INTRUSION DETECTION SYSTEM USING DEEP LEARNING

Project Review 3 for
CSE3501- Information Security Analysis and Audit (G2)

By:
YUVRAJ KUMAR: 18BIT0276
(CNN Implementation)

Complete Team Members:

AMAN AGARWAL: 18BIT0256
SHIVANSH SRIVASTAVA: 18BIT0324
YUVRAJ KUMAR: 18BIT0276

Submitted to

PROF. SUMAIYA THASEEN I.
School of Information Technology and Engineering,
Vellore Institute of Technology, Vellore-632014

FALL 2020-21

1. Design and Description of the System

Network Intrusion Detection Systems (NIDSs) are essential tools for the network system administrators to detect various security breaches inside an organization's network. An NIDS monitors and analyzes the network traffic entering into or exiting from the network devices of an organization and raises alarms if an intrusion is observed. Based on the methods of intrusion detection, NIDSs are categorized into two classes:

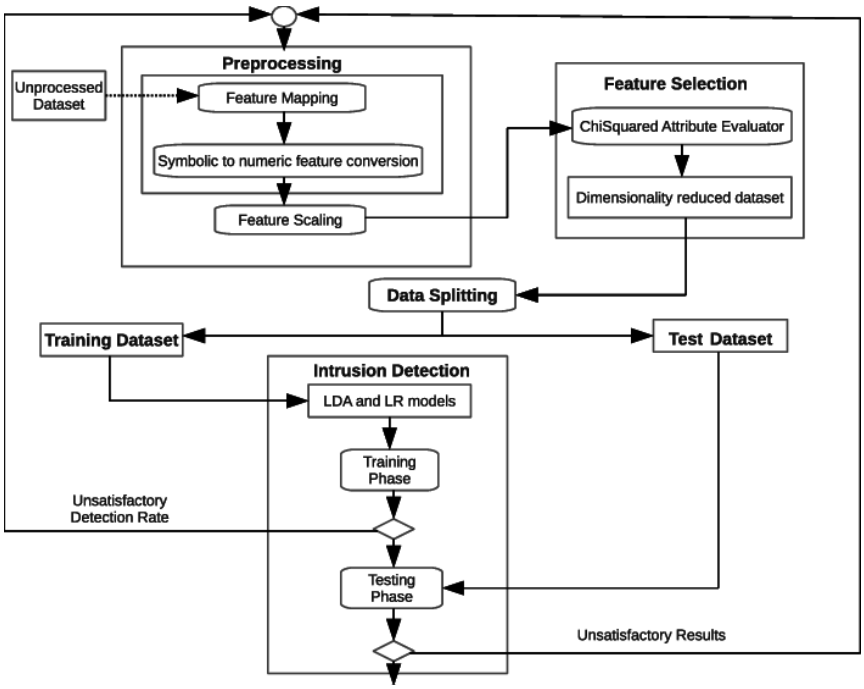
i) signature (misuse) based NIDS (SNIDS),
ii) anomaly detection-based NIDS (ADNIDS).

There are primarily two challenges that arise while developing an efficient and flexible NIDS for unknown future attacks. First, proper feature selections from the network traffic dataset for anomaly detection is difficult. The features selected for one class of attack may not work well for other categories of attacks due to continuously changing and evolving attack scenarios. Second, unavailability of labeled traffic dataset from real networks for developing an NIDS. Immense efforts are required to produce such a labeled dataset from the raw network traffic traces collected over a period or in real-time. Additionally, to preserve the confidentiality of the internal organizational network structure as well as the privacy of various users, network administrators are reluctant towards reporting any intrusion that might have occurred in their networks.

It is envisioned that the deep learning based approaches can help to overcome the challenges of developing an efficient NIDS . We can collect unlabeled network traffic data from different network sources and a good feature representation from these datasets using deep learning techniques can be obtained. These features can, then, be applied for supervised classification to a small, but labeled traffic dataset consisting of normal as well as anomalous traffic records. The traffic data for labeled dataset can be collected in a confined, isolated and private network environment. With this motivation, we use self-taught learning, a deep learning technique based on sparse autoencoder and soft-max regression, to develop an NIDS. We verify the usability of the self-taught learning based NIDS by applying on NSL-KDD intrusion dataset, an improved version of the benchmark dataset for various NIDS evaluations - KDD Cup 99. We provide a comparison of our current work with other techniques as well.

In the most widely used approach, the training data is used for both training and testing either using n-fold cross-validation or splitting the training data into training, cross-validation, and test sets. NIDSs based on this approach achieved very high accuracy and less false-alarm rates. The second approach uses the training and test data separately for the training and testing. Since the training and test data were collected in different environments, the accuracy obtained using the second approach is not as high as in the first approach. Therefore, we emphasize on the results of the second approach in our work for accurate evaluation of NIDS. However, we present the results of the first approach as well for completeness. We describe our NIDS implementation before discussing the results.

Architecture

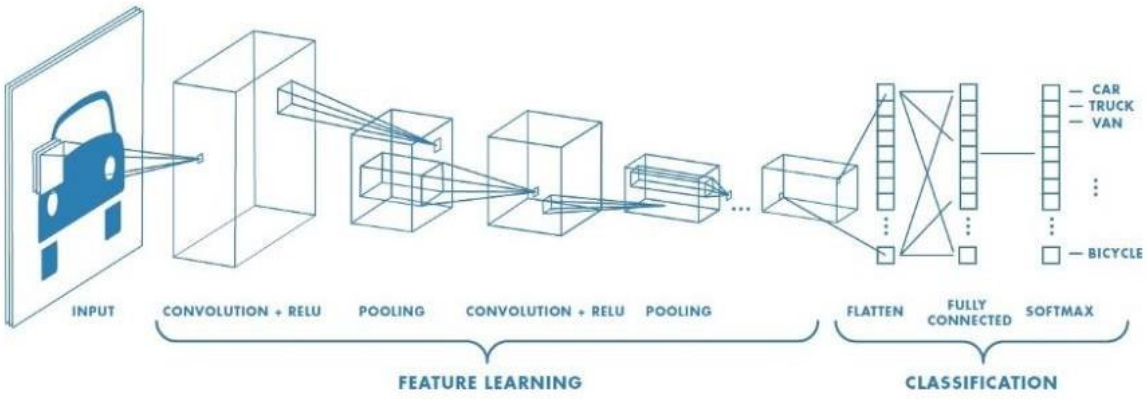


A Network Intrusion Detection System (NIDS) helps system administrators to detect network security breaches in their organizations. However, many challenges arise while developing a flexible and efficient NIDS for unforeseen and unpredictable attacks. We propose a deep learning based approach for developing such an efficient and flexible NIDS. We present the performance of our approach and compare it with a few previous work. Compared metrics include accuracy, precision, recall, and f-measure values.

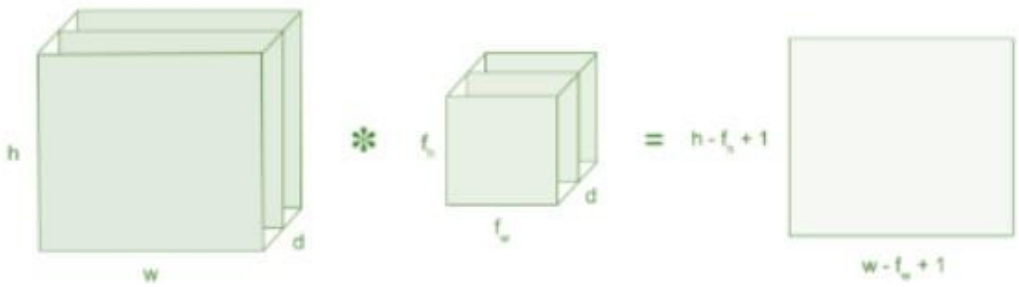
Convolutional Neural Network (CNN) — Deep Learning

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

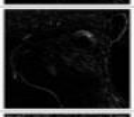
Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.



- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**



Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

2. PRE-PROCESSING (Normalization, Sampling – Oversampling, Undersampling (technique used for sampling - atleast 70%balanced)

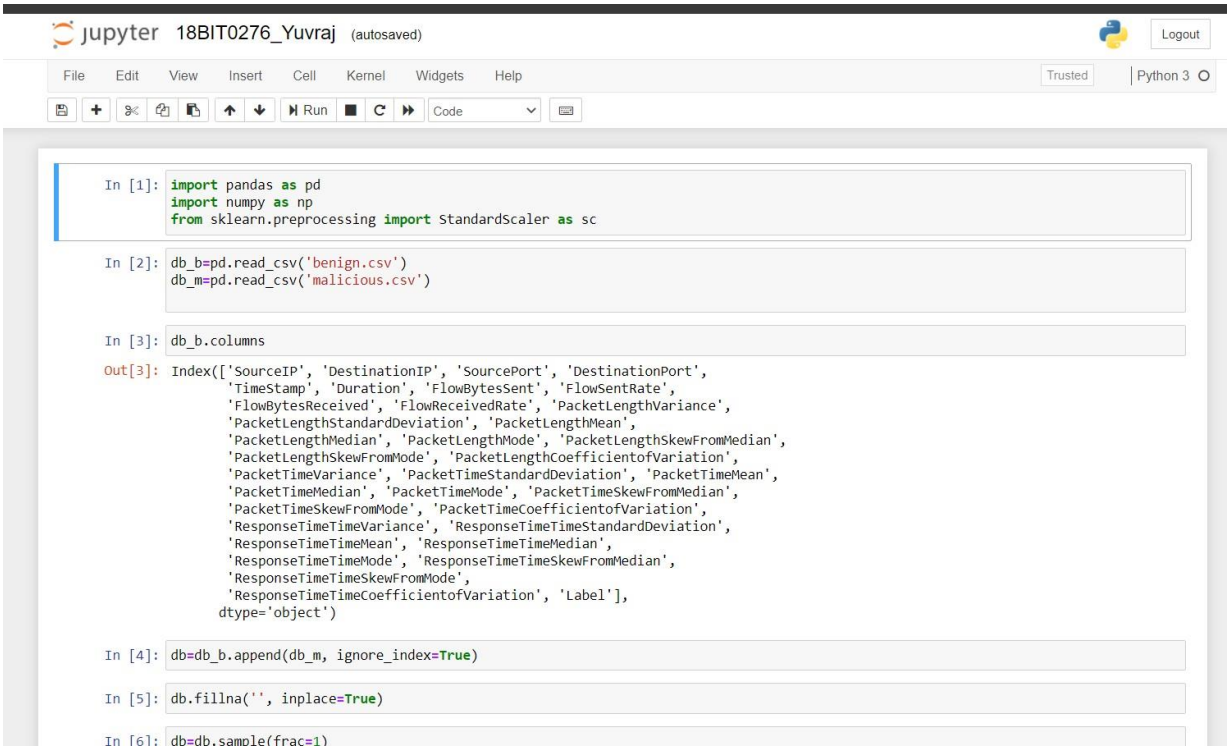
***Sampling is done before model creation. ->Pg.26*

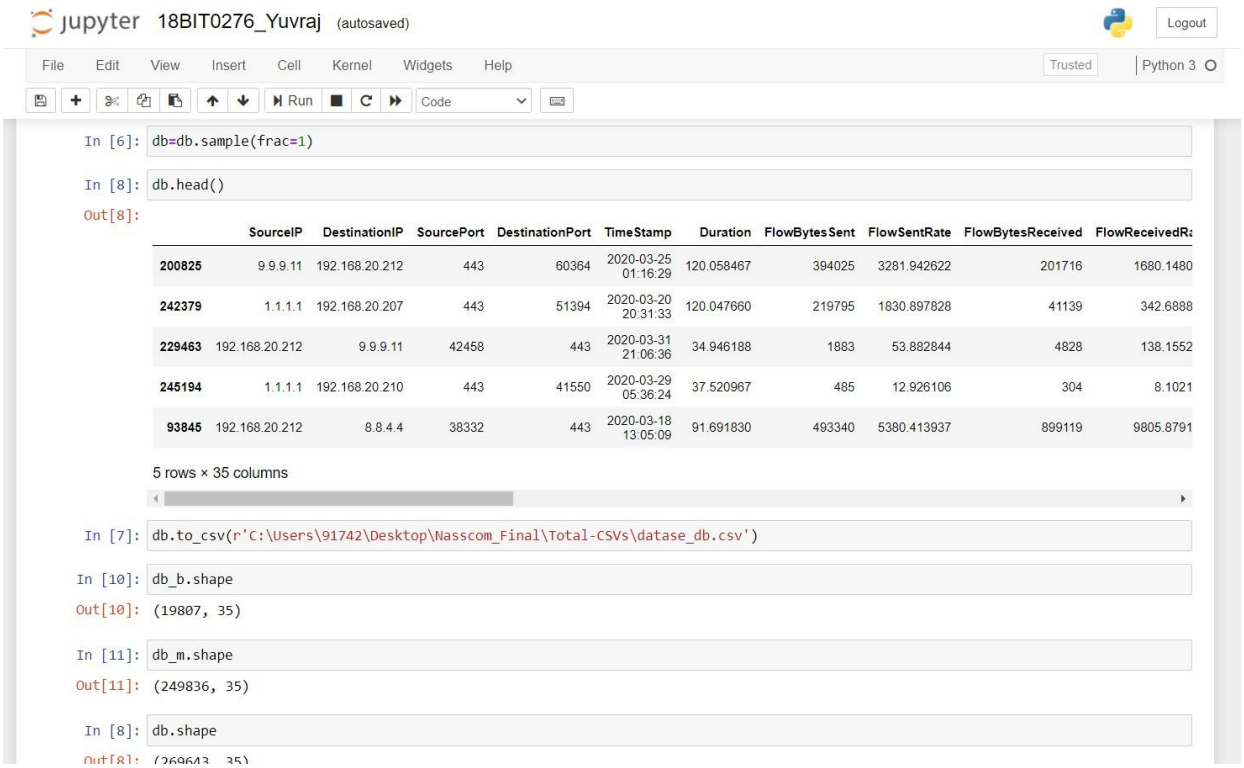
There were 4 datasets in a Total CSVs ZIP file provided to us by the Faculty.

Link to the Dataset :

https://www.unb.ca/cic/datasets/dohbrw-2020.html

One having **l2-benign** label and other containing **l2-malicious** were assigned to me. So, first step of processing was two combine two datasets into one. The combined dataset will have all Benign entries on top and all Malicious entries at the bottom or vice versa. This could create a problem will splitting into training and testing data. To avoid such problem I randomly shuffled the dataset so that both the labels get mixed.





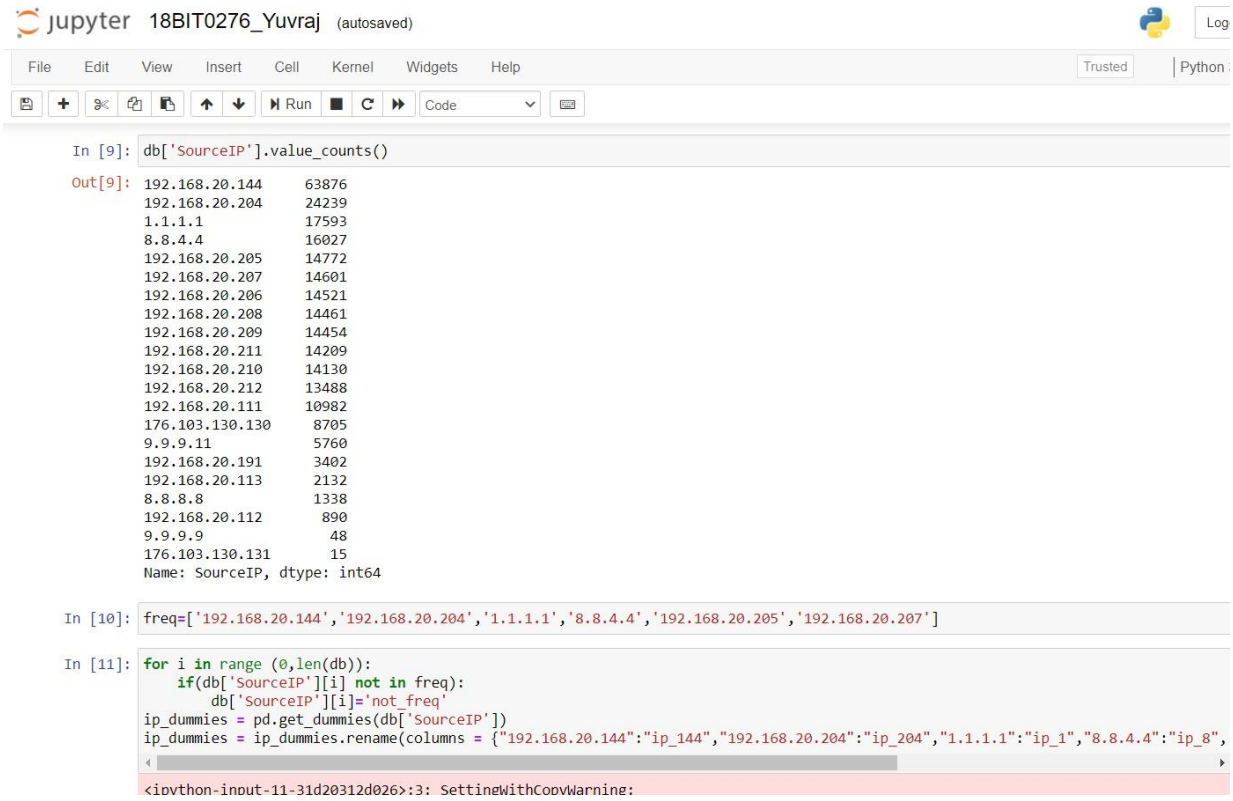
Now, the dataset basically had 2 kind of features, namely categorical and numeric. Categorical features comprised **SourceIP, DestinationIP, SourcePort, DestinationPort etc.** and numerical features included **Duration, FlowSentRate, FlowReceived, FlowBytesRate etc.** I have used different techniques to handle both kind of data.

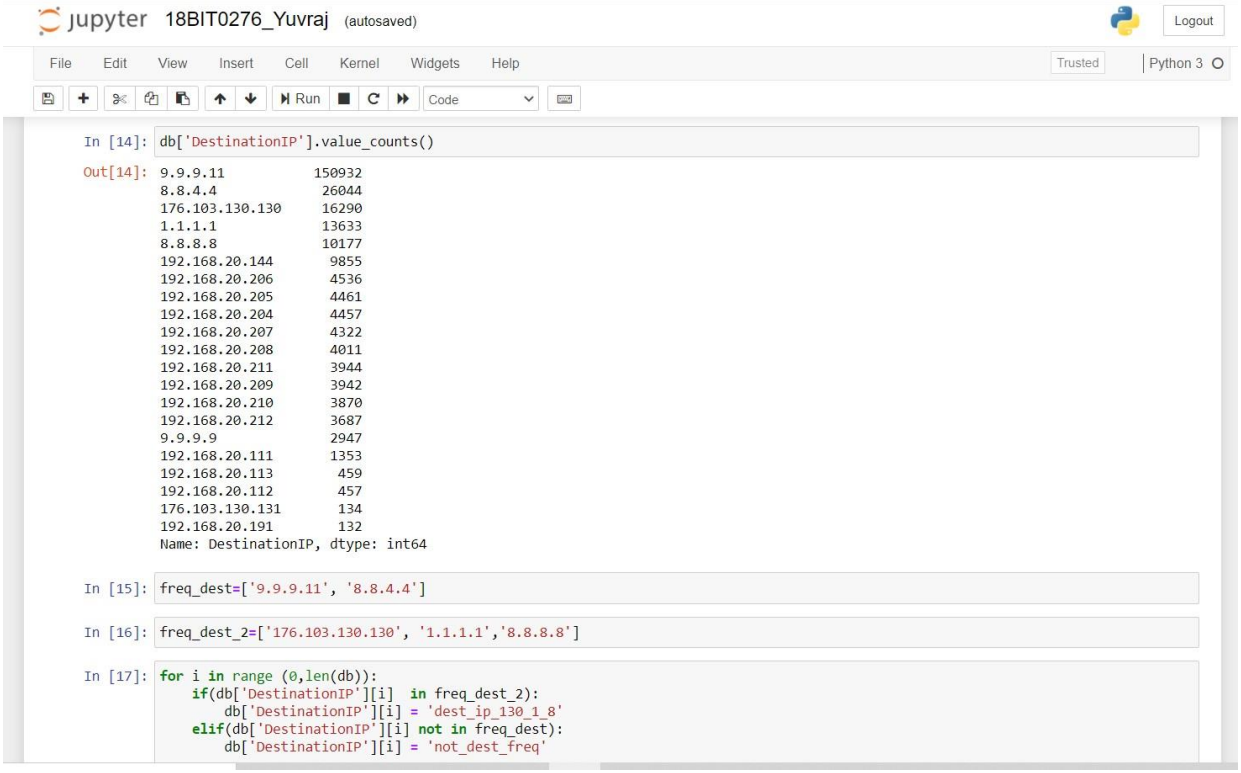
HANDLING CATEGORICAL DATA

Since total number of distinct values in each of the column are large, therefore encoding all them would have create a great sparsity in the dataset. So, in order to overcome the problem of sparsity, I analyzed each feature(column) of the dataset and choose a suitable threshold for the same. If the value is not in the threshold or given range, then I categorized it as other's. Different threshold criteria was chosen for each of the feature based on it's analysis. Then I one-hot encoded all of them so that I can feed them into my neural network. Below is the demonstration for the same.

This was done on the following column:

- **SOURCE IP COLUMN**
- **DESTINATION IP COLUMN**





SIMILARLY, AS ABOVE, THE SOURCE PORT AND DESTINATION PORT FEATURES HAVE BEEN PRE-PROCESSED AND CONCATENATING PRE-PROCESSED DATA INTO A NEW DATAFRAME

In [18]:
```
destination_dummy=pd.get_dummies(db['DestinationIP'])
```

In [19]:
```
destination_dummy = destination_dummy.rename(columns = {"9.9.9.11":"ip_11","8.8.4.4":"ip_4"})
```

In [20]:
```
db_create=pd.concat([db_create,destination_dummy], axis = 1)
```

In [21]:
```
db['SourcePort'].value_counts()
```

Out[21]:
```
443      49486
58759       73
41618       60
44886       60
40378       53
          ...
63314        1
64686        1
61443        1
60121        1
59999        1
Name: SourcePort, Length: 15446, dtype: int64
```

In [22]:
```
for i in range (0,len(db)):
    if(db['SourcePort'][i] != 443):
        db['SourcePort'][i] = 'Wrong_port'
    else:
        db['SourcePort'][i] = 'port_is_443'
```

```
<ipython-input-22-522c55c71581>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  db['SourcePort'][i] = 'Wrong_port'
C:\Users\91742\anaconda3\lib\site-packages\pandas\core\indexing.py:671: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  db['SourcePort'][i] = 'port_is_443'
```

In [23]:
```
for i in range (0,len(db)):
    if(db['DestinationPort'][i] != 443):
        db['DestinationPort'][i] = 'Wrong_port'
    else:
        db['DestinationPort'][i] = 'port_443'
```

```
<ipython-input-23-ec0a6933b03d>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  db['DestinationPort'][i] = 'port_443'
<ipython-input-23-ec0a6933b03d>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  db['DestinationPort'][i] = 'Wrong_port'
```

In [24]:
```
source_port_dummy = pd.get_dummies(db['SourcePort'])
```

In [25]:
```
destination_port_dummy = pd.get_dummies(db['DestinationPort'])
```

In [26]:
```
destination_port_dummy = destination_port_dummy.rename(columns = {"port_443":"destination_port_443"})
```

In [27]:
```
db_create = pd.concat([db_create,source_port_dummy],axis=1)
db_create = pd.concat([db_create,destination_port_dummy],axis=1)
```

HANDLING NUMERIC FEATURES / NORMALIZATION

Some values were really large and some values were small. To avoid biasness and create a balanced dataset, I used Standard Scaler to normalize the numeric data.

I WILL BE TAKING CARE OF ALL THE NUMERIC FEATURE BELOW LIKE, FlowSentRate, ByteRate, Duration etc. using different Methodologies like Median, Standard Scaling, Sc_Fit_Transform etc.

- **DURATION COLUMN**
- **FLOW BYTES SENT**
- **FLOW SENT RATE**
- **FLOW BYTES RECEIVED**
- **PACKET LENGTH (*USING MEDIAN METHOD*)**
- **PACKET LENGTH CoV**
- **PACKET TIME VARIANCE**
- **PACKET LENGTH MEAN**
- **OTHER SPECIAL NUMERIC FEATURES**

In [28]:
```
from sklearn.preprocessing import StandardScaler
```

In [29]:
```
sc=StandardScaler()
```

In [30]:
```
drn_db = db[["Duration"]]
drn_db = sc.fit_transform(drn_db)
drn_db = pd.DataFrame(drn_db)
```

In [31]:
```
drn_db = drn_db.rename(columns = {0:"Duration"})
```

In [32]:
```
db_create = pd.concat([db_create,drn_db],axis=1)
```

In [33]:
```
db['FlowBytesSent'].value_counts()
```

Out[33]:
```
1807      44384
1875      15189
1806       7627
1085       7474
1739       7471
          ...
211692        1
199402        1
201449        1
217825        1
421554        1
Name: FlowBytesSent, Length: 72805, dtype: int64
```

In [34]:
```
fs = db[['FlowBytesSent']]
```

In [35]:
```
fs.mean()
```

Out[35]:
```
FlowBytesSent    65757.889977
```

jupyter 18BIT0276_Yuvraj (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

0	1807

In [47]:
```
for i in range (0,len(db)):
    if(db['FlowBytesSent'][i] == 1807):
        db['FlowBytesSent'][i] = 'fs_1807'
    elif(db['FlowBytesSent'][i] >= 1810):
        df['FlowBytesSent'][i] = 'fsless'
    else:
        db['FlowBytesSent'][i] = 'fs_more'
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-47-67f2d0160206> in <module>
      3         db['FlowBytesSent'][i] = 'fs_1807'
      4     elif(db['FlowBytesSent'][i] >= 1810):
----> 5         df['FlowBytesSent'][i] = 'fsless'
      6     else:
      7         db['FlowBytesSent'][i] = 'fs_more'

NameError: name 'df' is not defined
```

In [38]:
```
for i in range (0,len(db)):
    if(db['FlowBytesSent'][i] == 1807):
        db['FlowBytesSent'][i] = 'fs_1807'
    elif(db['FlowBytesSent'][i] >= 1810):
        db['FlowBytesSent'][i] = 'fsless'
    else:
        db['FlowBytesSent'][i] = 'fs_more'
```

```
<ipython-input-38-4a8b68076bc7>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

jupyter 18BIT0276_Yuvraj (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

Run Code

```
rsus-a-copy
  db['FlowBytesSent'][i] = 'fs_1807'
```

In [39]:
```
fs_dummy = pd.get_dummies(db['FlowBytesSent'])
```

In [40]:
```
destination_port_dummy.head()
```

Out[40]:

	Wrong_port	destination_port_443
251170	0	1
250984	0	1
90316	0	1
6366	0	1
135318	0	1

In [41]:
```
db_create = pd.concat([db_create,fs_dummy],axis=1)
```

In [42]:
```
fs_r = db[["FlowSentRate"]]
```

In [43]:
```
fs_r = sc.fit_transform(fs_r)
```

In [44]:
```
fs_r = pd.DataFrame(fs_r)
```

In [45]:
```
fs_r = fs_r.rename(columns = {0:"FlowSentRate"})
```

In [46]:
```
db_create = pd.concat([db_create,fs_r],axis=1)
```

jupyter 18BIT0276_Yuvraj (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  db["FlowBytesReceived"][115] = 0
```

In [48]:
```
frec = db[['FlowBytesReceived']]
```

In [49]:
```
frec.mean()
```

Out[49]:
```
FlowBytesReceived    67362.488946
dtype: float64
```

In [50]:
```
frec.median()
```

Out[50]:
```
FlowBytesReceived    4896.0
dtype: float64
```

In [51]:
```
for i in range (0,len(db)):
    if(isinstance(db['FlowBytesReceived'][i], str)):
        db['FlowBytesReceived'][i] = 0

    if(db['FlowBytesReceived'][i] == 98):
        db['FlowBytesReceived'][i] = 'fr_98'
    elif(db['FlowBytesReceived'][i] > 4900):
        db['FlowBytesReceived'][i] = 'fr_more'
    else:
        db['FlowBytesReceived'][i] = 'fr_less'
```

```
<ipython-input-51-b979bb7d5b2d>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

jupyter 18BIT0276_Yuvraj (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [52]:
```
frec_dummies = pd.get_dummies(db['FlowBytesReceived'])
```

In [64]:
```
db_new = pd.concat([db_new,frec_dummies],axis=1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-64-34d07a978212> in <module>
----> 1 db_new = pd.concat([db_new,frec_dummies],axis=1)

NameError: name 'db_new' is not defined
```

In [53]:
```
bd_create = pd.concat([db_create,frec_dummies],axis=1)
```

In [54]:
```
frc = db[["FlowReceivedRate"]]
frc = sc.fit_transform(frc)
frc = pd.DataFrame(frc)
```

In [55]:
```
frc = frc.rename(columns = {0:"FlowRecvRate"})
```

In [56]:
```
db_create = pd.concat([db_create,frc],axis=1)
```

In [57]:
```
plv = db[["PacketLengthVariance"]]
plv = sc.fit_transform(plv)
plv = pd.DataFrame(plv)
```

In [58]:
```
plv = plv.rename(columns = {0:"Pcket_lenVar"})
```

In [59]:
```
db_create = pd.concat([db_create,plv],axis=1)
```

In [60]:
```
plsd = db[["PacketLengthStandardDeviation"]]
```

jupyter 18BIT0276_Yuvraj (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [62]:
```
db_create = pd.concat([db_create,plsd],axis=1)
```

In [63]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketLengthMean'][i], str)):
        df['PacketLengthMean'][i] = 0
```

In [64]:
```
plm = db[['PacketLengthMean']]
plm = sc.fit_transform(plm)
plm = pd.DataFrame(plm)
```

In [65]:
```
plm = plm.rename(columns = {0:"Length_mean"})
```

In [66]:
```
db_create = pd.concat([db_create,plm],axis=1)
```

In [67]:
```
db_temp_half = db_create
```

In [68]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketLengthMedian'][i], str)):
        db['PacketLengthMedian'][i] = 0
```

In [81]:
```
pl_med = df[['PacketLengthMedian']]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-81-c6e68f44c154> in <module>
----> 1 pl_med = df[['PacketLengthMedian']]
```

jupyter 18BIT0276_Yuvraj (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Py

Run Code

In [69]:
```
pl_med = db[['PacketLengthMedian']]
```

In [70]:
```
pl_med = sc.fit_transform(pl_med)
pl_med = pd.DataFrame(pl_med)
```

In [71]:
```
pl_med = pl_med.rename(columns = {0:"L_median"})
```

In [72]:
```
db_create = pd.concat([db_create,pl_med],axis=1)
```

In [73]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketLengthMode'][i], str)):
        db['PacketLengthMode'][i] = 0
```

In [74]:
```
pl_mode = db[['PacketLengthMode']]
pl_mode = sc.fit_transform(pl_mode)
pl_mode = pd.DataFrame(pl_mode)
```

In [75]:
```
pl_mode = pl_mode.rename(columns = {0:"L_mode"})
```

In [76]:
```
db_create = pd.concat([db_create,pl_mode],axis=1)
```

In [77]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketLengthSkewFromMedian'][i], str)):
        db['PacketLengthSkewFromMedian'][i] = 0
```

In [78]:
```
s_med = db[['PacketLengthSkewFromMedian']]
s_med = sc.fit_transform(s_med)
s_med = pd.DataFrame(s_med)
```

jupyter 18BIT0276_Yuvraj (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Pytho

Run Code

In [79]:
```
s_med = s_med.rename(columns = {0:"s_median"})
```

In [80]:
```
db_create = pd.concat([db_create,s_med],axis=1)
```

In [81]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketLengthSkewFromMode'][i], str)):
        db['PacketLengthSkewFromMode'][i] = 0
```

In [82]:
```
s_mode = db[['PacketLengthSkewFromMode']]
s_mode = sc.fit_transform(s_mode)
s_mode = pd.DataFrame(s_mode)
```

In [83]:
```
s_mode = s_mode.rename(columns = {0:"s_mode"})
```

In [84]:
```
db_create = pd.concat([db_create,s_mode],axis=1)
```

In [85]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketLengthCoefficientofVariation'][i], str)):
        db['PacketLengthCoefficientofVariation'][i] = 0
```

In [86]:
```
plcv = db[['PacketLengthCoefficientofVariation']]
plcv = sc.fit_transform(plcv)
plcv = pd.DataFrame(plcv)
```

In [87]:
```
plcv = plcv.rename(columns = {0:"PLCV"})
```

In [88]:
```
db_create = pd.concat([db_create,plcv],axis=1)
```

jupyter 18BIT0276_Yuvraj (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

Run Code

```
        db['PacketTimeVariance'][i] = 0
```

In [90]:
```
ptv = db[['PacketTimeVariance']]
ptv = sc.fit_transform(ptv)
ptv = pd.DataFrame(ptv)
```

In [91]:
```
ptv = ptv.rename(columns = {0:"PTV"})
```

In [92]:
```
db_create = pd.concat([db_create,ptv],axis=1)
```

In [93]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketTimeStandardDeviation'][i], str)):
        db['PacketTimeStandardDeviatione'][i] = 0
```

In [94]:
```
pt_std = db[['PacketTimeStandardDeviation']]
pt_std = sc.fit_transform(pt_std)
pt_std = pd.DataFrame(pt_std)
```

In [95]:
```
pt_std = pt_std.rename(columns = {0:"PT_std"})
db_create = pd.concat([db_create,pt_std],axis=1)
```

In [96]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketTimeMean'][i], str)):
        db['PacketTimeMean'][i] = 0

pt_mean = db[['PacketTimeMean']]
pt_mean = sc.fit_transform(pt_mean)
pt_mean = pd.DataFrame(pt_mean)
```

In [97]:
```
pt_mean = pt_mean.rename(columns = {0:"PT_mean"})
db_create = pd.concat([db_create,pt_mean],axis=1)
```

jupyter 18BIT0276_Yuvraj (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [98]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketTimeMedian'][i], str)):
        db['PacketTimeMedian'][i] = 0

pt_med = db[['PacketTimeMedian']]
pt_med = sc.fit_transform(pt_med)
pt_med = pd.DataFrame(pt_med)
```

In [99]:
```
pt_med = pt_med.rename(columns = {0:"PT_median"})
db_create = pd.concat([db_create,pt_med],axis=1)
```

In [100]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketTimeMode'][i], str)):
        db['PacketTimeMode'][i] = 0

packt_time_mode = db[['PacketTimeMode']]
packt_time_mode = sc.fit_transform(packt_time_mode)
packt_time_mode = pd.DataFrame(packt_time_mode)
```

In [101]:
```
packt_time_mode = packt_time_mode.rename(columns = {0:"Packt_Time_mode"})
db_create = pd.concat([db_create,packt_time_mode],axis=1)
```

In [102]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketTimeSkewFromMedian'][i], str)):
        db['PacketTimeSkewFromMedian'][i] = 0

s_med_time = db[['PacketTimeSkewFromMedian']]
s_med_time = sc.fit_transform(s_med_time)
s_med_time = pd.DataFrame(s_med_time)
```

In [103]:
```
s_med_time = s_med_time.rename(columns = {0:"s_med_time"})
dfb_create = pd.concat([db_create,s_med_time],axis=1)
```

jupyter 18BIT0276_Yuvraj (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Pytho

Run Code

In [104]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketTimeSkewFromMode'][i], str)):
        db['PacketTimeSkewFromMode'][i] = 0

s_mode_time = db[['PacketTimeSkewFromMode']]
s_mode_time = sc.fit_transform(s_mode_time)
s_mode_time = pd.DataFrame(s_mode_time)
```

In [105]:
```
s_mode_time = s_mode_time.rename(columns = {0:"s_mode_time"})
```

In [106]:
```
db_create = pd.concat([db_create,s_mode_time],axis=1)
```

In [107]:
```
for i in range (0,len(db)):
    if(isinstance(db['PacketTimeCoefficientofVariation'][i], str)):
        db['PacketTimeCoefficientofVariation'][i] = 0

pkt_cov_time = db[['PacketTimeCoefficientofVariation']]
pkt_cov_time = sc.fit_transform(pkt_cov_time)
pkt_cov_time = pd.DataFrame(pkt_cov_time)
```

In [108]:
```
pjt_cov_time = pkt_cov_time.rename(columns = {0:"pkt_cov_time"})
db_create = pd.concat([db_create,pkt_cov_time],axis=1)
```

In [109]:
```
for i in range (0,len(db)):
    if(isinstance(db['ResponseTimeTimeVariance'][i], str)):
        db['ResponseTimeTimeVariance'][i] = 0

r_time_var = db[['ResponseTimeTimeVariance']]
r_time_var = sc.fit_transform(r_time_var)
r_time_var = pd.DataFrame(r_time_var)
```

jupyter 18BIT0276_Yuvraj (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [110]:
```
r_time_var = r_time_var.rename(columns = {0:"r_time_var"})
db_create = pd.concat([db_create,r_time_var],axis=1)
```

In [111]:
```
for i in range (0,len(db)):
    if(isinstance(db['ResponseTimeTimeStandardDeviation'][i], str)):
        db['ResponseTimeTimeStandardDeviation'][i] = 0

r_time_std = db[['ResponseTimeTimeStandardDeviation']]
r_time_std = sc.fit_transform(r_time_std)
r_time_std = pd.DataFrame(r_time_std)
```

In [112]:
```
r_time_std = r_time_std.rename(columns = {0:"r_time_std"})
```

In [113]:
```
db_create = pd.concat([db_create,r_time_std],axis=1)
```

In [114]:
```
for i in range (0,len(db)):
    if(isinstance(db['ResponseTimeTimeMean'][i], str)):
        db['ResponseTimeTimeMean'][i] = 0
```

In [115]:
```
r_time_mean = db[['ResponseTimeTimeMean']]
r_time_mean = sc.fit_transform(r_time_mean)
r_time_mean = pd.DataFrame(r_time_mean)
```

In [116]:
```
r_time_mean = r_time_mean.rename(columns = {0:"r_time_mean"})
```

In [117]:
```
db_create = pd.concat([db_create,r_time_mean],axis=1)
```

In [118]:
```
for i in range (0,len(db)):
    if(isinstance(db['ResponseTimeTimeMedian'][i], str)):
        db['ResponseTimeTimeMedian'][i] = 0

r_time_med = db[['ResponseTimeTimeMedian']]
```

```
r_time_med = sc.fit_transform(r_time_med)
r_time_med = pd.DataFrame(r_time_med)
```

```
<ipython-input-118-4b739f030769>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
rsus-a-copy
  db['ResponseTimeTimeMedian'][i] = 0
```

In [119]:
```
r_time_med = r_time_med.rename(columns = {0:"r_time_med"})
db_create = pd.concat([db_create,r_time_med],axis=1)
```

In [120]:
```
for i in range (0,len(db)):
    if(isinstance(db['ResponseTimeTimeMode'][i], str)):
        db['ResponseTimeTimeMode'][i] = 0

r_time_mode = db[['ResponseTimeTimeMode']]
r_time_mode = sc.fit_transform(r_time_mode)
r_time_mode = pd.DataFrame(r_time_mode)
```

In [121]:
```
r_time_mode = r_time_mode.rename(columns = {0:"r_time_mode"})
db_create = pd.concat([db_create,r_time_mode],axis=1)
```

In [122]:
```
for i in range (0,len(db)):
    if(isinstance(db['ResponseTimeTimeSkewFromMedian'][i], str)):
        db['ResponseTimeTimeSkewFromMedian'][i] = 0

r_s_med_time = db[['ResponseTimeTimeSkewFromMedian']]
r_s_med_time = sc.fit_transform(r_s_med_time)
r_s_med_time = pd.DataFrame(r_s_med_time)
```

```
<ipython-input-122-1abd97a78dfc>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

jupyter 18BIT0276_Yuvraj (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [123]:
```
r_s_med_time = r_s_med_time.rename(columns = {0:"r_s_med_time"})
db_create = pd.concat([db_create,r_s_med_time],axis=1)
```

In [124]:
```
for i in range (0,len(db)):
    if(isinstance(db['ResponseTimeTimeSkewFromMode'][i], str)):
        db['ResponseTimeTimeSkewFromMode'][i] = 0
```

In [125]:
```
r_s_mode_time = db[['ResponseTimeTimeSkewFromMode']]
r_s_mode_time = sc.fit_transform(r_s_mode_time)
r_s_mode_time = pd.DataFrame(r_s_mode_time)
```

In [126]:
```
r_s_mode_time = r_s_mode_time.rename(columns = {0:"r_s_mode_time"})
db_create = pd.concat([db_create,r_s_mode_time],axis=1)
```

In [127]:
```
for i in range (0,len(db)):
    if(isinstance(db['ResponseTimeTimeCoefficientofVariation'][i], str)):
        db['ResponseTimeTimeCoefficientofVariation'][i] = 0
```

In [128]:
```
r_c_time = db[['ResponseTimeTimeCoefficientofVariation']]
r_c_time = sc.fit_transform(r_c_time)
r_c_time = pd.DataFrame(r_c_time)
```

In [129]:
```
r_c_time = r_c_time.rename(columns = {0:"r_c_time"})
db_create = pd.concat([db_create,r_c_time],axis=1)
```

In [142]:
```
labels = pd.get_dummies(db['Label'])
```

In [143]:
```
db_create = pd.concat([db_create,labels],axis=1)
```

jupyter 18BIT0276_Yuvraj (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [146]:
```
db_create.drop('Malicious' , axis=1)
```

Out[146]:

	ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_130_1_8	...	0	r_time_var	r_time_std	r_time_mean	r_time_med	r_
0	0	0	0	0	0	0	1	0	0	0	...	0.754077	-0.178941	-0.244670	-0.239715	-0.221598	
1	0	0	0	0	0	0	1	0	0	0	...	-0.814219	-0.132047	0.277059	-0.183655	-0.225091	
2	0	0	0	0	0	0	1	0	0	0	...	-0.817715	-0.178940	-0.244316	-0.238336	-0.221617	
3	0	0	0	0	0	0	1	0	0	0	...	-0.316040	-0.178946	-0.250146	-0.238243	-0.221523	
4	0	0	0	0	0	0	1	0	0	0	...	-0.770578	-0.178937	-0.242955	-0.240378	-0.221754	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
269638	1	0	0	0	0	0	0	0	0	0	...	-0.997728	-0.178940	-0.244337	-0.240169	-0.221636	
269639	1	0	0	0	0	0	0	0	0	0	...	0.947832	-0.178940	-0.244243	-0.240186	-0.221621	
269640	1	0	0	0	0	0	0	0	0	0	...	-0.858274	-0.178946	-0.250175	-0.243940	-0.226451	
269641	1	0	0	0	0	0	0	0	0	0	...	0.898332	-0.178940	-0.244267	-0.240184	-0.221634	
269642	1	0	0	0	0	0	0	0	0	0	...	-0.588891	-0.178785	-0.219341	-0.231216	-0.218044	

269643 rows × 45 columns

In [147]:
```
db_create['Benign'] = db_create['Benign'].replace(['1'],'Benign')
db_create['Benign'] = db_create['Benign'].replace (['0'],'Malicious')
```

In [148]:
```
db_create.head()
```

Out[148]:

	ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_130_1_8	...	r_time_var	r_time_std	r_time_mean	r_time_med	r_time_mode	r_s_
0	0	0	0	0	0	0	1	0	0	0	...	-0.178941	-0.244670	-0.239715	-0.221598	-0.159623	
1	0	0	0	0	0	0	1	0	0	0	...	-0.132047	0.277059	-0.183655	-0.225091	-0.159603	

jupyter 18BIT0276_Yuvraj (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [150]:
```
db_create.drop('Malicious' , inplace=True , axis=1)
```

In [151]:
```
db_create.head()
```

Out[151]:

	ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_130_1_8	...	0	r_time_var	r_time_std	r_time_mean	r_time_med	r_time_
0	0	0	0	0	0	0	1	0	0	0	...	0.754077	-0.178941	-0.244670	-0.239715	-0.221598	-0.1
1	0	0	0	0	0	0	1	0	0	0	...	-0.814219	-0.132047	0.277059	-0.183655	-0.225091	-0.1
2	0	0	0	0	0	0	1	0	0	0	...	-0.817715	-0.178940	-0.244316	-0.238336	-0.221617	-0.1
3	0	0	0	0	0	0	1	0	0	0	...	-0.316040	-0.178946	-0.250146	-0.238243	-0.221523	-0.1
4	0	0	0	0	0	0	1	0	0	0	...	-0.770578	-0.178937	-0.242955	-0.240378	-0.221754	-0.1

5 rows × 45 columns

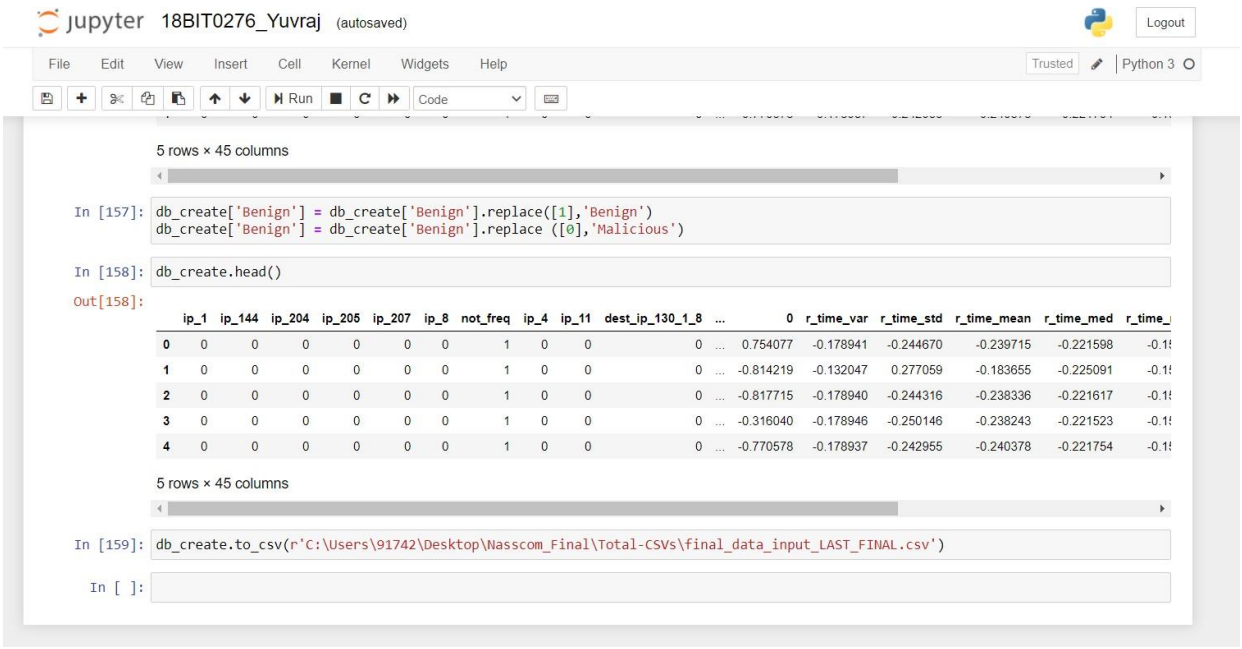
In [152]:
```
db_create.to_csv(r'C:\Users\91742\Desktop\Nasscom_Final\Total-CSVs\final_data_input_FINAL_Semifinal.csv')
```

In [153]:
```
db_create['Benign'] = db_create['Benign'].replace(['1'],'Benign')
db_create['Benign'] = db_create['Benign'].replace (['0'],'Malicious')
```

In [154]:
```
db_create.head()
```

Out[154]:

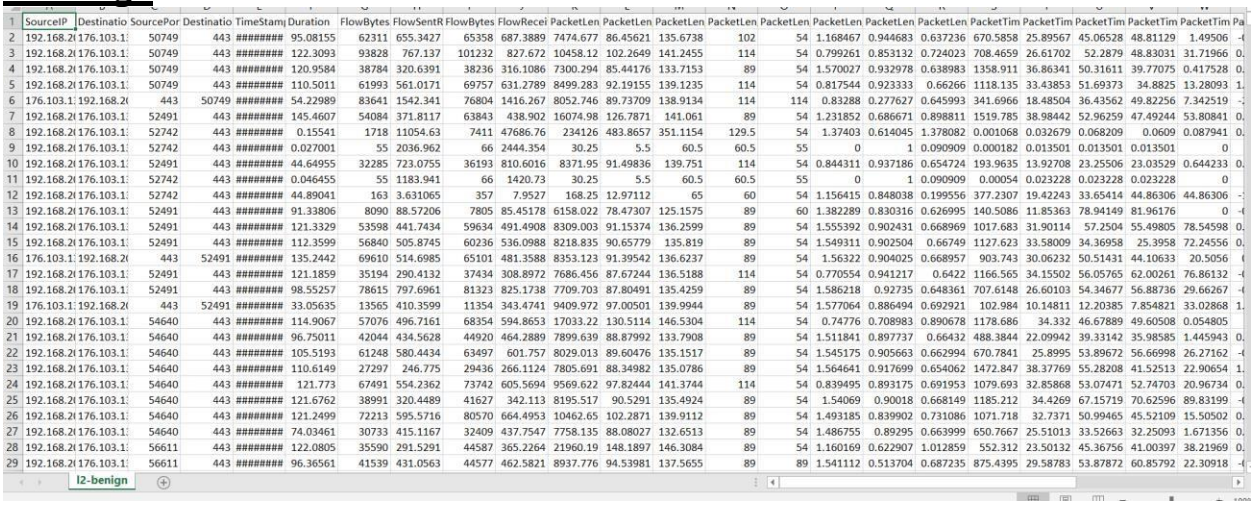
	ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_130_1_8	...	0	r_time_var	r_time_std	r_time_mean	r_time_med	r_time_
0	0	0	0	0	0	0	1	0	0	0	...	0.754077	-0.178941	-0.244670	-0.239715	-0.221598	-0.1
1	0	0	0	0	0	0	1	0	0	0	...	-0.814219	-0.132047	0.277059	-0.183655	-0.225091	-0.1
2	0	0	0	0	0	0	1	0	0	0	...	-0.817715	-0.178940	-0.244316	-0.238336	-0.221617	-0.1
3	0	0	0	0	0	0	1	0	0	0	...	-0.316040	-0.178946	-0.250146	-0.238243	-0.221523	-0.1



As I had 269643 records in DoH and only 19807 records in benign, I tried Under Sampling and went Ahead with 19807 Records and upon Random Sampling, Each Dataset Picked 19807 records to form 39614 Record Data Together.

DATASETS:

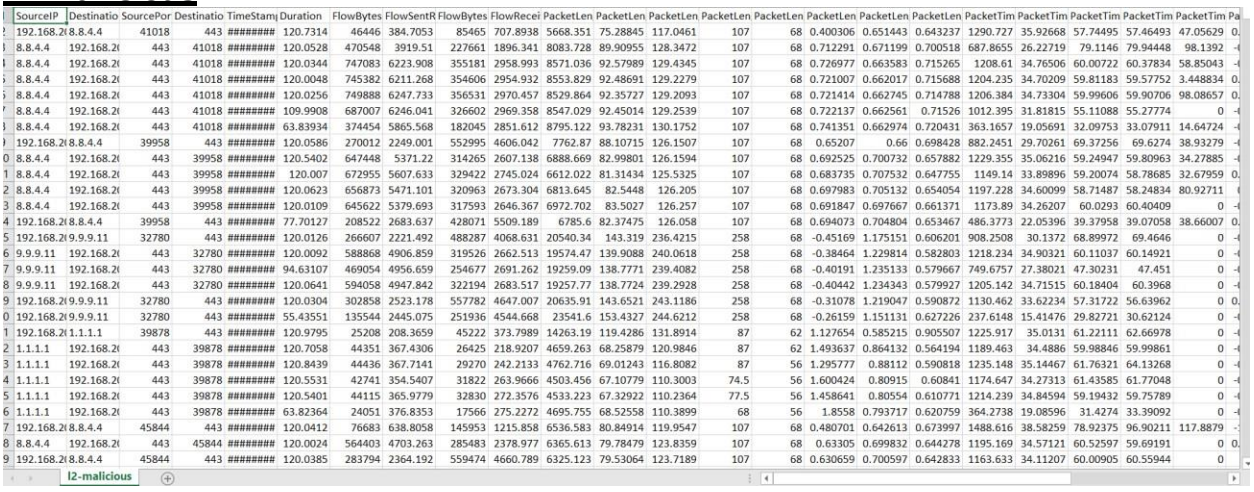
l2-benign



SourceIP	Destinatio	SourcePor	Destinatio	TimeStam	Duration	FlowBytes	FlowSentR	FlowBytes	FlowRecei	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketTim	PacketTim	PacketTim	PacketTim	PacketTim	Pa
192.168.2(	176.103.1:	50749	443	########	95.08155	62311	655.3427	65358	687.3889	7474.677	86.45621	135.6738	102	54	1.168467	0.944683	0.637236	670.5858	25.89567	45.06528	48.81129	1.49506	-(
192.168.2(	176.103.1:	50749	443	########	122.3093	93828	767.137	101232	827.672	10458.12	102.2649	141.2455	114	54	0.799261	0.853132	0.724023	708.4659	26.61702	52.2879	48.83031	31.71966	0.
192.168.2(	176.103.1:	50749	443	########	120.9584	38784	320.6391	38236	316.1086	7300.294	85.44176	133.7153	89	54	1.570027	0.932978	0.638983	1358.911	36.86341	50.31611	39.77075	0.417528	0.
192.168.2(	176.103.1:	50749	443	########	110.5011	61993	561.0171	69757	631.2789	8499.283	92.19155	139.1235	114	54	0.817544	0.923333	0.66266	1118.135	33.43853	51.69373	34.8825	13.28093	1.
176.103.1	192.168.2(	443	50749	########	54.22989	83641	1542.341	76804	1416.267	8052.746	89.73709	138.9134	114	114	0.83288	0.277627	0.645993	341.6966	18.48504	36.43562	49.82256	7.342519	-.
192.168.2(	176.103.1:	52491	443	########	145.4607	54084	371.8117	63843	438.902	16074.98	126.7871	141.061	89	54	1.231852	0.686671	0.898811	1519.785	38.98442	52.96259	47.49244	53.80841	0.
192.168.2(	176.103.1:	52742	443	########	0.15541	1718	11054.63	7411	47686.76	234126	483.8657	351.1154	129.5	54	1.37403	0.614045	1.378082	0.001068	0.032679	0.068209	0.0609	0.087941	0.
192.168.2(	176.103.1:	52742	443	########	0.027001	55	2036.962	66	2444.354	30.25	5.5	60.5	60.5	55	0	1	0.090909	0.000182	0.013501	0.013501	0.013501	0	
192.168.2(	176.103.1:	52491	443	########	44.64955	32285	723.0755	36193	810.6016	8371.95	91.49836	139.751	114	54	0.844311	0.937186	0.654724	193.9635	13.92708	23.25506	23.03529	0.644233	0.
192.168.2(	176.103.1:	52742	443	########	0.046455	55	1183.941	66	1420.73	30.25	5.5	60.5	60.5	55	0	1	0.090909	0.00054	0.023228	0.023228	0.023228	0	
192.168.2(	176.103.1:	52742	443	########	44.89041	163	3.631065	357	7.9527	168.25	12.97112	65	60	54	1.156415	0.848038	0.199556	377.2307	19.42243	33.65414	44.86306	44.86306	-:
192.168.2(	176.103.1:	52491	443	########	91.33806	8090	88.57206	7805	85.45178	6158.022	78.47307	125.1575	89	60	1.382289	0.830316	0.626995	140.5086	11.85363	78.94149	81.96176	0	-(
192.168.2(	176.103.1:	52491	443	########	121.3329	53598	441.7434	59634	491.4908	8309.003	91.15374	136.2599	89	54	1.555392	0.902431	0.668969	1017.683	31.90114	57.2504	55.49805	78.54598	0.
192.168.2(	176.103.1:	52491	443	########	112.3599	56840	505.8745	60236	536.0988	8218.835	90.65779	135.819	89	54	1.549311	0.902504	0.66749	1127.623	33.58009	34.36958	25.3958	72.24556	0.
176.103.1	192.168.2(	443	52491	########	135.2442	69610	514.6985	65101	481.3588	8353.123	91.39542	136.6237	89	54	1.56322	0.904025	0.668957	903.743	30.06232	50.51431	44.10633	20.5056	(
192.168.2(	176.103.1:	52491	443	########	121.1859	35194	290.4132	37434	308.8972	7686.456	87.67244	136.5188	114	54	0.770554	0.941217	0.6422	1166.565	34.15502	56.05765	62.00261	76.86132	-(
192.168.2(	176.103.1:	52491	443	########	98.55257	78615	797.6961	81323	825.1738	7709.703	87.80491	135.4259	89	54	1.586218	0.92735	0.648361	707.6148	26.60103	54.34677	56.88736	29.66267	-(
176.103.1	192.168.2(	443	52491	########	33.05635	13565	410.3599	11354	343.4741	9409.972	97.00501	139.9944	89	54	1.577064	0.886494	0.692921	102.984	10.14811	12.20385	7.854821	33.02868	1.
192.168.2(	176.103.1:	54640	443	########	114.9067	57076	496.7161	68354	594.8653	17033.22	130.5114	146.5304	114	54	0.74776	0.708983	0.890678	1178.686	34.332	46.67889	49.60508	0.054805	
192.168.2(	176.103.1:	54640	443	########	96.75011	42044	434.5628	44920	464.2889	7899.639	88.87992	133.7908	89	54	1.511841	0.897737	0.66432	488.3844	22.09942	39.33142	35.98585	1.445943	0.
192.168.2(	176.103.1:	54640	443	########	105.5193	61248	580.4434	63497	601.757	8029.013	89.60476	135.1517	89	54	1.545175	0.905663	0.662994	670.7841	25.8995	53.89672	56.66998	26.27162	-(
192.168.2(	176.103.1:	54640	443	########	110.6149	27297	246.775	29436	266.1124	7805.691	88.34982	135.0786	89	54	1.564641	0.917699	0.654062	1472.847	38.37769	55.28208	41.52513	22.90654	1.
192.168.2(	176.103.1:	54640	443	########	121.773	67491	554.2362	73742	605.5694	9569.622	97.82444	141.3744	114	54	0.839495	0.893175	0.691953	1079.693	32.85868	53.07471	52.74703	20.96734	0.
192.168.2(	176.103.1:	54640	443	########	121.6762	38991	320.4489	41627	342.113	8195.517	90.5291	135.4924	89	54	1.54069	0.90018	0.668149	1185.212	34.4269	67.15719	70.62596	89.83199	-(
192.168.2(	176.103.1:	54640	443	########	121.2499	72213	595.5716	80570	664.4953	10462.65	102.2871	139.9112	89	54	1.493185	0.839902	0.731086	1071.718	32.7371	50.99465	45.52109	15.50502	0.
192.168.2(	176.103.1:	54640	443	########	74.03461	30733	415.1167	32409	437.7547	7758.135	88.08027	132.6513	89	54	1.486755	0.89295	0.663999	650.7667	25.51013	33.52663	32.25093	1.671356	0.
192.168.2(	176.103.1:	56611	443	########	122.0805	35590	291.5291	44587	365.2264	21960.19	148.1897	146.3084	89	54	1.160169	0.622907	1.012859	552.312	23.50132	45.36756	41.00397	38.21969	0.
192.168.2(	176.103.1:	56611	443	########	96.36561	41539	431.0563	44577	462.5821	8937.776	94.53981	137.5655	89	89	1.541112	0.513704	0.687235	875.4395	29.58783	53.87872	60.85792	22.30918	-(

l2-benign

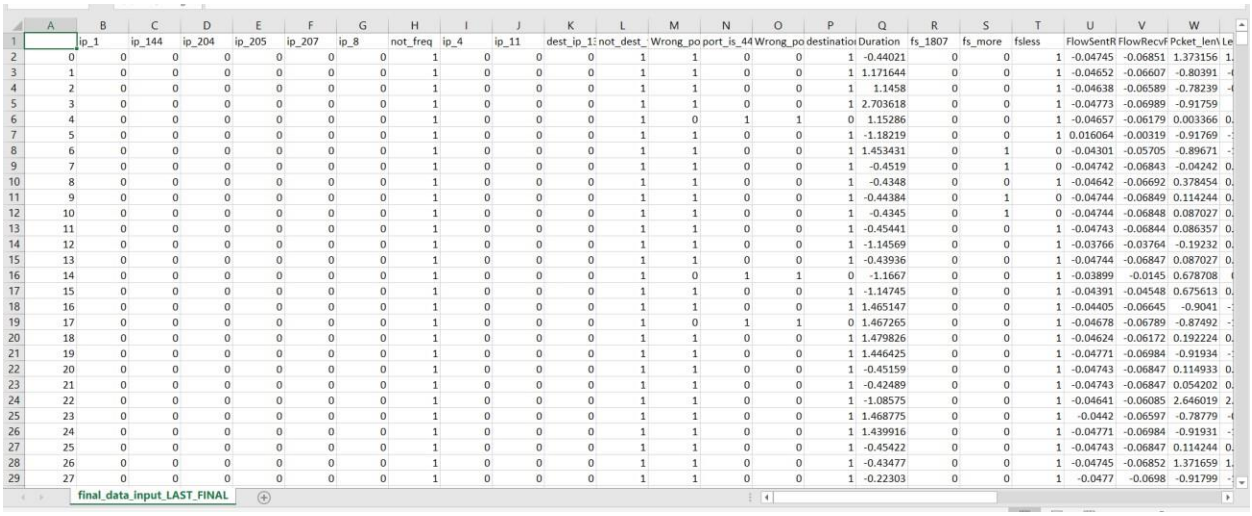
l2-malicious



SourceIP	Destinatio	SourcePor	Destinatio	TimeStam	Duration	FlowBytes	FlowSentR	FlowBytes	FlowRecei	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketLen	PacketTim	PacketTim	PacketTim	PacketTim	PacketTim	Pa
192.168.2(	8.8.4.4	41018	443	########	120.7314	46446	384.7053	85465	707.8938	5668.351	75.28845	117.0461	107	68	0.400306	0.651443	0.643237	1290.727	35.92668	57.74495	57.46493	47.05629	0.
8.8.4.4	192.168.2(	443	41018	########	120.0528	470548	3919.51	227661	1896.341	8083.728	89.90955	128.3472	107	68	0.712291	0.671199	0.700518	687.8655	26.22719	79.1146	79.94448	98.1392	-(
8.8.4.4	192.168.2(	443	41018	########	120.0344	747083	6223.908	355181	2958.993	8571.036	92.57989	129.4345	107	68	0.726977	0.663583	0.715265	1208.61	34.76506	60.00722	60.37834	58.85043	-(
8.8.4.4	192.168.2(	443	41018	########	120.0048	745382	6211.268	354606	2954.932	8553.829	92.48691	129.2279	107	68	0.721007	0.662017	0.715688	1204.235	34.70209	59.81183	59.57752	3.448834	0.
8.8.4.4	192.168.2(	443	41018	########	120.0256	749888	6247.733	356531	2970.457	8529.864	92.35727	129.2093	107	68	0.721414	0.662745	0.714788	1206.384	34.73304	59.99606	59.90706	98.08657	0.
8.8.4.4	192.168.2(	443	41018	########	109.9908	687007	6246.041	326602	2969.358	8547.029	92.45014	129.2539	107	68	0.722137	0.662561	0.71526	1012.395	31.81815	55.11088	55.27774	0	-(
8.8.4.4	192.168.2(	443	41018	########	63.83934	374454	5865.568	182045	2851.612	8795.122	93.78231	130.1752	107	68	0.741351	0.662974	0.720431	363.1657	19.05691	32.09753	33.07911	14.64724	-(
192.168.2(	8.8.4.4	39958	443	########	120.0586	270012	2249.001	552995	4606.042	7762.87	88.10715	126.1507	107	68	0.65207	0.66	0.698428	882.2451	29.70261	69.37256	69.6274	38.93279	-(
8.8.4.4	192.168.2(	443	39958	########	120.5402	647448	5371.22	314265	2607.138	6888.669	82.99801	126.1594	107	68	0.692525	0.700732	0.657882	1229.355	35.06216	59.24947	59.80963	34.27885	-(
8.8.4.4	192.168.2(	443	39958	########	120.007	672955	5607.633	329422	2745.024	6612.022	81.31434	125.5325	107	68	0.683735	0.707532	0.647755	1149.14	33.89896	59.20074	58.78685	32.67959	0.
8.8.4.4	192.168.2(	443	39958	########	120.0623	656873	5471.101	320963	2673.304	6813.645	82.5448	126.205	107	68	0.697983	0.705132	0.654054	1197.228	34.60099	58.71487	58.24834	80.92711	(
8.8.4.4	192.168.2(	443	39958	########	120.0109	645622	5379.693	317593	2646.367	6972.702	83.5027	126.257	107	68	0.691847	0.697667	0.661371	1173.89	34.26207	60.0293	60.40409	0	-(
192.168.2(	8.8.4.4	39958	443	########	77.70127	208522	2683.637	428071	5509.189	6785.6	82.37475	126.058	107	68	0.694073	0.704804	0.653467	486.3773	22.05396	39.37958	39.07058	38.66007	0.
192.168.2(	9.9.9.11	32780	443	########	120.0126	266607	2221.492	488287	4068.631	20540.34	143.319	236.4215	258	68	-0.45169	1.175151	0.606201	908.2508	30.1372	68.89972	69.4646	0	-(
9.9.9.11	192.168.2(	443	32780	########	120.0092	588868	4906.859	319526	2662.513	19574.47	139.9088	240.0618	258	68	-0.38464	1.229814	0.582803	1218.234	34.90321	60.11037	60.14921	0	-(
9.9.9.11	192.168.2(	443	32780	########	94.63107	469054	4956.659	254677	2691.262	19259.09	138.7771	239.4082	258	68	-0.40191	1.235133	0.579667	749.6757	27.38021	47.30231	47.451	0	-(
9.9.9.11	192.168.2(	443	32780	########	120.0641	594058	4947.842	322194	2683.517	19257.77	138.7724	239.2928	258	68	-0.40442	1.234343	0.579927	1205.142	34.71515	60.18404	60.3968	0	-(
192.168.2(	9.9.9.11	32780	443	########	120.0304	302858	2523.178	557782	4647.007	20635.91	143.6521	243.1186	258	68	-0.31078	1.219047	0.590872	1130.462	33.62234	57.31722	56.63962	0	0.
192.168.2(	9.9.9.11	32780	443	########	55.43551	135544	2445.075	251936	4544.668	23541.6	153.4327	244.6212	258	68	-0.26159	1.151131	0.627226	237.6148	15.41476	29.82721	30.62124	0	-(
192.168.2(	1.1.1.1	39878	443	########	120.9795	25208	208.3659	45222	373.7989	14263.19	119.4286	131.8914	87	62	1.127654	0.585215	0.905507	1225.917	35.0131	61.22111	62.66978	0	-(
1.1.1.1	192.168.2(	443	39878	########	120.7058	44351	367.4306	26425	218.9207	4659.263	68.25879	120.9846	87	62	1.493637	0.864132	0.564194	1189.463	34.4886	59.98846	59.99861	0	-(
1.1.1.1	192.168.2(	443	39878	########	120.8439	44436	367.7141	29270	242.2133	4762.716	69.01243	116.8082	87	56	1.295777	0.88112	0.590818	1235.148	35.14467	61.76321	64.13268	0	-(
1.1.1.1	192.168.2(	443	39878	########	120.5531	42741	354.5407	31822	263.9666	4503.456	67.10779	110.3003	74.5	56	1.600424	0.80915	0.60841	1174.647	34.27313	61.43585	61.77048	0	-(
1.1.1.1	192.168.2(	443	39878	########	120.5401	44115	365.9779	32830	272.3576	4533.223	67.32922	110.2364	77.5	56	1.458641	0.80554	0.610771	1214.239	34.84594	59.19432	59.75789	0	-(
1.1.1.1	192.168.2(	443	39878	########	63.82364	24051	376.8353	17566	275.2272	4695.755	68.52558	110.3899	68	56	1.8558	0.793717	0.620759	364.2738	19.08596	31.4274	33.39092	0	-(
192.168.2(	8.8.4.4	45844	443	########	120.0412	76683	638.8058	145953	1215.858	6536.583	80.84914	119.9547	107	68	0.480701	0.642613	0.673997	1488.616	38.58259	78.92375	96.90211	117.8879	-:
8.8.4.4	192.168.2(	443	45844	########	120.0024	564403	4703.263	285483	2378.977	6365.613	79.78479	123.8359	107	68	0.63305	0.699832	0.644278	1195.169	34.57121	60.52597	59.69191	0	0.
192.168.2(	8.8.4.4	45844	443	########	120.0385	283794	2364.192	559474	4660.789	6325.123	79.53064	123.7189	107	68	0.630659	0.700597	0.642833	1163.633	34.11207	60.00905	60.55944	0	

l2-malicious

Combined Final dataset



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1		ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_1	not_dest_	Wrong_po	port_is_44	Wrong_po	destinatio	Duration	fs_1807	fs_more	fsless	FlowSentR	FlowRecvF	Pcket_len\
2	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.44021	0	0	1	-0.04745	-0.06851	1.373156
3	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.171644	0	0	1	-0.04652	-0.06607	-0.80391
4	2	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.1458	0	0	1	-0.04638	-0.06589	-0.78239
5	3	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	2.703618	0	0	1	-0.04773	-0.06989	-0.91759
6	4	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1.15286	0	0	1	-0.04657	-0.06179	0.003366
7	5	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-1.18219	0	0	1	0.016064	-0.00319	-0.91769
8	6	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.453431	0	1	0	-0.04301	-0.05705	-0.89671
9	7	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.4519	0	1	0	-0.04742	-0.06843	-0.04242
10	8	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.4348	0	0	1	-0.04642	-0.06692	0.378454
11	9	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.44384	0	1	0	-0.04744	-0.06849	0.114244
12	10	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.4345	0	1	0	-0.04744	-0.06848	0.087027
13	11	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.45441	0	0	1	-0.04743	-0.06844	0.086357
14	12	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-1.14569	0	0	1	-0.03766	-0.03764	-0.19232
15	13	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.43936	0	0	1	-0.04744	-0.06847	0.087027
16	14	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	-1.1667	0	0	1	-0.03899	-0.0145	0.678708
17	15	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-1.14745	0	0	1	-0.04391	-0.04548	0.675613
18	16	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.465147	0	0	1	-0.04405	-0.06645	-0.9041
19	17	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1.467265	0	0	1	-0.04678	-0.06789	-0.87492
20	18	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.479826	0	0	1	-0.04624	-0.06172	0.192224
21	19	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.446425	0	0	1	-0.04771	-0.06984	-0.91934
22	20	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.45159	0	0	1	-0.04743	-0.06847	0.114933
23	21	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.42489	0	0	1	-0.04743	-0.06847	0.054202
24	22	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-1.08575	0	0	1	-0.04641	-0.06085	2.646019
25	23	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.468775	0	0	1	-0.0442	-0.06597	-0.78779
26	24	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.439916	0	0	1	-0.04771	-0.06984	-0.91931
27	25	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.45422	0	0	1	-0.04743	-0.06847	0.114244
28	26	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.43477	0	0	1	-0.04745	-0.06852	1.371659
29	27	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.22303	0	0	1	-0.0477	-0.0698	-0.91799

final_data_input_LAST_FINAL

3. TRAINING AND CROSS VALIDATION

IMPORTING NECESSARY LIBRARIES AND PACKAGES ON GOOGLE COLAB

FINAL_CNN_MODEL.ipynb

File Edit View Insert Runtime Tools Help Last edited on October 21

Comment Share

+ Code + Text

Connect Editing

```
import pandas as pd
import numpy as np
from google.colab import drive
#from imblearn.under_sampling import TomekLinks
```

```
[ ] drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] db = pd.read_csv('/content/drive/My Drive/final_data_input_LAST_FINAL.csv')
```

```
[ ] db.head()
```

	Unnamed: 0	ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_130_1_8	not_dest_freq	Wrong_port	port_is_443	Wrong_port.1	destination_port_443	Duration	fs
0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.440208	
1	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.171644	
2	2	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.145800	
3	3	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	2.703618	
4	4	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1.152860	

FINAL_CNN_MODEL.ipynb

File Edit View Insert Runtime Tools Help Last edited on October 21

Comment Share

+ Code + Text

Connect Editing

```
[ ] from keras.preprocessing import sequence
    from keras.models import Sequential
    from keras.layers import Dense, Dropout, Activation, Lambda
    from keras.layers import Embedding
    from keras.layers import Convolution1D,MaxPooling1D, Flatten
    from keras.datasets import imdb
    from keras import backend as K
    from sklearn.model_selection import train_test_split
    import pandas as pd
    from keras.utils.np_utils import to_categorical

    from sklearn.preprocessing import Normalizer
    from keras.models import Sequential
    from keras.layers import Convolution1D, Dense, Dropout, Flatten, MaxPooling1D
    from keras.utils import np_utils
    import numpy as np
    import h5py
    from keras import callbacks
    from keras.layers import LSTM, GRU, SimpleRNN
    from keras.callbacks import CSVLogger
    from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
```

FINAL_CNN_MODEL.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on October 21

Comment Share

+ Code + Text

Connect Editing

```
[ ] from keras.layers import LSTM, GRU, SimpleRNN
    from keras.callbacks import CSVLogger
    from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
```

```
[ ] db = db.iloc[:,1:46]
```

```
[ ] db = db.iloc[:,0:45]
```

```
[ ] db
```

	ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_130_1_8	not_dest_freq	Wrong_port	port_is_443	Wrong_port.1	destination_port_443	Duration	fs_1807
0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.440208	0
1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.171644	0
2	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.145800	0
3	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	2.703618	0
4	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1.152860	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
269638	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	-1.180485	0
269639	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	-0.434267	0
269640	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1.466744	0
269641	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	-0.446174	0
269642	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	-0.853504	0

269643 rows × 45 columns

FINAL_CNN_MODEL.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on October 21

Comment Share

+ Code + Text

Connect Editing

```
[ ] x1 = db.iloc[:,0:44]
```

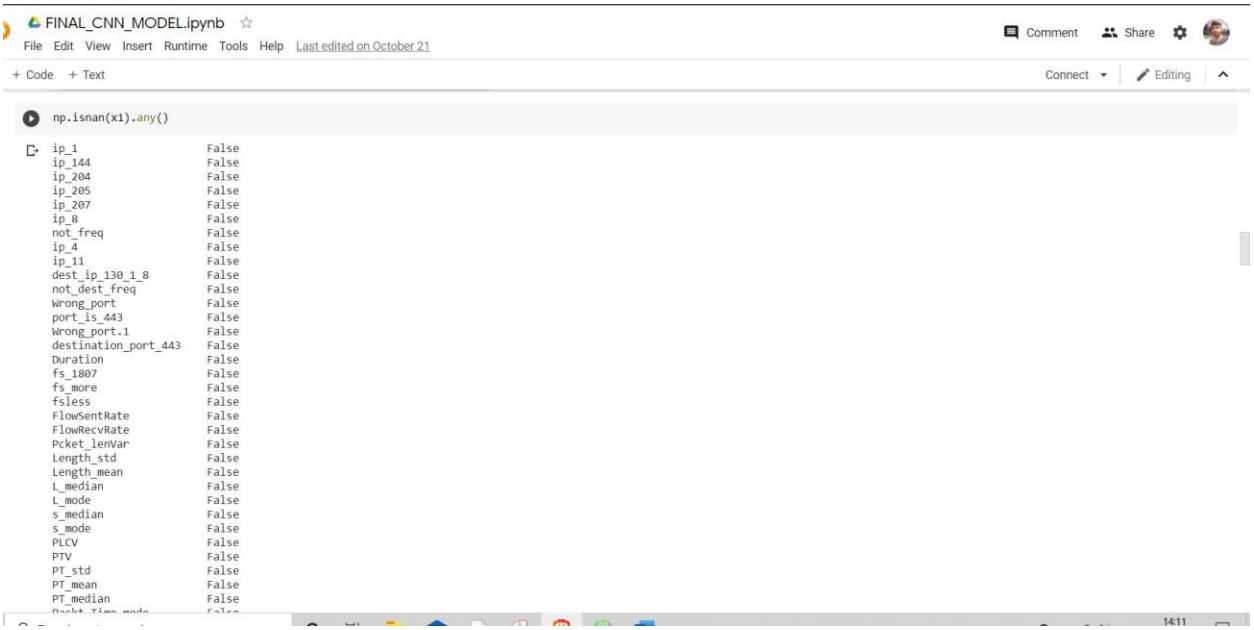
```
[ ] y1 = db.iloc[:,44]
```

```
[ ] x1
```

	ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_130_1_8	not_dest_freq	Wrong_port	port_is_443	Wrong_port.1	destination_port_443	Duration	fs_1807
0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	-0.440208	0
1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.171644	0
2	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1.145800	0
3	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	2.703618	0
4	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1.152860	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
269638	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	-1.180485	0
269639	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	-0.434267	0
269640	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1.466744	0
269641	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	-0.446174	0
269642	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	-0.853504	0

269643 rows × 44 columns

Now I checked any invalid python data type or any number is present in my data which I will be using or if array contains at least one non-numeric value.



Sampling Of the Dataset

Dataset given is highly unbalanced. So, to prevent overfitting, sampling was done. Also since this was network data, therefore creating dummy data might not be feasible in real time. Therefore I preferred undersampling instead of over-sampling. To overcome this, I used technique of random sampling

As I had 249836 records in malicious and only 19807 records in benign, I tried Under Sampling and went Ahead with 19807 Records and upon Random

Sampling, Each Dataset Picked 19807 records to form 39614 Record Data Together.

FINAL_CNN_MODEL.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on October 21

Comment Share

+ Code + Text

Connect Editing

```
[ ] df_c_0 = db[db['Benign'] == 'Malicious']
    df_c_1 = db[db['Benign'] == 'Benign']

[ ] df_c_0_under = df_c_0.sample(c_class_1)

[ ] df_under = pd.concat([df_c_0_under, df_c_1], axis=0)

[ ] print(df_under.Benign.value_counts())

    Benign       19807
    Malicious    19807
    Name: Benign, dtype: int64

[ ] df_Totoal = df_under

[ ] df_Totoal.head()
```

	ip_1	ip_144	ip_204	ip_205	ip_207	ip_8	not_freq	ip_4	ip_11	dest_ip_130_1_8	not_dest_freq	Wrong_port	port_is_443	Wrong_port.1	destination_port_443	Duration	fs_1807
253816	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1.443449	0
131414	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	-1.135324	0
144159	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	-0.440212	0
167495	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1	-1.150709	0
253272	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	1.140970	0

TRAINING AND TESTING ACCURACY

```
[ ]  x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_state=42)
```

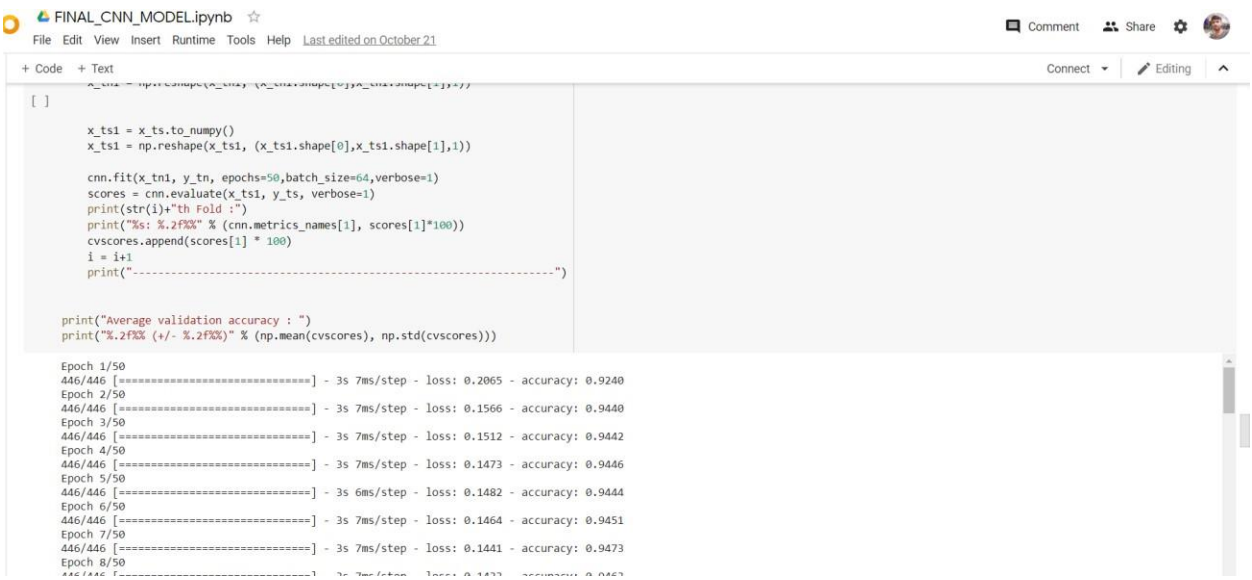
```
[ ]  from sklearn.model_selection import StratifiedKFold
     kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

MODEL CREATION

I also added two layers in my CNN model which are:

Convolution1D and MaxPooling1D

And the optimizer used here is “adam”




```
[ ] Epoch 36/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1385 - accuracy: 0.9480
Epoch 37/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1382 - accuracy: 0.9483
Epoch 38/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1378 - accuracy: 0.9480
Epoch 39/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1375 - accuracy: 0.9484
Epoch 40/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1376 - accuracy: 0.9487
Epoch 41/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1371 - accuracy: 0.9475
Epoch 42/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1371 - accuracy: 0.9486
Epoch 43/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1384 - accuracy: 0.9483
Epoch 44/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1390 - accuracy: 0.9484
Epoch 45/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1373 - accuracy: 0.9485
Epoch 46/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1363 - accuracy: 0.9486
Epoch 47/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1378 - accuracy: 0.9481
Epoch 48/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1376 - accuracy: 0.9488
Epoch 49/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1372 - accuracy: 0.9486
Epoch 50/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1377 - accuracy: 0.9487
100/100 [==============================] - 0s 2ms/step - loss: 0.1527 - accuracy: 0.9382
9th Fold :
accuracy: 93.82%
------------------------------------------------------------------
Average validation accuracy :
94.63% (+/- 0.47%)
```

Validation Accuracy is **: 94.63% (+/-0.47%):-**

Result of 10-fold cross validation with 60 epochs each:

94.63% (+/-0.47%)

Training accuracy: 94.83449459075928

Test accuracy: 94.44654583930969

HYPER PARAMETER TUNING

Parameters for the model like number of hidden units, number of stacked units, choice of optimizer, loss function chosen, value of Dropouts, number of epochs, folds variations and choice of metrics has been tuned to get the final model with better accuracy.

```
[ ] x_tr = x_train.to_numpy()
    x_tr = np.reshape(x_tr, (x_tr.shape[0], x_tr.shape[1],1))

[ ] x_ts = x_test.to_numpy()
    x_ts = np.reshape(x_ts, (x_ts.shape[0], x_ts.shape[1],1))

[ ] y_tr = pd.get_dummies(y_train)
    y_ts = pd.get_dummies(y_test)

[ ] _, train_acc = cnn.evaluate(x_tr, y_tr, verbose=1)
    _, test_acc = cnn.evaluate(x_ts, y_ts, verbose=1)

    991/991 [==============================] - 1s 2ms/step - loss: 0.1343 - accuracy: 0.9483
    248/248 [==============================] - 0s 2ms/step - loss: 0.1407 - accuracy: 0.9445

[ ] print("Training accuracy: "+str(train_acc*100))
    print("Test accuracy: "+str(test_acc*100))

    Training accuracy: 94.83449459075928
    Test accuracy: 94.44654583930969

[ ] y_probs = cnn.predict(x_ts, verbose=1).ravel()
    y_classes = cnn.predict_classes(x_ts, verbose=1)

    248/248 [==============================] - 0s 2ms/step
    WARNING:tensorflow:From <ipython-input-41-ed2a250cf2a8>:2: Sequential.predict_classes (from tensorflow.python.keras.engine.sequential) is deprecated and will be removed after 2021-01-€
    Instructions for updating:
    Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x)
    248/248 [==============================] - 0s 2ms/step
```

PERFORMANCE METRICS (Precision, Recall, F1-Score, Support)

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_ts,y_pred))

                  precision    recall  f1-score   support

               0       0.92      0.97      0.95      3926
               1       0.97      0.92      0.94      3997

       micro avg       0.94      0.94      0.94      7923
       macro avg       0.95      0.94      0.94      7923
    weighted avg       0.95      0.94      0.94      7923
     samples avg       0.94      0.94      0.94      7923

[ ] from sklearn.preprocessing import LabelEncoder

[ ] le = LabelEncoder()

[ ] y_test_le = le.fit_transform(y_test)

[ ] from sklearn.metrics import roc_curve
    fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test_le, y_classes)

[ ] from sklearn.metrics import auc
    auc_keras = auc(fpr_keras, tpr_keras)

[ ] from matplotlib import pyplot
```

NOTE : THE F-SCORE VALUE NEAR 1 (~0.95) PROVES THAT THE MODEL IS BEST TRAINED, NEITHER UNDER-FITTED OR OVER-FITTED.

AUC:

0.9446646242960366

```
print(auc_keras)

0.9446646242960366
```

ROC CURVE:

TOTAL COMPUTATION TIME :

ON LOCAL MACHINE (JUPYTER NOTEBOOK) : ~ 9-10 Hours (TOTAL TIME)

ON GOOGLE COLAB: <1 Hours

Overview of results:

(10 folds and 50 epochs)

```
---------------------------------------
Average validation accuracy :
94.63% (+/- 0.47%)
```

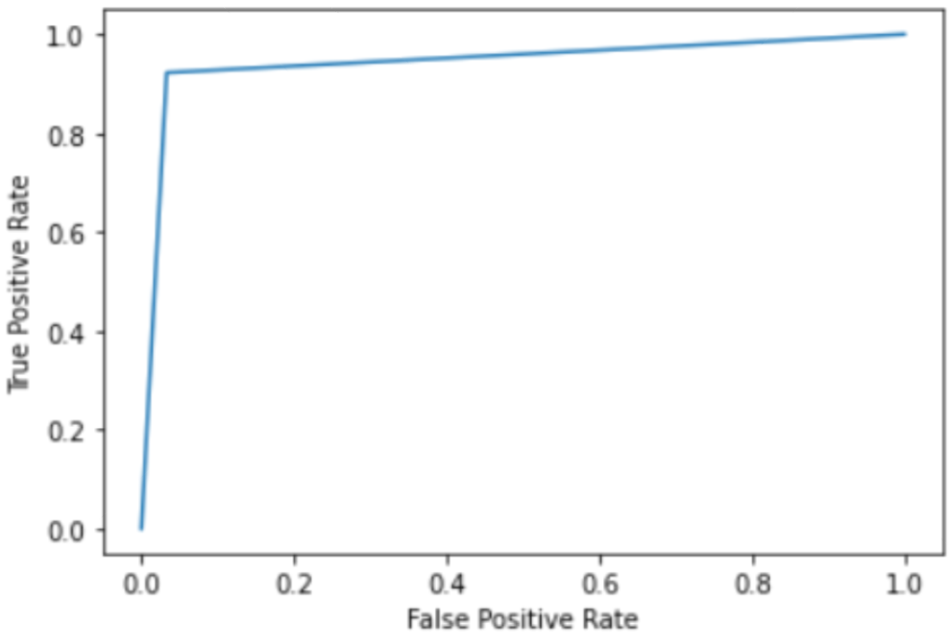
```
Training accuracy: 94.83449459075928
Test accuracy: 94.44654583930969
```

	precision	recall	f1-score	support
0	0.92	0.97	0.95	3926
1	0.97	0.92	0.94	3997
micro avg	0.94	0.94	0.94	7923
macro avg	0.95	0.94	0.94	7923
weighted avg	0.95	0.94	0.94	7923
samples avg	0.94	0.94	0.94	7923

AUC:

```
0.9446646242960366
```

```
pyplot.plot(fpr_keras, tpr_keras)
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.show()
```



Changing the type of optimizer used along with number of epochs:

Opimizer="RMSprop"
Epochs=50

Model:-

```
cvscores = []
i=0
for train, test in kfold.split(x_train, y_train):
    cnn = Sequential()
    cnn.add(Convolution1D(64, 3, padding="same",activation="relu",input_shape=(44,1)))
    cnn.add(MaxPooling1D(pool_size=(2)))
    cnn.add(Flatten())
    cnn.add(Dense(128, activation="relu"))
    cnn.add(Dropout(0.5))
    cnn.add(Dense(2, activation="sigmoid"))
    # define optimizer and objective, compile cnn


    cnn.compile(loss="binary_crossentropy", optimizer="RMSprop",metrics=['accuracy'])


    x_tn = x_train.iloc[train]
    y_tn = y_train.iloc[train]
    y_tn = pd.get_dummies(y_tn)
    x_ts = x_train.iloc[test]
    y_ts = y_train.iloc[test]
    y_ts = pd.get_dummies(y_ts)

    x_tn1 = x_tn.to_numpy()
    x_tn1 = np.reshape(x_tn1, (x_tn1.shape[0],x_tn1.shape[1],1))


    x_ts1 = x_ts.to_numpy()
    x_ts1 = np.reshape(x_ts1, (x_ts1.shape[0],x_ts1.shape[1],1))

    cnn.fit(x_tn1, y_tn, epochs=50,batch_size=64,verbose=1)
    scores = cnn.evaluate(x_ts1, y_ts, verbose=1)
    print(str(i)+"th Fold :")
    print("%s: %.2f%%" % (cnn.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
    i = i+1
    print("----------------------------------------------------------------")


print("Average validation accuracy : ")
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

Training and Testing accuracy:

```
446/446 [==============================] - 3s 7ms/step - loss: 0.1443 - ac
Epoch 49/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1439 - ac
Epoch 50/50
446/446 [==============================] - 3s 7ms/step - loss: 0.1438 - ac
100/100 [==============================] - 0s 2ms/step - loss: 0.1422 - ac
9th Fold :
accuracy: 94.32%
------------------------------------------------------------------------
Average validation accuracy :
94.37% (+/- 0.36%)
```

```
37] print("Training accuracy: "+str(train_acc*100))
    print("Test accuracy: "+str(test_acc*100))

    Training accuracy: 94.57574486732483
    Test accuracy: 94.34557557106018
```

PERFORMANCE METRICS (Precision, Recall, F1-Score, Support)

```
] from sklearn.metrics import classification_report
  print(classification_report(y_ts,y_pred))
```

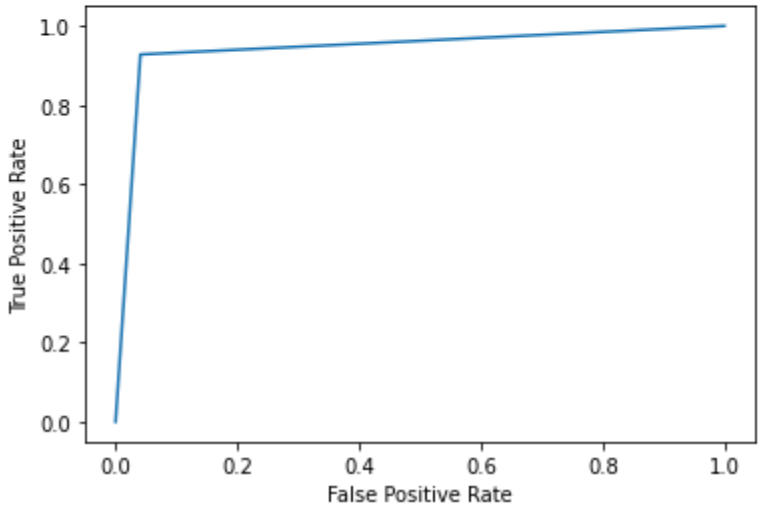
	precision	recall	f1-score	support
0	0.93	0.96	0.94	3926
1	0.96	0.93	0.94	3997
micro avg	0.94	0.94	0.94	7923
macro avg	0.94	0.94	0.94	7923
weighted avg	0.94	0.94	0.94	7923
samples avg	0.94	0.94	0.94	7923

AUC and ROC CURVE:

```
print(auc_keras)
```

```
0.943596005715443
```

```
pyplot.plot(fpr_keras, tpr_keras)
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.show()
```



Opimizer="RMSprop"
Epochs=65

Model:-

```
cvscores = []
i=0
for train, test in kfold.split(x_train, y_train):
    cnn = Sequential()
    cnn.add(Convolution1D(64, 3, padding="same",activation="relu",input_shape=(44,1)))
    cnn.add(MaxPooling1D(pool_size=(2)))
    cnn.add(Flatten())
    cnn.add(Dense(128, activation="relu"))
    cnn.add(Dropout(0.5))
    cnn.add(Dense(2, activation="sigmoid"))
    # define optimizer and objective, compile cnn


    cnn.compile(loss="binary_crossentropy", optimizer="RMSprop",metrics=['accuracy'])


    x_tn = x_train.iloc[train]
    y_tn = y_train.iloc[train]
    y_tn = pd.get_dummies(y_tn)
    x_ts = x_train.iloc[test]
    y_ts = y_train.iloc[test]
    y_ts = pd.get_dummies(y_ts)

    x_tn1 = x_tn.to_numpy()
    x_tn1 = np.reshape(x_tn1, (x_tn1.shape[0],x_tn1.shape[1],1))


    x_ts1 = x_ts.to_numpy()
    x_ts1 = np.reshape(x_ts1, (x_ts1.shape[0],x_ts1.shape[1],1))

    cnn.fit(x_tn1, y_tn, epochs=65,batch_size=64,verbose=1)
    scores = cnn.evaluate(x_ts1, y_ts, verbose=1)
    print(str(i)+"th Fold :")
    print("%s: %.2f%%" % (cnn.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
    i = i+1
    print("----------------------------------------------------------------")


print("Average validation accuracy : ")
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

Training and Testing accuracy:

```
9th Fold :
accuracy: 93.88%
----------------------------------
Average validation accuracy :
94.60% (+/- 0.41%)
```

```
print("Training accuracy: "+str(train_acc*100))
print("Test accuracy: "+str(test_acc*100))
```

```
Training accuracy: 94.59152221679688
Test accuracy: 94.59800720214844
```

PERFORMANCE METRICS (Precision, Recall, F1-Score, Support)

```
from sklearn.metrics import classification_report
print(classification_report(y_ts,y_pred))
```

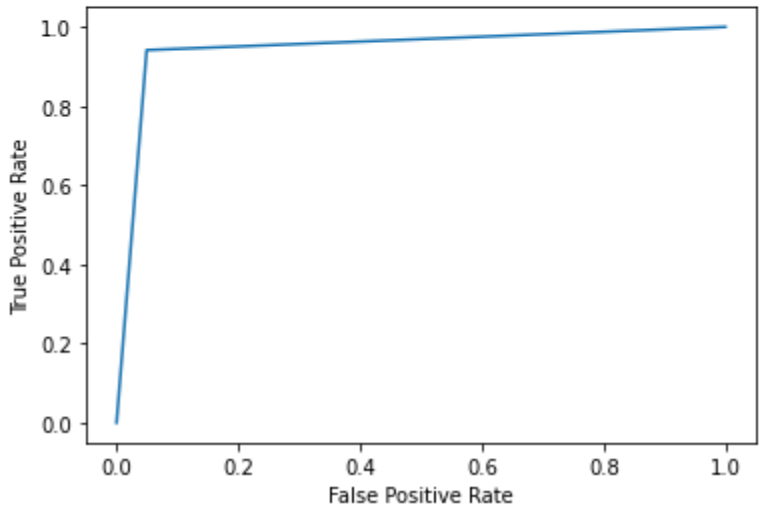
```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95      3926
           1       0.95      0.94      0.95      3997

   micro avg       0.95      0.95      0.95      7923
   macro avg       0.95      0.95      0.95      7923
weighted avg       0.95      0.95      0.95      7923
 samples avg       0.95      0.95      0.95      7923
```

AUC and ROC CURVE:

```
print(auc_keras)
```

```
0.946020965036054
```

```
pyplot.plot(fpr_keras, tpr_keras)
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.show()
```



Opimizer=” Adagrad”
Epochs=50

Model:-

```
cvscores = []
i=0
for train, test in kfold.split(x_train, y_train):
    cnn = Sequential()
    cnn.add(Convolution1D(64, 3, padding="same",activation="relu",input_shape=(44,1)))
    cnn.add(MaxPooling1D(pool_size=(2)))
    cnn.add(Flatten())
    cnn.add(Dense(128, activation="relu"))
    cnn.add(Dropout(0.5))
    cnn.add(Dense(2, activation="sigmoid"))
    # define optimizer and objective, compile cnn


    cnn.compile(loss="binary_crossentropy", optimizer="Adagrad",metrics=['accuracy'])


    x_tn = x_train.iloc[train]
    y_tn = y_train.iloc[train]
    y_tn = pd.get_dummies(y_tn)
    x_ts = x_train.iloc[test]
    y_ts = y_train.iloc[test]
    y_ts = pd.get_dummies(y_ts)

    x_tn1 = x_tn.to_numpy()
    x_tn1 = np.reshape(x_tn1, (x_tn1.shape[0],x_tn1.shape[1],1))


    x_ts1 = x_ts.to_numpy()
    x_ts1 = np.reshape(x_ts1, (x_ts1.shape[0],x_ts1.shape[1],1))

    cnn.fit(x_tn1, y_tn, epochs=50,batch_size=64,verbose=1)
    scores = cnn.evaluate(x_ts1, y_ts, verbose=1)
    print(str(i)+"th Fold :")
    print("%s: %.2f%%" % (cnn.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
    i = i+1
    print("----------------------------------------------------------------")


print("Average validation accuracy : ")
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

Training and Testing accuracy:

```
9th Fold :
accuracy: 94.32%
------------------------------------------------------------
Average validation accuracy :
94.23% (+/- 0.58%)
```

```
print("Training accuracy: "+str(train_acc*100))
print("Test accuracy: "+str(test_acc*100))
```

```
Training accuracy: 94.40219402313232
Test accuracy: 94.53489780426025
```

PERFORMANCE METRICS (Precision, Recall, F1-Score, Support)

```
from sklearn.metrics import classification_report
print(classification_report(y_ts,y_pred))
```

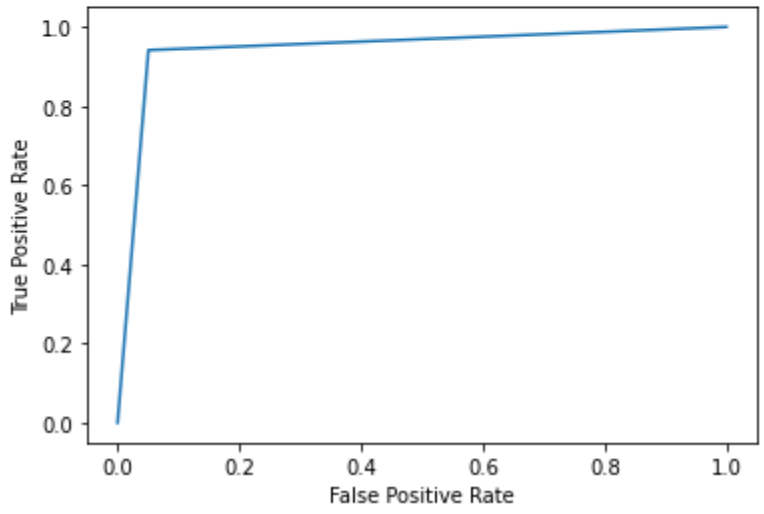
```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95      3926
           1       0.95      0.94      0.95      3997

   micro avg       0.95      0.95      0.95      7923
   macro avg       0.95      0.95      0.95      7923
weighted avg       0.95      0.95      0.95      7923
 samples avg       0.95      0.95      0.95      7923
```

AUC and ROC CURVE:

```
print(auc_keras)
```

```
0.9453841845979493
```

```
pyplot.plot(fpr_keras, tpr_keras)
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.show()
```



Opimizer=” Adagrad”
Epochs=65

Model:-

```
cvscores = []
i=0
for train, test in kfold.split(x_train, y_train):
    cnn = Sequential()
    cnn.add(Convolution1D(64, 3, padding="same",activation="relu",input_shape=(44,1)))
    cnn.add(MaxPooling1D(pool_size=(2)))
    cnn.add(Flatten())
    cnn.add(Dense(128, activation="relu"))
    cnn.add(Dropout(0.5))
    cnn.add(Dense(2, activation="sigmoid"))
    # define optimizer and objective, compile cnn


    cnn.compile(loss="binary_crossentropy", optimizer="Adagrad",metrics=['accuracy'])


    x_tn = x_train.iloc[train]
    y_tn = y_train.iloc[train]
    y_tn = pd.get_dummies(y_tn)
    x_ts = x_train.iloc[test]
    y_ts = y_train.iloc[test]
    y_ts = pd.get_dummies(y_ts)

    x_tn1 = x_tn.to_numpy()
    x_tn1 = np.reshape(x_tn1, (x_tn1.shape[0],x_tn1.shape[1],1))


    x_ts1 = x_ts.to_numpy()
    x_ts1 = np.reshape(x_ts1, (x_ts1.shape[0],x_ts1.shape[1],1))

    cnn.fit(x_tn1, y_tn, epochs=65,batch_size=64,verbose=1)
    scores = cnn.evaluate(x_ts1, y_ts, verbose=1)
    print(str(i)+"th Fold :")
    print("%s: %.2f%%" % (cnn.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
    i = i+1
    print("----------------------------------------------------------------")


print("Average validation accuracy : ")
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

Training and Testing accuracy:

```
9th Fold :
accuracy: 94.76%
----------------------------------------
Average validation accuracy :
94.57% (+/- 0.42%)
```

```
] print("Training accuracy: "+str(train_acc*100))
  print("Test accuracy: "+str(test_acc*100))

  Training accuracy: 94.55366134643555
  Test accuracy: 95.01451253890991
```

PERFORMANCE METRICS (Precision, Recall, F1-Score, Support)

```
from sklearn.metrics import classification_report
print(classification_report(y_ts,y_pred))
```

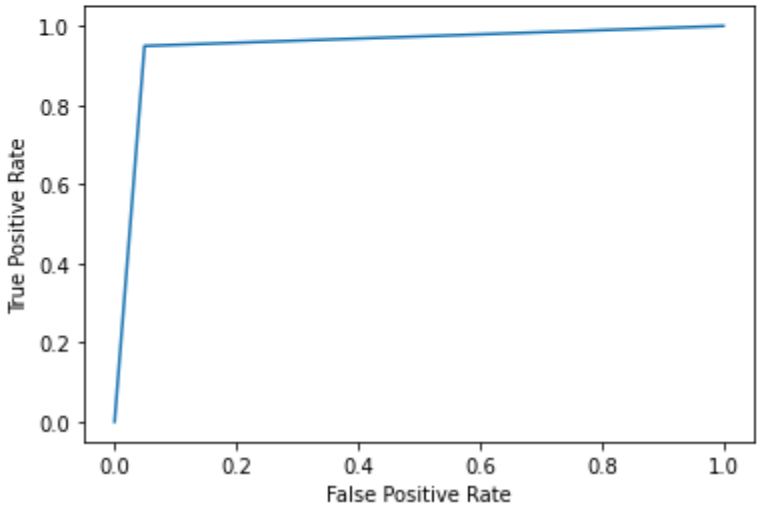
	precision	recall	f1-score	support
0	0.95	0.95	0.95	3926
1	0.95	0.95	0.95	3997
micro avg	0.95	0.95	0.95	7923
macro avg	0.95	0.95	0.95	7923
weighted avg	0.95	0.95	0.95	7923
samples avg	0.95	0.95	0.95	7923

AUC and ROC CURVE:

```
print(auc_keras)
```

```
0.9501513233753638
```

```
pyplot.plot(fpr_keras, tpr_keras)
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.show()
```



Running CNN on Aman's and Shivansh's dataset with optimizer="Adam" and epoch=50:

Aman(NonDoH, Benign):

Dataset:

Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV
Length_m	Length_m	Length_m	skew_med	skew_moc	Length_co	Time_var	Time_std	Time_mea	Time_med	Time_mod	skew_med	skew_moc	cov_time	res_time_	res_time_	res_time_	res_time_	res_time_	res_skew_	res_skew_	res_cov_ti	Label
3.450375	1.33275	1.387531	0.133807	-1.48866	-0.1352	-0.28203	-0.35867	-0.23029	-0.10095	0.456427	-1.06931	-1.42116	-0.82871	-0.11313	-0.12664	-0.05018	0.140587	-0.04807	0.179427	0.691027	0.091464	NonDoH
-0.1633	-0.65795	-0.1806	-1.95845	-1.13494	-0.85858	-0.28469	-0.45267	-0.40968	-0.31316	-0.12911	-1.50409	0.328504	-0.46039	-0.11348	-0.1828	0.166193	0.322024	0.506174	-1.44439	-1.4897	-0.5275	NonDoH
-0.2898	0.392213	-0.23887	0.916266	0.305363	0.355579	-0.28473	-0.4607	-0.41677	-0.32717	-0.12911	-0.54841	0.867792	-0.80344	-0.11348	-0.18273	-0.133	-0.0559	0.009551	0.838991	0.692224	-0.52518	NonDoH
-0.2898	-0.42536	-0.19392	0.360076	0.238742	0.107735	-0.27802	-0.29691	-0.34145	-0.32198	-0.0774	0.919694	-0.57546	1.265228	-0.11347	-0.17298	-0.14424	-0.05677	-0.04806	0.264739	0.727377	0.010519	NonDoH
-0.2898	0.018507	-0.25385	0.308179	-0.00277	1.328569	-0.28474	-0.46255	-0.42152	-0.33332	-0.12911	0.176719	0.465554	-0.57207	-0.11348	-0.17881	-0.15147	-0.0749	-0.03969	0.349409	0.778978	-0.21312	NonDoH
-0.2898	-0.02211	-0.16729	0.547834	0.201478	0.49324	-0.28473	-0.46205	-0.4194	-0.33061	-0.12911	-0.20837	0.810025	-0.77665	-0.11347	-0.17493	-0.14505	-0.05945	-0.04783	0.226929	0.782825	-0.08087	NonDoH
-0.2898	0.155428	-0.28714	0.775609	0.212716	0.524299	-0.26629	-0.18548	-0.28201	-0.32899	-0.12911	1.099388	-0.43139	1.19843	-0.11343	-0.16253	-0.13129	-0.05567	-0.04814	0.706986	0.657305	0.190419	NonDoH
-0.2568	-0.68215	-0.2938	0.924183	0.255064	-0.92773	-0.28473	-0.46134	-0.42231	-0.33374	-0.12911	-0.0383	-0.04398	0.010963	-0.11348	-0.1828	-0.05285	0.045362	0.142751	-1.44439	-1.4897	-0.5275	NonDoH
-0.28705	-0.72956	-0.31211	-0.48773	0.577172	-1.07042	-0.28474	-0.46544	-0.42597	-0.33731	-0.12911	-0.03522	-0.05055	0.02271	-0.11348	-0.1828	-0.14708	-0.07366	-0.01365	-1.44439	-1.4897	-0.5275	NonDoH
-0.14955	-0.63622	-0.16396	-1.95845	-1.13494	-0.93075	-0.28474	-0.46496	-0.42503	-0.33555	-0.12911	-1.50372	0.328504	-0.46039	-0.11348	-0.1828	-0.13401	-0.05715	0.00809	-1.44439	-1.4897	-0.5275	NonDoH
-0.2568	-0.2448	-0.2938	0.232738	-0.07835	1.424362	6.985	5.108939	4.491384	2.843157	-0.12911	0.660287	-0.05121	0.023903	-0.11346	-0.17235	-0.1368	-0.05363	-0.04814	0.472266	0.794226	-0.09182	NonDoH
-0.2568	1.844555	-0.2938	1.0992	0.339457	0.374174	2.012322	2.667616	0.688238	-0.32061	-0.12911	0.788583	-0.59685	2.465835	-0.11346	-0.17179	-0.15657	-0.09685	-0.04813	0.987668	0.566348	0.731486	NonDoH
-0.1633	-0.49101	-0.1806	0.134242	-0.12585	0.059821	0.942791	1.824645	2.982883	1.772787	-0.12911	1.238119	0.577147	-0.64835	-0.11347	-0.17375	-0.13484	-0.0528	-0.04797	0.495669	0.87214	-0.17263	NonDoH
-0.28705	-0.72956	-0.31211	-0.48773	0.577172	-1.07042	-0.28474	-0.46481	-0.42541	-0.33676	-0.12911	-0.03522	-0.05055	0.02271	-0.11348	-0.1828	-0.13251	-0.05526	0.01058	-1.44439	-1.4897	-0.5275	NonDoH
-0.2733	-0.22608	-0.20724	0.303247	0.043836	0.838338	-0.28473	-0.46116	-0.41941	-0.33222	-0.10304	0.565121	-1.66555	-0.59865	-0.11345	-0.16693	-0.10976	-0.03766	0.009236	0.89124	0.619569	-0.17336	NonDoH
-0.2568	-0.01889	-0.26717	0.191587	-0.08661	1.771481	-0.28471	-0.4546	-0.40538	-0.3192	-0.12911	0.36034	0.921608	-0.82691	-0.1134	-0.15575	-0.07223	0.04492	-0.0376	0.406871	0.832194	-0.16072	NonDoH
-0.28705	-0.72956	-0.31211	-0.48773	0.577172	-1.07042	-0.28474	-0.46324	-0.42402	-0.33542	-0.12911	-0.03522	-0.05055	0.02271	-0.11348	-0.1828	-0.09646	-0.00972	0.070398	-1.44439	-1.4897	-0.5275	NonDoH
-0.28705	-0.72956	-0.31211	-0.48773	0.577172	-1.07042	-0.28474	-0.46544	-0.42597	-0.33731	-0.12911	-0.03522	-0.05055	0.02271	-0.11348	-0.1828	-0.1471	-0.07368	-0.01363	-1.44439	-1.4897	-0.5275	NonDoH
-0.2568	1.858754	-0.2938	0.413568	0.008846	1.562845	1.355003	2.181515	0.816208	-0.30867	-0.12911	1.042655	-0.47904	1.475064	-0.11345	-0.16849	-0.14263	-0.04525	-0.04812	0.192615	0.652523	0.206642	NonDoH
-0.2898	-0.72183	-0.31378	0.02051	0.291957	-0.92831	-0.20052	0.133753	0.565388	0.902131	-0.12911	-1.14447	0.748218	-0.746	-0.11347	-0.1748	-0.14916	-0.06447	-0.04809	0.227816	0.719673	0.025777	NonDoH
-0.2898	0.637818	-0.20724	0.558092	0.12311	0.91355	-0.25902	-0.13476	-0.22242	-0.27876	-0.12911	0.982247	-0.32683	0.736022	-0.11244	-0.08621	-0.00812	-0.06438	-0.04812	1.201676	0.639396	0.256862	NonDoH
-0.2898	-0.13268	-0.17394	0.333959	0.089386	0.794669	0.207114	0.983843	0.234223	-0.33452	-0.12911	1.031507	-0.49317	1.567826	8.210831	8.476549	5.704928	-0.07304	-0.0481	0.937475	0.525254	1.385799	NonDoH
-0.2568	-0.09346	-0.26717	0.41863	0.028668	0.964085	-0.28467	-0.44953	-0.40464	-0.306	-0.12911	-1.55512	0.398055	-0.51987	-0.11346	-0.17166	-0.14935	-0.07687	-0.04814	0.686859	0.640792	0.251057	NonDoH
-0.2733	0.058149	-0.1806	0.525207	0.149009	0.638995	-0.28464	-0.44564	-0.41142	-0.32979	-0.11615	0.739657	-0.46763	0.338204	-0.11347	-0.17511	-0.13707	-0.05697	-0.04772	0.517364	0.911086	-0.204	NonDoH
-0.2898	-0.21518	-0.2938	0.591899	0.132005	0.586714	-0.28463	-0.44484	-0.39287	-0.3091	-0.12911	0.412681	0.668155	-0.70284	-0.11348	-0.18276	0.014311	0.130211	0.253733	0.486308	0.756486	-0.52727	NonDoH
-0.2898	0.463518	-0.31378	0.21739	-0.08069	2.105162	2.916401	3.233298	2.039795	-0.27051	-0.12911	1.48961	-0.27453	0.557258	-0.11338	-0.15333	-0.06256	0.061539	-0.04814	0.383663	0.864095	-0.16458	NonDoH
-0.2568	-0.32143	-0.2938	0.275877	-0.05755	1.142643	4.770838	4.183012	3.507186	2.259715	-0.12911	0.604228	-0.08854	0.094142	-0.11347	-0.17839	-0.14087	-0.06439	-0.04808	0.579454	1.162102	-0.31594	NonDoH
-0.28705	-0.72956	-0.31211	-0.48773	0.577172	-1.07042	-0.28474	-0.46474	-0.42534	-0.3367	-0.12911	-0.03522	-0.05055	0.02271	-0.11348	-0.1828	-0.13085	-0.05316	0.013333	-1.44439	-1.4897	-0.5275	NonDoH

```
cvscores = []
i=0
for train, test in kfold.split(x_train, y_train):
    cnn = Sequential()
    cnn.add(Convolution1D(64, 3, padding="same",activation="relu",input_shape=(44,1)))
    cnn.add(MaxPooling1D(pool_size=(2)))
    cnn.add(Flatten())
    cnn.add(Dense(128, activation="relu"))
    cnn.add(Dropout(0.5))
    cnn.add(Dense(2, activation="sigmoid"))
    # define optimizer and objective, compile cnn


    cnn.compile(loss="binary_crossentropy", optimizer="adam",metrics=['accuracy'])


    x_tn = x_train.iloc[train]
    y_tn = y_train.iloc[train]
    y_tn = pd.get_dummies(y_tn)
    x_ts = x_train.iloc[test]
    y_ts = y_train.iloc[test]
    y_ts = pd.get_dummies(y_ts)

    x_tn1 = x_tn.to_numpy()
    x_tn1 = np.reshape(x_tn1, (x_tn1.shape[0],x_tn1.shape[1],1))


    x_ts1 = x_ts.to_numpy()
    x_ts1 = np.reshape(x_ts1, (x_ts1.shape[0],x_ts1.shape[1],1))

    cnn.fit(x_tn1, y_tn, epochs=50,batch_size=64,verbose=1)
    scores = cnn.evaluate(x_ts1, y_ts, verbose=1)
    print(str(i)+"th Fold :")
    print("%s: %.2f%%" % (cnn.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
    i = i+1
    print("------------------------------------------------------------------------")


print("Average validation accuracy : ")
```

Training and Testing accuracy:

```
446/446 [==============================] - 3s 6ms/step - loss: 0.3365 - accuracy: 0.8373
Epoch 47/50
446/446 [==============================] - 3s 6ms/step - loss: 0.3369 - accuracy: 0.8369
Epoch 48/50
446/446 [==============================] - 3s 6ms/step - loss: 0.3373 - accuracy: 0.8374
Epoch 49/50
446/446 [==============================] - 3s 6ms/step - loss: 0.3370 - accuracy: 0.8367
Epoch 50/50
446/446 [==============================] - 3s 7ms/step - loss: 0.3348 - accuracy: 0.8369
100/100 [==============================] - 0s 2ms/step - loss: 0.3113 - accuracy: 0.8571
9th Fold :
accuracy: 85.71%
------------------------------------------------------------------------
Average validation accuracy :
83.93% (+/- 1.00%)
```

```
[48] print("Training accuracy: "+str(train_acc*100))
     print("Test accuracy: "+str(test_acc*100))

     Training accuracy: 84.1185212135315
     Test accuracy: 83.99596214294434
```

PERFORMANCE METRICS (Precision, Recall, F1-Score, Support)

```
] from sklearn.metrics import classification_report
  print(classification_report(y_ts,y_pred))
```

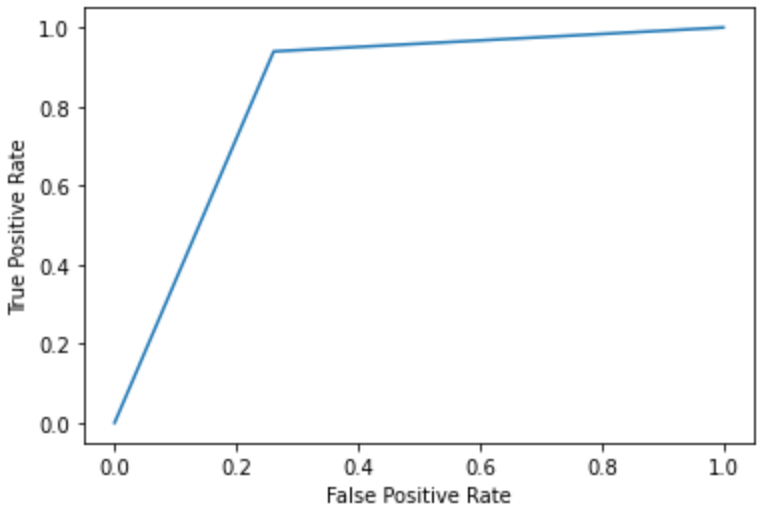
	precision	recall	f1-score	support
0	0.92	0.74	0.82	3926
1	0.79	0.94	0.86	3997
micro avg	0.84	0.84	0.84	7923
macro avg	0.85	0.84	0.84	7923
weighted avg	0.85	0.84	0.84	7923
samples avg	0.84	0.84	0.84	7923

AUC and ROC CURVE:

```
[58] print(auc_keras)
```

```
0.8390599495724697
```

```
[59] pyplot.plot(fpr_keras, tpr_keras)
     pyplot.xlabel('False Positive Rate')
     pyplot.ylabel('True Positive Rate')
     pyplot.show()
```



Shivansh(DoH, Benign):

Dataset:

U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ
ngth_me	Length_me	Length_m	skew_med	skew_moc	Length_co	Time_var	Time_std	Time_med	Time_mod	skew_med	skew_moc	cov_time	skew_med	res_time_	res_time_	res_time_	res_time_	res_time_	res_skew_	res_skew_	res_cov_ti	Label
.062383	-0.60254	-0.19077	0.717028	0.187329	0.794178	-0.55644	-0.85171	-0.64247	-0.181	1.254661	-0.40539	0.327724	1.254661	-0.14549	-0.24009	-0.20009	-0.1529	-0.11249	0.246676	0.488486	-0.25187	DoH
.682527	-0.60254	-0.19077	0.494456	0.091133	1.044114	-0.31955	-0.00296	-0.64133	-0.181	1.17284	-0.70034	1.218502	1.17284	-0.1455	-0.24321	-0.20296	-0.15359	-0.1125	-0.32153	0.497271	-0.26058	DoH
.417151	-0.72591	-0.19077	0.349595	0.012761	2.044237	-0.55651	-0.86352	-0.6416	-0.16704	0.226964	-2.81537	-0.80369	0.226964	-0.14549	-0.24035	-0.19891	-0.14972	-0.11239	-0.16168	0.564216	-0.31047	DoH
.682527	-0.60254	-0.19077	0.49471	0.091254	1.043262	-0.31391	0.007251	-0.64126	-0.181	1.178104	-0.69748	1.206346	1.178104	-0.1455	-0.24277	-0.20291	-0.15381	-0.11257	-0.20427	0.472583	-0.23483	DoH
0.58539	0.353578	-0.19077	0.04301	0.258015	-0.4018	-0.2336	0.141517	0.252861	-0.181	-0.04316	0.255948	-0.52549	-0.04316	-0.1455	-0.24277	-0.20304	-0.15383	-0.11258	-0.23061	0.46301	-0.22495	DoH
.820781	-0.60254	-0.19077	0.289844	-0.00741	2.076458	-0.31163	0.011332	-0.64188	-0.181	1.240541	-0.66568	1.077417	1.240541	-0.14549	-0.24083	-0.20132	-0.15383	-0.11257	0.209601	0.461404	-0.22383	DoH
.682124	-0.60254	-0.19077	0.494789	0.091296	1.042808	-0.32449	-0.01201	-0.64197	-0.181	1.167933	-0.70422	1.23516	1.167933	-0.1455	-0.24206	-0.20171	-0.15367	-0.11256	0.070198	0.51039	-0.2676	DoH
.935822	5.01081	-0.19077	-0.31709	0.491806	-0.23653	-0.08648	0.349619	0.702179	-0.181	-0.12868	0.602667	-0.74915	-0.12868	-0.14258	-0.10296	-0.15625	-0.14879	-0.09646	0.689661	0.165507	0.400294	DoH
0.75002	-0.60254	-0.19077	0.506804	0.096371	1.033396	-0.31916	-0.00226	-0.64197	-0.181	1.201185	-0.68665	1.161194	1.201185	-0.1455	-0.24317	-0.20327	-0.15384	-0.11257	-0.3252	0.472969	-0.23505	DoH
.570402	-0.60254	-0.19077	0.468103	0.0797	1.079515	-0.30044	0.031174	-0.64076	-0.181	1.223954	-0.67235	1.10352	1.223954	-0.14549	-0.23792	-0.20147	-0.15388	-0.11257	0.22364	0.342012	-0.05943	DoH
.918549	-0.32496	-1.07705	0.517806	0.149749	0.934837	-0.37334	-0.10727	-0.64272	-0.17758	0.91514	-0.84255	1.95586	0.91514	-0.1455	-0.24396	-0.20274	-0.15374	-0.11198	-0.30244	0.548701	-0.32485	DoH
.735221	5.01081	-0.19077	-0.60053	0.805161	-0.61599	1.005191	1.350061	1.659632	-0.181	-0.06169	0.515481	-0.70047	-0.06169	-0.1455	-0.24425	-0.20728	-0.16045	-0.11235	0.608367	0.140659	0.817417	DoH
.592327	5.01081	-0.19077	-0.83545	0.839773	-0.66055	-0.3516	-0.06348	0.237313	-0.181	-0.16668	0.566034	-0.72922	-0.16668	-0.14549	-0.24007	-0.19984	-0.15375	-0.10329	0.447304	0.090111	-0.26279	DoH
0.27561	0.862481	-0.19077	0.26016	0.517399	-0.58544	1.008678	1.352534	1.738546	-0.181	-0.17916	0.515319	-0.70038	-0.17916	-0.1288	0.099871	-0.0601	-0.16062	-0.11257	0.917709	0.240569	0.235049	DoH
0.51907	0.353578	-0.19077	0.168241	0.31502	-0.44966	1.001326	1.347317	1.710988	0.957554	-0.10698	0.173744	-0.71214	-0.10698	-0.14548	-0.23589	-0.20249	-0.16061	-0.11256	0.96594	0.25681	0.167614	DoH
.619923	-0.60254	-0.19077	0.47875	0.084267	1.067583	-0.32422	-0.01153	-0.64194	-0.181	1.142958	-0.71742	1.293208	1.142958	-0.1455	-0.2432	-0.20327	-0.15383	-0.11252	-0.32965	0.472794	-0.23783	DoH
0.92906	0.291893	2.15155	-1.65098	-0.89916	-1.14124	3.822928	2.845217	1.707772	-0.181	0.036843	-0.16507	-0.08688	0.036843	-0.1455	-0.24877	-0.20067	-0.15374	-0.10319	0.710021	0.327343	-0.63931	DoH
0.63261	0.353578	-0.19077	0.066902	0.485785	-0.7536	1.056085	1.385886	1.801238	2.613076	-0.42655	-0.46734	-0.63253	-0.42655	-0.14461	-0.16797	-0.20097	-0.16061	-0.11256	0.440022	0.08319	3.384907	DoH
.682527	-0.60254	-0.19077	0.49471	0.091254	1.043262	-0.31455	0.006093	-0.64186	-0.181	1.177784	-0.69887	1.212274	1.177784	-0.1455	-0.24305	-0.20301	-0.15382	-0.11256	-0.25065	0.48556	-0.24672	DoH
.506819	-0.60254	-0.19077	0.479138	0.085711	1.007474	-0.32753	-0.01763	-0.6394	-0.181	1.091652	-0.73917	1.393831	1.091652	-0.14548	-0.23492	-0.20058	-0.1538	-0.11257	0.324046	0.30711	0.017023	DoH
.605377	-0.60254	-0.19077	0.466235	0.068871	1.682257	-0.54864	-0.709	-0.64073	-0.181	1.61088	-0.43082	0.383461	1.61088	-0.1455	-0.24349	-0.19696	-0.15213	-0.10219	0.82119	0.297117	-0.46812	DoH
.107465	-0.60254	-0.19077	0.598605	0.136955	0.886715	-0.55577	-0.81796	-0.59805	-0.181	-0.13805	0.334659	-0.58405	-0.13805	-0.14063	-0.06044	-0.12978	-0.14882	-0.11258	0.805138	0.236885	0.251437	DoH
.077985	-0.72591	-0.19077	0.564582	0.113024	1.027083	-0.55648	-0.8563	-0.64229	-0.181	1.555399	-0.09555	-0.17862	1.555399	-0.14549	-0.23939	-0.20354	-0.15868	-0.11257	0.720013	0.285742	0.073505	DoH
.506819	-0.60254	-0.19077	0.479138	0.085711	1.007474	-0.33295	-0.02774	-0.642	-0.181	1.088073	-0.74653	1.429362	1.088073	-0.1455	-0.2429	-0.20375	-0.15381	-0.11258	-0.42187	0.415211	-0.17219	DoH
.560982	-0.60254	-0.19077	0.493838	0.092126	0.987025	-0.3357	-0.0329	-0.64183	-0.181	1.106607	-0.73629	1.380174	1.106607	-0.1455	-0.24277	-0.20366	-0.15382	-0.11257	-0.38225	0.414074	-0.17155	DoH
0.11987	2.358347	-0.19077	-0.56053	0.831372	-0.78912	0.996312	1.343752	1.700547	-0.181	-0.03272	0.569635	-0.73121	-0.03272	-0.1455	-0.2438	-0.19458	-0.14881	-0.09645	0.46043	0.090235	-0.50625	DoH
.729418	0.106838	-0.19077	0.800895	0.260773	0.546152	0.346465	0.819007	1.342227	-0.181	-0.2749	0.653874	-0.77582	-0.2749	-0.14393	-0.14193	-0.17883	-0.14888	-0.11257	0.527366	0.170463	0.703526	DoH
.062383	-0.60254	-0.19077	0.717028	0.187329	0.794178	-0.55617	-0.83379	-0.64154	-0.181	1.334147	-0.45392	0.436659	1.334147	-0.14548	-0.2371	-0.1964	-0.14847	-0.11251	0.10414	0.523771	-0.27859	DoH

shiv_ds

```
cvscores = []
i=0
for train, test in kfold.split(x_train, y_train):
    cnn = Sequential()
    cnn.add(Convolution1D(64, 3, padding="same",activation="relu",input_shape=(41,1)))
    cnn.add(MaxPooling1D(pool_size=(2)))
    cnn.add(Flatten())
    cnn.add(Dense(128, activation="relu"))
    cnn.add(Dropout(0.5))
    cnn.add(Dense(2, activation="sigmoid"))
    # define optimizer and objective, compile cnn


    cnn.compile(loss="binary_crossentropy", optimizer="adam",metrics=['accuracy'])


    x_tn = x_train.iloc[train]
    y_tn = y_train.iloc[train]
    y_tn = pd.get_dummies(y_tn)
    x_ts = x_train.iloc[test]
    y_ts = y_train.iloc[test]
    y_ts = pd.get_dummies(y_ts)

    x_tn1 = x_tn.to_numpy()
    x_tn1 = np.reshape(x_tn1, (x_tn1.shape[0],x_tn1.shape[1],1))


    x_ts1 = x_ts.to_numpy()
    x_ts1 = np.reshape(x_ts1, (x_ts1.shape[0],x_ts1.shape[1],1))

    cnn.fit(x_tn1, y_tn, epochs=50,batch_size=64,verbose=1)
    scores = cnn.evaluate(x_ts1, y_ts, verbose=1)
    print(str(i)+"th Fold :")
    print("%s: %.2f%%" % (cnn.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
    i = i+1
    print("----------------------------------------------------------------")


print("Average validation accuracy : ")
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

Training and Testing accuracy:

```
446/446 [==============================] - 3s 7ms/step - loss
Epoch 50/50
446/446 [==============================] - 3s 7ms/step - loss
100/100 [==============================] - 0s 2ms/step - loss
9th Fold :
accuracy: 87.85%
------------------------------------------------------------
Average validation accuracy :
88.71% (+/- 0.55%)
```

```
print("Training accuracy: "+str(train_acc*100))
print("Test accuracy: "+str(test_acc*100))
```

```
Training accuracy: 88.70972990989685
Test accuracy: 88.24940323829651
```

PERFORMANCE METRICS (Precision, Recall, F1-Score, Support)

```
from sklearn.metrics import classification_report
print(classification_report(y_ts,y_pred))
```

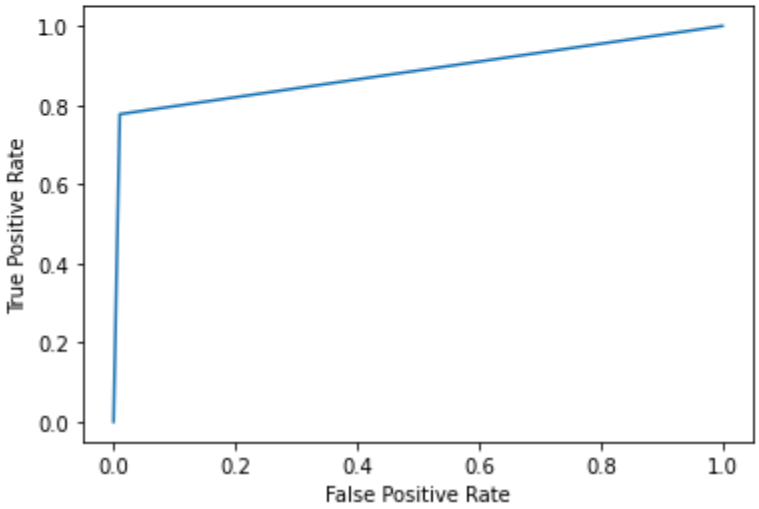
	precision	recall	f1-score	support
0	0.81	0.99	0.89	3926
1	0.99	0.78	0.87	3997
micro avg	0.88	0.88	0.88	7923
macro avg	0.90	0.88	0.88	7923
weighted avg	0.90	0.88	0.88	7923
samples avg	0.88	0.88	0.88	7923

AUC and ROC CURVE:

```
print(auc_keras)
```

```
0.8834471625497013
```

```
pyplot.plot(fpr_keras, tpr_keras)
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.show()
```



Results and comparison :

1) Dataset used here for comparison consists of Malicious and Benign data:

OPTIMIZER	EPOCH	TRAINING ACCURACY (%)	TEST ACCURACY (%)
Adam	50	94.83449459075928	94.44654583930969
RMSprop	50	94.57574486732483	94.34557557106018
RMSprop	65	94.59152221679688	94.59800720214844
Adagrad	50	94.40219402313232	94.53489780426025
Adagrad	65	94.55366134643555	95.01451253890991

Feature/ Name	**OPTIMIZER** Adam **Epoch** 50	**OPTIMIZER** RMSprop **Epoch** 50	**OPTIMIZER** RMSprop **Epoch** 65	**OPTIMIZER** Adagrad **Epoch** 50	**OPTIMIZER** Adagrad **Epoch** 65
Precision	0.92	0.93	0.94	0.94	0.95
Recall	0.97	0.96	0.95	0.95	0.95
f-Score	0.95	0.94	0.95	0.95	0.95
AUC Score	0.9446646 242960366	0.9435960 05715443	0.9460209 65036054	0.9453841 845979493	0.9501513 233753638
Avg. Validation Accuracy	94.63% (+/- 0.47%)	94.37% (+/- 0.36%)	94.60% (+/- 0.41%)	94.23% (+/- 0.58%)	94.57% (+/- 0.42%)
Training Accuracy	94.834494 59075928	94.575744 86732483	94.591522 21679688	94.402194 02313232	94.55366 134643555
Testing Accuracy	94.446545 83930969	94.345575 57106018	94.59800 720214844	94.53489 780426025	95.014512 53890991
Computation Time	1533.566s	1513.191s	2026.284s	1627.167s	1996.546s

2) Dataset here used are all four datasets i.e. Malicious, Benign, NonDoH and NonDoH on CNN model.

Feature/Name	Shivansh (DoH + Benign)	Yuvraj (Malicious + Benign)	Aman (Non-DoH + Benign)
Precision	0.81	0.92	0.92
Recall	0.99	0.97	0.74
f-Score	0.89	0.95	0.82
AUC Score	0.8834471625497013	0.9446646242960366	0.8390599495724697
Avg. Validation Accuracy	88.71% (+/- 0.55%)	94.63% (+/- 0.47%)	83.93% (+/- 1.00%)
Training Accuracy	88.70972990989685	94.83449459075928	84.1185212135315
Testing Accuracy	88.24940323829651	94.44654583930969	83.99596214294434
Computation Time	1530.099s	1533.566s	1466.379s

CONCLUSION:

I have successfully made a deep neural network consisting of stacked Convolution1D and MaxPooling1D layers along with appropriate change in optimizers and epochs. Appropriate preprocessing & sampling was done to avoid biasness in the model & fed the data to algorithm for training. The model has been validated using 10-fold cross validation and hyper-parameter tunning was done to improve the accuracy.

I found that Adagrad optimizer gave the best results on the same dataset and same number of epoch which is 50. Also when the number of epoch increased to 65, the accuracy shown a great increase in accuracy which even crossed 95% accuracy.

Apart from trying combinations on my dataset (malicious and benign), I tried CNN model on dataset of Aman (Non-DoH and benign) and Shivansh (DoH and Benign). I found that CNN model performed exceptionally well on dataset for Malicious and Benign.

Link for video demonstration:

https://drive.google.com/file/d/1oMzIOdPpv9-bDKQuQuxsSugWBMRM143Z/view?usp=sharing

Link to my Github project repository (Network-Intrusion-Detection-using-Deep-Learning):

https://github.com/yuvrajkumarViT/Network-Intrusion-Detection-using-Deep-Learning

Yuvraj Kumar
(18BIT0276)

Student's Signature