Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

B. Tech., Fall Semester 2022-2023
CSE3501 - Information Security Analysis and Audit
Review - 3

Cyber Attack Detection using Deep Learning

Team Members
Manigandan R (20BCE1223)
Onkar Apte (20BCE1341)
Rishi Nair (20BCE1922)

Course Faculty
Dr. Prasad M

Contents

Abstract:

A wide range of cyberattacks have been reported in recent years due to the rapid advancement of internet technologies. In the context of the modern cyber world, detection of these attacks is more important Understanding the above needs, we build an IDS which scans the network or the system to check the policy breaching and informs the concerned authorities or applications. Our contributions to this concept include developing IDS leveraging our domain expertise in AI and Deep Learning by developing a Classification Model using the datasets that are currently accessible. The network's packets will be scanned. Each packet has some data that can be examined. Using this extracted feature, we test our model by generating a fake attack. The attack would be detected by our model and a notification would be sent regarding the attack. delivered.

Keywords: Cyberattack, Deep Learning, Packets Capture, Feature Extraction, Cyber-attack Detection, Classification Model and Notification System.

Introduction:

Cyberattacks are growing exponentially, rendering present detection systems inadequate and necessitating the development of more pertinent prediction models and methodologies. Since present attack prediction algorithms are unable to keep up with the vast number and variety of attacks, this challenge is still unresolved. With so many new cybersecurity threats, there is an immediate need to improvise existing detection techniques to thwart these threats. AI and ML plays a vital role in in current cyber security world, which motivated us to develop a self-built IDS model which could send a notification once. In this project, we will use deep learning techniques for detecting intrusions in networks and compare it to the result of various machine learning techniques. We will be using supervised machine learning techniques and deep learning techniques to train and build multiple classification models that can classify attack type of network traffic versus normal type of network traffic. Then we will compare the accuracies of the multiple classification models. As of now, we will be importing the dataset and preprocessing the dataset.

Literature Survey:

In [1], they work with intrusion detection systems and networks to improve classification effectiveness by developing an ensemble method facilitated by stacked generalization techniques to analyses outliers that makes use of deep models like the Deep Neural Network (DNN) and Long Short-Term Memory (LSTM) and a meta-classifier (i.e., logistic regression) adhering to the principle of swarm intelligence. The method employs a two-step procedure for the detection of network anomalies in order to increase the capabilities of the suggested methodology. A Deep Sparse Autoencoder (DSAE) is used for the feature engineering challenge in the initial step of data pre-processing.

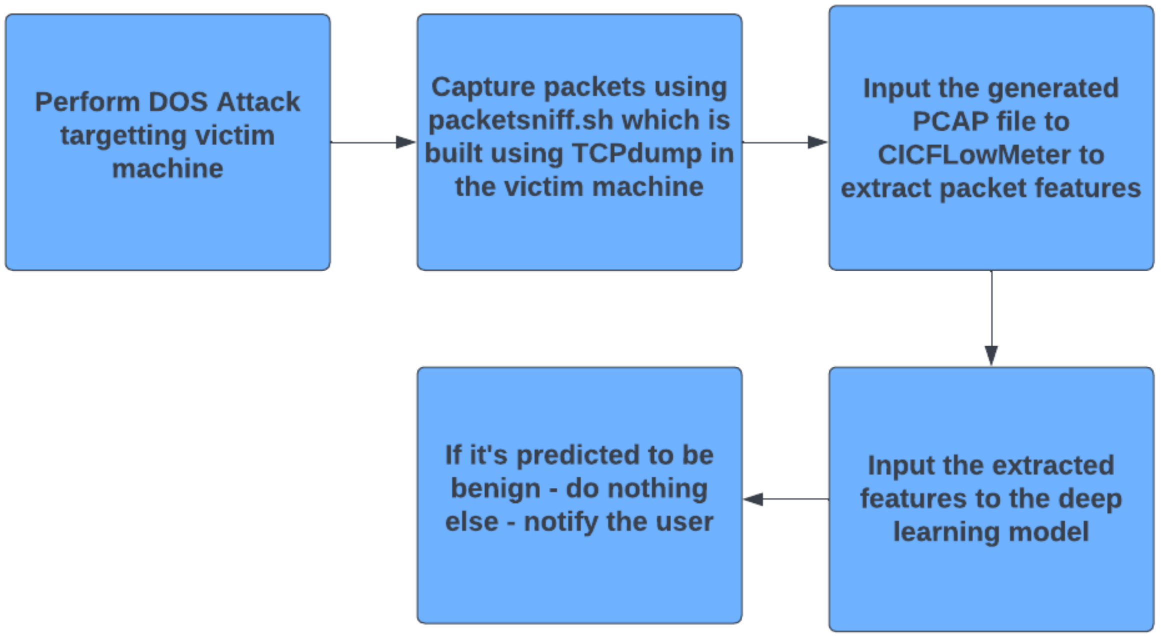
A stacking ensemble learning strategy is used for classification in the second phase. Artificial learning techniques have been applied to the Intrusion Detection System (IDS) in [2]. For Network Intrusion Detection, two different Machine Learning approaches—supervised and unsupervised—are prepared. The strategies that are described include Naive Bayes (supervised learning) and Self Organizing Maps (unsupervised learning). The naive Bayes classifier combines the Bayes model with decision criteria such as the hypothesis that represents the most likely result. For feature extraction, deep learning methods like CNN are used.

Malware infections and pirated software can readily target the industrial IoT-cloud for malicious purposes, compromising security by reusing the code or logic in another form of source code where malware can be injected. As a result, [3] proposes a TensorFlow deep neural network to detect software piracy through source code plagiarism. The malware attacks are typically designed to invade the online privacy of computers, smartphones, and IoT nodes. A variety of scanning methods are suggested to find malware that targets Windows by employing particular signatures. Static and dynamic approaches are the two primary divisions of the malware identification analysis. In a dynamic approach, virus patterns are discovered while code is run in a virtual environment in real time. Function calls, function parameter research, data flow, instruction traces, and visual code analysis can all be used to spot malicious behavior. To analyses the dynamic behavior of harmful software, automated web tools are available like TT analyzer, Anubis, and CW Sandbox, for example. For static methods, the control flow graph, opcode frequency, n-gram, and string signature malware identification techniques are employed.

In [4], a deep neural network-based classifier for attack identification was put forth. This system, which is an end-to-end early intrusion detection system, aims to stop network attacks before they could do any further harm to the system that is being attacked while avoiding unanticipated downtime and interruption. Along with that, a brand-new statistic termed earliness has been developed to assess how quickly their suggested method recognizes attacks.

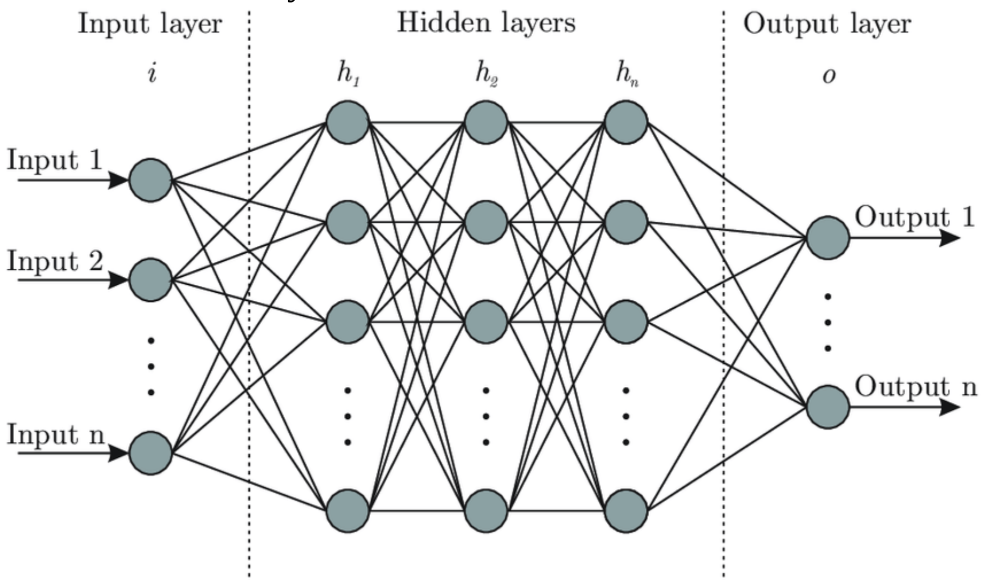
Proposed Work:

Our main idea is to detect cyber-attacks. To do this, we have developed an IDS using deep learning and machine learning techniques. We will be deep learning models like artificial neural network (ANN) and machine learning models using random forest algorithm, decision tree, KNN algorithm to compare the accuracy and recall score to find the best model. First, we will be initiating DoS attack using hping3 to target machine. These packets will be captured by packet sniffer (packetsniff.sh) which is a shell script developed using tcpdump. This shell script generates PCAP file as an output. This PCAP file will be fed into CICFlowMeter which extracts the features from the packets. CICFlowMeter is a network traffic flow generator and analyzer developed by Canadian Institute of Cybersecurity for research purposes. More than 80 statistical network traffic features, such as Duration, Number of packets, Number of bytes, Length of packets, etc., can be extracted separately in the forward and backward directions when using it to generate bidirectional flows, where the first packet determines the forward (source to destination) and backward (destination to source) directions. The output of the application is the CSV format file that has six columns labeled for each flow (FlowID, Source IP, DestinationIP, SourcePort, DestinationPort, and Protocol) with more than 80 network traffic analysis features. This will be fed as an input to the developed ANN model which will predict whether the flow is benign or malicious flow. When malicious network flow is detected, the user will be notified. This is the overall workflow of our project and this has been summarized into a flowchart below.

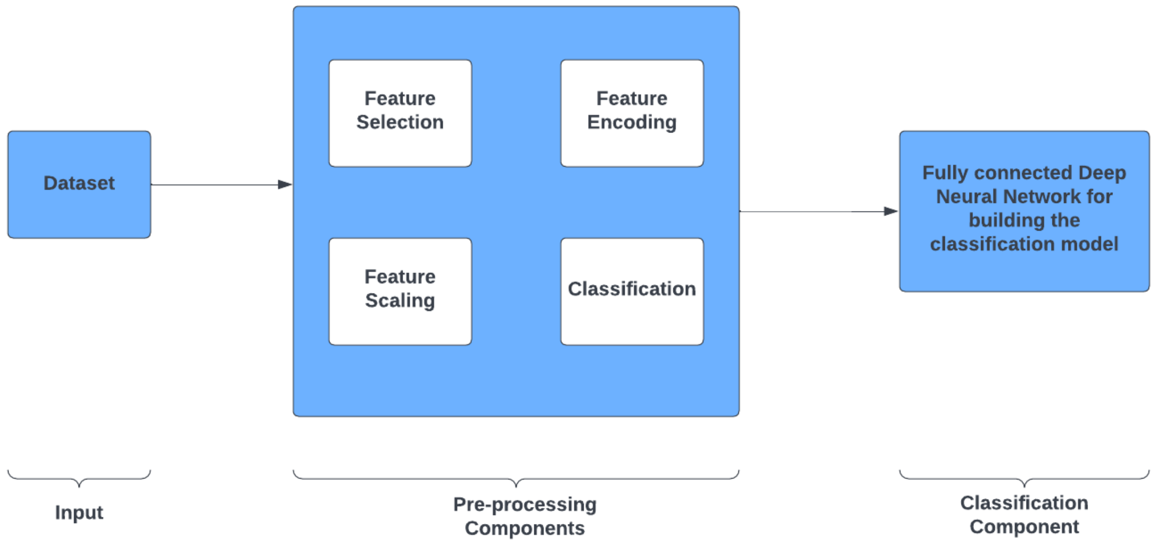

Perform DOS Attack targetting victim machine
Capture packets using packetsniff.sh which is built using TCPdump in the victim machine
Input the generated PCAP file to CICFLowMeter to extract packet features
Input the extracted features to the deep learning model
If it's predicted to be benign - do nothing else - notify the user

Architecture:

Artificial neural networks (ANNs) are composed of a node layer, containing an input layer, one or more hidden layers, and an output layer. Each artificial neuron, or node, is connected to another and has its own weight and threshold. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's next layer.

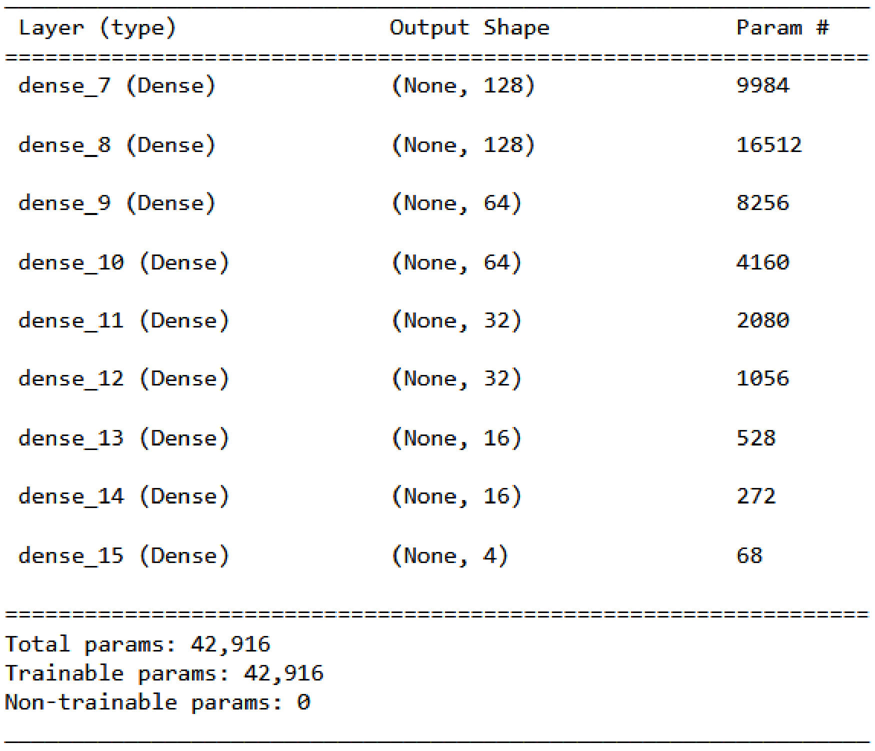


Artificial Neural Network (ANN) model architecture:


Dataset
Feature Selection
Feature Encoding
Feature Scaling
Classification
Fully connected Deep Neural Network for building the classification model
Input
Pre-processing Components
Classification Component

For developing ANN, we will first import the dataset. Second, we will preprocess the data by applying feature selection, feature encoding and feature scaling on the data. Finally, we a sequential ANN and train the model with the preprocessed data.

Artificial Neural Network (ANN) model layer wise Architecture:

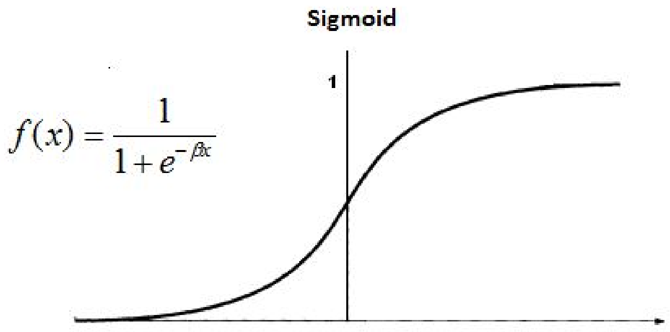


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 128)               9984

 dense_8 (Dense)             (None, 128)               16512

 dense_9 (Dense)             (None, 64)                8256

 dense_10 (Dense)            (None, 64)                4160

 dense_11 (Dense)            (None, 32)                2080

 dense_12 (Dense)            (None, 32)                1056

 dense_13 (Dense)            (None, 16)                528

 dense_14 (Dense)            (None, 16)                272

 dense_15 (Dense)            (None, 4)                 68

=================================================================
Total params: 42,916
Trainable params: 42,916
Non-trainable params: 0
_________________________________________________________________
```

This is the layer wise architecture of the developed ANN model which has 1 input layer, 7 dense layer with sigmoid activation function and 1 output layer with softmax activation function along with 1 optimization function at the last layer to reduce the loss.

Activation functions used in our model:

- Activation function is used to determine the output of each node in the neural network. It is used to introduce non-linearity to the neural network.
- Sigmoid activation function takes a real value as input and outputs another value between 0 and 1. It's good for a classifier.



- We have also used the softmax function which is a generalization of the sigmoid function. It gives the probabilities of each class for being output and thus, the sum of softmax values will always equal to 1.

Optimizer function used in our model:

- Optimizers are algorithms or methods used to change the attributes of a neural network such as weights and learning rate in order to reduce the losses and to provide the most accurate results.
- Adam is the most used optimizer while working with fully connected deep neural networks. Adam is fast and converges rapidly, when compared to other optimizers

Dataset Information:

The CICIDS2017 dataset, which is developed by Canadian Institute of Cybersecurity closely reflects actual real-world data, contains the most recent and benign prevalent attacks (PCAPs) and its especially designed for research purpose for developing an IDS. It also contains the outcomes of a network traffic analysis performed using CICFlowMeter, with flows categorized according to the time stamp, source and destination IP addresses, source and destination ports, protocols, and attack (CSV files).

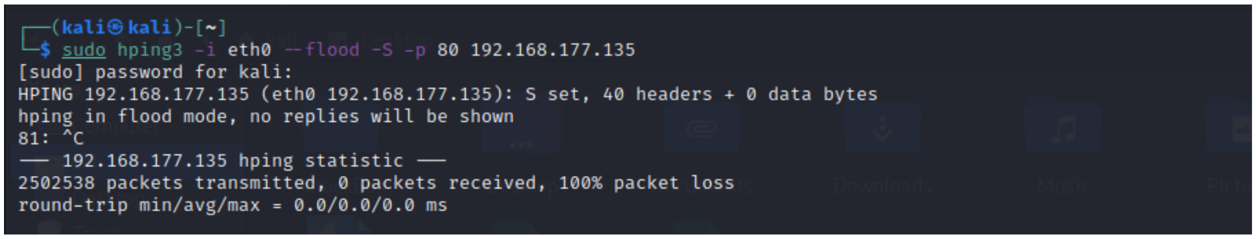
We chose this dataset above others because many of these datasets have low traffic diversity and volume, some don't cover the range of known attacks, and others anonymize packet payload data, which makes it impossible for them to reflect current trends. Some also lack information and feature sets.

Dataset URL: http://205.174.165.80/CICDataset/CIC-IDS-2017/Dataset/

Implementation:

Initiating DoS attack:

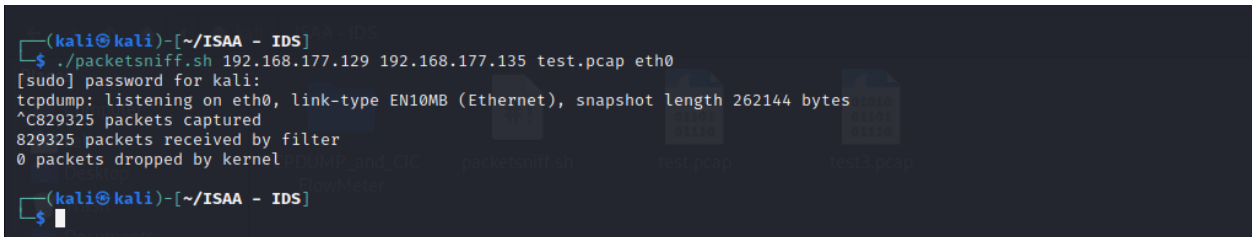
We initiate a DoS attack using hping3 to target machine which has a HTTP server hosted at port 80.



```
┌──(kali㉿kali)-[~]
└─$ sudo hping3 -i eth0 --flood -S -p 80 192.168.177.135
[sudo] password for kali:
HPING 192.168.177.135 (eth0 192.168.177.135): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
81: ^C
--- 192.168.177.135 hping statistic ---
2502538 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Packet Sniffer (packetsniff.sh):

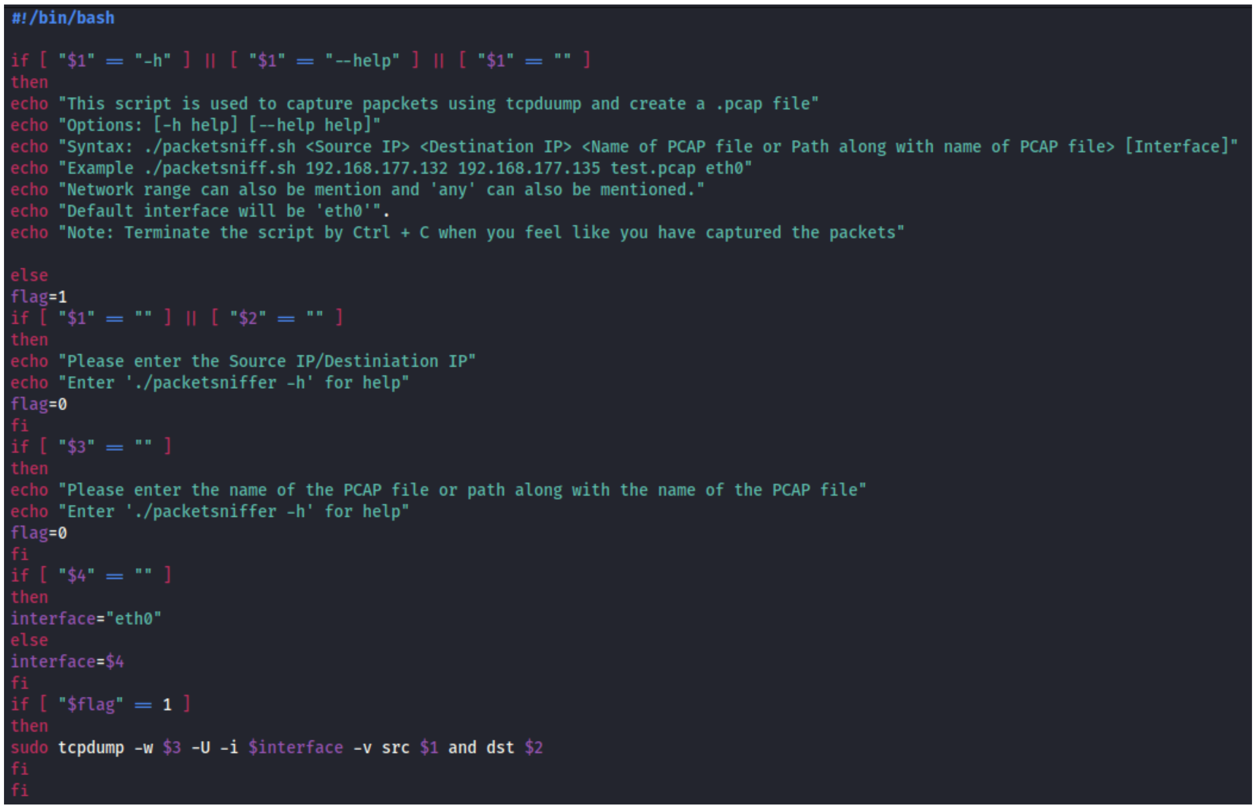
We have written a shell script using tcpdump tool to capture packets. This shell script will capture packets and store it in a PCAP file. We have handled few exceptions in the script.
This shell script implementation is as follows in the target machine:



```
┌──(kali㉿kali)-[~/ISAA - IDS]
└─$ ./packetsniff.sh 192.168.177.129 192.168.177.135 test.pcap eth0
[sudo] password for kali:
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C829325 packets captured
829325 packets received by filter
0 packets dropped by kernel

┌──(kali㉿kali)-[~/ISAA - IDS]
└─$ █
```

The shell script requires source IP, destination IP and output PCAP file name as command line input. Giving interface name as input is optional. Source IP or

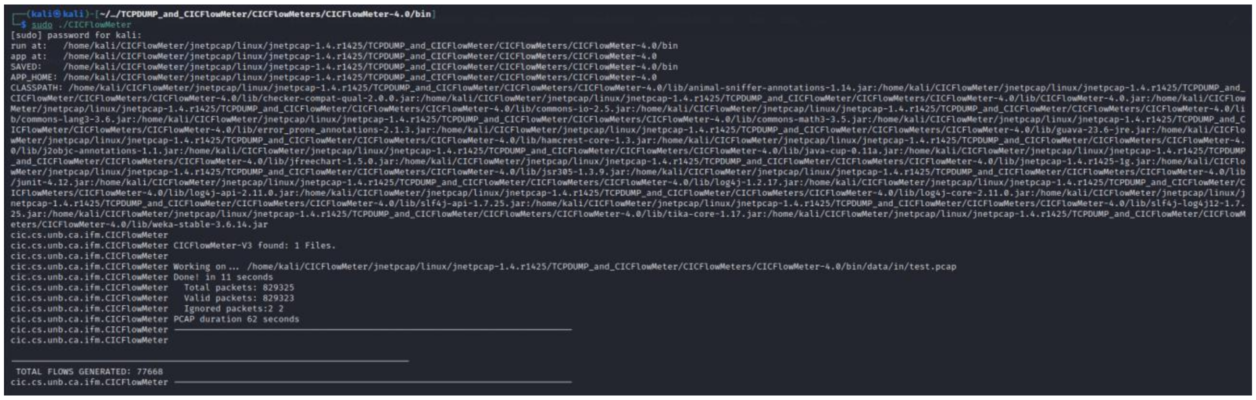
Destination IP can also be a range in network. We have developed this script along with help page. The shell script is as follows:



```bash
#!/bin/bash

if [ "$1" = "-h" ] || [ "$1" = "--help" ] || [ "$1" = "" ]
then
echo "This script is used to capture papckets using tcpduump and create a .pcap file"
echo "Options: [-h help] [--help help]"
echo "Syntax: ./packetsniff.sh <Source IP> <Destination IP> <Name of PCAP file or Path along with name of PCAP file> [Interface]"
echo "Example ./packetsniff.sh 192.168.177.132 192.168.177.135 test.pcap eth0"
echo "Network range can also be mention and 'any' can also be mentioned."
echo "Default interface will be 'eth0'".
echo "Note: Terminate the script by Ctrl + C when you feel like you have captured the packets"

else
flag=1
if [ "$1" = "" ] || [ "$2" = "" ]
then
echo "Please enter the Source IP/Destiniation IP"
echo "Enter './packetsniffer -h' for help"
flag=0
fi
if [ "$3" = "" ]
then
echo "Please enter the name of the PCAP file or path along with the name of the PCAP file"
echo "Enter './packetsniffer -h' for help"
flag=0
fi
if [ "$4" = "" ]
then
interface="eth0"
else
interface=$4
fi
if [ "$flag" = 1 ]
then
sudo tcpdump -w $3 -U -i $interface -v src $1 and dst $2
fi
fi
```

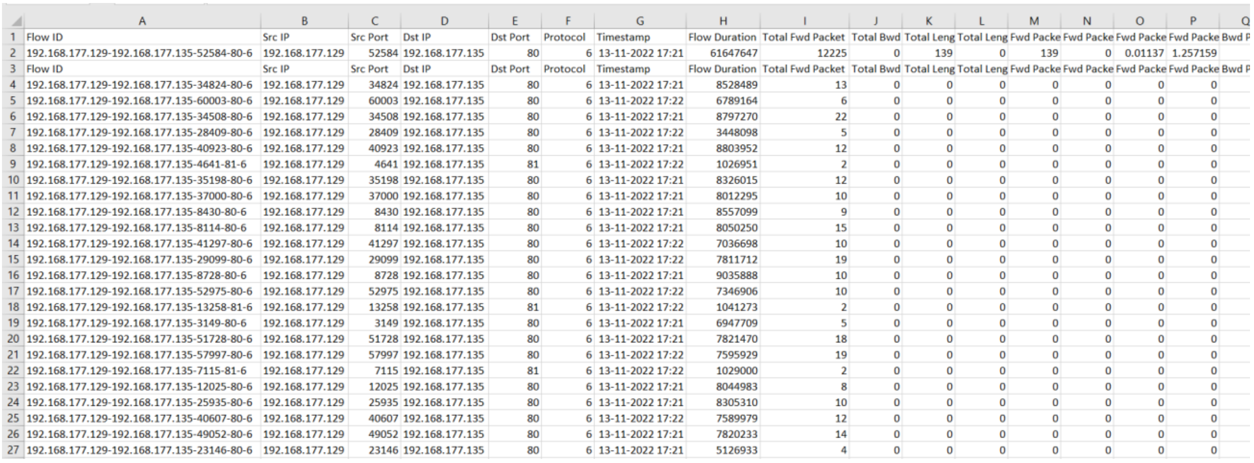
Extracting Features using CICFlowMeter:

CICFlowMeter takes the generated PCAP file as input and goes through the PCAP file and extracts the features from the packets which will be given as input to the ANN model. This tool generated a CSV file as an output with flows categorized according to the time stamp, source and destination IP addresses, source and destination ports, protocols.



```
┌──(kali㉿kali)-[~/…/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/bin]
└─$ sudo ./CICFlowMeter
[sudo] password for kali:
run at:   /home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/bin
app at:   /home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0
SAVED:    /home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/bin
APP_HOME: /home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0
CLASSPATH: /home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/animal-sniffer-annotations-1.14.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_
CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/checker-compat-qual-2.0.0.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/CICFlowMeter-4.0.jar:/home/kali/CICFlow
Meter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/commons-io-2.5.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/li
b/commons-lang3-3.6.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/commons-math3-3.5.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_C
ICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/error_prone_annotations-2.1.3.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/guava-23.6-jre.jar:/home/kali/CICFlo
wMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/hamcrest-core-1.3.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.
0/lib/j2objc-annotations-1.1.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/java-cup-0.11a.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP
_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/jfreechart-1.5.0.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/jnetpcap-1.4.r1425-1g.jar:/home/kali/CICFlo
wMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/jsr305-1.3.9.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib
/junit-4.12.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/log4j-1.2.17.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/C
ICFlowMeters/CICFlowMeter-4.0/lib/log4j-api-2.11.0.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/log4j-core-2.11.0.jar:/home/kali/CICFlowMeter/jnetpcap/linux/j
netpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/slf4j-api-1.7.25.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/slf4j-log4j12-1.7.
25.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/lib/tika-core-1.17.jar:/home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowM
eters/CICFlowMeter-4.0/lib/weka-stable-3.6.14.jar
cic.cs.unb.ca.ifm.CICFlowMeter
cic.cs.unb.ca.ifm.CICFlowMeter CICFlowMeter-V3 found: 1 Files.
cic.cs.unb.ca.ifm.CICFlowMeter
cic.cs.unb.ca.ifm.CICFlowMeter Working on... /home/kali/CICFlowMeter/jnetpcap/linux/jnetpcap-1.4.r1425/TCPDUMP_and_CICFlowMeter/CICFlowMeters/CICFlowMeter-4.0/bin/data/in/test.pcap
cic.cs.unb.ca.ifm.CICFlowMeter Done! in 11 seconds
cic.cs.unb.ca.ifm.CICFlowMeter   Total packets: 829325
cic.cs.unb.ca.ifm.CICFlowMeter   Valid packets: 829323
cic.cs.unb.ca.ifm.CICFlowMeter   Ignored packets:2 2
cic.cs.unb.ca.ifm.CICFlowMeter PCAP duration 62 seconds
cic.cs.unb.ca.ifm.CICFlowMeter ──────────────────────────────────────────────────
cic.cs.unb.ca.ifm.CICFlowMeter

──────────────────────────────────────────────────
 TOTAL FLOWS GENERATED: 77668
cic.cs.unb.ca.ifm.CICFlowMeter ──────────────────────────────────────────────────
```

This is the generated CSV file which we will using for predicting.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd	Total Leng	Total Leng	Fwd Packe	Fwd Packe	Fwd Packe	Fwd Packe	Bwd P
2	192.168.177.129-192.168.177.135-52584-80-6	192.168.177.129	52584	192.168.177.135	80	6	13-11-2022 17:21	61647647	12225	0	139	0	139	0	0.01137	1.257159	
3	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd	Total Leng	Total Leng	Fwd Packe	Fwd Packe	Fwd Packe	Fwd Packe	Bwd P
4	192.168.177.129-192.168.177.135-34824-80-6	192.168.177.129	34824	192.168.177.135	80	6	13-11-2022 17:21	8528489	13	0	0	0	0	0	0	0	
5	192.168.177.129-192.168.177.135-60003-80-6	192.168.177.129	60003	192.168.177.135	80	6	13-11-2022 17:22	6789164	6	0	0	0	0	0	0	0	
6	192.168.177.129-192.168.177.135-34508-80-6	192.168.177.129	34508	192.168.177.135	80	6	13-11-2022 17:21	8797270	22	0	0	0	0	0	0	0	
7	192.168.177.129-192.168.177.135-28409-80-6	192.168.177.129	28409	192.168.177.135	80	6	13-11-2022 17:22	3448098	5	0	0	0	0	0	0	0	
8	192.168.177.129-192.168.177.135-40923-80-6	192.168.177.129	40923	192.168.177.135	80	6	13-11-2022 17:21	8803952	12	0	0	0	0	0	0	0	
9	192.168.177.129-192.168.177.135-4641-81-6	192.168.177.129	4641	192.168.177.135	81	6	13-11-2022 17:22	1026951	2	0	0	0	0	0	0	0	
10	192.168.177.129-192.168.177.135-35198-80-6	192.168.177.129	35198	192.168.177.135	80	6	13-11-2022 17:21	8326015	12	0	0	0	0	0	0	0	
11	192.168.177.129-192.168.177.135-37000-80-6	192.168.177.129	37000	192.168.177.135	80	6	13-11-2022 17:21	8012295	10	0	0	0	0	0	0	0	
12	192.168.177.129-192.168.177.135-8430-80-6	192.168.177.129	8430	192.168.177.135	80	6	13-11-2022 17:21	8557099	9	0	0	0	0	0	0	0	
13	192.168.177.129-192.168.177.135-8114-80-6	192.168.177.129	8114	192.168.177.135	80	6	13-11-2022 17:21	8050250	15	0	0	0	0	0	0	0	
14	192.168.177.129-192.168.177.135-41297-80-6	192.168.177.129	41297	192.168.177.135	80	6	13-11-2022 17:22	7036698	10	0	0	0	0	0	0	0	
15	192.168.177.129-192.168.177.135-29099-80-6	192.168.177.129	29099	192.168.177.135	80	6	13-11-2022 17:22	7811712	19	0	0	0	0	0	0	0	
16	192.168.177.129-192.168.177.135-8728-80-6	192.168.177.129	8728	192.168.177.135	80	6	13-11-2022 17:21	9035888	10	0	0	0	0	0	0	0	
17	192.168.177.129-192.168.177.135-52975-80-6	192.168.177.129	52975	192.168.177.135	80	6	13-11-2022 17:22	7346906	10	0	0	0	0	0	0	0	
18	192.168.177.129-192.168.177.135-13258-81-6	192.168.177.129	13258	192.168.177.135	81	6	13-11-2022 17:22	1041273	2	0	0	0	0	0	0	0	
19	192.168.177.129-192.168.177.135-3149-80-6	192.168.177.129	3149	192.168.177.135	80	6	13-11-2022 17:21	6947709	5	0	0	0	0	0	0	0	
20	192.168.177.129-192.168.177.135-51728-80-6	192.168.177.129	51728	192.168.177.135	80	6	13-11-2022 17:21	7821470	18	0	0	0	0	0	0	0	
21	192.168.177.129-192.168.177.135-57997-80-6	192.168.177.129	57997	192.168.177.135	80	6	13-11-2022 17:22	7595929	19	0	0	0	0	0	0	0	
22	192.168.177.129-192.168.177.135-7115-81-6	192.168.177.129	7115	192.168.177.135	81	6	13-11-2022 17:22	1029000	2	0	0	0	0	0	0	0	
23	192.168.177.129-192.168.177.135-12025-80-6	192.168.177.129	12025	192.168.177.135	80	6	13-11-2022 17:21	8044983	8	0	0	0	0	0	0	0	
24	192.168.177.129-192.168.177.135-25935-80-6	192.168.177.129	25935	192.168.177.135	80	6	13-11-2022 17:21	8305310	10	0	0	0	0	0	0	0	
25	192.168.177.129-192.168.177.135-40607-80-6	192.168.177.129	40607	192.168.177.135	80	6	13-11-2022 17:22	7589979	12	0	0	0	0	0	0	0	
26	192.168.177.129-192.168.177.135-49052-80-6	192.168.177.129	49052	192.168.177.135	80	6	13-11-2022 17:21	7820233	14	0	0	0	0	0	0	0	
27	192.168.177.129-192.168.177.135-23146-80-6	192.168.177.129	23146	192.168.177.135	80	6	13-11-2022 17:21	5126933	4	0	0	0	0	0	0	0	

Developing Artificial Neural Network (ANN) – Deep Learning Model:

Importing necessary libraries:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import math
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from sklearn import metrics
import joblib
import matplotlib.pyplot as plt
import hiplot as hip
from tensorflow.keras.utils import plot_model
import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix
```

Importing the dataset:

The following code reads the CICIDS2017 dataset into a Pandas data frame.

```python
df = pd.read_csv("O:\\VIT\\Fall Semester 2022-2023\\Information Security Analysis and Audit\\Project\\Dataset\\dataset.csv")
```

```python
df=df.drop([' Fwd Header Length.1'], axis=1)
```

```python
df.head()
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	min_seg_size_forward	Active Mean	Active Std	Active Max	Act
0	54865	3	2	0	12	0	6	6	6.0	0.0	...	20	0.0	0.0	0	
1	55054	109	1	1	6	6	6	6	6.0	0.0	...	20	0.0	0.0	0	
2	55055	52	1	1	6	6	6	6	6.0	0.0	...	20	0.0	0.0	0	
3	46236	34	1	1	6	6	6	6	6.0	0.0	...	20	0.0	0.0	0	
4	54863	3	2	0	12	0	6	6	6.0	0.0	...	20	0.0	0.0	0	

5 rows × 78 columns

```python
df.shape
```

```
(1042557, 78)
```

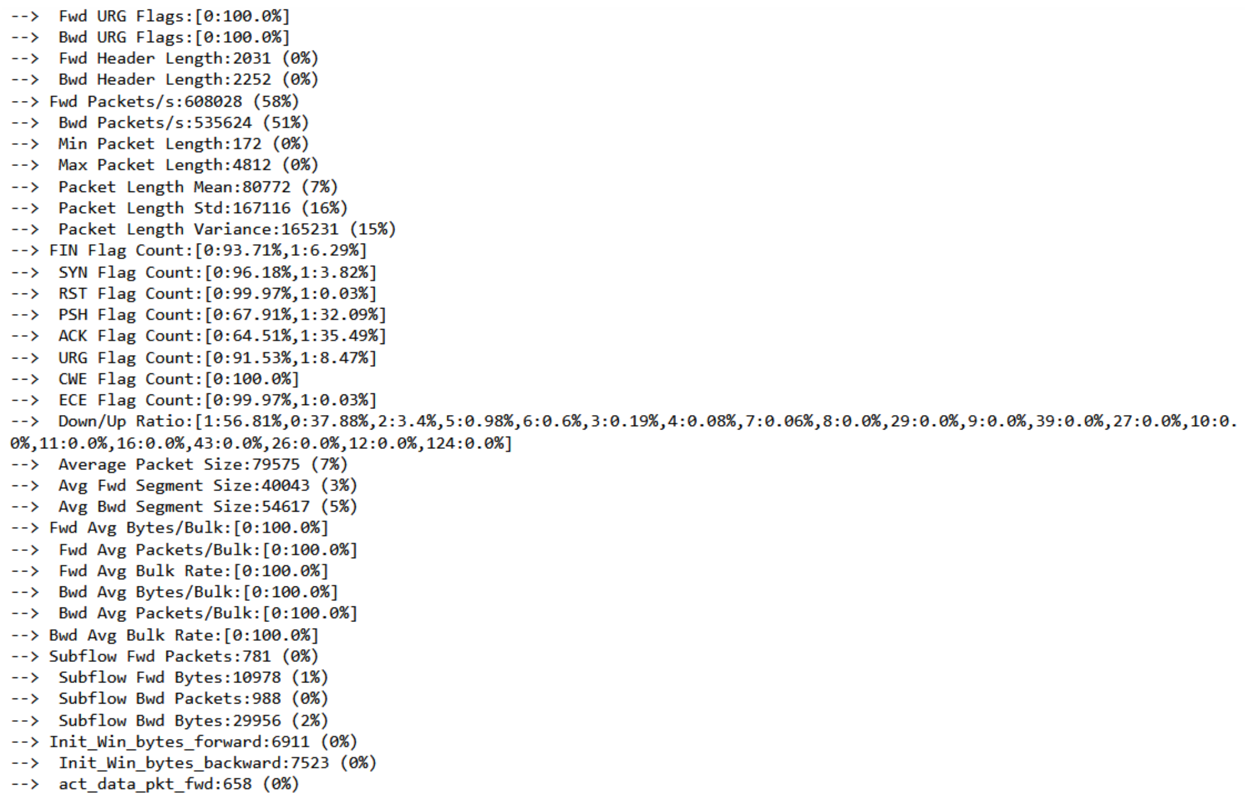
The dataset has 78 unique features and 1,042,557 samples.

Analyzing the data:

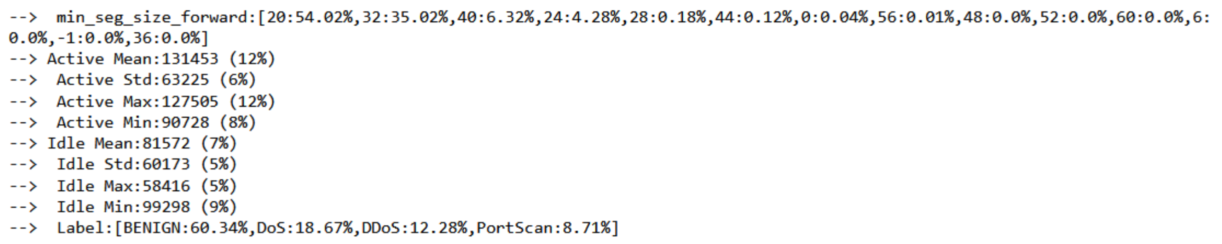
```
def expand_categories(values):
    result = []
    s = values.value_counts()
    t = float(len(values))
    for v in s.index:
        result.append("{}:{}%".format(v,round(100*(s[v]/t),2)))
    return "[{}]".format(",".join(result))

def analyze(df):
    cols = df.columns.values
    total = float(len(df))
    for col in cols:
        uniques = df[col].unique()
        unique_count = len(uniques)
        if unique_count>100:
            print("--> {}:{} ({}%)".format(col,unique_count,int(((unique_count)/total)*100)))
        else:
            print("--> {}:{}".format(col,expand_categories(df[col])))
            expand_categories(df[col])
```

```
analyze(df)
```

```
-->  Destination Port:43225 (4%)
-->  Flow Duration:574207 (55%)
-->  Total Fwd Packets:781 (0%)
-->  Total Backward Packets:988 (0%)
--> Total Length of Fwd Packets:10978 (1%)
-->  Total Length of Bwd Packets:29953 (2%)
-->  Fwd Packet Length Max:4132 (0%)
-->  Fwd Packet Length Min:266 (0%)
-->  Fwd Packet Length Mean:40043 (3%)
-->  Fwd Packet Length Std:85441 (8%)
--> Bwd Packet Length Max:4138 (0%)
-->  Bwd Packet Length Min:452 (0%)
-->  Bwd Packet Length Mean:54617 (5%)
-->  Bwd Packet Length Std:77400 (7%)
--> Flow Bytes/s:719867 (69%)
-->  Flow Packets/s:617140 (59%)
-->  Flow IAT Mean:598172 (57%)
-->  Flow IAT Std:427642 (41%)
-->  Flow IAT Max:290294 (27%)
-->  Flow IAT Min:61326 (5%)
--> Fwd IAT Total:185351 (17%)
-->  Fwd IAT Mean:283373 (27%)
-->  Fwd IAT Std:261890 (25%)
-->  Fwd IAT Max:181285 (17%)
-->  Fwd IAT Min:47672 (4%)
--> Bwd IAT Total:212738 (20%)
-->  Bwd IAT Mean:318909 (30%)
-->  Bwd IAT Std:352303 (33%)
-->  Bwd IAT Max:203355 (19%)
-->  Bwd IAT Min:23881 (2%)
--> Fwd PSH Flags:[0:96.18%,1:3.82%]
-->  Bwd PSH Flags:[0:100.0%]
```



```
-->  Fwd URG Flags:[0:100.0%]
-->  Bwd URG Flags:[0:100.0%]
-->  Fwd Header Length:2031 (0%)
-->  Bwd Header Length:2252 (0%)
--> Fwd Packets/s:608028 (58%)
-->  Bwd Packets/s:535624 (51%)
-->  Min Packet Length:172 (0%)
-->  Max Packet Length:4812 (0%)
-->  Packet Length Mean:80772 (7%)
-->  Packet Length Std:167116 (16%)
-->  Packet Length Variance:165231 (15%)
--> FIN Flag Count:[0:93.71%,1:6.29%]
-->  SYN Flag Count:[0:96.18%,1:3.82%]
-->  RST Flag Count:[0:99.97%,1:0.03%]
-->  PSH Flag Count:[0:67.91%,1:32.09%]
-->  ACK Flag Count:[0:64.51%,1:35.49%]
-->  URG Flag Count:[0:91.53%,1:8.47%]
-->  CWE Flag Count:[0:100.0%]
-->  ECE Flag Count:[0:99.97%,1:0.03%]
-->  Down/Up Ratio:[1:56.81%,0:37.88%,2:3.4%,5:0.98%,6:0.6%,3:0.19%,4:0.08%,7:0.06%,8:0.0%,29:0.0%,9:0.0%,39:0.0%,27:0.0%,10:0.
0%,11:0.0%,16:0.0%,43:0.0%,26:0.0%,12:0.0%,124:0.0%]
-->  Average Packet Size:79575 (7%)
-->  Avg Fwd Segment Size:40043 (3%)
-->  Avg Bwd Segment Size:54617 (5%)
--> Fwd Avg Bytes/Bulk:[0:100.0%]
-->  Fwd Avg Packets/Bulk:[0:100.0%]
-->  Fwd Avg Bulk Rate:[0:100.0%]
-->  Bwd Avg Bytes/Bulk:[0:100.0%]
-->  Bwd Avg Packets/Bulk:[0:100.0%]
--> Bwd Avg Bulk Rate:[0:100.0%]
--> Subflow Fwd Packets:781 (0%)
-->  Subflow Fwd Bytes:10978 (1%)
-->  Subflow Bwd Packets:988 (0%)
-->  Subflow Bwd Bytes:29956 (2%)
--> Init_Win_bytes_forward:6911 (0%)
-->  Init_Win_bytes_backward:7523 (0%)
-->  act_data_pkt_fwd:658 (0%)
```



```
-->  min_seg_size_forward:[20:54.02%,32:35.02%,40:6.32%,24:4.28%,28:0.18%,44:0.12%,0:0.04%,56:0.01%,48:0.0%,52:0.0%,60:0.0%,6:
0.0%,-1:0.0%,36:0.0%]
--> Active Mean:131453 (12%)
-->  Active Std:63225 (6%)
-->  Active Max:127505 (12%)
-->  Active Min:90728 (8%)
--> Idle Mean:81572 (7%)
-->  Idle Std:60173 (5%)
-->  Idle Max:58416 (5%)
-->  Idle Min:99298 (9%)
-->  Label:[BENIGN:60.34%,DoS:18.67%,DDoS:12.28%,PortScan:8.71%]
```

We are analyzing the data by looking at the unique values present in the dataset by calling analyze() function. For example, the destination port column has 43225 unique values, and there is a 4% overlap. We have displayed the percentage of the unique value in the column if the number of unique values is less than 100 to save display space. This is done by the expand_categories() function. For example, a text or categorical feature such as label has a few unique values, and the program shows the percentages of each category. Finally, to be on the safer side, we checked whether there are any null or infinity values or not and we are dropping those rows.

```
df['Flow Bytes/s'].max()
```

```
inf
```

```
df = df.replace([np.inf, -np.inf], np.nan).dropna(axis=0)
```

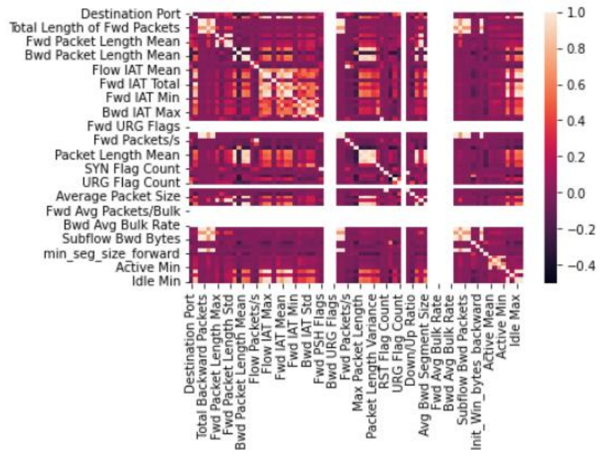
```
df
```

```
         Destination    Flow    Total      Total    Total    Total     Fwd     Fwd      Fwd       Fwd  ... min_seg_size_forward  Active  Active  Ac
                Port Duration    Fwd   Backward   Length   Length  Packet  Packet   Packet    Packet
                                Packets  Packets   of Fwd   of Bwd  Length  Length   Length    Length                            Mean     Std
                                                  Packets  Packets     Max     Min     Mean       Std
      0        54865        3       2          0       12        0       6       6      6.0   0.00000 ...                   20     0.0     0.0
      1        55054      109       1          1        6        6       6       6      6.0   0.00000 ...                   20     0.0     0.0
      2        55055       52       1          1        6        6       6       6      6.0   0.00000 ...                   20     0.0     0.0
      3        46236       34       1          1        6        6       6       6      6.0   0.00000 ...                   20     0.0     0.0
      4        54863        3       2          0       12        0       6       6      6.0   0.00000 ...                   20     0.0     0.0
    ...          ...      ...     ...        ...      ...      ...     ...     ...      ...       ... ...                  ...     ...     ...
1042552           53    32215       4          2      112      152      28      28     28.0   0.00000 ...                   20     0.0     0.0
1042553           53      324       2          2       84      362      42      42     42.0   0.00000 ...                   20     0.0     0.0
1042554        58030       82       2          1       31        6      31       0     15.5  21.92031 ...                   32     0.0     0.0
1042555           53  1048635       6          2      192      256      32      32     32.0   0.00000 ...                   20     0.0     0.0
1042556           53    94939       4          2      188      226      47      47     47.0   0.00000 ...                   20     0.0     0.0

1041899 rows × 78 columns
```

Then we are looking at heatmap to find correlated features which will be useful while developing machine learning model.

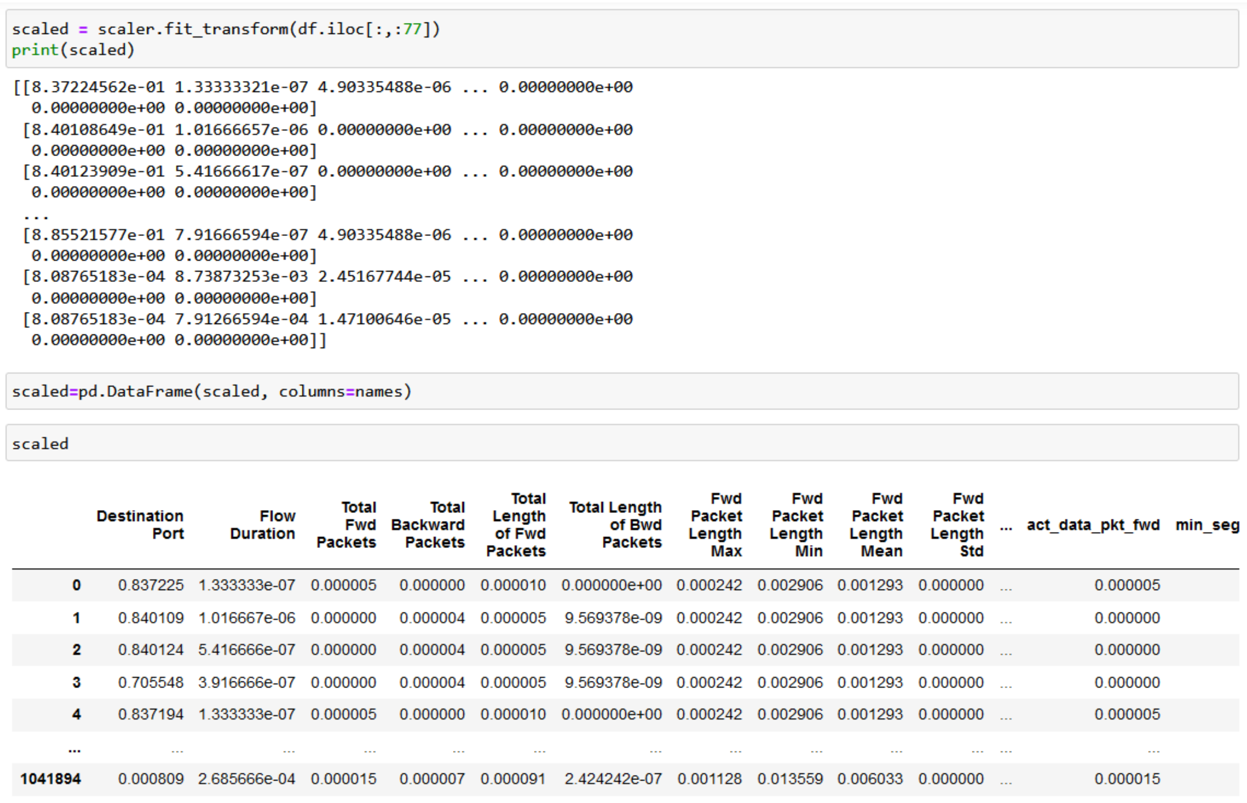
```
sns.heatmap(df.corr())
```

```
<AxesSubplot:>
```



Preprocessing the data:

We are scaling since the range of the data is large. We use MinMaxScaler which uses maximum and minimum as its range and scales the columns and keeps the values of the columns between 0 and 1. This is done to speed up the computation.



```
scaled = scaler.fit_transform(df.iloc[:,:77])
print(scaled)
```

```
[[8.37224562e-01 1.33333321e-07 4.90335488e-06 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [8.40108649e-01 1.01666657e-06 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [8.40123909e-01 5.41666617e-07 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 ...
 [8.85521577e-01 7.91666594e-07 4.90335488e-06 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [8.08765183e-04 8.73873253e-03 2.45167744e-05 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [8.08765183e-04 7.91266594e-04 1.47100646e-05 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]]
```

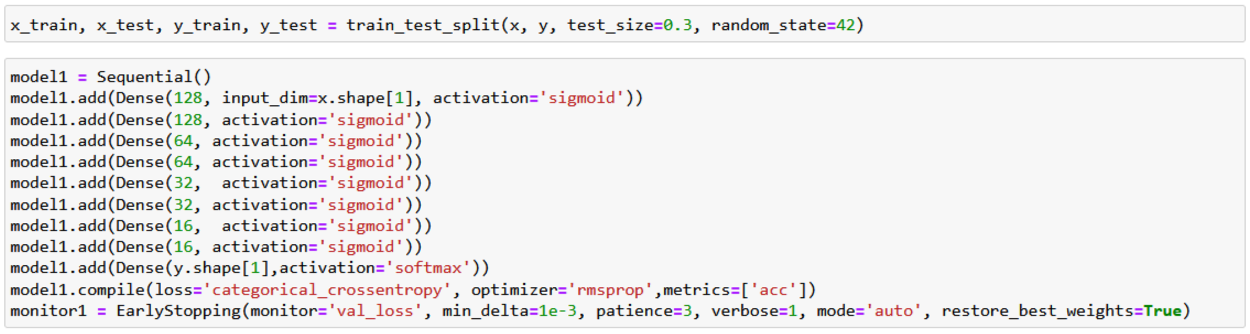
```
scaled=pd.DataFrame(scaled, columns=names)
```

```
scaled
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	act_data_pkt_fwd	min_seg
0	0.837225	1.333333e-07	0.000005	0.000000	0.000010	0.000000e+00	0.000242	0.002906	0.001293	0.000000	...	0.000005	
1	0.840109	1.016667e-06	0.000000	0.000004	0.000005	9.569378e-09	0.000242	0.002906	0.001293	0.000000	...	0.000000	
2	0.840124	5.416666e-07	0.000000	0.000004	0.000005	9.569378e-09	0.000242	0.002906	0.001293	0.000000	...	0.000000	
3	0.705548	3.916666e-07	0.000000	0.000004	0.000005	9.569378e-09	0.000242	0.002906	0.001293	0.000000	...	0.000000	
4	0.837194	1.333333e-07	0.000005	0.000000	0.000010	0.000000e+00	0.000242	0.002906	0.001293	0.000000	...	0.000005	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1041894	0.000809	2.685666e-04	0.000015	0.000007	0.000091	2.424242e-07	0.001128	0.013559	0.006033	0.000000	...	0.000015	

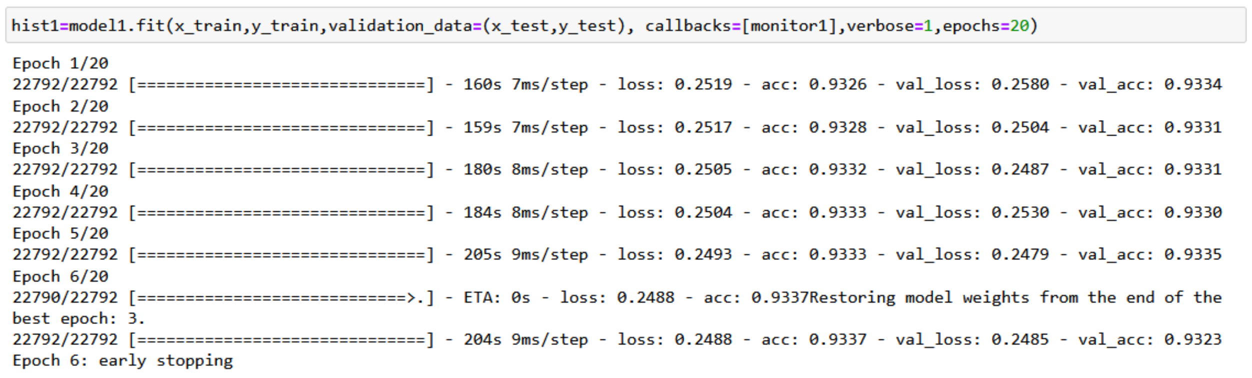
Training the model:

We have first split the dataset into training (70%) and test (30%) dataset. We developed an artificial neural network which is sequential. We added one input layer with 77 neurons, seven hidden layers with sigmoid as activation layer with 128 or 64 or 32 or 16 neurons and one output layer with softmax as activation layer with 4 neurons.



```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

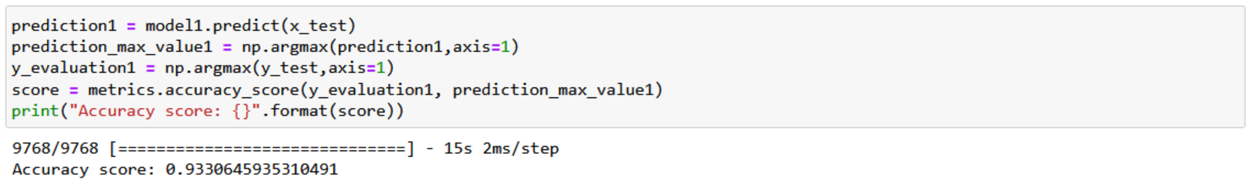
```
model1 = Sequential()
model1.add(Dense(128, input_dim=x.shape[1], activation='sigmoid'))
model1.add(Dense(128, activation='sigmoid'))
model1.add(Dense(64, activation='sigmoid'))
model1.add(Dense(64, activation='sigmoid'))
model1.add(Dense(32,  activation='sigmoid'))
model1.add(Dense(32, activation='sigmoid'))
model1.add(Dense(16,  activation='sigmoid'))
model1.add(Dense(16, activation='sigmoid'))
model1.add(Dense(y.shape[1],activation='softmax'))
model1.compile(loss='categorical_crossentropy', optimizer='rmsprop',metrics=['acc'])
monitor1 = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=3, verbose=1, mode='auto', restore_best_weights=True)
```



```
hist1=model1.fit(x_train,y_train,validation_data=(x_test,y_test), callbacks=[monitor1],verbose=1,epochs=20)
```

```
Epoch 1/20
22792/22792 [==============================] - 160s 7ms/step - loss: 0.2519 - acc: 0.9326 - val_loss: 0.2580 - val_acc: 0.9334
Epoch 2/20
22792/22792 [==============================] - 159s 7ms/step - loss: 0.2517 - acc: 0.9328 - val_loss: 0.2504 - val_acc: 0.9331
Epoch 3/20
22792/22792 [==============================] - 180s 8ms/step - loss: 0.2505 - acc: 0.9332 - val_loss: 0.2487 - val_acc: 0.9331
Epoch 4/20
22792/22792 [==============================] - 184s 8ms/step - loss: 0.2504 - acc: 0.9333 - val_loss: 0.2530 - val_acc: 0.9330
Epoch 5/20
22792/22792 [==============================] - 205s 9ms/step - loss: 0.2493 - acc: 0.9333 - val_loss: 0.2479 - val_acc: 0.9335
Epoch 6/20
22790/22792 [============================>.] - ETA: 0s - loss: 0.2488 - acc: 0.9337Restoring model weights from the end of the
best epoch: 3.
22792/22792 [==============================] - 204s 9ms/step - loss: 0.2488 - acc: 0.9337 - val_loss: 0.2485 - val_acc: 0.9323
Epoch 6: early stopping
```

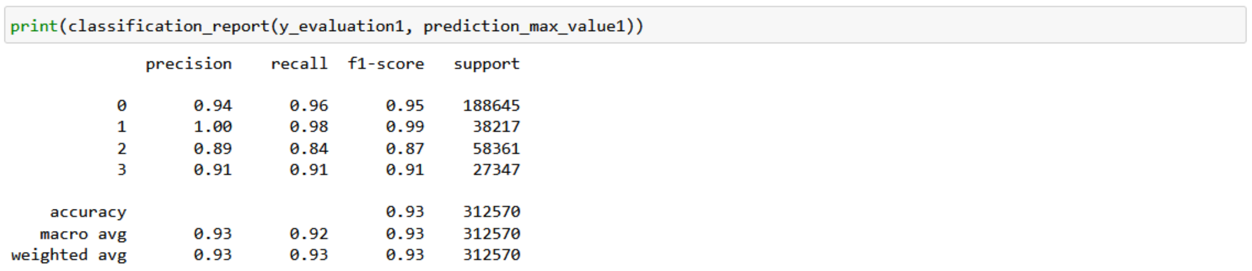
Accuracy and Recall Score:



```
prediction1 = model1.predict(x_test)
prediction_max_value1 = np.argmax(prediction1,axis=1)
y_evaluation1 = np.argmax(y_test,axis=1)
score = metrics.accuracy_score(y_evaluation1, prediction_max_value1)
print("Accuracy score: {}".format(score))
```

```
9768/9768 [==============================] - 15s 2ms/step
Accuracy score: 0.9330645935310491
```

As we can see, we have got an accuracy of 93.30% which is really good for an ANN model.



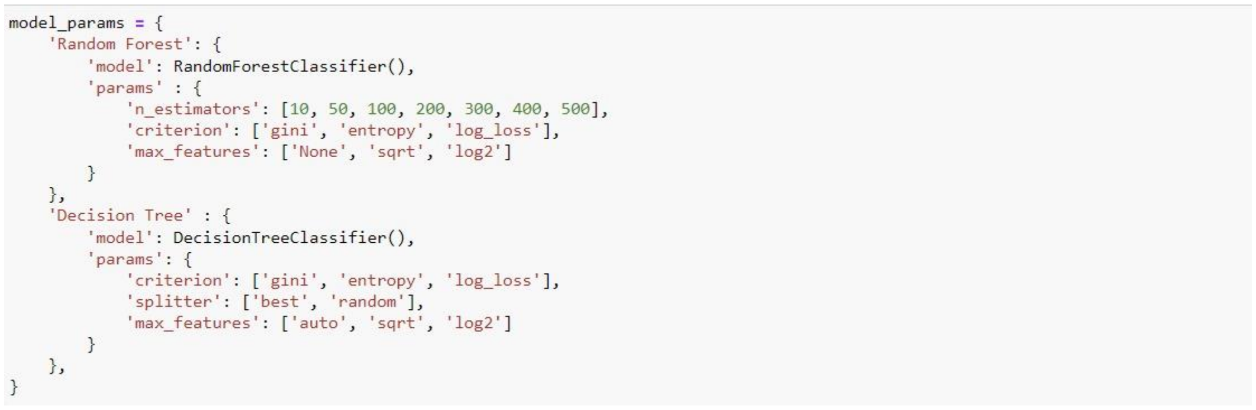
```
print(classification_report(y_evaluation1, prediction_max_value1))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95    188645
           1       1.00      0.98      0.99     38217
           2       0.89      0.84      0.87     58361
           3       0.91      0.91      0.91     27347

    accuracy                           0.93    312570
   macro avg       0.93      0.92      0.93    312570
weighted avg       0.93      0.93      0.93    312570
```

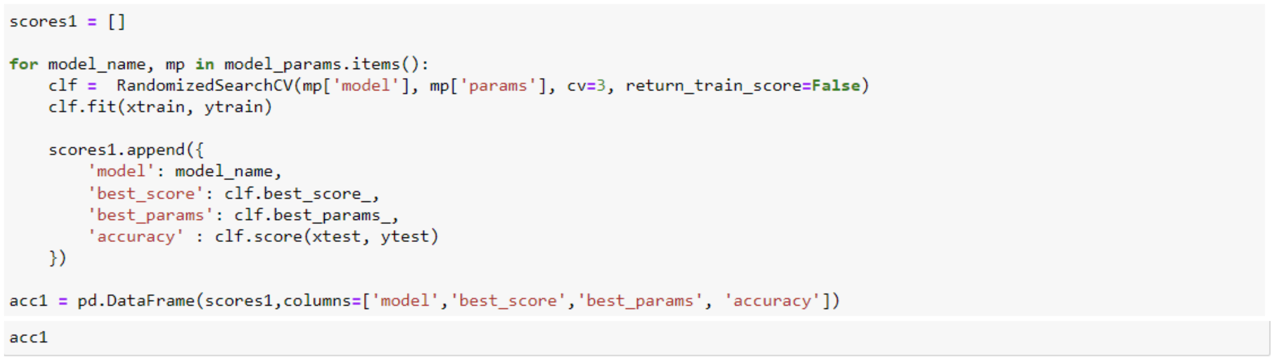
Recall score is used to assess how well a model performs in terms of accurately counting true positives among all real positive values. The recall score of the ANN model is 93% which is better than all other models.

Developing Machine Learning Models:

We developed machine learning models using random forest algorithm, decision tree algorithm and KNN algorithm. We preprocessed the data and did feature selection and then we used RandomizedSearchCV with multiple hyper parameters for hyper parameter tuning.

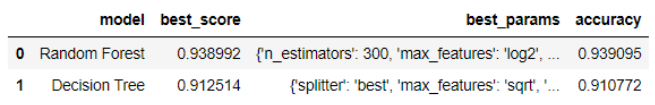


```python
model_params = {
    'Random Forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [10, 50, 100, 200, 300, 400, 500],
            'criterion': ['gini', 'entropy', 'log_loss'],
            'max_features': ['None', 'sqrt', 'log2']
        }
    },
    'Decision Tree' : {
        'model': DecisionTreeClassifier(),
        'params': {
            'criterion': ['gini', 'entropy', 'log_loss'],
            'splitter': ['best', 'random'],
            'max_features': ['auto', 'sqrt', 'log2']
        }
    },
}
```



```python
scores1 = []

for model_name, mp in model_params.items():
    clf =  RandomizedSearchCV(mp['model'], mp['params'], cv=3, return_train_score=False)
    clf.fit(xtrain, ytrain)

    scores1.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_,
        'accuracy' : clf.score(xtest, ytest)
    })

acc1 = pd.DataFrame(scores1,columns=['model','best_score','best_params', 'accuracy'])
```

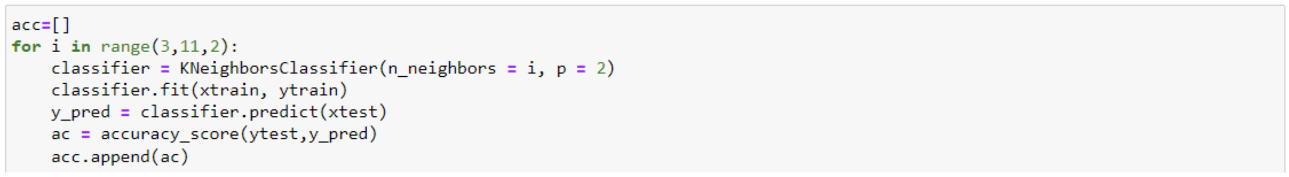
```python
acc1
```



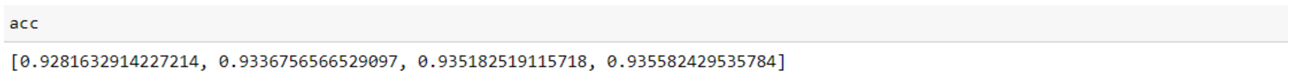
	model	best_score	best_params	accuracy
0	Random Forest	0.938992	{'n_estimators': 300, 'max_features': 'log2', ...	0.939095
1	Decision Tree	0.912514	{'splitter': 'best', 'max_features': 'sqrt', '...	0.910772

We got an accuracy of 93.90% for Random Forest algorithm and 91.07% for Decision Tree algorithm which is really good but recall score for Random Forest algorithm is 93.1% and for Decision Tree is 91.07%. The hyperparameters for the best obtained accuracy has been mentioned in the above picture.

Finally, we develop a model suing KNN algorithm.



```python
acc=[]
for i in range(3,11,2):
    classifier = KNeighborsClassifier(n_neighbors = i, p = 2)
    classifier.fit(xtrain, ytrain)
    y_pred = classifier.predict(xtest)
    ac = accuracy_score(ytest,y_pred)
    acc.append(ac)
```



```python
acc
```

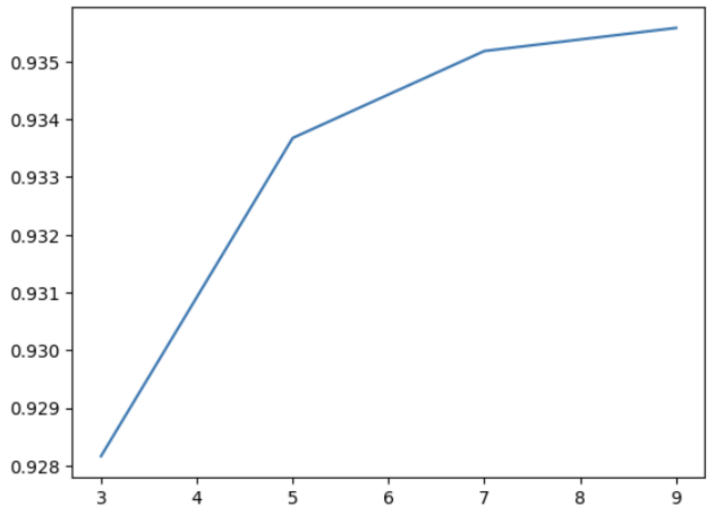
```
[0.9281632914227214, 0.9336756566529097, 0.935182519115718, 0.935582429535784]
```

We get highest accuracy of 93.55% when K value is 9 using KNN algorithm.

Plotting accuracy with respect to K value (number of neighbours).

```
plt.plot([3,5,7,9],acc)
```

```
[<matplotlib.lines.Line2D at 0x2ef6e594f10>]
```



Prediction:

We load the pickled ANN model and use it to predict the network flow generated by CICFlowMeter.

```
from keras.models import load_model
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

```
model = load_model("O:\\VIT\\Fall Semester 2022-2023\\Information Security Analysis and Audit\\Project\\Models\\ISAA_ANN.h5")
df=pd.read_csv("C:\\Users\\manig\\Downloads\\test_ISCX.csv")
```

```
df = df.drop(df[df.Protocol  == 'Protocol'].index)
```

```
res=df[['Src IP']]
```

```
res['Src Port']=df['Src Port']
```

```
drop=['Flow ID','Src IP','Src Port','Dst IP','Protocol','Timestamp']
df.drop(drop, axis=1, inplace=True)
```

```
df.drop('Label', axis=1, inplace=True)
```

```
df=df.astype(np.float32)
```

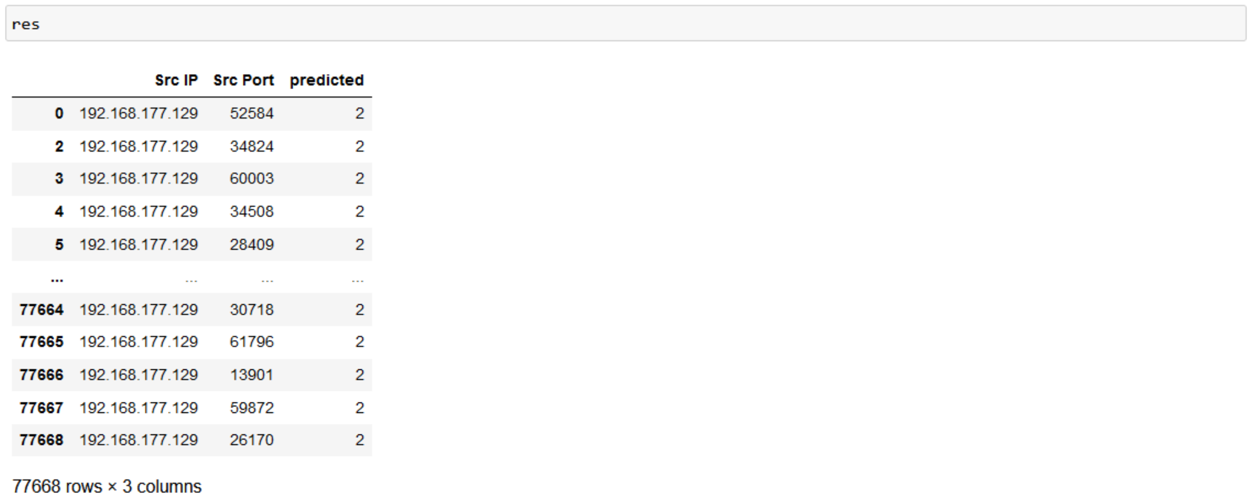
```
pred=model.predict(df)
```

```
2428/2428 [==============================] - 5s 2ms/step
```

```
predicted=[]
for i in range(len(pred)):
    predicted.append(np.argmax(pred[i]))
```

```
res['predicted']=predicted
```

The prediction result is as follows:

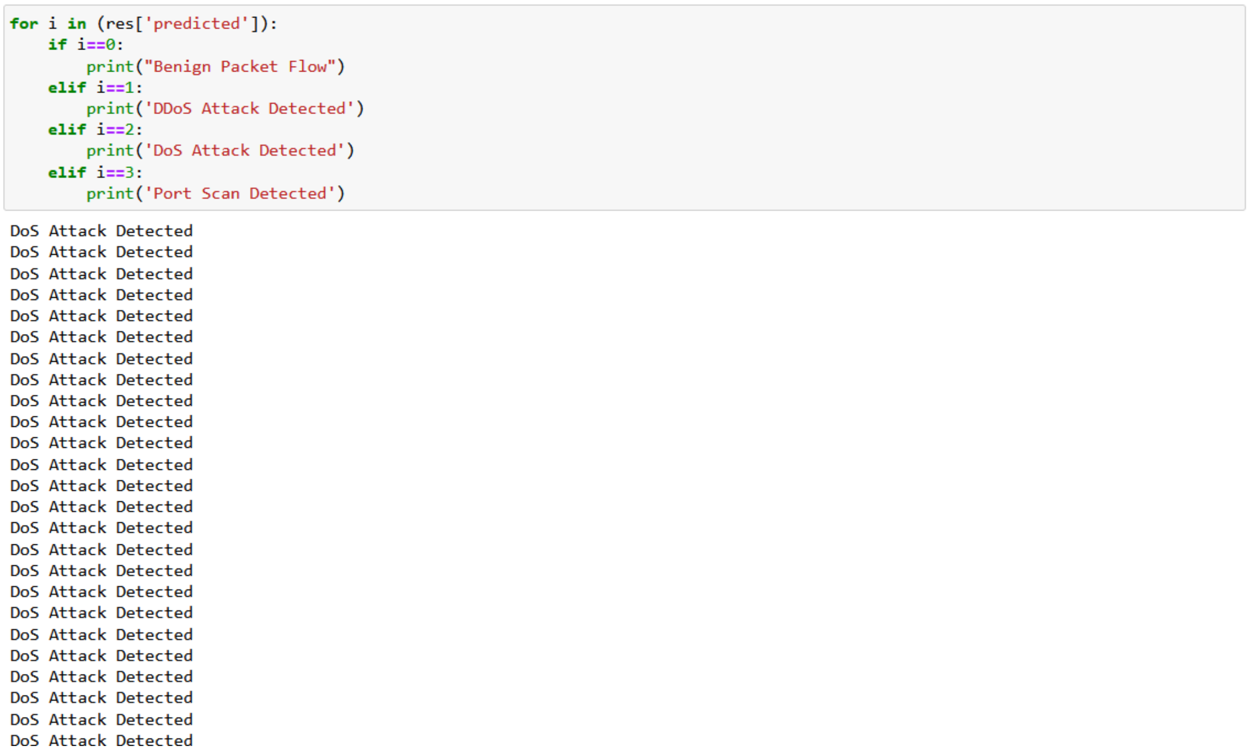


```
res
```

	Src IP	Src Port	predicted
0	192.168.177.129	52584	2
2	192.168.177.129	34824	2
3	192.168.177.129	60003	2
4	192.168.177.129	34508	2
5	192.168.177.129	28409	2
...	...	...	...
77664	192.168.177.129	30718	2
77665	192.168.177.129	61796	2
77666	192.168.177.129	13901	2
77667	192.168.177.129	59872	2
77668	192.168.177.129	26170	2

77668 rows × 3 columns

We can see that we are getting DoS as prediction result which is the expected result.



```
for i in (res['predicted']):
    if i==0:
        print("Benign Packet Flow")
    elif i==1:
        print('DDoS Attack Detected')
    elif i==2:
        print('DoS Attack Detected')
    elif i==3:
        print('Port Scan Detected')
```

```
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
DoS Attack Detected
```

Results and Discussions:

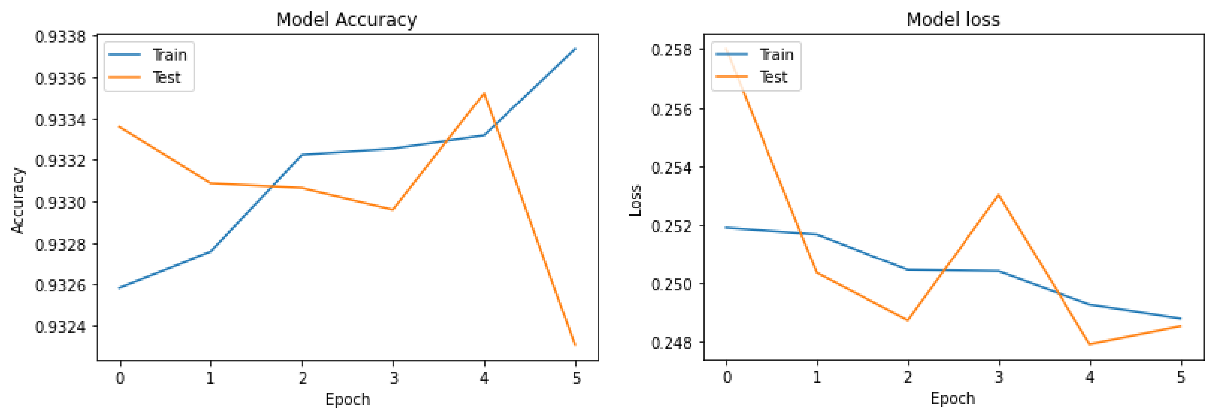
After implementing the various techniques of Machine Learning as well as Deep Learning which we had proposed earlier, we carried out an analysis to figure out which models performed the best. For this, we considered multiple metrics for evaluation including accuracy, recall performance, plot for visualizing loss and accuracy convergence.

To conclude we can say that the Artificial Neural Network (ANN) model performed quite well as compared to Random Forest and KNN.
Comparison of Accuracies:

ANN	-	93.306%
KNN	-	93.5%
Random Forest	-	93.8%

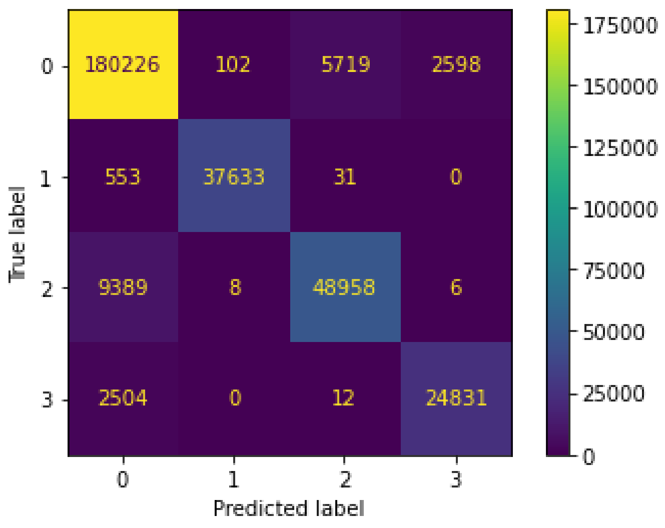
All these accuracies were conducted on our training data, therefore our random forest and KNN showed higher accuracies as compared to ANN.

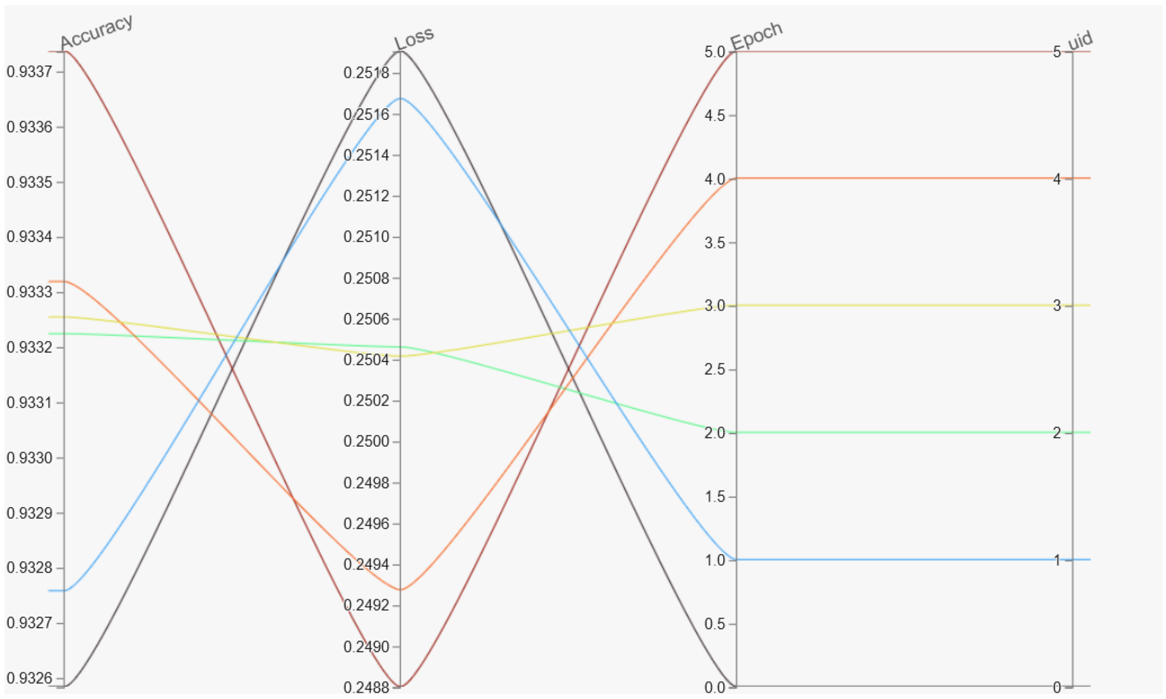
Though this is the case, our ANN model is more robust when it comes to predicting values other than the training data. We can see this evidently by looking at the precision and recall values of our ANN model which are 93.0% and 93.0% respectively. These scores are much better as compared to the scores of Random Forest and KNN.



Since our most preferred model is ANN in this case, we further made plots to support our conclusion. The Graph of Loss shows that the training and testing loss reduces overtime and converges in the end to a score of 0.248. In the accuracy plot, we see that the accuracy increases overtime and the maximum variation in accuracy is about 0.0014 which is very low and is a good indication.

The Confusion Matrix which we acquired shows that the True Positive values dominate over the diagonal and there are hardly few values which are wrongly predicted. This not only shows us the correct predictions but also shows us which features were specifically predicted incorrectly along with numerical data to support the conclusion.





The last plot shows us the accuracy and loss matched for each epoch, and just shows us how much both of them are varying over time. At the accuracy increase to a maximum of 93.37 and the loss decreases to 0.2488 which tells us ANN is the best model.

References:

1. Dutta, V., Choraś, M., Pawlicki, M., & Kozik, R. (2020, August 15). A Deep Learning Ensemble for Network Anomaly and Cyber-Attack Detection. Sensors, 20(16), 4583. https://doi.org/10.3390/s20164583

2. Kumar, P., Kumar, A. A., Sahayakingsly, C., & Udayakumar, A. (2020b, October 31). Analysis of intrusion detection in cyber attacks using DEEP learning neural networks. Peer-to-Peer Networking and Applications, 14(4), 2565–2584. https://doi.org/10.1007/s12083-020-00999-y

3. Ullah, F., Naeem, H., Jabbar, S., Khalid, S., Latif, M. A., Al-turjman, F., & Mostarda, L. (2019). Cyber Security Threats Detection in Internet of Things Using Deep Learning Approach. IEEE Access, 7, 124379–124389. https://doi.org/10.1109/access.2019.2937347

4. Ahmad, T., Truscan, D., Vain, J., & Porres, I. (2022, April). Early Detection of Network Attacks Using Deep Learning. 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). https://doi.org/10.1109/icstw55395.2022.00020

5. Team, E. (2022, August 16). 2022 Must-Know Cyber Attack Statistics and Trends. Embroker. Retrieved September 9, 2022, from https://www.embroker.com/blog/cyber-attack-statistics/

6. Chang, J. (2022, January 14). Cybercrimes are now an everyday concern for businesses. Cybersecurity statistics indicate a significant rise i. Financesonline.com. Retrieved September 9, 2022, from https://financesonline.com/cybersecurity-statistics/

7. Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”, 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018