National College of Ireland

Botnet Detection Using Multi-D Convolutional Neural Network

Academic Internship
MSc Cybersecurity

Shalini Srinivasan
Student ID: x21208000

School of Computing
National College of Ireland

Supervisor: Mr Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shalini Srinivasan
Student ID:	x21208000
Programme:	MSc in Cybersecurity **Year:** 2023
Module:	Academic Internship
Supervisor:	Mr Vikas Sahni
Submission Due Date:	15th August 2023
Project Title:	Botnet detection using Multi D Convolutional Neural Network
Word Count:	7265

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shalini Srinivasan

Date: 10/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Botnet Detection Using Multi-D Convolutional Neural Network

Shalini Srinivasan
21208000

Abstract

DDoS attacks pose a significant threat to the network of every organisation. The damage caused by DDoS on a renowned company Bandwidth Inc. was around 10 million for a fiscal year.[1] And, as cybercriminals are building more and more sophisticated botnets, there is need for newer techniques, therefore, this paper presents an approach for the detection of HTTP, IRC, and P2P botnets. The dataset used is CTU 13 containing 13 different scenarios to study upon. To avoid any false positives, the data is further categorised based on the flow information in bytes. After the pre-processing of the data, a unique approach of CNN called Multi-D CNN model is considered, that detects legitimate, suspicious, or malicious traffic. Upon analysis, it was concluded that using categorical prediction, the Multi-D CNN model has an accuracy of 73.5%.

1 Introduction

Recently, the number of DDoS attacks has surged dramatically. According to the report (Cook, 2023), 2022 was the turning point for DDoS attacks as most attacks were larger and lasted longer than in previous years. Initially, DDoS attacks lasted 30 minutes on average in Q2 of 2021, but by the end of the year, they lasted over 50 hours. Online industries were the most targeted, with a 131% increase each quarter and a 300% increase year on year.
(Mccart, 2022) suggests that 5.4 million DDoS attacks were launched in the first half of 2021, with the number increasing at a rate of 11% year on year. However, most of these attacks are carried out using botnets. Botnets are compromised networks of devices used to gain control of a company's network. Previously, most of these attacks were carried out by centralized botnets, also known as traditional botnets, which use a single command and control server to receive commands. IRC and HTTP are examples of centralized botnets that use a single point of failure communication channel. However, in recent days, Peer-to-Peer botnets have emerged. P2P botnets are difficult to detect because to their dispersed structure.

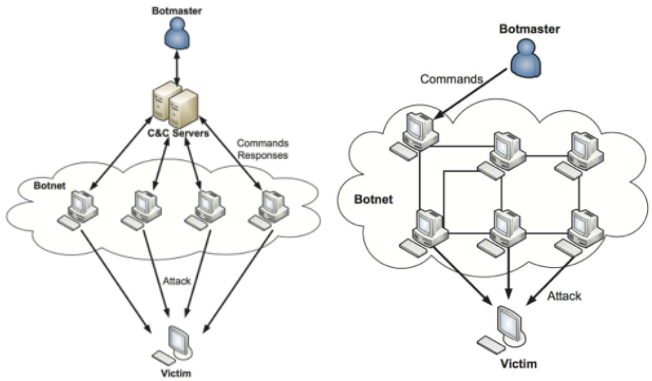

Figure 1: Centralised and P2P botnet architecture (Mahmoud et al., 2015)

[1] https://www.indusface.com/blog/ddos-attack-cost-bandwidth-com-nearly-12-million-how-to-protect-your-site-against-one/

Previous botnet detection research has included signature-based detection, network-based detection, DNS-based detection, anomaly based, and Hybrid based detection. Network based focuses on finding specific patterns in the network traffic. This detection technique has various subcategories, signature based is one of them. Signature-based detection uses existing signatures to identify botnets and so cannot detect new or unknown bots. DNS detection relies on DNS information provided by the botnet. As connections to the C2C server are established to receive commands, the bots utilize DNS queries to identify and communicate to their C2C server. As this detection method solely rely on DNS traffic, using only this method for the detection is not ideal for the detection. Hybrid-based detection, on the other hand, combines the two different types of detections to improve accuracy of the detection.

However, in terms of the algorithms used, most of CNN detections have provided ample evidence that they can detect botnet activities and analyze network behavior. Existing solutions, on the other hand, primarily target centralized botnets and P2P botnets with known signatures and a specific monitored network. (Khoh Choon Hwa & Mahmood Al-Shareeda, n.d.; Shetu et al., 2019; Yang & Wang, 2019)

Therefore, this paper proposes an approach to detect IRC, HTTP and P2P botnets from the CTU 13 dataset. The flow characteristics of the network traffic is considered to determine whether the traffic is malicious, suspicious, or legitimate. The approach employs a unique technique of CNN called Multi-D CNN through categorical prediction as it can handle huge datasets effectively and, in fact, produces better results with large datasets.

Research Question: How well does Multi–D CNN detect malicious, suspicious, and legitimate traffic?

2 Related Work

2.1 Network Based Detection

(Saad et al., 2011a) collects data from network traffic, which is then categorized using flow-based and hybrid analysis. Storm and Waledac are two datasets that feature a variety of packet communication. A combined dataset was created by mapping IP addresses to other devices during traffic production. To replay traffic intercepted using Wireshark, a tool called tcpreplay was utilized. This dataset produced around 129,453 feature vectors, which were merged and classified into three categories: botnet normal traffic, P2P traffic, and non-P2P traffic.

On the other hand, (Duan et al., 2022) describes a method for generating data in which a zombie is used. A Deep Flow Inspection (DFI) analysis was utilized, allowing the packets to be combined into a quintuple data stream. The primary columns are SourceIP, DestinationIP, SourcePort, DestinationPort, and Protocol, which includes TCP and UDP protocols. The dataset contained many spatial and temporal features, which reduced duplicated data. However, the payload is destroyed upon the retrieval of the length, IP address, time, and port. After processing, the data is separated into streams using statistical analysis to produce feature vectors. From 4gb of data, approximately 500,000 streams and 248 characteristics were retrieved. The limitation of this article highlights that the paper (Saad et al., 2011a) may apply the method to prior version bots, but as the bots upgrade themselves, it is difficult to detect them as new commands are given across the network. (Duan et al., 2022) research article, on

the other hand, overcomes this problem but cannot deal with online data since it solely considers static traffic.

Finally, the articles employ a variety of machine learning techniques, with (Saad et al., 2011a) paper's method achieving 90% accuracy for SVM. Whereas (Duan et al., 2022) paper demonstrated 99.2% accuracy due to the use of various methods.

On the contrary, the (Guntuku et al., n.d.) paper research provides an alternative way to the paper (Duan et al., 2022), in which real-time traffic can be employed for prediction. A network mirroring was built that mirrored all traffic that passed across it. This aided in running the application servers concurrently. Tshark was used to further investigate the packets, which were stored in libpcap format in 200 mb chunks. After this procedure is completed, the packets become flows, and each of them is further examined for potentially malicious content.

(Saad et al., 2011b) research proposes a method for identifying network traffic using a behavior-based methodology that investigates various network characteristics. The factors evaluated are packet size, the number of requests between two devices, and so on. payload information, detailed packet information such as the duration of the replicated packet, and active ports on the featured set are considered by the model. The methodology is divided into two distinctions: flow-based and host-based. The features associate the flows with a certain network traffic type and provide the necessary classification, P2P and non-P2P traffic. Other features allow host-based tracking between the two hosts which helps identify different communication patterns.

(Saad et al., 2011b) is a continuation of (Saad et al., 2011a) which uses the same algorithm for the detection of the botnet. Except that the datasets considered in (Saad et al., 2011a) are for storm and Walowdac. The labels on both papers are similar in type, with the source and destination IP addresses and port information, packet length and payload size, and so on. These papers also employ comparable machine learning models.

The paper (Alauthaman et al., 2018) presents an entirely different technique for detecting abnormalities in traffic by the inspection of TCP packets allowing significant decrease in the volume of network traffic. The two-stage filtering process provides the data to the feature extractor which divides the data into 29 tuples every 30 seconds. As the paper discusses only about TCP packets, UDP packets cannot be processed by the model.

(Ravindra Vishwakarma, n.d.) used the CTU 13 dataset, and it was further classified using a feature selection technique such as univariate feature with various statistical tests, random selection by recursive elimination, random forest, and logistic regression. Through various under sampling and oversampling strategies, as well as ensemble learning, the major algorithms used were random forest and decision tree. Through imbalance learning and under sampling, the results achieved are 83%.

2.2 Signature Based Detection

The author of (Suthar et al., 2022) provides a technique that considers the port, location, and IP address. The reputation of the IP address is used to generate a severity score. The Snort IDS routes the IP using two primary components: the port and the protocol. A table is constructed that contains all the criteria and validates the signatures. The generator is alerted using a generic signature with no parameters. The generator has two output settings: low and high. If the

severity is high, the generator will block all traffic; if the severity is low, it will allow communication over the port and protocol after showing a message. The emotet botnet was used to conduct the analysis. The analysis cannot be performed on any other signature-based research because only the emotet botnet was used.

However, the PGA filter discussed in (Szynkiewicz, 2022) research is an example of an intrusion detection system (IDS) primarily used for the detection of botnets conducting DoS attacks. A third approach is provided, in which the signatures are generated by implementing packet filters in network traffic. With a protocol (which primarily focuses on the TCP number sequence and the IP address of the destination), the observed patterns are monitored and categorised as a single byte or a variety of multiple bytes of data. Port scanning is used in the procedure for comparing the IP address and the TCP sequence number. This allows all the PGA to be transferred from the IP address to the sequence number without than having to create a new one. The main goal was to improve the calculation of the structure by applying a power constraint. The primary limitation is that it can only detect known bots.

The research on (Hossein Rouhani Zeidanloo & Saman Shojae Chaeikar, 2010) framework gives a mechanism for locating botnets with comparable patterns and signatures that participate in malicious activities. The detection system is built around four primary components: filtering, traffic monitoring, a malicious activity detector, and an analyzer. The filtering tool removes unnecessary traffic and flows, increasing system efficiency. The malware detector scans network data for harmful behaviour on hosts and isolates them for further investigation. Traffic monitoring ensures that the network is being monitored to identify hosts that display similar behaviour and communication. Finally, to acquire the best findings, the analyzer compares the results of traffic monitoring and harmful activity detection.

(Alzahrani & Ghorbani, 2015) describes a signature-based strategy for detecting botnets using SMS. After training the model with existing signatures, the returned signatures are compared to the known ones to decide whether they are dangerous or not. The approach proposed in the study employs a model that includes up-to-date signatures of known botnets and can correctly detect malware. The sender's phone number, SMS text, and any embedded URLs are analysed and fed into the model as a dataset for effective detection. If a match is detected, the SMS is banned. If no matches are found, the system scans the SMS body further, recognising token strings and comparing them to the established signatures to extract URLs, phone numbers, and commands.

2.3 Hybrid Detection Techniques

(Bhatia et al., 2020) proposed a hybrid based analysis that uses a combination of heuristic and statistical. The heuristic approach provides a collection that is used for the categorisation of traffic whereas the statistical analysis provides a traffic flow based approach. The data computed the mean, average, variance and duration. It provides identification of P2P using heuristic technique by classification of traffic. The part that is unclassified is further classified through flow analysis. Factors such as the source, destination of IP is used to calculate the hash whereas irrelevant data is eliminated. A hask key is used for the mapping of the P2P database. If a match is found for the mapping, the packets are selected and the procedure is repeated. The main problem with the technique is the function with limited number of applications.

The paper (Almutairi et al., 2020) presents a hybrid detection method based on network and host analysis. To process the dataset based on certain features, an algorithm named HANAbot

(Host and Network Analyzer for Botnet identification) was developed. This method is divided into two stages. To begin, malware signatures were acquired using a honeypot to infect the VM host and monitor the activity. A sniffer, Wireshark, and a malware analysis programmed are employed to extract patterns between the server and the host. Other tools are employed to analyse the acquired data. The second stage involves developing the algorithm to extract certain features. Data pre-processing is used to remove unnecessary elements before concluding. This study also discusses the size of packets and bytes flow for regular, suspicious, and abnormal traffic.

(Kingma & Ba, 2015) proposes a study of identifying botnet attacks on Iot networks. The model was a combination of convolutional neural networks and long short-term memory which focuses on the detection of BASHLITE and mirai. The data was collected from security cameras in an Iot environment. According to the authors, they achieved a precision of 88%, a recall of 87% and the F1 score of 83% for a provision PT-737E camera. Whereas, for provision PT-838, the recall value was 89%, F1 score was 85% and precision of 94%.

2.4 Behaviour Detection

(Garg et al., 2013) focuses on the C&C phase to detect the botnet. The network traffic is categorised into legitimate and malicious traffic by selecting specific attributes of the network. In case of legitimate p2p botnet, they have packets of larger size as compared to C&C botnets, this is done so that they remain unnoticed. The paper considers a combination of different characteristics which uses packet size, the number of packets, payload size to classify the traffic in the model. The C&C behaviour is analyzed by selecting features of the traffic which can distinguish the traffic between normal and botnet traffic. The paper adopted ML algorithms which used perl script to extract flow vectors which was sent to the MySQL database. Normal non p2p contains data of HTTP, FTP, SMTP and normal P2P consisted of skype, bittorrent, e donkey etc.

(Nagaraja, 2014) proposes a detection of P2P botnets which combines the traffic flows and the collaboration of the infected hosts. The main combination of the theory is a graph-theoretic and statistical analysis to ensure that the feedback loops are considered. This loop was designed to ensure that the stochastic diffusion process is followed over traffic flows which are similar. Random walks are used for the graph partitioning techniques. Edge similarity is incorporated through theoretical tools where each traffic flow is considered in scalar terms though cartesian coordinates. These points constitute to multi-dimensional geometric space in the algorithm with structural information. The algorithm created a geometric surface on the communication graph and includes the traffic flow as well. The traffic is represented through the traces formed on the communication graph which defines the special random walk.

The approach proposed (Ibrahim et al., 2021) uses flow as well as behavior based. It consists of two modules, filtering module and the detecting C&C server. The main purpose was to filter out the network traffic so that it can be used for the next phase. The filtering module uses a semi supervised concept of labelling the dataset partly using patterns of another unlabeled dataset (normal and botnet). As the main purpose is to filter the network, the number of features and group considered was at a minimum time interval (1s interval). After the clustering the data was transferred to C&C server for detection.

(Alharbi & Alsubhi, 2021) suggests a graph-based approach which uses naïve bayes, decision tree and random forest to extract features for the detection of botnets through various

behavioral characteristics suitable for large scale and zero-day attacks. The method uses different feature evaluation measures to improve the algorithms. The results obtained were extraordinary as they received a 100% recall on both the datasets used, with extratrees providing an accuracy between 99 and 100%.

The paper (Wang et al., 2014) presents a behavior-based botnet detection technique that detects botnets in two stages using fuzzy pattern recognition. The domain name service uses client DNS queries and conducts investigations depending on the malicious URL given by the bot. The transmission control protocol (TCP) employs request and response packets to identify access patterns that are unique in terms of packet size. The five detection stages proposed are traffic reduction, feature extraction, data partitioning, DNS detection, and TCP detection. The first phases reduce the amount of data that can be processed.
Behavior-based botnet detection in parallel (BBDP) is recommended for detection to improve efficiency. This BBDP focuses on DNS and TCP query inquiries. The detection method improves accuracy by employing a technique that retrieves domain names and IP addresses before sending data to the appropriate firewalls or IDS. The research conducted experiments in which the BPDP detected 95% of the bots with a false positive rate of less than 3.5%. They employ genuine traces from over 670 bots as well as malicious traffic from 240 active bots.

(Wang et al., 2011) suggests a detection method based on fuzzy pattern. In network traffic, domain names and IP addresses are examined. The methodology was developed by first reducing network traffic. The traffic goes through a reduction phase before being filtered to extract features in the feature extraction phase. The pattern recognition stage identifies the domain names and IP addresses that have been influenced by the extracted features. During testing, the model was able to eliminate approximately 70% of the traffic and obtain a detection rate of 95% with few false positives. Another feature of the model is that it is resource efficient and detects inactive botnets.

2.5 Anomaly based Detection

The study (Arshad et al., 2011) provides a method for detecting botnet traffic based on anomalies in real time. One advantage of this strategy is that the prototype is based on real-time network traces that distinguish between botnet and normal traffic. To analyse traffic, the technique consists of nine components. The traffic dispatching component, as well as the NetFlow generating and alert generating components, are used to route traffic to domain IP mapping. The domain IP mapping maps DNS domains to specific IP addresses for further filtration. TCP flows between hosts are generated by Netflow. However, the alter generating component detects malicious traffic from other hosts through scanning. The alter filtering ensures that unnecessary alerts are removed, while the NetFlow filtering filters flows generated by the NetFlow generating component through the database. The correlation engine is triggered at the end of each window to create alerts for clusters to detect bot-infected hosts. The method correctly recognized three out of four bots and received an accuracy of 100%.

(Nomm & Bahsi, 2019) demonstrates the ability of obtaining excellent accuracy from an unsupervised machine learning model with a reduced feature set. Initially, the flow is applied to the data to evaluate model performance irrespective of device type. The method builds a distinct model for each IoT device and trains it on unique feature sets. For the experiments, two distinct class labels were examined in each iteration. The original dataset was constructed by randomly separating 20292 malicious records from 2747 normal records. The dataset is inconsistent because the normal data used is only 11%. The second dataset, on the other hand,

is balanced since it contains nearly equal amounts of malicious and normal data. Although two different learning models were explored for accuracy and precision of greater than 90% for both datasets, it was determined that separate models for each IoT device perform better than a single common model.

(View of A Study Of Machine Learning Classifiers for Anomaly-Based Mobile Botnet Detection, n.d.) detects botnets on mobile devices by monitoring network traffic with machine learning algorithms. One significant advantage of machine learning models is that the computational burden is reduced. The research employs a variety of machine learning models, such navies bayes, decision trees, and k nearest neighbors (KNN). The procedure is comparable to any other machine learning process in that it involves data gathering, feature selection, data extraction, and data classification. The data is divided into regular and malicious traffic and collected separately. The classifier inspects the attributes in the following stage. The two types of data are combined, randomized, and labelled. The dataset is then put into and compared using five different classifiers. When compared to other models, KNN had the highest AUC value of 1.00. The decision tree came in second with a value of 0.995, followed by the SVM with a value of 0.994, while the naive bayes and MLP both had a value of 0.979.

3 Research Methodology

3.1 Dataset collection and experimental setup

This section discusses the datasets used for the analysis. An experimental setup was formed to check the actual prediction accuracy of the model.

3.1.1 Dataset description

The CTU 13 dataset is used in this paper for the analysis of variety of botnet attacks. It was initially compiled by CTU in Prague which contains data of the network captured in a real time setup. There are 13 separate botnet samples in the dataset. IRC, DDoS, Port Scan, and HTTP traffic are a few examples. Most of these attacks include Distributed Denial of Service (DDoS) as well botnet activity. The dataset contains network traffic information which includes source IP address, Destination IP address, kind of service, total bytes, total packets, and traffic label. The dataset shows a link between several characteristics such as time, amount of packets, netflows, and the size of pcap files, as well as the bot used for request capture for further categorisation. The table below shows the type of data produced by each kind of botnet.

Id	Duration(hrs)	# Packets	#NetFlows	Size	Bot	#Bots
1	6.15	71,971,482	2,824,637	52GB	Neris	1
2	4.21	71,851,300	1,808,123	60GB	Neris	1
3	66.85	167,730,395	4,710,639	121GB	Rbot	1
4	4.21	62,089,135	1,121,077	53GB	Rbot	1
5	11.63	4,481,167	129,833	37.6GB	Virut	1
6	2.18	38,764,357	558,920	30GB	Menti	1
7	0.38	7,467,139	114,078	5.8GB	Sogou	1
8	19.5	155,207,799	2,954,231	123GB	Murlo	1
9	5.18	115,415,321	2,753,885	94GB	Neris	10
10	4.75	90,389,782	1,309,792	73GB	Rbot	10
11	0.26	6,337,202	107,252	5.2GB	Rbot	3
12	1.21	13,212,268	325,472	8.3GB	NSIS.ay	3
13	16.36	50,888,256	1,925,150	34GB	Virut	1

Figure 2: CTU 13 Netflow Table

As the dataset was labelled, it was also classified according to the type of flows and the percentage of flows in each group. The labelling was done within the NetFlows files. Table

below shows the link between the total flows for each label and provides distribution of labels assigned to netflows for each type. [2]

Scen.	Total Flows	Botnet Flows	Normal Flows	C&C Flows	Background Flows
1	2,824,636	39,933(1.41%)	30,387(1.07%)	1,026(0.03%)	2,753,290(97.47%)
2	1,808,122	18,839(1.04%)	9,120(0.5%)	2,102(0.11%)	1,778,061(98.33%)
3	4,710,638	26,759(0.56%)	116,887(2.48%)	63(0.001%)	4,566,929(96.94%)
4	1,121,076	1,719(0.15%)	25,268(2.25%)	49(0.004%)	1,094,040(97.58%)
5	129,832	695(0.53%)	4,679(3.6%)	206(1.15%)	124,252(95.7%)
6	558,919	4,431(0.79%)	7,494(1.34%)	199(0.03%)	546,795(97.83%)
7	114,077	37(0.03%)	1,677(1.47%)	26(0.02%)	112,337(98.47%)
8	2,954,230	5,052(0.17%)	72,822(2.46%)	1,074(2.4%)	2,875,282(97.32%)
9	2,753,884	179,880(6.5%)	43,340(1.57%)	5,099(0.18%)	2,525,565(91.7%)
10	1,309,791	106,315(8.11%)	15,847(1.2%)	37(0.002%)	1,187,592(90.67%)
11	107,251	8,161(7.6%)	2,718(2.53%)	3(0.002%)	96,369(89.85%)
12	325,471	2,143(0.65%)	7,628(2.34%)	25(0.007%)	315,675(96.99%)
13	1,925,149	38,791(2.01%)	31,939(1.65%)	1,202(0.06%)	1,853,217(96.26%)

Figure 3: Netflow distribution of each label

3.1.2 Dataset Preprocessing

To remove any additional noise from the dataset, further categorisation of the dataset based on the srcbytes, destbytes and totbytes depending on the protocol used. The paper (Almutairi et al., 2020)a clear indication in case of a HTTP protocol, there are three ranges which can be considered depending on the source and the destination. The ranges categorised are as follows: 0-500, 501-1000, and 1001-1500 bytes. The different cases classified are: i) if the http source bytes are in the first range (0-500) and the destination bytes is in the same range, then the behaviour is considered malicious. ii) If the source bytes are in the range 0-500 and the destination bytes is in a different range (501-1000 or 1001-1500), then the traffic is considered normal.

However, if the protocol is UDP and the source bytes is between 36 and 67, then it is an abnormal behaviour but not a malicious one. Smaller bytes of packets are sent to ensure that the connection is maintained. Whereas, if the source packet size is greater than 120 bytes and the destination bytes is more than 400 bytes, then it is most likely a malicious connection. In case of a TCP connection, as a normal handshake takes place, the detection should be performed in the data transmission stage after the connection has been established. The main criterion for the identification includes large packet size which indicates that the traffic is legitimate as this reduces the overhead. However, in case of botnets, they use smaller packet size to ensure that they transmit data in an undetected manner.

Some of these conditions hold good only for a P2P network. But as the dataset is categorised based on the protocol and the label, it is assumed that any noise will be eliminated.

3.1.3 Experimental setup

The experimental setup shows how the packets were captured to train and test the model.

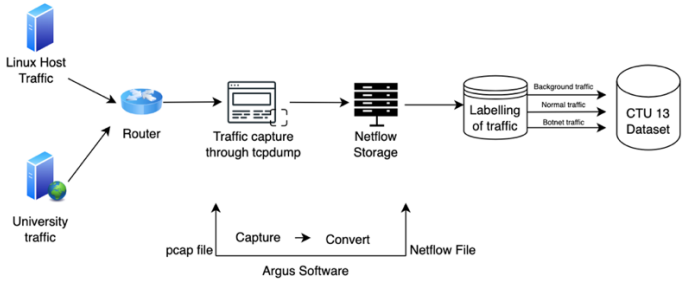

Figure 4: Experimental Setup

[2] https://www.stratosphereips.org/datasets-ctu13

The dataset contains 13 scenarios with different captures of real time botnet samples. The pcap file is generated from real time traffic. The features or characteristics are displayed through the tcpdump tool. By considering the characteristics of the traffic, the pcap files are converted into bidirectional argus storage. These bidirectional argus storage are then transformed into Netflow, resulting in the final Netflow file. Following this, the data is saved in NetFlow storage, and the traffic is further classified as background, regular, and botnet activity. The labelling is done by using a specialised filter that classifies background traffic as normal traffic. The botnet label is categorised based on the IP address, where any traffic incoming or outgoing from an infected IP address is considered as botnet. (Shamshirband & Chronopoulos, n.d.)

3.2 Design Architecture

3.2.1 Multi D CNN

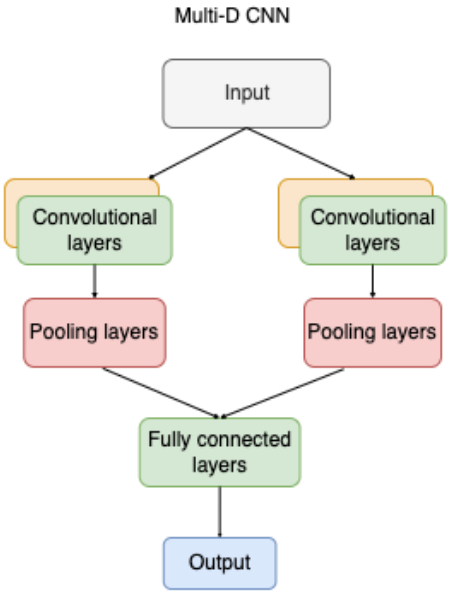

Figure 5: Multi-Dimensional CNN proposed architecture

3.2.1.1 Input Layer

The labelled pcap data are then processed further to extract certain botnet characteristics. This processed input file is then normalised for the model to understand. This layer converts the alphabetic data into numeric data through label encoding and then feeding it into the convolutional layers for further processing.

3.2.1.2 Layers of Convolution

These layers perform computation in the model. The method used employs two single CNNs to improve the model's efficiency by examining new features. The filters are employed while categorising the model so that the matrix can be reduced to fewer elements for faster processing. The kernel's size remains constant during the dot multiplication procedure. The formula for calculating the volume of output would be as follows:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Figure 6: Formula for Convolutional Layers

3.2.1.3 Pooling Layers

The pooling layer is typically separated into two layers: the max pool layer and the dropout layer. The max pool layer computes the highest weights of the matrices once the weights have been computed, indicating that these weights can be employed by the model. The dropout layer removes the lighter weights. The ReLU function is used to consider an activation map. Consider a map with dimensions W x W x D and a spatial size F and a stride S; the output is,

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Figure 7: Formula for Pooling Layer

Note: The two above layers are each considered for two separate CNNs and then combined through the fully connected layer using the concatenate function.

3.2.1.4 Fully Connected Layers

These layers identify the characteristics of the provided matrices and distinguish between malicious and legitimate traffic. The layer employs the activation function ReLU, which means that if the output is negative, the output is considered 0. If the output is positive, the actual value is considered. [3]

3.2.1.5 Dense Layer

The dense layers are a part of the fully connected layers where the classification of the model occurs.

3.2.1.6 Output

This layer gives the output of whether the traffic is malicious or legitimate.

3.3 Evaluation

3.3.1 Normalisation

The data is processed initially through normalisation to provide a standard format for classification. The formula used for normalisation is,

$$x_{normalized} = \frac{x - m}{x_{max} - x_{min}}$$

Figure 8: Formula for Normalization

[3] https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243

3.3.2 Accuracy

The model's accuracy is calculated to determine whether the predictions are correct. The collected samples must be from the same class.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Figure 9: Formula for Accuracy

The metrics used are:

1. **Confusion matrix**: Provides the accuracy, precision as well as the recall value with which the model can be interpreted. There are 4 values considered in the confusion matrix. True positive (TP), false positive (FP), false negative (FN), True negative (TN). [4]
2. **Recall**: Provides a score based on the correctly predicted classes. Therefore, this score should be as high as possible.
3. **Precision**: Indicates the accuracy of the predicted value, i.e. how the true values are actually true. Therefore, should be high. [5]

4 Design Specification

4.1 Multi D CNN Model

The unique model proposed for the detection of botnet is the Multi-Dimensional CNN. This model is made up of many single CNNs that are integrated to boost efficiency. Convolutional layers, pooling layers, and fully connected layers are the model's three primary layers. The dense layers are part of the fully connected layers that receive and concatenate the output of the two models to produce the Multi-Dimensional CNN model.

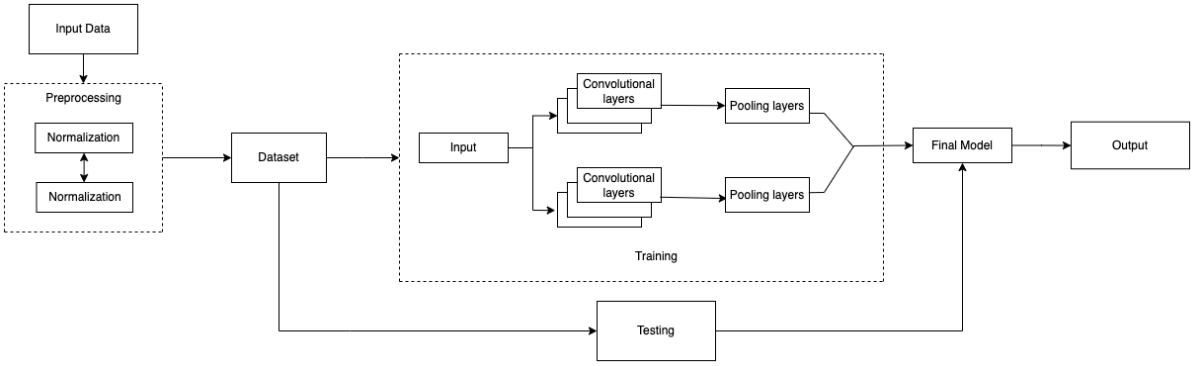

Figure 10: Multi-Dimensional CNN Architecture

4.1.1 Input Layer

The input layer accepts images, which include features that can be retrieved and used to train the model. The layer also aids in correlating the features that must be taken from the image.

[4] https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62
[5] https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

4.1.2 Convolutional Layers

Each image's feature maps are extracted using convolutional layers. Convolutional layers include learnable parameters known as kernels, whereas the other matrix has a constrained component with a specified field as input. The kernel is smaller in size but has a larger depth. The dot product of the two matrices is performed by this layer. The kernel size is 32 with 3x3 filters, whereas the following layer employs a 64-filter size with 3x3 filters, and so on. Two different models are constructed using these filter sizes.

4.1.3 Pooling Layer

This layer reduces the output of the convolutional layer by taking statistics produced from close outputs into account. The image size is decreased while essential characteristics are retained. Pooling is done individually on each slice of the matrix representation. The Max pooling function is employed in this model with a pool size of 2x2 to allow the model to recognise distinguishing aspects of each image such as edges and corners, resulting in better prediction.[6]

4.1.4 Activation Layer

The ReLu correction layer is then implemented to ensure that the negative values are removed and replaced with zeros to comprehend the deep learning model's positive aspects. The Softmax function is frequently used to normalise the model's output. Similarly, softmax is used in this model for the same functionality.

4.1.5 Hyperparameters

Batch normalisation is a data normalisation function that is executed between layers, primarily in the pooling layer. This provides for better training and easier learning. [7]
The dropout layer is used to nullify the effect of a few neurons on the next layer. It is useful in CNN because it prevents data overfitting.

4.1.6 Fully Connected Layer

The fully connected layer receives the output from both the models. A concatenate function is used to combine the output of the model and is further sent into dense layers. [8]

4.1.6.1 Combination of the two CNNs

The outputs of the two models built using the pooling and activation layers are combined in this layer. The concatenate function is used to combine the layers; this layer is known as the dense layer. The layer obtains the combined output and proceeds through the sizes 512, 256, and 128. A dropout layer is introduced after each computation with its corresponding size to prevent the model from overfitting.
The dense layer is activated using a ReLU activation function, and the final output is obtained using the Softmax function. The keras.model function creates the model, but the RMS prop

---

[6] https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

[7] https://www.baeldung.com/cs/batch-normalization-cnn

[8] https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243

function increases the model's learning. Finally, the model is constructed using the "categorical_crossentropy" loss. The final model.fit method is used to evaluate the model's performance using the epoch value and batch size. The model is stored and visualised using the matplotlib tool. This functionality is carried out throughout the model's training phase. Furthermore, testing is carried out on the H5 file produced by the training model. [9]

5 Implementation

5.1 Architecture used for Botnet Detection

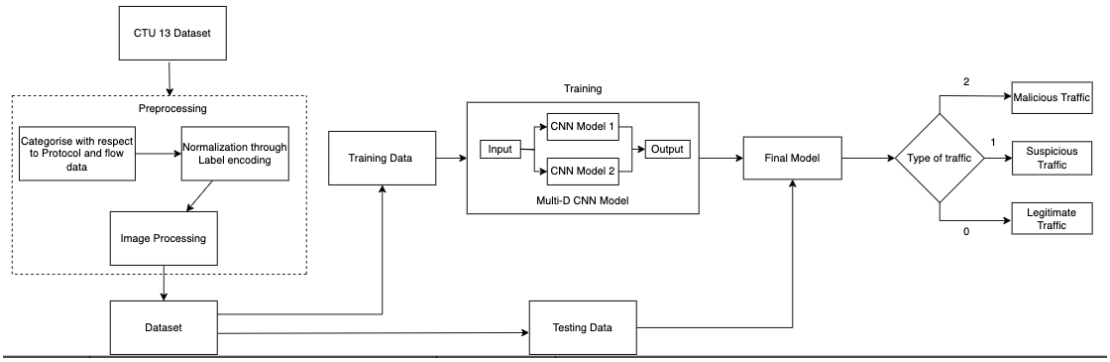

Figure 11: Architecture of Botnet Prediction

5.1.1 Dataset Preprocessing

The CTU 13 dataset is considered for the training and testing with its labels. The labels are background traffic, normal traffic, and botnet traffic. Feature selection was performed on the dataset which included removing the null values, extracting certain features by applying functions to enhance the detection technique. One such function, was suggested by the paper (Le et al., n.d.)that the background traffic does not have only normal traffic and may contain botnet or CC traffic. The flow information in bytes is assessed based on the protocol to identify and categorise the data. Different protocols, such as TCP, UDP, and HTTP, are used to categorise traffic. This aids in identifying malicious traffic based on the bytes in the flow. Dataset was augmented to ensure that multiple dimensions of the images are considered to better train the model. [10]

Further category prediction is conducted on the dataset, which categorises the traffic into three distinct groups that the user can use for further analysis. There are three types: legitimate, malicious, and suspicious.

Label encoding is conducted after normalisation of the input for the model to understand. Label encoding is a unique approach that converts categorical columns into numerical columns, allowing the machine to be more optimal as it only accepts numerical data.

The code incorporates several libraries and frameworks, including keras, a tensorflow-based library that enables for the computation of deep learning models. Similarly, to examine and manipulate the dataset, a package called pandas is employed. The label encoded data is then

[9] https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[10] https://www.linkedin.com/advice/0/how-do-you-implement-data-augmentation-techniques#:~:text=Data%20augmentation%20can%20address%20a,classes%20by%20applying%20different%20transformations.

turned into images using the function "data_to_img" as the model understands and classifies only through images. [11]

5.1.2 Training and Testing the Model

The images generated are then loaded and used for the training of the model. There are two models, model 1 and model 2 that are created. Each with filters of 32, 64 and 128 and kernel sizes of 3x3. Activation function is added to ensure that the negative values are further nullified and replaced with zero to consider the positive aspects of the images. A max pooling function is computed with the pool size of 2x2 which provides the weights and considers the maximum weights obtained.
Batch normalisation is added to each model to ensure that each set of data has been normalised to bring into a standard format. Dropout is added to ignore a few nodes in the process of the computation, this is done so to avoid overfitting. A dropout of 0.5 and 0.1 is used for filters 32 and 64 respectively. This process allows the data to be processed and the features to be extracted for the model to identify different characteristics of the traffic. Finally, the two models are concatenated with the concatenate function, to obtain a final model with epoch of 10 and batch size of 200.
This model is then used for the testing purposes where the accuracy is obtained of the model. The accuracy indicated how well the model can predict a malicious traffic, suspicious traffic and a legitimate one. This allows companies applying this strategy to further categorise suspicious traffic as needed. The obtained accuracy was 73.5%.

6 Evaluation

This section highlights the dataset sizes that were used, as well as the number of iterations required to train the model.

6.1 Experiment / Case Study 1

The dataset size used for case study 1 is 270000 and there was different epoch performed using the same image dataset.

6.1.1 Train

An example of the training model with 13 epochs is mentioned below. A similar training was performed with 5, 7 and 10 epochs which created different models.

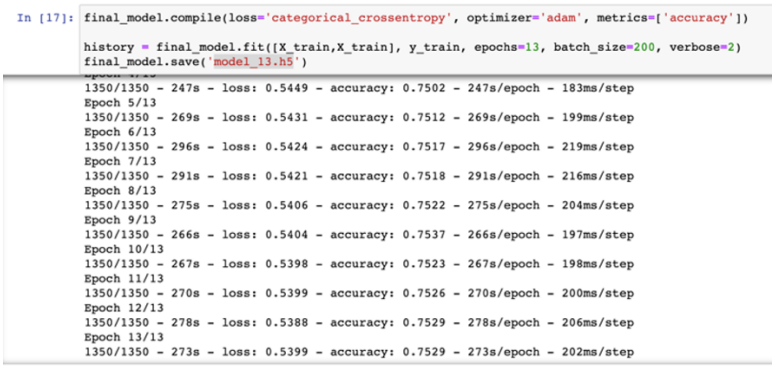

Figure 12: Train with 13 epochs

[11] https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

```
In [11]: import matplotlib.pyplot as plt

         %matplotlib inline
         plt.figure(1, figsize = (15,8))

         plt.subplot(221)
         plt.plot(history.history['accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')

         plt.subplot(222)
         plt.plot(history.history['loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')

         plt.show()
```

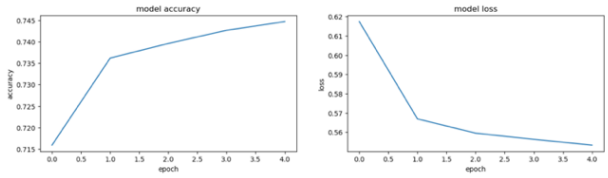

Figure 13: Graph for the train model

6.1.2 Test

The models developed with various epochs were utilised for testing to ensure accuracy. The accuracy for a model with 5 epochs is 69.5, for a model with 7 epochs is 63.4, for a model with 10 epochs is 46.62, and for a model with 13 epochs is 69.59. The model with ten epochs indicated overfitting, which reduced accuracy.

```
In [15]: model= load_model('model_5_2.7L.h5')
         #model.summary()

In [16]: pred = model.predict([X_test, X_test])

         5625/5625 [==============================] - 42s 7ms/step

In [17]: from sklearn.metrics import accuracy_score

         print(accuracy_score(y_test,pred.round()))

         0.69545

In [13]: from sklearn.metrics import confusion_matrix, classification_report
         print(classification_report(y_test,pred.round()))

                       precision    recall  f1-score   support

                    0       0.53      0.40      0.46     59720
                    1       0.82      0.59      0.69     60176
                    2       0.77      0.91      0.83     60104

            micro avg       0.72      0.63      0.67    180000
            macro avg       0.71      0.63      0.66    180000
         weighted avg       0.71      0.63      0.66    180000
          samples avg       0.63      0.63      0.63    180000
```

Figure 14: Test with 5 epochs

```
In [10]: model= load_model('model_7.h5')
         #model.summary()

In [11]: pred = model.predict([X_test, X_test])

         5625/5625 [==============================] - 46s 8ms/step

In [12]: from sklearn.metrics import accuracy_score

         print(accuracy_score(y_test,pred.round()))

         0.6340777777777777

In [13]: from sklearn.metrics import confusion_matrix, classification_report
         print(classification_report(y_test,pred.round()))

                       precision    recall  f1-score   support

                    0       0.53      0.40      0.46     59720
                    1       0.82      0.59      0.69     60176
                    2       0.77      0.91      0.83     60104

            micro avg       0.72      0.63      0.67    180000
            macro avg       0.71      0.63      0.66    180000
         weighted avg       0.71      0.63      0.66    180000
          samples avg       0.63      0.63      0.63    180000
```

Figure 15: Test with 7 epochs

```
In [27]: X_test = data/255

In [28]: y_test = to_categorical(labels)
         num_classes = y_test.shape[1]

In [29]: model= load_model('model_10_2.7L.h5')
         #model.summary()

In [30]: pred = model.predict([X_test, X_test])

         5625/5625 [==============================] - 41s 7ms/step

In [31]: from sklearn.metrics import accuracy_score

         print(accuracy_score(y_test,pred.round()))

         0.4662277777777778

In [32]: from sklearn.metrics import confusion_matrix, classification_report
         print(classification_report(y_test,pred.round()))

                       precision    recall  f1-score   support

                    0       0.32      0.54      0.40     59720
                    1       0.77      0.73      0.75     60176
                    2       0.35      0.13      0.19     60104

            micro avg       0.47      0.47      0.47    180000
            macro avg       0.48      0.47      0.45    180000
         weighted avg       0.48      0.47      0.45    180000
          samples avg       0.47      0.47      0.47    180000
```

Figure 16: Test with 10 epochs

```
In [10]: model= load_model('model_13_2.7L.h5')
         #model.summary()

In [11]: pred = model.predict([X_test, X_test])

         5625/5625 [==============================] - 44s 8ms/step

In [12]: from sklearn.metrics import accuracy_score

         print(accuracy_score(y_test,pred.round()))

         0.6959388888888889

In [13]: from sklearn.metrics import confusion_matrix, classification_report
         print(classification_report(y_test,pred.round()))

                       precision    recall  f1-score   support

                    0       0.63      0.28      0.39     59720
                    1       0.75      0.81      0.78     60176
                    2       0.67      0.99      0.80     60104

            micro avg       0.70      0.70      0.70    180000
            macro avg       0.69      0.70      0.66    180000
         weighted avg       0.69      0.70      0.66    180000
          samples avg       0.70      0.70      0.70    180000
```

Figure 17: Test with 13 epochs

6.2 Experiment / Case Study 2

Similarly, the dataset size used for case study 2 is 360000 and there was different epoch performed using the same image dataset.

6.2.1 Train

An example of the training model with 10 epochs and a batch size of 150. A similar training for performed with 5, 7 and 10 epochs which created different models.

```
In [10]: final_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

         history = final_model.fit([X_train,X_train], y_train, epochs=5, batch_size=200, verbose=2)
         final_model.save('model_3.6L.h5')

         Epoch 1/5
         1800/1800 - 313s - loss: 0.6174 - accuracy: 0.7159 - 313s/epoch - 174ms/step
         Epoch 2/5
         1800/1800 - 324s - loss: 0.5669 - accuracy: 0.7362 - 324s/epoch - 180ms/step
         Epoch 3/5
         1800/1800 - 327s - loss: 0.5595 - accuracy: 0.7396 - 327s/epoch - 182ms/step
         Epoch 4/5
         1800/1800 - 323s - loss: 0.5563 - accuracy: 0.7427 - 323s/epoch - 180ms/step
         Epoch 5/5
         1800/1800 - 316s - loss: 0.5532 - accuracy: 0.7447 - 316s/epoch - 176ms/step
```

Figure 18: Train with 5 epochs

```
In [10]: final_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

         history = final_model.fit([X_train,X_train], y_train, epochs=10, batch_size=150, verbose=2)
         final_model.save('model_3.6L_10_150.h5')
         Epoch 4/10
         2400/2400 - 307s - loss: 0.5564 - accuracy: 0.7427 - 307s/epoch - 128ms/step
         Epoch 5/10
         2400/2400 - 309s - loss: 0.5539 - accuracy: 0.7447 - 309s/epoch - 129ms/step
         Epoch 6/10
         2400/2400 - 306s - loss: 0.5510 - accuracy: 0.7463 - 306s/epoch - 127ms/step
         Epoch 7/10
         2400/2400 - 305s - loss: 0.5490 - accuracy: 0.7478 - 305s/epoch - 127ms/step
         Epoch 8/10
         2400/2400 - 305s - loss: 0.5483 - accuracy: 0.7482 - 305s/epoch - 127ms/step
         Epoch 9/10
         2400/2400 - 306s - loss: 0.5484 - accuracy: 0.7468 - 306s/epoch - 127ms/step
         Epoch 10/10
         2400/2400 - 313s - loss: 0.5470 - accuracy: 0.7491 - 313s/epoch - 130ms/step
```

Figure 19: Train with 10 epochs and a batch size of 150

6.2.2 Test

The models developed with different epochs were utilised for testing to ensure the model's accuracy. The accuracy of the images below is 73.5 for a model with 5 epochs, 68.7 for a model with 7 epochs, and 69.6 for a model with 10 epochs. A confusion matrix is constructed to check for true positives and false positives since the best obtained accuracy is with a dataset size of 360000 and an accuracy of 73.5.

```
In [15]: model= load_model('model_3.6L.h5')
         #model.summary()

In [16]: pred = model.predict([X_test, X_test])
         7500/7500 [==============================] - 66s 9ms/step

In [17]: from sklearn.metrics import accuracy_score

         print(accuracy_score(y_test,pred.round()))
         0.7351708333333333

In [18]: from sklearn.metrics import confusion_matrix, classification_report
         print(classification_report(y_test,pred.round()))
                       precision    recall  f1-score   support

                    0       0.76      0.35      0.48     80095
                    1       0.70      0.94      0.80     79650
                    2       0.76      0.91      0.83     80255

            micro avg       0.74      0.74      0.74    240000
            macro avg       0.74      0.74      0.71    240000
         weighted avg       0.74      0.74      0.71    240000
          samples avg       0.74      0.74      0.74    240000
```

Figure 20: Test with 5 epochs

```
In [23]: import matplotlib.pyplot as plt
         import matplotlib.cm as cm
         import seaborn as sn
         labels = np.unique(np.concatenate((pred_labels, true_labels)))

         # Plot the confusion matrix as a seaborn heatmap
         plt.figure(figsize=(8, 6))
         sn.heatmap(conf_matrix, annot=True, cmap='Reds', fmt='d', xticklabels=labels, yticklabels=labels)
         plt.xlabel('Predicted Labels')
         plt.ylabel('True Labels')
         plt.title('Confusion Matrix')
         plt.show()
```

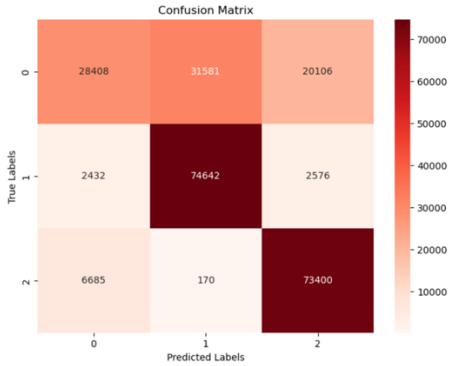

Figure 21: Confusion matrix

A confusion matrix represents the machine learning algorithm's performance with two or more classes. There are two types of values: true values and predicted values. According to the confusion matrix above, 28408 legitimate traffic, 74642 suspicious traffic, and 73400 malicious traffic were properly predicted. 31581 legitimate traffic was predicted to be malicious, whereas 20106 legitimate traffic was predicted to be suspicious. Likewise, 2432 malicious traffic was predicted to be legitimate, whereas 2576 malicious traffic was predicted to be suspicious. Similarly, 6685 suspicious traffic was predicted to be legitimate, while 170 suspicious traffic was predicted to be malicious.

```
In [9]: model= load_model('model_3.6L_7.h5')
        #model.summary()

In [10]: pred = model.predict([X_test, X_test])
         7500/7500 [==============================] - 58s 8ms/step

In [11]: from sklearn.metrics import accuracy_score

         print(accuracy_score(y_test,pred.round()))
         0.6871708333333333

In [12]: from sklearn.metrics import confusion_matrix, classification_report
         print(classification_report(y_test,pred.round()))
                       precision    recall  f1-score   support

                    0       0.73      0.16      0.26     80095
                    1       0.70      0.91      0.79     79650
                    2       0.67      0.99      0.80     80255

            micro avg       0.69      0.69      0.69    240000
            macro avg       0.70      0.69      0.62    240000
         weighted avg       0.70      0.69      0.62    240000
          samples avg       0.69      0.69      0.69    240000
```

Figure 22: Test with 7 epochs

```
In [17]: X_test = data/255

In [18]: y_test = to_categorical(labels)
         num_classes = y_test.shape[1]

In [19]: model= load_model('model_3.6L_10_150.h5')
         #model.summary()

In [20]: pred = model.predict([X_test, X_test])
         7500/7500 [==============================] - 57s 8ms/step

In [21]: from sklearn.metrics import accuracy_score

         print(accuracy_score(y_test,pred.round()))
         0.6962916666666666

In [22]: from sklearn.metrics import confusion_matrix, classification_report
         print(classification_report(y_test,pred.round()))
                       precision    recall  f1-score   support

                    0       0.64      0.29      0.40     80095
                    1       0.77      0.80      0.78     79650
                    2       0.67      0.99      0.80     80255

            micro avg       0.70      0.70      0.70    240000
            macro avg       0.69      0.70      0.66    240000
         weighted avg       0.69      0.70      0.66    240000
          samples avg       0.70      0.70      0.70    240000
```

Figure 23: Test with 10 epochs

6.2.3 Prediction

A separate notebook with a dataset to predict the model was created. The image below shows the dataframe with the predicted value.

```
In [11]: path = "IMGl/Pred/"
         predimgs = loadImages(path)

         data = np.array(predimgs)
         data = data.reshape([-1,50,50,1])


         X_pred = data/255
         print(X_pred.shape)
         (200000, 50, 50, 1)

In [ ]:

In [12]: model= load_model('model_3.6L.h5')
         pred = np.argmax(model.predict([X_pred, X_pred]),axis=1)
         df['Detection']=pred
         df['Detection'] = df['Detection'].replace(k)
         6250/6250 [==============================] - 50s 8ms/step

In [13]: df
Out[13]:
```

	dur	proto	state	tot_pkts	tot_bytes	src_bytes	label	dst_bytes	Detection
7960182	0.169702	1	21	2	514	454	7	60	Malicious traffic
6084324	3391.819824	1	21	4	1856	471	7	1385	Malicious traffic
612710	2428.668945	1	21	14	2278	1382	7	896	Suspicious traffic
8300545	0.003416	0	136	10	1430	955	5	475	Legitimate traffic
4852672	0.337290	1	21	2	135	75	3	60	Suspicious traffic
...	...	...	...	...	...	...	...	...	...
5516910	561.315430	0	136	117	7496	3173	5	4323	Legitimate traffic
14356	0.157974	1	21	2	240	85	1362	155	Malicious traffic
2588838	0.000229	1	21	2	222	83	1362	139	Malicious traffic
2435732	0.000619	1	21	2	255	193	7	62	Legitimate traffic
8974294	0.008163	0	136	10	1348	876	5	472	Suspicious traffic

200000 rows × 9 columns

Figure 24: Prediction

6.3 Discussion

Following the completion of two independent case studies with data sizes of 270000 and 360000. Each has a different number of epochs to train the model, such as 5, 7, 10, and 13 for 270000 and 5, 7, 10 for 360000. The model's accuracy was estimated to be 73.5%. The author (Ravindra Vishwakarma, n.d.) adopts the same dataset and employs the SMOTE and Adaptive Synthetic methods to handle the imbalance problem, achieving an accuracy of 83%. The methodology employed in the research performs under sampling to ensure that the number of samples from a major class is eliminated and made equal to the number of samples from the minor class, but with an accuracy of 73.5%. Therefore, to improve the model's performance, a different sort of approach can be employed to ensure that the dataset is categorized mainly focusing on the legitimate traffic.

7 Conclusion and Future Work

The essential question of whether Multi D CNN can identify between legitimate, malicious, and suspicious communications has been answered: yes. The proposed approach classifies the dataset using source and destination bytes for each protocol. This method produces significant outcomes through categorical prediction as it distributes the detection types evenly for training

the model. The methodology proposed in this research has a 73.5% accuracy rate. If the model's accuracy needs to be improved, one area to focus on is how the dataset is classified after the logic has been applied.

References

Alauthaman, M., Aslam, N., Zhang, L., Alasem, R., & Hossain, M. A. (2018). A P2P Botnet detection scheme based on decision tree and adaptive multilayer neural networks. *Neural Computing and Applications*, *29*(11), 991–1004. https://doi.org/10.1007/S00521-016-2564-5

Alharbi, A., & Alsubhi, K. (2021). Botnet Detection Approach Using Graph-Based Machine Learning. *IEEE Access*, *9*, 99166–99180. https://doi.org/10.1109/ACCESS.2021.3094183

Almutairi, S., Mahfoudh, S., Almutairi, S., & Alowibdi, J. S. (2020). Hybrid Botnet Detection Based on Host and Network Analysis. *Journal of Computer Networks and Communications*, *2020*. https://doi.org/10.1155/2020/9024726

Alzahrani, A. J., & Ghorbani, A. A. (2015). Real-time signature-based detection approach for SMS botnet. *2015 13th Annual Conference on Privacy, Security and Trust, PST 2015*, 157–164. https://doi.org/10.1109/PST.2015.7232968

Arshad, S., Abbaspour, M., Kharrazi, M., & Sanatkar, H. (2011). An anomaly-based botnet detection approach for identifying stealthy botnets. *ICCAIE 2011 - 2011 IEEE Conference on Computer Applications and Industrial Electronics*, 564–569. https://doi.org/10.1109/ICCAIE.2011.6162198

Bhatia, M., Sharma, V., Singh, P., & Masud, M. (2020). Multi-Level P2P Traffic Classification Using Heuristic and Statistical-Based Techniques: A Hybrid Approach. *Symmetry 2020, Vol. 12, Page 2117*, *12*(12), 2117. https://doi.org/10.3390/SYM12122117

Cook, S. (2023, June 13). *20+ DDoS attack statistics and facts for 2018-2023*. https://www.comparitech.com/blog/information-security/ddos-statistics-facts/

Duan, L., Zhou, J., Wu, Y., & Xu, W. (2022). A novel and highly efficient botnet detection algorithm based on network traffic analysis of smart systems. *International Journal of Distributed Sensor Networks*, *18*(3). https://doi.org/10.1177/15501477211049910/ASSET/IMAGES/LARGE/10.1177_15501477211049910-FIG18.JPEG

Garg, S., Singh, A. K., Sarje, A. K., & Peddoju, S. K. (2013). Behaviour analysis of machine learning algorithms for detecting P2P botnets. *2013 15th International Conference on Advanced Computing Technologies, ICACT 2013*. https://doi.org/10.1109/ICACT.2013.6710523

Guntuku, S. C., Narang, P., & Hota, C. (n.d.). *Real-time Peer-to-Peer Botnet Detection Framework based on Bayesian Regularized Neural Network*.

Hossein Rouhani Zeidanloo, A. B. A. M., & Saman Shojae Chaeikar. (2010, April). *A Proposed Framework for P2P Botnet Detection*. http://www.ijetch.org/papers/116--T260.pdf

Ibrahim, W. N. H., Anuar, S., Selamat, A., Krejcar, O., Gonzalez Crespo, R., Herrera-Viedma, E., & Fujita, H. (2021). Multilayer Framework for Botnet Detection Using Machine Learning Algorithms. *IEEE Access*, *9*, 48753–48768. https://doi.org/10.1109/ACCESS.2021.3060778

Khoh Choon Hwa, st, & Mahmood Al-Shareeda, rd A. (n.d.). *Review of Peer-to-Peer Botnets and Detection Mechanisms*.

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Le, D. C., Nur Zincir-Heywood, A., & Heywood, M. I. (n.d.). *Data Analytics on Network Traffic Flows for Botnet Behaviour Detection*. Retrieved August 4, 2023, from https://ieeexplore.ieee.org/document/7850078

Mahmoud, M., Nir, M., & Matraw, A. (2015). *A Survey on Botnet Architectures, Detection and Defences*. http://ijns.jalaxy.com.tw

Mccart, C. (2022, March 15). *15+ Shocking Botnet Statistics and Facts for 2023*. https://www.comparitech.com/blog/information-security/botnet-statistics/

Nagaraja, S. (2014). Botyacc: Unified P2P botnet detection using behavioural analysis and graph analysis. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8713 LNCS*(PART 2), 439–456. https://doi.org/10.1007/978-3-319-11212-1_25/COVER

Nomm, S., & Bahsi, H. (2019). Unsupervised Anomaly Based Botnet Detection in IoT Networks. *Proceedings - 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018*, 1048–1053. https://doi.org/10.1109/ICMLA.2018.00171

Ravindra Vishwakarma, A. (n.d.). *Network Traffic Based Botnet Detection Using Machine Learning*. https://doi.org/10.31979/etd.4nd6-m6hp

Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., Felix, J., & Hakimian, P. (2011a). Detecting P2P botnets through network behavior analysis and machine learning. *2011 9th Annual International Conference on Privacy, Security and Trust, PST 2011*, 174–180. https://doi.org/10.1109/PST.2011.5971980

Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., Felix, J., & Hakimian, P. (2011b). Detecting P2P botnets through network behavior analysis and machine learning. *2011 9th Annual International Conference on Privacy, Security and Trust, PST 2011*, 174–180. https://doi.org/10.1109/PST.2011.5971980

Shamshirband, S., & Chronopoulos, A. T. (n.d.). *A New Malware Detection System Using a High Performance-ELM method*.

Shetu, S. F., Saifuzzaman, M., Moon, N. N., & Nur, F. N. (2019). A survey of botnet in cyber security. *2019 2nd International Conference on Intelligent Communication and Computational Techniques, ICCT 2019*, 174–177. https://doi.org/10.1109/ICCT46177.2019.8969048

Suthar, F., Patel, N., & Khanna, S. V. O. (2022). A Signature-Based Botnet (Emotet) Detection Mechanism. *International Journal of Engineering Trends and Technology*, *70*, 185–193. https://doi.org/10.14445/22315381/IJETT-V70I5P220

Szynkiewicz, P. (2022). Signature-Based Detection of Botnet DDoS Attacks. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13300 LNCS*, 120–135. https://doi.org/10.1007/978-3-031-04036-8_6/FIGURES/8

View of A Study Of Machine Learning Classifiers for Anomaly-Based Mobile Botnet Detection. (n.d.). Retrieved August 4, 2023, from http://adum.um.edu.my/index.php/MJCS/article/view/6785/4458

Wang, K., Huang, C. Y., Lin, S. J., & Lin, Y. D. (2011). A fuzzy pattern-based filtering algorithm for botnet detection. *Computer Networks*, *55*(15), 3275–3286. https://doi.org/10.1016/J.COMNET.2011.05.026

Wang, K., Huang, C. Y., Tsai, L. Y., & Lin, Y. D. (2014). Behavior-based botnet detection in parallel. *Security and Communication Networks*, *7*(11), 1849–1859. https://doi.org/10.1002/SEC.898

Yang, Z., & Wang, B. (2019). A Feature Extraction Method for P2P Botnet Detection Using Graphic Symmetry Concept. *Symmetry 2019, Vol. 11, Page 326*, *11*(3), 326. https://doi.org/10.3390/SYM11030326