
3.2.3 CNN 34
3.3 Evaluation and Visualization 37

4 Experiments **41**
4.1 Environment Setup 41
4.2 CIC_IoT23 42
4.2.1 Random Forest Results 43
4.2.2 SVM Results 44
4.2.3 CNN Results 46
4.3 UGR'16 48
4.3.1 Random Forest Results 48
4.3.2 SVM Results 49
4.3.3 CNN Results 52
4.4 Result Analysis 54

5 Conclusion **56**
5.1 summary 56
5.2 Achievements 57
5.3 Future Work 57

References **58**

Word Count: 10012

List of Tables

2.1 Number of Features and Classes of Different Datasets 17
2.2 Details of features in CIC_IoT23 19

3.1 Numbers of flows in each .csv file 23
3.2 Number of Each Class in Train Test Set in CIC_IoT23 24
3.3 Numbers and Size of dataset for each day of UGR'16 25
3.4 Numbers of different classes in each day 25
3.5 Example of Network Flows without Label 26
3.6 Example of One-hot Encoded Sample 27
3.7 Introduction to Different Metrics 37

4.1 Accuracy with Differen Hyperparameters on CIC_IoT23 with Random Forest 43
4.2 Time Spent with Differen Hyperparameters on CIC_IoT23 with Random Forest 43
4.3 Different Metrics for Each Class of Random Forest Model on CIC_IoT23 44
4.4 Different Metrics for Each Class of SVM Model on CIC_IoT23 45
4.5 Different Metrics for Each Class of CNN Model on CIC_IoT23 47
4.6 Accuracy with Differen Hyperparameters on UGR'16 with Random Forest 49
4.7 Time Spent with Differen Hyperparameters on CIC_IoT23 with Random Forest 49
4.8 Different Metrics for Each Class of Random Forest Model on UGR'16 . 49
4.9 Accuracy with Differen Hyperparameters on UGR'16 with SVM 50
4.10 Time Spent with Differen Hyperparameters on CIC_IoT23 with SVM . 51
4.11 Different Metrics for Each Class of SVM Model on UGR'16 51
4.12 Different Metrics for Each Class of SVM Model on UGR'16 53

4.13 Accuracies for All Models 54

List of Figures

2.1 Number of instances for different classes of CIC_IoT23 20

3.1 Illustration of Random Forest's majority voting mechanism 30
3.2 Hyperplanes of SVM 32
3.3 Structure of CNN model 36
3.4 Screenshot of ExplainerDashboard 38

4.1 Confusion Matrix for Primitive Experiments on CIC_IOT 42
4.2 Confusion Matrix of Random Forest Model on CIC_IoT23 44
4.3 SHAP Values for CIC_IOT23 45
4.4 Confusion Matrix of SVM Model on CIC_IoT23 46
4.5 Loss Reduction Trend over Epochs on CIC_IOT23 47
4.6 Confusion Matrix of SVM Model on CIC_IoT23 48
4.7 Confusion Matrix of Random Forest Model on UGR'16 50
4.8 Confusion Matrix of SVM Model on UGR'16 52
4.9 Loss Reduction Trend over Epochs on UGR'16 52
4.10 Confusion Matrix of CNN Model on UGR'16 53

Abstract

NETWORK ANOMALY DETECTION
Sihan Zeng
A dissertation submitted to The University of Manchester
for the degree of Bachelor of Science, 2024

With the rapid development of internet technology, cybersecurity has become a global focus. Although existing Intrusion Detection Systems (IDS) can protect networks from unauthorized access threats to some extent, these traditional technologies are often inadequate when facing increasingly complex cyberattacks.

Machine Learning (ML) technology, with its powerful data analysis and pattern recognition capabilities, offers a new way to detect anomalous network behaviours. By learning and analyzing network traffic data, ML algorithms can effectively identify and classify normal and abnormal traffic. This project aims to explore and evaluate the effectiveness of different ML algorithms in network anomaly detection.

This research is based on two trustworthy open-source datasets, CIC_IOT23 and UGR'16, which respectively represent the network traffic characteristics in the Internet of Things (IoT) environments and large-scale network environments. After basic exploration of various ML algorithms for network anomaly detection, this project has selected three representative ML algorithms for further study: Random Forest, Convolutional Neural Networks (CNN), and Support Vector Machine (SVM). These algorithms are widely applied in the fields of data classification and pattern recognition and have shown good performance. Experimental results indicate that these three ML algorithms can achieve high accuracy and detection efficiency in network anomaly detection tasks.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see http://www.library.manchester.ac.uk/about/regulations/) and in The University's policy on presentation of Theses

Acknowledgements

Personally, this project is challenging for me. I cloud not have completed this project without the people who helped and supported me in my 3rd year. Hence, I wish to acknowledge my supervisor Dr. Ahmed Saeed and Dr. Hongpeng Zhou, for their continuous support and for guiding me through numerous challenges during the project's implementation.

Chapter 1

Introduction

My project aims to implement and compare different machine learning algorithms on two distinct network flow datasets. Primarily, three algorithms were implemented: Random Forest, SVM, and CNN. The two datasets used are CIC_IOT23 and UGR'16. Models based on these three algorithms were trained on these two datasets. The performance of these models in classifying different types of network flows was evaluated. Finally, there are three trained models for each dataset, six models in total. To understand how different algorithms perform in network anomaly detection, my project studies and compares the results these models generated as classifiers. It also aims to expose the unique advantages and potential disadvantages of using these algorithms to recognize and classify network flows. Since CIC_IOT23 and UGR'16 are two large datasets obtained from the real Internet environment, this project can evaluate the applicability and effectiveness of the three algorithms in real-world.

Before training the models, the datasets were sampled and pre-processed to ensure the samples were available and suitable for model training. During the model training phase, special attention was paid to the adjust of hyperparameters and model structures to ensure the models could fully learn and accurately classify different types of network flows. Furthermore, the SHAP method was adopted to evaluate the importance of features, enabling this project to explore the key factors affecting model performance.

Overall, through comprehensive comparison and in-depth analysis, my project aims to identify machine learning algorithms that are more effective and accurate in network anomaly detection. Through experimental verification, this project demonstrates the

application potential of Random Forest, CNN, and SVM in network anomaly detection.

1.1 Motivation

Before machine learning techniques were applied to network anomaly detection, traditional IDS was primarily based on signature-based and behavior analysis techniques. These techniques were effective in detecting known anomalies, but they were facing some important flaws. Firstly, the traditional IDS based on matching signature and rules are weak at detecting zero-day attacks and unknown threats. What's more, IDS based on signature might have a high false-positive rate. Furthermore, as a lot of new threats appear, signatures databases need maintenance very often, which comes with high costs and leads to detection delays.

Hence, the machine learning techniques were introduced to network anomaly detection to overcome the above flaws. There are some advantages of network anomaly detection systems with machine learning. Firstly, IDS with machine learning can learn the difference between normal and abnormal behaviours of network flows to recognize unknown and zero-day attacks. Secondly, through learning the patterns of the network flows, IDS with machine learning can reduce the false-positive rate, since they can tell if the flow is normal based on the many features, not only a few key signatures. What's more, machine learning models can continuously learn from the latest network flow data and implement incremental learning to adapt to the changeable and unpredictable Internet environment. This approach may reduce a lot of cost compared to the traditional IDS.

1.2 Objectives

Further in the context discussed in Chapter 1, there were a few main objectives of this project. Firstly, I need to choose what machine learning algorithms should be implemented and evaluated. After research that is discussed in Section 2.1.1, Random Forest, CNN and SVM are selected. Secondly, suitable datasets should be selected, which should not be outdated and whose features and labels should be proper for machine

learning algorithms to learn and make classifications. CIC_IoT23 and UGR'16 are selected as discussed in Section 2.3. What's more, different models with selected algorithms need to be evaluated and compared by different metrics like accuracy, recall, precision, and F1-Score. And finally, to display the results of the model performance more intuitively, visualization should be implemented.

1.3 Report Structures

This report includes four main sections. First, it illustrates the technical background information for the project, including machine learning algorithms and how they are used in network anomaly detection, and an introduction to the open-source network flows dataset. Second, there is the description of my design and implementation of the machine learning models for network anomaly detection. Before the models are trained, detailed decisions and information of how the dataset should be preprocessed are discussed. It also discusses how models are designed to be a classifier of network flows and how they should be evaluated. After that, there will be an explanation of how I implemented the machine learning models, and how they can be used to classify the network flows among the two datasets. And finally, the results of my models will be evaluated, and a conclusion will be discussed.

Chapter 2

Background

In this chapter, the technical background information needed in this project will be discussed. First, I will discuss the basic concepts of network anomaly detection, and some disadvantages of traditional network anomaly detection systems. Then, I will discuss how machine learning techniques are used to help improve the flaws mentioned above. I will also provide detailed information on which machine learning algorithms are selected. After that, I will discuss the selected datasets and provide an introduction to them.

2.1 Network Anomaly Detection

Network anomaly detection includes the identification and analysis of unusual patterns or behaviours in network traffic that deviate from the normal flows. It can indicate the potential problems like equipment failures or cyber attacks. Network anomaly detection is crucial for the maintenance of network integrity and security, as it can detect issues early and prevent network disruption or data damage.[1]

2.1.1 Traditional Network Anomaly Detection Systems

Basically, there are two types of anomaly detection systems. One is Signatures or knowledge-based, and the other is Baseline or statistical-based. Signatures or knowledge-based detection relies on matching predefined patterns or characteristics, such as IP addresses or specific packet content. Baseline or statistical-based detection uses historical

data to establish a normal behavior pattern, such as the typical number of TCP connections, and identifies anomalies when current activities significantly deviate from this established baseline.[1] As the internet environment is changing and developing very fast, the traditional network anomaly detection systems based on signature and baseline show their weaknesses when detecting changeable network traffic. For example, normal traffic may turn into anomaly within a short period of time. To recognize and classify the dynamic traffic, manually adding signatures and baselines is becoming very inefficient and expensive. What's more, this will also lead to to delays in anomaly detection . Therefore, the application of machine learning techniques to network anomaly detection is very important.

2.1.2 Application of Machine Learning in Network Anomaly Detection

The characteristics of data extracted from network traffic are large amounts, noisy, and high-dimensional.[2] For traditional network anomaly detection systems, dealing with these characteristics is very difficult, which may lead to problems of high false-positive rates and low accuracy. To avoid the disadvantages of traditional network anomaly detection systems, machine learning techniques are applied to this field. As a statistical-based analytical tool, machine learning techniques have been widely discussed and applied across various fields. In the case of known attacks, machine learning can understand their characteristics based on knowledge learned from existing data. For unknown attacks, machine learning can identify outliers through data patterns. Machine learning can also build various models according to the required abilities.[3] In this project, supervised models are trained to classify different classes of network flows, thus the models can identify malicious and benign network behaviors.

2.2 Machine Learning Algorithm

Basically, there are four types of machine learning techniques: Supervised Learning (SL), Unsupervised Learning (UL), Semi-Supervised Learning (SSL) and Reinforcement Learning. In the field of network anomaly detection, they have their own advantages and disadvantages. SL make predictions after being trained with a large, labelled

dataset, and its performance can be validated through labelled data. Its predictions are more reliable when facing conditions similar to the training set. However, when SL encounters unfamiliar data, error rate of predictions increases. UL makes predictions without labelled data being passed. It can detect new threats with patterns distinct from normal data. However, it could be very computing-consuming when facing complex scenarios and making classifications. SSL can initialize supervised learning with a limited amount of labelled data. It gains more confidence from labelled data than UL. However, it has the disadvantage that incorrectly predicted unlabeled data could mislead the classifier. RL uses a method called trial and error to test all possible state-action pairs to find the best long-term reward strategy. It is suited for complex problems but it is very resource-intensive.[3]

2.2.1 Algorithm Selections

The primary objective of this project is trying to train different machine learning models to detect network anomaly, to compare how machine learning helps with network anomaly detection and to compare their performance. It is obvious that Supervised Learning can help to reach the objective more easily, since it is intuitive and effective to evaluate the performance of SL models. For many popular network traffic datasets (e.g. UNSW-NB15 Dataset, NSL-KDD, UGR'16, CIC_IoT23), it is common that they are labelled with different classes. There are basically 6 machine learning algorithms for supervised learning, which are linear regression, logistic regression, decision trees, random forest, support vector machines (SVM) and Neural Networks. Random Forest is a classifier which can be used for classification or regression, which is also considered as ensemble learning as it is based on decision tree.[4] The characteristics of random forest include diversity, efficiency, and flexibility. It can handle various data anomalies and noise, get results faster when processing large-scale datasets and can be easily adapted to different datasets. Hence, random forest is an algorithm that is ideal for network anomaly detection and has been widely used. When investigating the characteristics of the network traffic datasets, it is obvious that most of them contain many features, which makes those data high-dimensional. Table 2.1 shows the number of features and classes in different popular datasets. The Support Vector Machine (SVM) operates by finding the hyperplane that separates different classes best in the feature space and maximizing

	UNSW-NB15	NSL-KDD	UGR'16	CIC_IoT23
Features	42	41	9	46
Classes	2	2	9	34

Table 2.1: Number of Features and Classes of Different Datasets

the margin between the closest points of the classes. To obtain the objective of this project, SVM is chosen for its advantage in handling high dimensional and nonlinear data, which are the characteristics of network traffic data.[5] A Convolutional Neural Network is a class of deep learning models. A CNN is composed of convolution, pooling, and fully connected layers. In this project, CNN is chosen for the ability to deal with high dimensional network traffic data. And CNN can deal with the complexity of network traffic patterns by modeling it, based on its deep learning capabilities efficiently. Additionally, CNN can automatically and adaptively learn spatial hierarchies of features of network traffic data, which makes it able to classify network traffic as different classes in a high accuracy.[6]

2.3 Dataset Introduction

To achieve the objectives of this project, the network traffic dataset should satisfy the following characteristics. First, the dataset should be of a large scale. Only a dataset that is large enough, could allow the machine learning models to learn the patterns inside the data from them and gain generalization ability. Second, the dataset should contain different classes of network behaviour, including normal traffic and different types of network attacks such as DDoS attack, anomaly scan and so on. Only trained by a dataset with great diversity, the machine learning models can have the ability to detect different types of network traffic. Third, the dataset should not be outdated. As the network environment changes very quickly, the patterns of network attacks also change and develop rapidly. If the dataset is too old to maintain timeliness, the patterns that machine learning models learn form it will not be able to detect the network attacks that are up-to-date. Moreover, it's also meaningless to evaluate their performances since they may not work as expected when facing the real and new network attacks. What's more, the network traffic among the dataset should have a great quality, Data with low

quality would improve the difficulty for training machine learning models, leading to over-fitting or low performance. Lastly, the network traffic flows within the dataset should be labelled. As mentioned above, our objective is to evaluate the performance of different machine learning algorithms. Only with a dataset that is properly labelled can our evaluation be easier and more credible. Therefore, for this project, I've chosen CIC_IoT23 and UGR'16 as the datasets for machine learning models to train on.

2.3.1 CIC_IoT23

CIC_IoT23 is a new realistic IoT attack dataset. This dataset is built using an extensive topology, which is composed of a few different kinds of real IoT devices. And in this topology, IoT devices act as attackers and victims.[7] In this dataset, there are 33 types of attacks, and they are separated into 7 classes. To describe the network flows, there are 46 features for each network flow. The details of the features are described in Table 2.2. And the 7 classes of the network attacks are DDoS, DoS, Mirai, Spoofing, Recon, Web, and BruteForce. However, the numbers of different classes are not even, for example, the DDoS class has the most significant ratio among the dataset, which is almost 78%. The uneven is caused by the implement when creating this dataset. During the implementation of dataset creation, DDoS attacks are executed against all devices and that's why it has the largest number. Meanwhile, web-based attacks are only executed against the devices supporting web applications.[7] Figure 2.1 illustrates the number of instances for different classes. The advantages of this dataset are they are collected from the real attacks in IoT devices, hence in the aspect of doing research about realistic network anomaly detection, it is meaningful. And it has a very large scale, which can help the machine learning models to gain a reasonable robustness and generalization ability. What's more, all data among this dataset is well structured and has a high quality. However, the biggest disadvantage of it is the uneven distribution of different classes. And this uneven distribution has led to some problems when training machine learning models and forced me to abandon the web-based and BruteForce attacks later in my experiment. This will be discussed in more detail in Section 4.

Feature	mean	min	median	max
flow_duration	5.765449	0	0	394357.2
Header_Length	76705.96	0	54	9907148
Protocol type	9.06569	0	6	47
Duration	66.35072	0	64	255
Rate	9064.057	0	15.75423	8388608
Srate	9064.057	0	15.75423	8388608
Drate	5.46E-06	0	0	29.71522
fin_flag_number	0.086572	0	0	1
syn_flag_number	0.207335	0	0	1
rst_flag_number	0.090505	0	0	1
psh_flag_number	0.08775	0	0	1
ack_flag_number	0.123432	0	0	1
ece_flag_number	1.48E-06	0	0	1
cwr_flag_number	7.28E-07	0	0	1
ack_count	0.090543	0	0	7.7
syn_count	0.330358	0	0	12.87
fin_count	0.099077	0	0	248.32
urg_count	6.239824	0	0	4401.7
rst_count	38.46812	0	0	9613
HTTP	0.048234	0	0	1
HTTPS	0.055099	0	0	1
DNS	0.000131	0	0	1
Telnet	2.14E-08	0	0	1
SMTP	6.43E-08	0	0	1
SSH	4.09E-05	0	0	1
IRC	1.50E-07	0	0	1
TCP	0.573834	0	1	1
UDP	0.211918	0	0	1
DHCP	1.71E-06	0	0	1
ARP	6.62E-05	0	0	1
ICMP	0.163722	0	0	1
IPv	0.999887	0	1	1
LLC	0.999887	0	1	1
Tot sum	1308.323	42	567	127335.8
Min	91.60735	42	54	13583
Max	181.9634	42	54	49014
AVG	124.6688	42	54	13583
Std	33.32481	0	0	12385.24
Tot size	124.6916	42	54	13583
IAT	83182526	0	83124522	1.68E+08
Number	9.498489	1	9.5	15
Magnitue	13.12182	9.165151	10.3923	164.8211
Radius	47.09498	0	0	17551.27
Covariance	30724.36	0	0	1.55E+08
Variance	0.096438	0	0	1
Weight	141.5124	1	141.55	244.6

Table 2.2: Details of features in CIC_IoT23

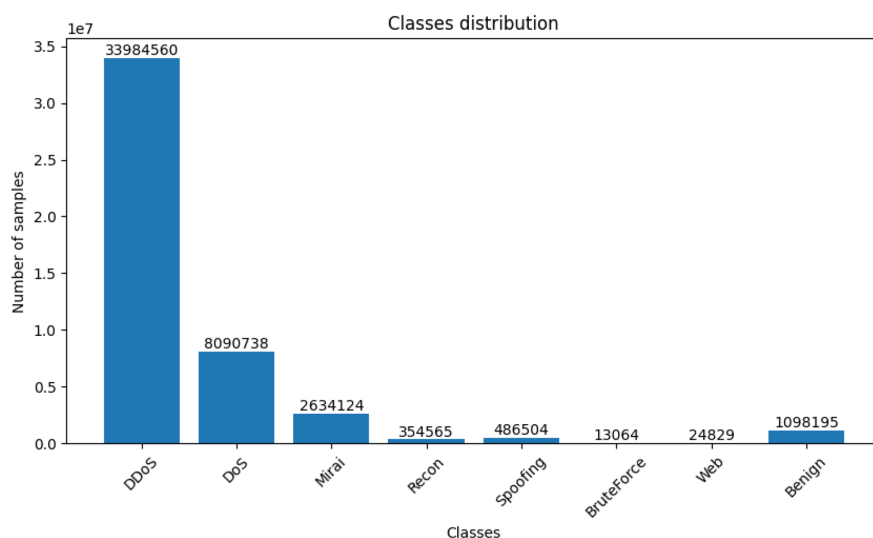

Figure 2.1: Number of instances for different classes of CIC_IoT23

2.3.2 UGR'16

UGR'16 is a dataset composed of real anonymized network traffic flows collected from a tier-3 ISP for 4 months. And it was collected by NetFlow traces capture.[8] To improve the practicality, the researchers have simulated different kinds of real attacks, and infused them into the dataset, which makes the dataset contain not only normal network flows data, but also network attacks data which is big enough for testing and machine learning model training. This dataset is divided into two parts: a CALIBRATION set and a TEST set. The CALIBRATION set contains 4 months of real background network traffic data gathered from ISP. And the TEST set includes another 5 weeks of real background data while network attacks were embedded to it.[8] Eight types of network attacks were implemented in the dataset, which are dos, nerisbotnet, scan11, scan44, blacklist, anomaly-spam, anomaly-sshscan and anomaly-udpscan. One objective for building this dataset was to study network attacks with consideration of traffic periodicity, which has not been considered in this project. Therefore, we will only include one week of data from the TEST set. To describe the patterns of network traffic flows, there are nine features for each network flow, including IP, port, period, packets and so on. The main advantages of this dataset are that it includes the realistic attack scenarios, and machine learning models training with this data is about to detect attacks in the

real internet environment. Furthermore, all data is labelled, as mentioned above, it can help with the evaluation of the performance of machine learning models. Its large-scale also contributes to improving the accuracy of the models. However, since this dataset is gathered from the real network environment from ISP, it may contain flaws in data quality. This results in some data being unusable due to missing features or duplication, but these issues can be addressed in the data-preprocessing section.

Chapter 3

Design

In this section, we will discuss the methods and algorithms used in this project in detail. Further information on how the datasets were preprocessed, as mentioned in the previous section, will be illustrated here. We will also thoroughly discuss the design and implementation of the machine learning models referred to earlier.

During the data preprocessing stage, tasks such as cleaning, feature selection, sampling, and splitting into training and testing sets are carried out to ensure the models can effectively learn the data patterns. Then, we explore the design of three machine learning models: Random Forest, SVM, and CNN. The implementation details of these models are also discussed. Finally, we introduce the methods and tools used to evaluate the models' performance.

3.1 Data Preprocessing

In this section, we will explore the data processing for the CIC_IoT23 and UGR'16 datasets separately. The process for CIC_IoT23 is expected to be simpler than that for UGR'16, as its data is in high-quality, which suggests it is almost ready for training machine learning models. The primary task at hand is to sample the dataset. However, the UGR'16 dataset requires more process due to some minor flaws and the need for restructuring its features.

Mean	Median	Min	Max
276,252	239,887	211,834	451,498

Table 3.1: Numbers of flows in each .csv file

3.1.1 CIC_IOT23

To analyse this dataset, first, we need to investigate the structure of the data. For the dataset itself, there are 168 .csv files in total, the sizes of each of them vary from 60,000 KB to 120,000 KB. And the numbers of instances vary from 211,834 to 451,498, which are illustrated in Table 3.1. The total size of the dataset is 13,431,798 KB. Due to the large size of the dataset, it would be impossible to load all the data at once into the memory and train the machine learning models with it. As the traditional machine learning method Random Forest and SVM do not support incremental learning by their nature, it is very challenging to implement incremental learning in Random Forest and SVM.[9] Therefore, we can not simply take all data from the dataset, instead, we need to sample from it and use the sampled dataset to train the models. Also, as illustrated in Figure 2.1, we can see that the distribution of the data is very uneven. Hence, when sampling the data, we need to take under-sampling strategy to make the training and testing dataset even. When looking deeper into the numbers of instances for each class, we can notice that the numbers of 2 classes, Web and BruteForce, are very small, which brings the coincidence that models trained with these 2 classes are very difficult to classify them, resulting in almost none of testing network flows being classified as Web or BruteForce. And this influences the overall performance of models in a very bad way. This problem has been explored and will be discussed in the Section 4.2. As a result, we must drop these 2 classes, and extract an even sampled dataset with other classes. Finally, we design to put all data from the dataset together, and randomly sample 40,000 flows from the whole dataset for each class (excluding Web and BruteForce) as training set, and randomly sample 20,000 flows from the rest of the dataset for each class. And the numbers of instances of each class in training and testing set are illustrated in Table 3.2.

In conclusion, for the CIC_IOT23 dataset, we have taken the following steps to preprocess the data. Firstly, we combined all data from 168 .csv files together. Secondly, we removed flows of the Web and BruteForce classes. Thirdly, we extracted 40,000

	Training	Testing
Benign	40,000	20,000
DDoS	40,000	20,000
DoS	40,000	20,000
Mirai	40,000	20,000
Recon	40,000	20,000
Spoofing	40,000	20,000

Table 3.2: Number of Each Class in Train Test Set in CIC_IoT23

flows for each class to serve as the training set and 20,000 flows for each class to serve as the test set.

3.1.2 UGR'16

As mentioned earlier in this project, the UGR'16 dataset is gathered from real Internet environment from an ISP. It contains 23 weeks of network flows, and for each week, the size of the dataset is around 80GB.[8] Therefore, the size of the whole dataset is too large for us to train the machine learning models effectively. Regarding the CALIBRATION set, it consists mainly of normal network flows, while we need to train the machine learning models not only with normal flows but also network attacks. Hence, we focus on the TEST set, and choose the data from the first week of August in the TEST set. Since the dataset for this week contains all 7 days' data in one single .csv file, we need to split them into separate days and work out their distribution. While splitting them, we also notice that there are some flaws in the data. Not all flows in the dataset are usable, which means some flows recorded in the dataset are not in the correct format, lack some features, or are duplicate in some features, indicated by their numbers of features being not 12 or some features being in wrong data type. Therefore, we will only keep the usable flows when splitting the dataset. What's more, the reason why network traffic flows were labelled as blacklist is not about what these flows are doing, but what they are. When the researchers were creating this dataset, they labelled the blacklist flows according to their IP only, hence the patterns among these flows could vary a lot.[8] Therefore, we must remove the blacklist class from the dataset to achieve our objective, which is allowing the machine learning models to learn the patterns within the data and classify network flows. Otherwise, the blacklist class may significantly affect

Day	Number of instances	Size (KB)
1	123,186,619	7,590,197
2	126,837,542	7,804,402
3	125,070,667	7,703,453
4	121,658,121	7,487,820
5	119,236,848	7,333,370
6	111,696,037	6,870,789
7	120,685,717	7,441,426
Total	848,371,551	52,231,457

Table 3.3: Numbers and Size of dataset for each day of UGR'16

Day	**Background**	**DOS**	**Nerisbotnet**	**Scan11**	**Scan44**	**Anomaly-Spam**	**Anomaly-SSHScan**	**Anomaly-UDPScan**
1	120,779,166	783,640	151,525	76,284	406,077	47	8	989,872
2	125,204,583	784,186	151,641	78,282	370,198	248,650	2	0
3	124,285,649	391,527	151,964	48,139	188,554	4,830	4	0
4	120,263,351	783,842	151,490	83,310	376,128	0	0	0
5	117,867,837	782,356	151,368	68,278	367,007	0	2	0
6	104,462,147	783,901	152,419	92,234	366,955	5,838,381	0	0
7	97,517,366	783,680	82,168	92,491	402,284	21,807,728	0	0
Total	810,380,099	5,093,132	992,575	539,018	2,477,203	27,899,636	16	989,872

Table 3.4: Numbers of different classes in each day

the machine learning models, and result in low accuracy and poor performance of the classifications made by the models. After finishing these works, we are able to separate the data from day 1 to day 7 from this week's dataset. And Table 3.3 illustrates the size and number of instances of the dataset for each day.

After checking the table, we notice that the size of each day's dataset is still too large for our experiment to handle. Therefore, we cannot just simply take one dataset from a random day and train our machine learning models on it. And we also need to make sure that the train and test set we plan to use include sufficient and even data for all classes. Therefore, it is necessary to investigate deeper and find out the distribution of all classes on each day. Table 3.4 illustrates the number of flows for different classes on each day.

From Table 3.4, we can clearly notice that the number of anomaly-sshscan sums up to 16, which means that it is impossible for our machine learning models to learn this class. And there's not a single day that has all classes with a proper number for us to train and test. Therefore, we need to concatenate them back together, and sample from the whole dataset. And to investigate the usability of the dataset and if the features can

td	sp	dp	pr	flg	fwd	stos	pkt	byt	bpp	bps
0.052	56097	443	TCP	.AP.S.	0	0	3	190	63.333	3653.846
0.052	56096	443	TCP	.AP.S.	0	0	3	190	63.333	3653.846
0.256	443	58676	TCP	.AP.S.	0	0	6	5298	883	20695.31
0.28	55674	80	TCP	.AP.S.	0	0	5	439	87.8	1567.857
0.828	42068	443	TCP	.AP.S.	0	0	12	1829	152.417	2208.937
0.928	59669	443	TCP	.AP...	0	0	3	274	91.333	295.259
0.992	0	769	ICMP	.A....	0	0	4	224	56	225.806
0.992	54213	53	UDP	.A....	0	0	2	130	65	131.048
1.068	49900	80	TCP	.AP.S.	0	72	5	2019	403.8	1890.449

Table 3.5: Example of Network Flows without Label

be used to train, we need to look at the shape of the sample. Table 3.5 is an example of one of the network flows extracted from the dataset, with labels removed.

And the features are as follows: duration (td), source port (sp), destination port (dp), protocol (pr), flags (flg), forwarding status (fwd), type of service (stos), packet count (pkt), byte count (byt), bytes per packet (bpp), bits per second (bps).[8] As we can see, the protocol and flags are described by a text string, which is the data type that machine learning algorithms cannot handle. Therefore, we need to convert them a into one-hot encoding format. By analyzing other samples, we can find out there are seven types of protocols in total, which are 'TCP', 'ICMP', 'UDP', 'IPIP', 'GRE', 'ESP', 'IPv6'. And for the flags, it is a combination of the TCP flag indicators: URG (U), ACK (A), PSH (P), RST (R), SYN (S), and FIN (F). These flags represent different control messages used in the TCP protocol to manage the state of a network connection. To one-hot encode the 'pr' (protocol) feature, we need to expand it into 7 distinct features. Each of these new features indicates whether a specific protocol is used in each network flow, marked by a 1 for presence and 0 for absence. In this case, only one out of seven features would be marked as 1, with the rest of them set to 0. Similarly, we need to expand the 'flg' (flags) feature into six features, each representing whether a specific flag state is present. Unlike the protocol feature, the flags feature can have more than one flag marked as 1, as multiple TCP flags can be set for a single packet, indicating various states or actions of the network flow. What's more, as the network flows are gathered from the real Internet environment and were meant to be used to research the periodicity of the network traffic, each sample comes with a timestamp. However, our project is aimed at identifying the patterns of network flows to classify them, we

td	sp	dp	TCP	ICMP	UDP	IPIP	GRE	ESP	IPv6	flg_u	flg_a	flg_p	flg_r	flg_s	flg_f	fwd	stos	pkt	byt	bpp	bps
0.12	25	44718	1	0	0	0	0	0	0	0	1	1	0	1	1	0	72	3	234	78	1950

Table 3.6: Example of One-hot Encoded Sample

don't need the timestamp feature, and we don't want the machine learning models to be influenced by it. We decided to remove the timestamp. Otherwise, the machine learning models may associate the attacks with a specific time rather than the behaviours themselves, since there may be some connections between the attacks and the timestamp among the dataset.[10] Similarly, we need the machine learning models to learn from the flows' behaviours, and not be limited by the IP. Because we know that some IPs are malicious, just like the blacklist we mentioned before. If the machine learning models learn well about the IP, they might perform very well in the test set, since likely most attacks are executed by the same group of IPs. However, this is not reliable, because the IPs are changing out of this dataset and the machine learning models may have ignored the other hidden patterns in the data. Further more, among the samples, the features 'td' (time duration), 'pkt' (packet) and 'byt' (byte) are considered in isolation. Intuitively, the features can reveal more insights when they are connected to each other. This is because analyzing these features together allows the machine learning models to gain a deeper understanding of network traffic behaviour over time and to isolate the abnormal network flows when the packets contain bytes varying form the normal flows. To achieve this, we add the features 'bpp' (Bytes per packet) and 'bps' (Bytes per second). In addition, as mentioned by Dion, it's difficult for machine learning models to distinguish between the class scan11 and scan44, and these two attacks are very similar. They have similar effects on the victims, and solutions to them are both closing certain ports.[10] Therefore, we decided to merge these two classes into one class. And the sample after processing as mentioned above, should look like the one in Table 3.6. As a summary, the following are what we have done to pre-process the UGR'16 dataset: filtering out unusable data, removing blacklist data, removing anomaly-sshscan data, removing IP, removing timestamp, calculating 'bpp' and 'bps', merging scann11 and scan44 into one class.

Finally, we needed to sample from the dataset and split into training and testing sets. We randomly sampled 20,000 flows for each class to create the training set. Then, we sampled 20,000 flows for each class from the rest of the dataset to form the testing set.

3.1.3 Normalization

The last step before using the dataset to train the machine learning models is data normalization. Data normalization is an essential data-preprocessing step in machine learning, aiming to adjust the range of data values and normalize them to a same scale without changing the distribution of the data.[11] It is essential for machine learning models as it can help improve the models' convergence speed and performance by mitigating the issues related to value calculations. In this project, we look at two methods for normalization, which are Min-Max Scaling and Z-Score normalization (or Standardization). The formula for 0-1 Min-Max Scaling is given below:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{3.1}$$

The formula for Z-score normalization (Standardization) is as below, where μ is the mean and σ is the standard deviation:

$$X_{\text{std}} = \frac{X - \mu}{\sigma} \tag{3.2}$$

After Min-Max scaling, all values of data will be transformed into values within a fixed range, which is from 0 to 1 in this example. It helps to reduce the pressure for machine learning models to calculate, however, it is very sensitive to the outlier values. For example, if there's a value significantly higher than the others, then the rest of the values will be compressed into a very small range near 0, resulting in a gap within the normalized range.

After applying standardization, the mean of the normalized values will be 0, and their standard deviation will be 1. Unlike Min-Max scaling, Standardization shows more tolerance to outliers. Having outliers in the dataset would not have a great impact on the normalized data.

At the beginning, we had decided to use standardization for its better tolerance towards outliers, which are very common in network traffic flows data. By checking Table 2.2, we found most features have a max value significantly higher than the mean value. However, after we investigated deeper into the structure of CNN models, we realized that CNN cannot take negative values. There are activation layers called ReLU in CNN, and when numbers lower than 0 enter the activation layer, the ReLU turns them into

0.[12] This means, if we normalize the data with Standardization and train the CNN models with it, there would be nearly half of the data being dropped by being turned into 0.

Therefore, we decided to use 0-1 Min-Max Scaling to normalize the data. To do so, we trained a MinMaxScaler from python library Scikit-Learn with the training set and stored the scaler for later usage. And every time we train or test a model, we apply the scaler to normalize the dataset first.

3.2 Machine Learning Algorithms

In this section, we will go through the design of the machine learning algorithms implemented in this project, which are Random Forest, SVM and CNN. We will discuss how the models are trained, how structures of models are designed and how hyperparameters are chosen.

3.2.1 Random Forest

Random Forest is a classifier based on decision trees and considered as an ensemble machine learning method.[13] It is designed to improve accuracy and robustness by combining the predictions of multiple decision trees. Basically, it classifies a sample by having every single tree to make a classification and vote, and it chooses the class by the majority voting. Figure 3.1 illustrates how Random Forest model works. For the Random Forest model, there are two hyper parameters that can be adjusted: number of trees and maximum depth of each tree. Generally, setting a higher number of trees can lead to better model performance because by averaging more trees, the model can reduce the risk of overfitting. However, this also increases the computational cost and time needed for model training and classifying. Maximum depth is another important parameter for Random Forest. It controls the maximum number of levels in each decision tree. A deeper tree can learn and model more complex patterns and relationships by having more splits. However, setting this parameter too high can lead to overfitting, as the model learns the patterns from the training data too well but also learns the noise, which negatively affects its performance on the data it has never seen. Likely, setting the

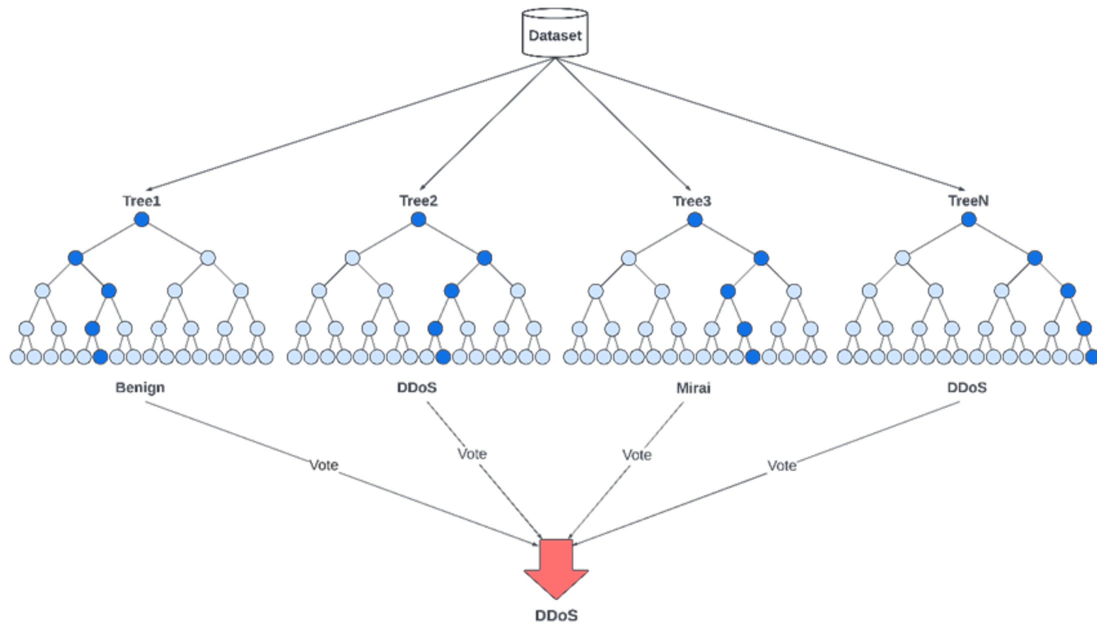

Figure 3.1: Illustration of Random Forest's majority voting mechanism

maximum depth too low may result in underfitting, which means the model cannot capture the underlying patterns and relations in the data well, leading to poor performance on the test set.

In this project, different numbers of trees and maximum depths have been chosen to find an optimal balance between performance and computational cost. The numbers of trees that we chose are [10, 50, 100, 200, 500], and numbers of maximum depth are [10, 20, 30]. While implementing the Random Forest models, we will try all the combinations of hyperparameters to find the best combination by implementing a grid search. Grid search is a traditional method for hyperparameters optimization that finds best hyperparameters space by making complete search over a subset given to it.[14]

To implement the Random Forest models, we need to use a machine learning library of Python to train and evaluate the models. In this project, we decided to use Scikit-learn. Scikit-learn is an open-source machine learning library for Python, aiming to integrate the machine learning methods into Python code.[15] The key reason why we choose Scikit-learn to implementation is its ease of use when implementing the Random Forest model. It only requires a single line of code to define a Random Forest model. The code we used to define our Random Forest model is as shown in code 3.1.

```
model = RandomForestClassifier(n_estimators=200,
max_depth=10, random_state=0)
```

Listing 3.1: Source Code for Random Forest

And to implement a Grid search to find the optimal hyperparameters, we need to call the function GridSearchCV from Scikit-earn. The code we used to find optimal hyperparameters using grid search with a 5-fold cross-validation is listed as code 3.2.

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [10,50,100,500],
    'max_depth': [10,20,30],
    'random_state': [0]
}
grid_search = GridSearchCV(RandomForestClassifier(),
param_grid, refit=True, verbose=2, cv=5)
grid_search.fit(X_numpy, y_numpy)
```

Listing 3.2: Source Code for Grid Search for Random Forest

3.2.2 SVM

SVM is a classifier that is based on convex optimization techniques. It can learn the underlying patterns of different classes from the known dataset and make predictions on the unknown dataset.[16] The core idea of SVM learning is to find a hyperplane that maximizes the margin between the closest points of different classes, which are known as support vectors. Figure 3.2 illustrates the hyperplanes found by SVM and how the hyperplanes classify different classes of data. By maximizing the margin, it can improve the model's generalization ability when classifying unseen data. In our project, the network traffic flows are not linearly separable, and the SVM uses the kernel trick to map the original data into a higher-dimensional space where the flows become linearly separable. Common kernels that SVM uses are the Linear Kernel, Polynomial Kernel, and Radial Basis Function (RBF) Kernel. For the training of SVM model, there are

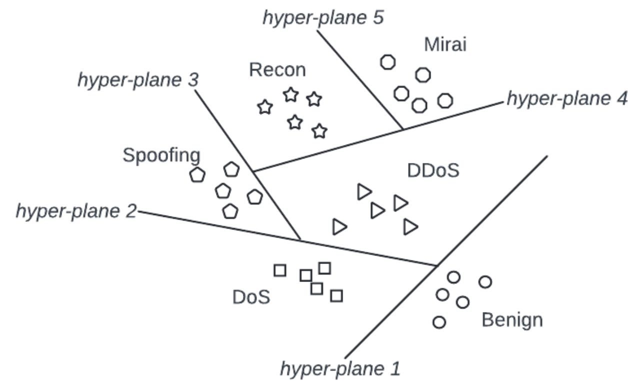

Figure 3.2: Hyperplanes of SVM

a few hyperparameters that can be chosen, which are C, Kernel, and Gamma. C is a regularization parameter that controls the tolerance of errors for individual data points. Specifically, a smaller C value means the model will try to make the margin as big as possible, however, this may result in some misclassifications for some data. Hence, C is a parameter to balance the size of the margin and the accuracy of classifications. As mentioned above, there are a few kernels that SVM can use. In this project, we chose the RBF Kernel. With its non-linear properties, the RBF kernel can adapt to various shapes of decision boundaries.[17] And this flexibility ensures that SVM models with an RBF kernel can adapt to the network traffic flows in this project and perform well in classification. The last hyperparameter we can adjust is gamma. The Gamma parameter is used in the RBF kernel and determines the range of influence of a single training example. Specifically, a smaller gamma value means the range of influence for each example becomes larger, which could lead to a smoother decision boundary. On the opposite, a larger gamma value means a smaller range of influence, which may result in a more complex decision boundary and a higher risk of overfitting. Since the kernel is decided, we need to find the optimal pair for C and gamma. The values for C we chose are [0.1, 1, 10, 100], and the values for gamma are [1, 0.1]. Similarly to what we did when training the Random Forest, we need to perform a grid search to find the optimal hyperparameters.

Before we can train SVM models with CIC_IoT23 and UGR'16 datasets, we need to

perform feature selection to select the features necessary for training the SVM model. As we mentioned above, SVM transforms data into a higher dimensional space to make it linearly separable. Therefore, SVM models are very sensitive to the number of features, especially since our datasets contain many features. As a result, feature selection must be carried out to reduce the training time for SVM models and improve its performance.[18] We can only train SVM with features that are important, otherwise, the training process will take too long, and its accuracy will be low. Therefore, we introduce SHAP algorithm to evaluate the feature importance. SHAP (SHapley Additive exPlanations) is a game-theoretic approach used to explain the output of any machine learning model. SHAP values assigned to each feature represent their contribution to the model's output, encoding the importance a model places on a feature.[19] Therefore, by computing SHAP values, we can understand which features are important for classification for the network traffic flows. After that, we can choose the features with high importance and train the SVM models exclusively with those only.

To implement the feature selection, we decided to use a Python library called ExplainerDashboard to calculate the SHAP values, which will be discussed later in Section 3.3. To train the SVM models and perform the grid search, we need to use the python library Scikit-learn, which is mentioned above and used to train Random Forest models. The code listing in 3.3 provides a sample for defining a SVM model.

```
model = SVC(kernel='rbf', C=100, gamma=1,
random_state=0, verbose=2, probability=True)
```

Listing 3.3: Source Code for SVM

Code 3.4is the code for performing grid search.

```
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [1, 0.1],
    'kernel': ['rbf']
}
grid_search = GridSearchCV(SVC(), param_grid,
refit=True, verbose=2, cv=5)
```

Listing 3.4: Source Code for Grid Search for SVM

3.2.3 CNN

CNN has been used in many fields for its advantage in handling high dimensional and non-linear data.[6] To learn the patterns underlying among the data, there are some essential layers in the structure of a CNN model.

Convolutional Layer: This layer is the core of CNN model. It extracts the features from the input data and captures local features through convolution.

Activation Function: This function commonly follows each convolutional layer, and its purpose is to introduce non-linearity into the network to help the network learn complex patterns. In this project, we use the ReLU (Rectified Linear Unit) as our activation function.

Pooling Layer: A pooling layer should be applied after the convolution operation. This layer can reduce the spatial dimensions and reduce the amount of computation. In this project, the Max Pooling layer is implemented to the CNN model.

Fully Connected Layer: This layer should be applied towards the end of a CNN. And in our CNN model, this layer performs classification.

Hence, we need to design a CNN model and train it with the network traffic dataset to classify network behaviours. To make sure the CNN can learn the underlying patterns from the network flows with non-linearity, we decided to apply two convolutional layers to it. As the network traffic flows are not image data, we use convolution 1D layers. Right after each convolution layer, there is a batch normalization layer, which can speed up the convergence when training the model and prevent the model from overfitting to some degree. Then an activation function is applied after each batch normalization layer, and a pooling layer is applied after the activation function. After that, a flatten layer is included to handle the data from convolution layers and pass it to the fully connected layers. At the end of the CNN, we applied three fully connected layers, the first two of which are followed by activation layers. The basic structure of the CNN model we designed is illustrated in figure 3.3. Code 3.5 shows the source code for building this CNN model with PyTorch.

As the network traffic data is not an image, it contains only 1 channel, which is why the first convolutional layer in the CNN model was set to take in 1 channel data. Before we start training the CNN model, we need to reshape the data into the shape that convolution layers can handle. In this case, the data was reshaped to (number_of_samples,

1, number_of_features). After that, the data was transferred to a tensor and stored in a tensor dataset. Then a data loader was created with a batch size, and the data loader was used to train the CNN model. During the training process, there are three hyperparameters that we can adjust, which are batch size, epochs, and learning rate. We use Adam as our optimizer and calculate the loss with the Cross Entropy function. We designed to fix the batch size to 64 because if it was set larger, the generalization ability would be lower than expected, and the model would be difficult to converge during the training process. But if it was set to smaller, the time it would take for training would be too long. And the epochs and learning rate change during the whole training, which will be discussed in Chapter 4.

```
class CNNModel(nn.Module):
    def __init__(self, num_classes):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=1,
        out_channels=64, kernel_size=3)
        self.bn1 = nn.BatchNorm1d(num_features=64)
        self.conv2 = nn.Conv1d(in_channels=64,
        out_channels=128, kernel_size=3)
        self.bn2 = nn.BatchNorm1d(num_features=128)
        self.pool = nn.MaxPool1d(kernel_size=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(number_of_Conv1d_output, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, num_classes)
    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.pool(x)
        x = F.relu(self.bn2(self.conv2(x)))
        x = self.pool(x)
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

Listing 3.5: Source Code for CNN

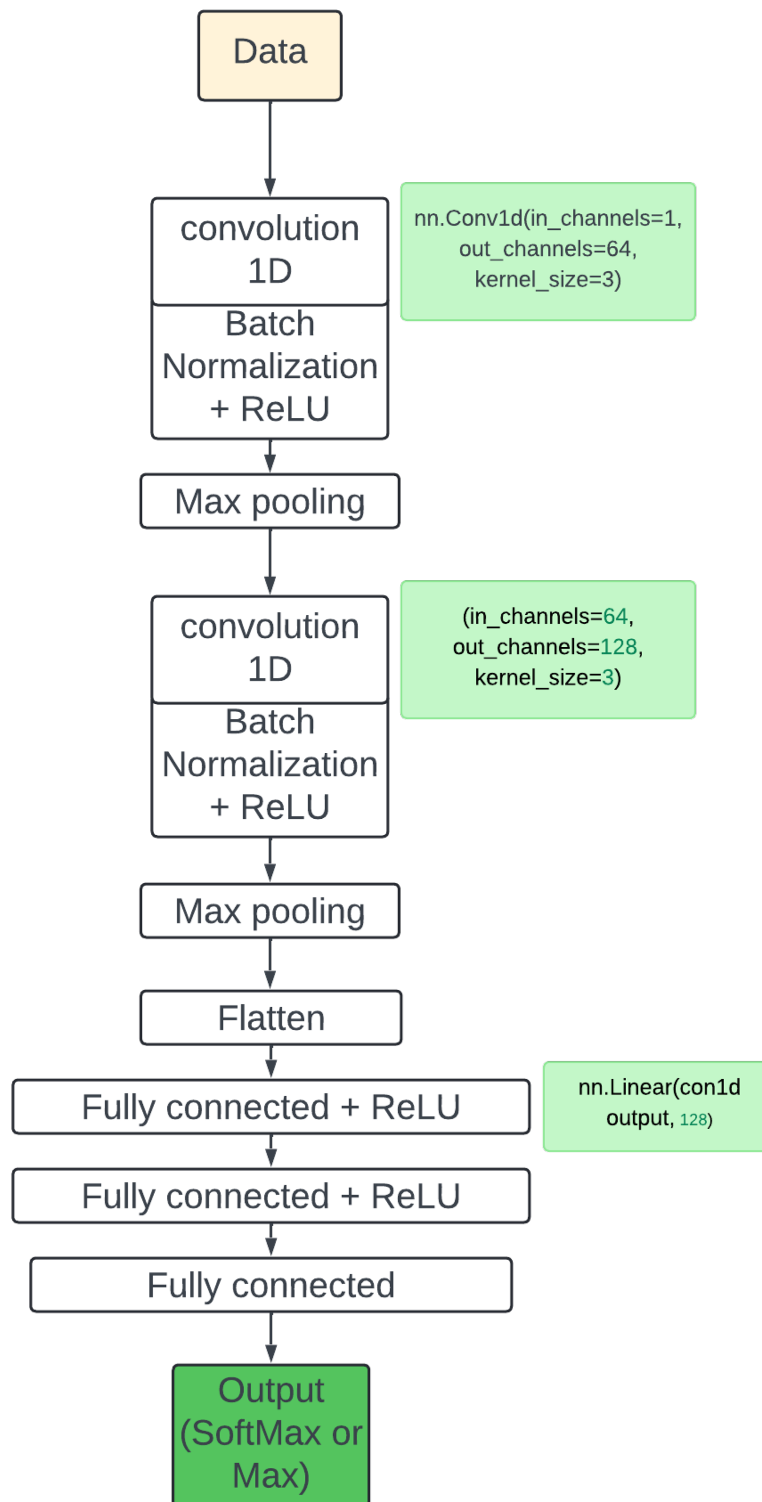

Figure 3.3: Structure of CNN model

3.3 Evaluation and Visualization

The most important objective of this project is to evaluate the performances of different machine learning models. Therefore, we need to find a method and metrics to evaluate how well the models trained are performing. As stated by Hossin and Sulaiman, a suitable evaluation metric is essential for achieving a better classifier.[20] As all models trained are classifier, and all train and test data are labelled with specific classes, the way to evaluate them is straightforward. The main metrics we are using for evaluation are accuracy, recall, precision, F1-Score and the Confusion Matrix. Table 3.7 illustrates the formulas for calculating and characteristics of different evaluation metrics.

Metrics	*Formula*	*Evaluation Focus*
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	Model's overall correctness in predicting positives and negatives.
Precision	$\frac{TP}{TP+FP}$	Proportion of actual positives among the positive predictions.
Recall	$\frac{TP}{TP+FN}$	Proportion of actual positives that are correctly classified.
F1-Score	$\frac{2\times Precision\times Recall}{Precision+Recall}$	The harmonic mean of precision and recall.

Table 3.7: Introduction to Different Metrics

These metrics are easy to calculate and obtain. In this project, we use Python functions from Scikit-learn library. Code 3.6 lists the code to calculate the metrics mentioned above. The first metric we look at to compare different models' performance is overall accuracy score. The overall accuracy tells us how the models work as classifiers in general. After that, we look deeper to assess other metrics, and investigate the differences between different models.

However, merely looking at different metrics does not provide sufficient information about the models' performance. Even if it could, it is not intuitive. Therefore, we require a tool to visualize the results obtained from different models. To do so, we decided to use a Python library called ExplainerDashboard. ExplainerDashboard is designed

```
over_all_accuracy = accuracy_score(preds_classes, y_test)
recall = recall_score(preds_classes, y_test,
average=None, labels=classes_labels, zero_division=0)
precision = precision_score(preds_classes, y_test,
average=None, labels=classes_labels, zero_division=0)
f1 = f1_score(preds_classes, y_test,
average=None, labels=classes_labels, zero_division=0)
```

Listing 3.6: Source Code for Metrics Calculation

to create interactive dashboard and visualize the performance of the machine learning models. It is compatible with models created with Scikit-learn. With ExplainerDashboard, we can visualize the models' performance with a few lines of code. Furthermore, as mentioned earlier, ExplainerDashboard supports the investigation of SHAP values, which can help find out the importance of each feature. Figure 3.4 is a screenshot from ExplainerDashboard. As naturally supported in ExplainerDashboard, Random Forest

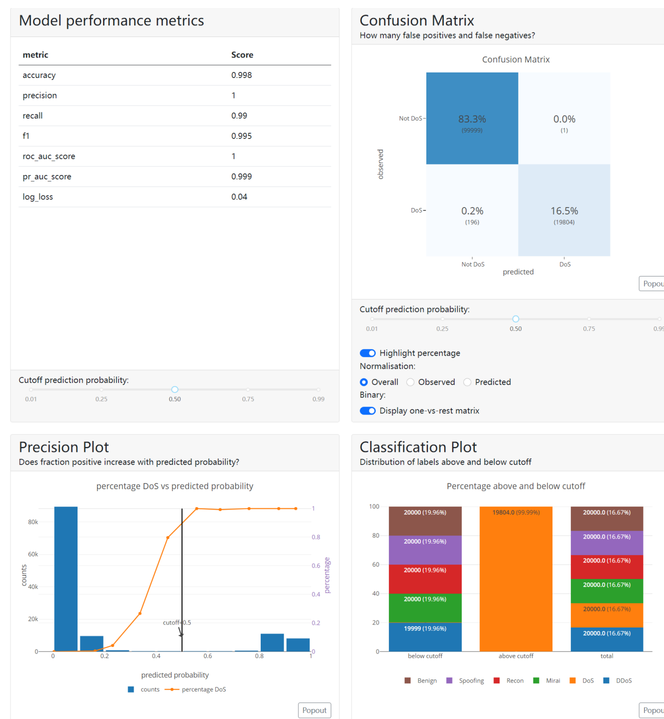

Figure 3.4: Screenshot of ExplainerDashboard

models and SVM models are very easy to analyze with ExplainerDashboard. We just

need to load the processed test set and load the pre-trained models into the Explainer-Dashboard, then all statistics will be calculated automatically. Source code to visualize a Random Forest or SVM model with explainer dashboard is shown in Code 3.7.

```
model = load('Randomforest_IoT_Even.joblib')
explainer = ClassifierExplainer(model, X_explain,
y_explain, labels=classes_labels)
dashboard = ExplainerDashboard(explainer)
```

Listing 3.7: Source Code for Random Forest in Explainer Dashboard

Unfortunately, CNN models are not naturally supported in ExplainerDashboard. Therefore, the code to integrate a CNN model into ExplainerDashboard would be much more complicated. Firstly, we need to define the class for CNN Model again, including all layers and parameters. Then, we need to define a class as the wrapper for the model, which can adapt the model into formats that ExplainerDashboard can handle. To do so, the wrapper class would accept a CNN model and test set as parameters and use the model to make predictions on the testing set. As a result, the wrapper will output a list containing all predictions, with each prediction contains probabilities for each class, known as SoftMax probabilities, which sum up to 1. Hence, this behaviour meets the expectation of ExplainerDashboard, allowing it to visualize the performance of the model. Code 3.8 shows basic code for a CNN model wrapper.

```
class Model_Wrapper():
    def __init__(self, model, device):
        self.model = model
        self.device = device

    def predict_proba(self, X):
        test_loader = preprocess(X)
        preds_list=[]
        for inputs in tqdm(test_loader):
            with torch.no_grad():
                inputs = inputs[0].to(self.device)
                outputs = self.model(inputs)
                # return softmax probability
                preds_list.append(F.softmax(outputs,
                dim=1).cpu().numpy())
        preds = np.concatenate(preds_list, axis=0)
        return preds
```

Listing 3.8: Source Code for CNN in Explainer Dashboard

Chapter 4

Experiments

This section will begin by introducing the environment setup for the experiments in this project. Then we will review the experiments that were performed to train and compare the performance of all different algorithms, Random Forest, SVM and CNN on two different datasets, CIC_IoT23 and UGR'16. And the results will be presented and discussed.

4.1 Environment Setup

The hardware used in this project includes a PC with Windows 10 OS, 32 GB RAM, and an Nvidia 3070ti graphic card with 8GB graphics memory. At the beginning of the experiments, everything works as expected, until we need to combine all data from CIC_IoT23 together and sample from them. The PC ran out of memory when all data was loaded into memory. Therefore, the RAM of the PC was increased to 48 GB.

As for the software aspect, the main environment used in this project is Jupyter Notebook. As mentioned above, we used many python libraries to accomplish the objectives of this project. The main library used for Random Forest and SVM models training is Scikit-learn. At the beginning, we chose TensorFlow as the deep learning python library to train the CNN model. However, TensorFlow had stopped supporting Windows OS. Hence, we would have to change to PyTorch, though some scripts have already been written using TensorFlow. Fortunately, it turns out that PyTorch is a much better tool for deep learning than TensorFlow.

4.2 CIC_IoT23

In this section, the details of the experiments performed with CIC_IoT23 dataset will be introduced. Since the code snippets for implementing the machine learning models are already listed in the earlier section, the process and the results of the experiments will be discussed more thoroughly.

As mentioned in section 3.3, before we started sampling from the original dataset and create an even dataset for training and testing, we did some primitive experiments. In these experiments, we did not attempt to make the dataset's distribution even but simply tried to use the entire dataset to train the machine learning models. And the result was very clear that none of the models could recognize or classify the two classes, Web and BruteForce. Figure 4.1 shows a confusion matrix from one of these experiments. After that, we tried to undersample the classes with too much data to make the dataset more even. However, the overall performance improved slightly, but the models still cannot recognize or classify the two classes with too few instances. Finally, we did what we said in section 3.1.1, the two classes, Web and BruteForce, were dropped form the training and testing datasets.

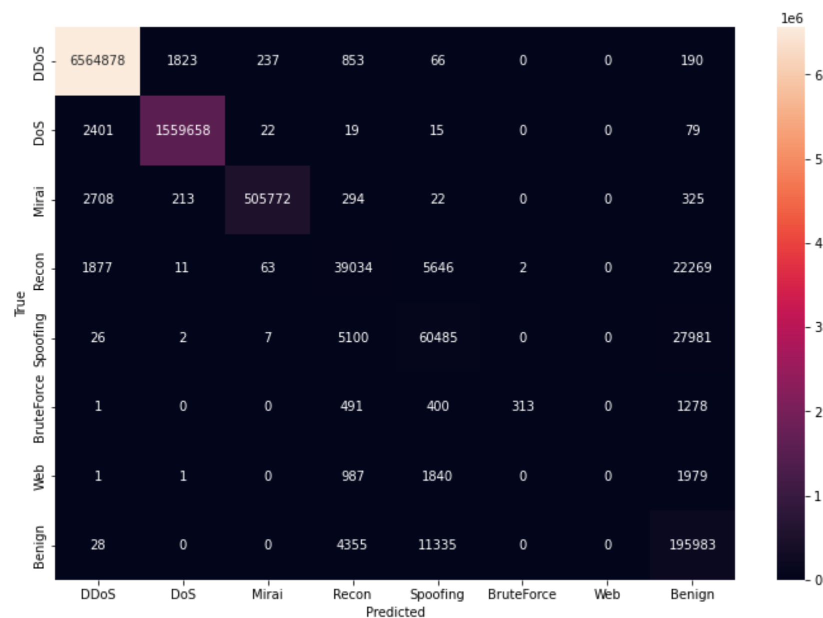

Figure 4.1: Confusion Matrix for Primitive Experiments on CIC_IOT

4.2.1 Random Forest Results

As mentioned in section 3.2.1, a grid search was performed to find the optimal hyperparameters for Random Forest. The optimal hyperparameters that the grid search identified were n_estimators=500, max_depth=30. To further investigate the impact of different hyperparameters, we trained the Random Forest Model with different hyperparameters and record the overall accuracy. The accuracies are illustrated in table 4.1 and the time cost in seconds for training the model is illustrated in table 4.2.

Overall Accuracy	**max depth=10**	**max depth=20**	**max depth=30**
estimator=10	0.9142	0.9434	0.9397
estimator=50	0.9165	0.9478	0.9491
estimator=100	0.9186	0.9483	0.9502
estimator=500	0.9209	0.949	0.9505

Table 4.1: Accuracy with Differen Hyperparameters on CIC_IoT23 with Random Forest

Time Spent (Second)	**max depth=10**	**max depth=20**	**max depth=30**
estimator=10	2.91	4.19	4.75
estimator=50	14.15	20.69	21.24
estimator=100	27.59	40.28	42.11
estimator=500	138.90	196.73	213.24

Table 4.2: Time Spent with Differen Hyperparameters on CIC_IoT23 with Random Forest

We can notice that increasing either estimator or max depth can both increase the accuracy. Although increasing the max depth is more efficient since it improves accuracy without increasing processing time as increasing the number of estimators does. Then, we used these hyperparameters to train the Random Forest model on the training set, and we got the result of performance. The overall accuracy of the model when classifying the test set is 0.951. Table 4.3 shows different metrics for each class.

Figure 4.2 illustrates the classifications distribution.

With these results, it is clearly notice that Random Forest model performs nearly perfectly for the classes DDoS, DoS and Mirai. However, the performance for classifying Recon, Spoofing and Benign is not good enough, as these classes are likely to be misclassified as one another.

Class	Recall	Precision	F1-Score
DDoS	0.999	0.998	0.999
DoS	0.999	0.999	0.999
Mirai	1	0.998	0.999
Recon	0.903	0.897	0.9
Spoofing	0.919	0.881	0.9
Benign	0.884	0.931	0.907

Table 4.3: Different Metrics for Each Class of Random Forest Model on CIC_IoT23

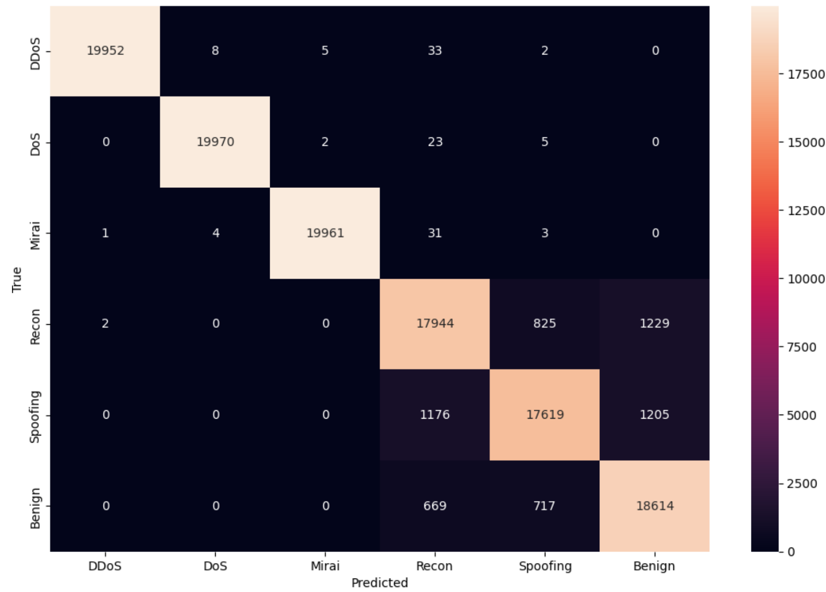

Figure 4.2: Confusion Matrix of Random Forest Model on CIC_IoT23

4.2.2 SVM Results

As mentioned in section 3.2.2, SVM models are very sensitive to the number of features, and we need to perform feature selection to preserve important features for training only. Hence, SHAP algorithm was applied using explainerdashboard, and SHAP values were calculated for each feature. The result of these SHAP values is illustrated in figure 4.3. We decided to preserve only the first 15 features for SVM model training and testing, as the importance of other features is much lower and may pollute the model and extend the training time. After that, we performed a grid search, which indicated that the optimal hyperparameters are C=100 and gamma=1. Since training SVM models takes much

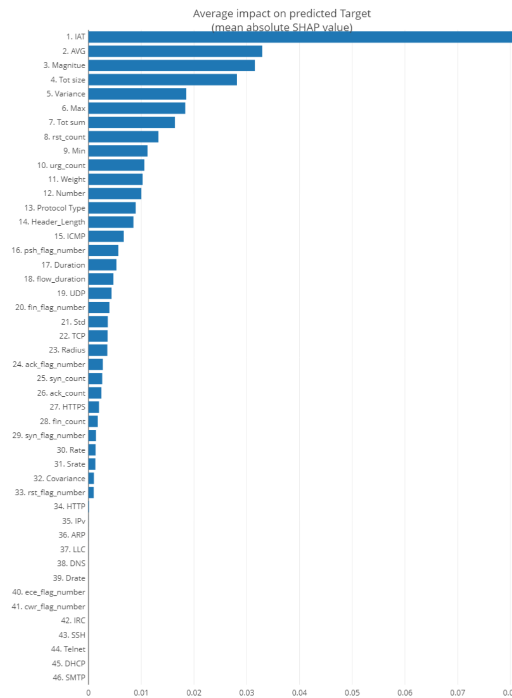

Figure 4.3: SHAP Values for CIC_IOT23

more time than training Random Forest (nearly 1 hour), we decided not to train SVM models with all hyperparameters pairs to test the accuracy. With these hyperparameters, we trained the SVM model and used it to classify the test set. The results obtained are as follows. The overall accuracy of the model is 0.797. Table 4.4 shows all the metrics for different classes. Figure 4.4 shows the confusion matrix.

Class	**Recall**	**Precision**	**F1-Score**
DDoS	0.985	0.599	0.745
DoS	0.712	0.989	0.828
Mirai	0.999	0.995	0.997
Recon	0.648	0.823	0.725
Spoofing	0.891	0.644	0.747
Benign	0.724	0.736	0.73

Table 4.4: Different Metrics for Each Class of SVM Model on CIC_IoT23

The results clearly show that SVM model only performs well when classifying Mirai class but has low accuracy for all other classes.

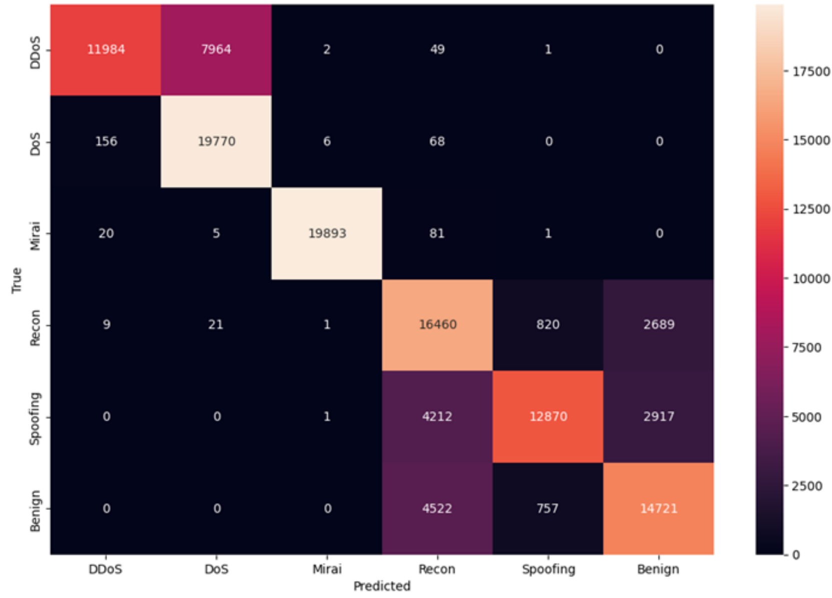

Figure 4.4: Confusion Matrix of SVM Model on CIC_IoT23

4.2.3 CNN Results

To train the CNN model, we decided to train for 10 epochs with a learning rate of 0.001 first. In this case, the loss decreased fast and steadily, from 0.4435 to 0.3391. Then, we reduced the learning rate to 0.0001 and trained the model for another 50 epochs. We noticed that the loss is dropping in small steps. Finally, we set the learning rate to 0.00001 and trained it for 200 epochs. Figure 4.5 illustrates the trend of loss reduction over epochs with different learning rates applied. After CNN model was trained, we used it to classify the test set. And we got the overall accuracy of 0.9158. Table 4.5 shows all the metrics for different classes. Figure 4.6 shows the confusion matrix.

It's obvious that the performance of CNN model is quite similar to the Random Forest model, it performs well with the DDoS, DoS and Mirai classes, but accuracy declines when classifying the Recon, Spoofing and Benign classes.

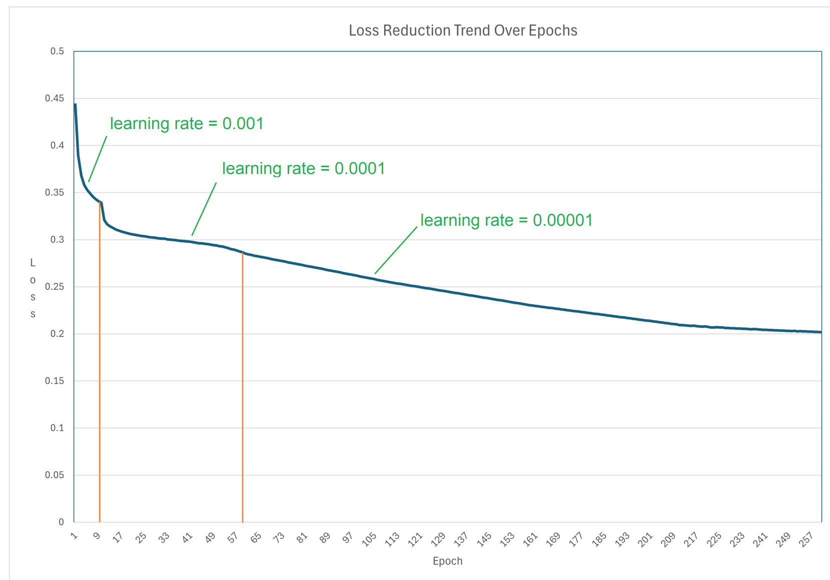

Figure 4.5: Loss Reduction Trend over Epochs on CIC_IOT23

Class	Recall	Precision	F1-Score
DDoS	0.994	0.988	0.991
DoS	0.988	0.993	0.99
Mirai	0.998	0.998	0.998
Recon	0.834	0.84	0.837
Spoofing	0.849	0.814	0.831
Benign	0.833	0.863	0.848

Table 4.5: Different Metrics for Each Class of CNN Model on CIC_IoT23

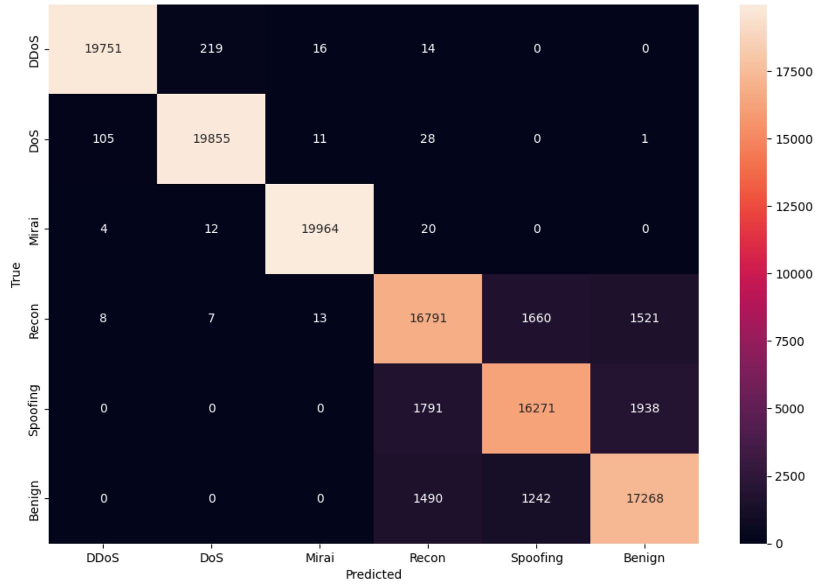

Figure 4.6: Confusion Matrix of SVM Model on CIC_IoT23

4.3 UGR'16

After training and testing different models on the CIC_IoT23 dataset, we have gained the better understanding of using the machine learning models as network anomaly classifier. Next, we will discuss the details about implementing the experiments of training and testing machine learning models on UGR'16 dataset.

4.3.1 Random Forest Results

Before training the Random Forest model, a grid search was performed to find the optimal hyperparameters for the model as we done previously with CIC_IoT23 dataset. The result from the grid search indicated that the optimal hyperparameters are 100 for the number of estimators and 20 for max depth. Also, we have trained different Random Forest models with all combinations of hyperparameters and calculated the overall accuracies for classification on the test set. The accuracies are illustrated in table 4.6 and the time cost in seconds for training the model is illustrated in table 4.7.

In this case, we can observe that the Random Forest model has already reached its best performance when the max depth was set to 20 and the number of estimators was

Overall Accuracy	**max depth=10**	**max depth=20**	**max depth=30**
estimator=10	0.9845	0.9964	0.9961
estimator=50	0.9956	0.9967	0.9964
estimator=100	0.9958	0.9967	0.9964
estimator=500	0.9957	0.9966	0.9963

Table 4.6: Accuracy with Differen Hyperparameters on UGR'16 with Random Forest

Time Spent (Second)	**max depth=10**	**max depth=20**	**max depth=30**
estimator=10	0.61	0.72	0.72
estimator=50	2.98	3.62	3.56
estimator=100	5.72	7.27	7.22
estimator=500	26.34	32.55	33.31

Table 4.7: Time Spent with Differen Hyperparameters on CIC_IoT23 with Random Forest

50 or 100.

With the optimal hyperparameters, we started training the Random Forest model on the training set. The overall accuracy we got when the trained model classified the test set is 0.997. Different metrics for each class are illustrated in table 4.8. Figure 4.7 shows the confusion matrix.

Class	**Recall**	**Precision**	**F1-Score**
background	0.99	0.99	0.99
dos	0.999	1	1
nerisbotnet	0.995	0.992	0.994
scan	0.998	0.999	0.998
anomaly-spam	0.998	0.999	0.998
anomaly-udpscan	1	1	1

Table 4.8: Different Metrics for Each Class of Random Forest Model on UGR'16

By checking the results, it is shows that the Random Forest model performs very well in classifying network traffics on UGR'16 dataset.

4.3.2 SVM Results

A grid search has been performed to find the optimal hyperparameters pair. And the best combination of hyperparameters identified by the grid search is C=100 and gamma=1.

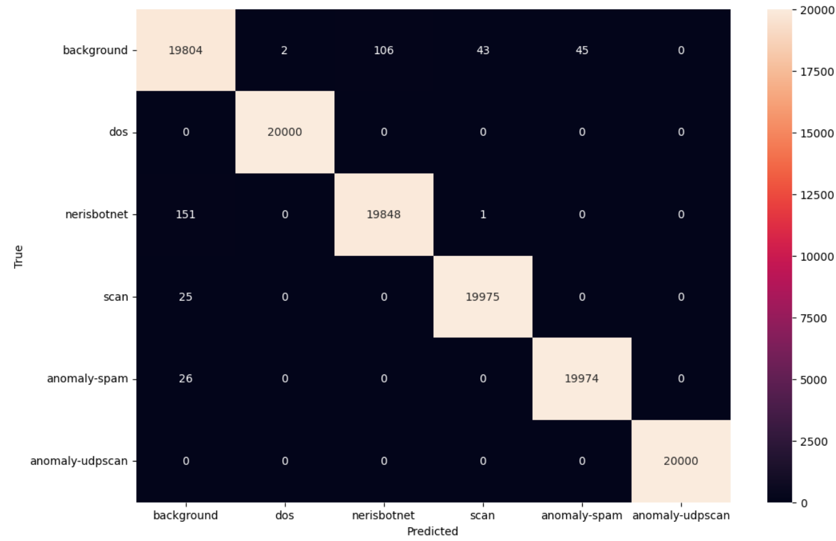

Figure 4.7: Confusion Matrix of Random Forest Model on UGR'16

Unlike the situation in training SVM model on CIC_IoT23 dataset, the time required to train a SVM model on UGR'16 dataset is acceptable. Therefore, we can investigate the overall accuracies of SVM models when set with different hyperparameters. Table 4.9 shows the accuracies and Table 4.10 shows the time spent on training and predicting.

Accuracy	**gamma=1**	**gamma=0.1**
C=0.1	0.932	0.9185
C=1	0.9438	0.9297
C=10	0.9556	0.9393
C=100	0.9685	0.9522

Table 4.9: Accuracy with Differen Hyperparameters on UGR'16 with SVM

Then, we applied the optimal hyperparameters to train and test the SVM model on UGR'16 dataset. The overall accuracy we got is 0.968. Table 4.11 illustrates the metrics for different classes being predicted. Figure 4.8 shows the confusion matrix:

From the results, we can notice that the SVM model we trained generally performs well in classifying the UGR'16 dataset. However, it has a little issue, which would misclassify the background flows as nerisbotnet attacks.

Time Spent (Second)	**gamma=1**	**gamma=0.1**
C=0.1	383.52	763.66
C=1	298.91	445.38
C=10	265.6	363.71
C=100	289.91	301.1

Table 4.10: Time Spent with Differen Hyperparameters on CIC_IoT23 with SVM

Class	**Recall**	**Precision**	**F1-Score**
background	0.971	0.842	0.902
dos	0.998	1	0.999
nerisbotnet	0.883	0.986	0.932
scan	0.977	0.999	0.988
anomaly-spam	0.994	0.983	0.989
anomaly-udpscan	0.999	1	0.999

Table 4.11: Different Metrics for Each Class of SVM Model on UGR'16

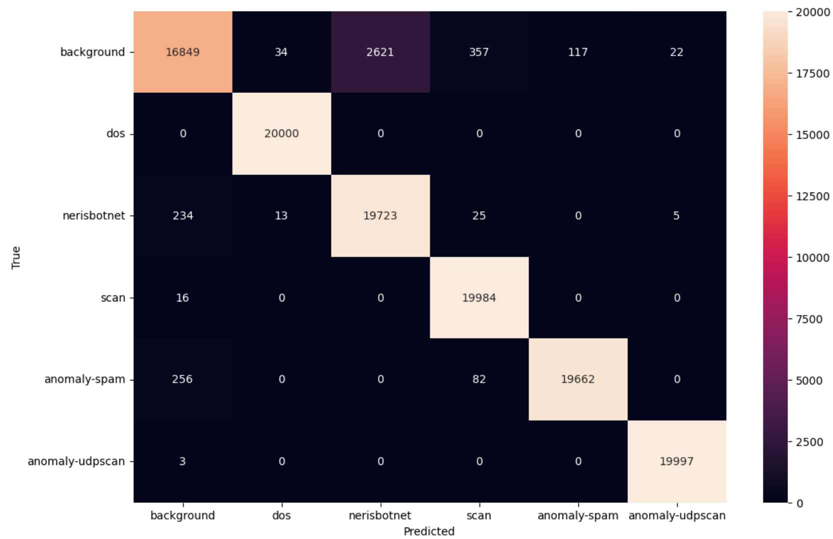

Figure 4.8: Confusion Matrix of SVM Model on UGR'16

4.3.3 CNN Results

When training the CNN model on the UGR'16 dataset, we initially trained for 10 epochs with learning rate set to 0.001. After that, we performed training for 50 epochs with learning rate set to 0.0001. Finally, we trained the model for another 50 epochs with learning rate set to 0.00001. Figure 4.9 illustrates the trend of loss reduction over epochs with different learning rates applied.

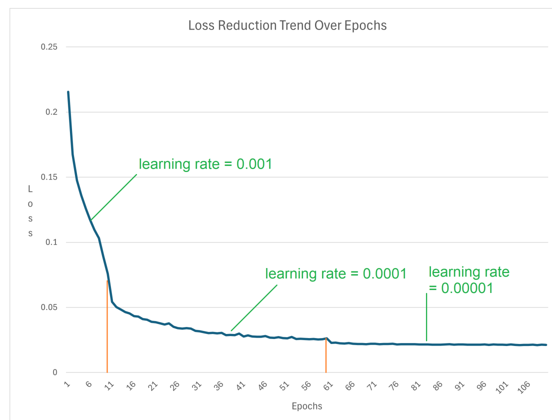

Figure 4.9: Loss Reduction Trend over Epochs on UGR'16

It is obvious that after 60th epoch, the loss the loss no longer decreased. Then we

used this trained CNN model to make predictions on the test set, and the overall accuracy we got is 0.994. Table 4.12 shows the important metrics for all classes. Figure 4.10 shows the confusion matrix.

Class	Recall	Precision	F1-Score
background	0.985	0.977	0.981
dos	0.999	1	0.999
nerisbotnet	0.985	0.99	0.987
scan	0.997	0.999	0.998
anomaly-spam	0.996	0.996	0.996
anomaly-udpscan	0.999	1	0.999

Table 4.12: Different Metrics for Each Class of SVM Model on UGR'16

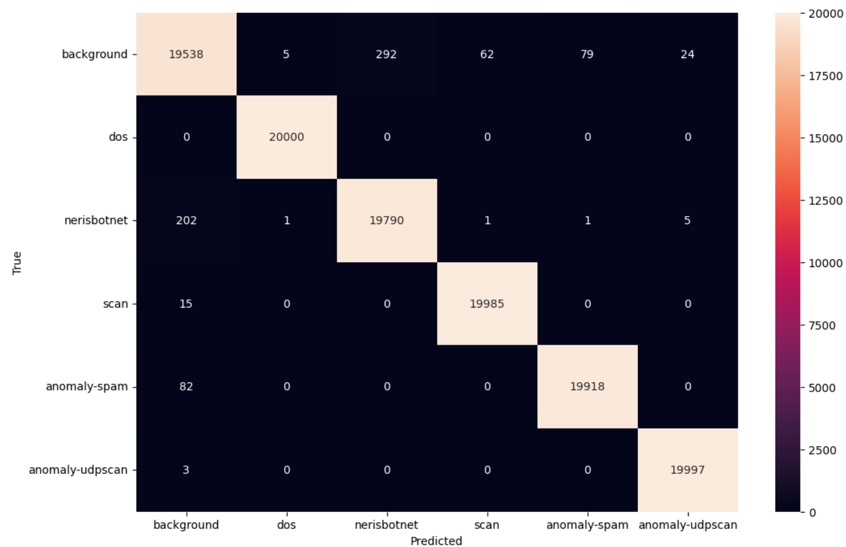

Figure 4.10: Confusion Matrix of CNN Model on UGR'16

From the results, it is clear that the CNN model performs nearly perfectly when classifying different classes of network traffic flows in UGR'16 dataset.

4.4 Result Analysis

Table 4.13 illustrates all the overall accuracies of all models in 2 datasets.

Overall Accuracy	CIC_IoT23	UGR'16
Random Forest	0.951	0.997
SVM	0.797	0.968
CNN	0.916	0.994

Table 4.13: Accuracies for All Models

First, we focus on the results of CIC_IoT23 dataset. It's obvious that when machine learning models are trained based on this dataset, only the Random Forest model can perform as expected. If we look at the Confusion Matrix in figure 4.2 and figure 4.6, we can identify the main problem faced by machine learning models is in classifying the Recon, Spoofing and Benign classes. Since this problem occurs in all three models, we can deduce that data for these three classes have some underlying patterns in common, which would make it difficult for the machine learning models to separate them.

Then, we consider the results of UGR'16 dataset. We observe that all 3 models perform well when classifying the network traffic flows in UGR'16 dataset. Compared to the model performance in CIC_IoT23 dataset, we can make a judgement that in this project, the UGR'16 dataset's data quality is better than the data quality of CIC_IoT23 dataset. We believe the method for network attacks implementation to the dataset may be the reason that makes machine learning models easier to classify different classes in UGR'16 dataset. The attacks that researchers conducted when creating UGR'16 dataset may have limited patterns, while the attacks in CIC_IoT23 dataset might be more changeable.

For both datasets, the SVM model don't have a great performance. There could be a few reasons for this problem. First, the data in both datasets is high-dimensional and non-linear. During the learning of SVM, it tries to keep increasing the dimensions to find linearity, which may lead to the Curse of Dimensionality.[21] And compared to Random Forest and CNN, SVM model may lack generalization ability when facing large-scale dataset.[22]

For CNN model in CIC_IoT23 dataset, its performance did not meet our expectation. At the beginning, we thought it could be improved by adding more layers to the CNN.

However, the overall accuracy did not increase after more convolutional layers were added. Then we consider that might be fixed by training for more epochs. However, during the training from 300 to 500 epochs, the loss of CNN model did not decrease at all. And as a result, the overall accuracy was worse than before, indicating that the model was overfitting. However, when the CNN model with the same structure was trained on the UGR'16 dataset, the speed for loss reduction over epochs was very quick. As mentioned above, the loss is low enough when it reached the 60th epoch. And the overall accuracy for classification is over 99%. When we look at the confusion matrix in figure 4.10, it is obvious that the CNN model has high accuracy for all classes, unlike its poor performance in some classes when classifying the CIC_IoT23 dataset, which means it successfully learned the underlying patterns among all classes and was able to distinguish all classes with different patterns. Therefore, we believe the reason the CNN model performs much better in the UGR'16 dataset than in the CIC_IoT23 dataset is due to the data quality of UGR'16 being better than that of CIC_IOT23. The key factor in the CNN model having a better performance in UGR'16 than in CIC_IoT23 is that all classes in UGR'16 are distinct from each other, but some classes in CIC_IoT23 are mixed with each other.

Chapter 5

Conclusion

This section will discuss what have been achieved in this project, and what could be improved and expanded in the future.

5.1 summary

At the beginning of this project, I successfully found the datasets for later usage in the project, which are CIC_IoT'23 and UGR'16. After that, the machine learning algorithms were also found based on the investigation of the existing machine learning algorithms for network anomaly detection. Then, primitive experiments were performed to help me understand the datasets and algorithms. Furthermore, data analysis and data pre-processing were executed to prepare for the later model training. In the model training part, different models with different hyperparameters were trained to seek the best method to classify the network traffic flows. Finally, I analyzed the results for classifications from the trained machine learning models and compared their performances. To visualize the results, I've integrated all models into the Python library ExplainerDashboard.

5.2 Achievements

This research successfully accomplished its objectives, which are training machine learning models with different algorithms on different datasets and comparing their performance. After the experiments and comparisons, we can conclude that with the same presets as in this project, Random Forest and CNN models perform better than SVM model. What's more, due to the short training time, Random Forest can be considered more effective than CNN.

Personally, I've learned and practised the techniques for pre-processing network traffic data and training machine learning models on it, as well as the techniques to adjust hyperparameters and analyse the results.

5.3 Future Work

In this project, there are two flaws that were overlooked. Firstly, feature selection wasn't applied to Random Forest models and CNN models. Feature selection is an essential technique in machine learning, which can improve the models' performance by helping models to learn the more important patterns. Secondly, while experimenting on the CIC_IoT23, 2 classes were dropped due to their lack of samples. However, these 2 classes might still be learnable through some specialised techniques.

Looking ahead, I believe there are two objectives that can be achieved. The first is to use unsupervised learning to train a model that can learn the patterns of normal network traffic flows and classify whether the incoming network flows are behaving normally. Initially, an unsupervised learning model, an Autoencoder, was planned to be implemented. However, the training of this model did not proceed as expected, so it had to be abandoned. The second objective, which I was eager to achieve before choosing this project, is to apply machine learning models to real Internet environments to detect anomalous network behaviours. Due to the challenges in implementation, it was not feasible to undertake this within the scope of this project. However, with the foundation laid by this project, I might be able to realise it in the future.

References

[1] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials*, 16(1):303–336, 2014.

[2] T. Ahmed, B. Oreshkin, and M. Coates. Machine learning approaches to network anomaly detection. In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, pages 1–6. USENIX Association, April 2007.

[3] S. Wang, J. F. Balarezo, S. Kandeepan, A. Al-Hourani, K. G. Chavez, and B. Rubinstein. Machine learning in network anomaly detection: A survey. *IEEE Access*, 9:152379–152396, 2021.

[4] R. Primartha and B. A. Tama. Anomaly detection using random forest: A performance revisited. In *2017 International Conference on Data and Software Engineering (ICoDSE)*, pages 1–6, Palembang, Indonesia, 2017.

[5] Y. Zhang, Q. Yang, S. Lambotharan, K. Kyriakopoulos, I. Ghafir, and B. AsSadhan. Anomaly-based network intrusion detection using svm. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, Xi'an, China, 2019.

[6] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim. An empirical study on network anomaly detection using convolutional neural networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1595–1598, Vienna, Austria, 2018.

[7] E.C.P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A.A. Ghorbani. Ciciot2023: A real-time dataset and benchmark for large-scale attacks in iot environment. *Sensors*, 23:5941, 2023.

[8] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. Ugr'16: A new dataset for the evaluation of cyclostationarity-based network idss. *Computers & Security*, 2018.

[9] M. Ristin, M. Guillaumin, J. Gall, and L. Van Gool. Incremental learning of random forests for large-scale image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(3):490–503, March 2016.

[10] Dion de Hoog. A personalized approach for communicating found anomalies in netflow data to end-users. *Delft University of Technology*, 2021.

[11] Peshawa J. Muhammad Ali and Rezhna H. Faraj. Data normalization and standardization: A technical report. *Machine Learning Technical Reports*, 1(1):1–6, 2014.

[12] A. F. Agarap. Deep learning using rectified linear units (relu). *arXiv*, 2018.

[13] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley & Sons, 2004.

[14] P. Liashchynskyi and P. Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas. *arXiv*, 2019.

[15] O. Kramer. Scikit-learn. In *Studies in Big Data*, pages 45–53. Springer International Publishing, 2016.

[16] Y. Zhong, X. Lin, and L. Zhang. A support vector conditional random fields classifier with a mahalanobis distance boundary constraint for high spatial resolution remote sensing imagery. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(4):1314–1330, 2014.

[17] S. Han, Qubo Cao, and Meng Han. Parameter selection in svm with rbf kernel function. In *World Automation Congress 2012*, pages 1–4, Puerto Vallarta, Mexico, 2012. IEEE.

[18] Y.-W. Chen and C.-J. Lin. Combining svms with various feature selection strategies. In *Feature Extraction*, pages 315–324. Springer Berlin Heidelberg, 2006.

[19] W. E. Marcilio and D. M. Eler. From explanations to feature selection: assessing shap values as feature selection mechanism. In *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. IEEE, 2020.

[20] M. Hossin and M. N. Sulaiman. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):1, 2015.

[21] Y. Bengio, O. Delalleau, and N. Le Roux. The curse of dimensionality for local kernel machines. Technical Report 1258(12), Techn. Rep, 2005.

[22] T. Joachims. Estimating the generalization performance of a svm efficiently. Technical Report 2001, 20, Technical Report, 2001.