THAPAR INSTITUTE
OF ENGINEERING AND TECHNOLOGY
(Deemed to be University)

DDOS TRAFFIC DETECTION USING MACHINE LEARNING AND DEEP LEARNING

Submitted to: Mrs.Archana Singh

Efforts by:
Gautam Oberoi(102116095)
Puneet Gupta(102116073)

Title

DDOS attack detection using deep learning technique and machine learning algorithm.

Objective

The primary objective of this project is to design, implement, and evaluate a robust DDoS (Distributed Denial of Service) attack detection system leveraging both deep learning (specifically Deep Neural Networks - DNN) and traditional machine learning algorithms, including Stochastic Gradient Descent (SGD), k-Nearest Neighbors (k-NN), and Decision Trees. The aim is to develop a comprehensive solution that can effectively identify and mitigate DDoS attacks, enhancing the security and resilience of network infrastructures.

Conclusion

In conclusion, this project successfully explored the application of deep learning (DNN) and traditional machine learning algorithms (SGD, k-NN, and Decision Trees) for DDoS attack detection.

WHAT IS MEANT BY DDOS ATTACKS ?

1. Introduction to DDoS Attacks:

Definition: DDoS (Distributed Denial of Service) attacks involve overwhelming a target system, network, or service with a flood of traffic, rendering it unavailable to users. Unlike traditional DoS attacks, DDoS attacks employ multiple compromised systems (often a botnet) to amplify the impact.

Attack Mechanisms: Discuss various techniques used in DDoS attacks, such as flooding attacks (e.g., ICMP, UDP, SYN), application-layer attacks, and amplification attacks.

2. Importance of DDoS Attack Detection:

Service Availability: DDoS attacks can significantly impact the availability of online services, leading to downtime and financial losses for businesses. Highlight the critical need to detect and mitigate these attacks promptly.

Data Security: DDoS attacks may serve as a distraction while attackers attempt to breach security defenses. Discuss how effective detection can prevent or minimize the impact of concurrent security incidents.

Customer Trust: Underscore the link between service availability and customer trust. Frequent or prolonged disruptions can erode customer confidence, impacting the long-term reputation of an organization.

Financial Implications: Discuss the financial consequences of DDoS attacks, including the costs associated with downtime, potential ransom payments, and investments in cybersecurity measures for prevention and detection.

Working methodologies

Steps involved

1. Data Collection
2. Data processing
3. Train the model with deep learning technique(DNN) and machine learning techniques (Decision trees, KNN, SGD)
4. Test the model

Libraries imported:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from tensorflow import keras
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import tree
import seaborn as sns

#reading csv file
df = pd.read_csv('dataset_sdn.csv')
df.head(10)
```

```
      dt  switch       src       dst  pktcount  bytecount  dur
dur_nsec  \
0  11425       1  10.0.0.1  10.0.0.8     45304   48294064  100
716000000
1  11605       1  10.0.0.1  10.0.0.8    126395  134737070  280
734000000
2  11425       1  10.0.0.2  10.0.0.8     90333   96294978  200
744000000
3  11425       1  10.0.0.2  10.0.0.8     90333   96294978  200
744000000
4  11425       1  10.0.0.2  10.0.0.8     90333   96294978  200
744000000
5  11425       1  10.0.0.2  10.0.0.8     90333   96294978  200
744000000
6  11425       1  10.0.0.1  10.0.0.8     45304   48294064  100
716000000
7  11425       1  10.0.0.1  10.0.0.8     45304   48294064  100
716000000
8  11425       1  10.0.0.1  10.0.0.8     45304   48294064  100
716000000
9  11425       1  10.0.0.2  10.0.0.8     90333   96294978  200
744000000

        tot_dur  flows  ...  pktrate  Pairflow  Protocol  port_no
tx_bytes  \
0  1.010000e+11      3  ...      451         0       UDP        3
143928631
1  2.810000e+11      2  ...      451         0       UDP        4
```

```
3842
2  2.010000e+11      3  ...      451         0       UDP         1
3795
3  2.010000e+11      3  ...      451         0       UDP         2
3688
4  2.010000e+11      3  ...      451         0       UDP         3
3413
5  2.010000e+11      3  ...      451         0       UDP         1
3795
6  1.010000e+11      3  ...      451         0       UDP         4
3665
7  1.010000e+11      3  ...      451         0       UDP         1
3775
8  1.010000e+11      3  ...      451         0       UDP         2
3845
9  2.010000e+11      3  ...      451         0       UDP         4
354583059

  rx_bytes  tx_kbps  rx_kbps  tot_kbps  label
0     3917        0      0.0       0.0      0
1     3520        0      0.0       0.0      0
2     1242        0      0.0       0.0      0
3     1492        0      0.0       0.0      0
4     3665        0      0.0       0.0      0
5     1402        0      0.0       0.0      0
6     3413        0      0.0       0.0      0
7     1492        0      0.0       0.0      0
8     1402        0      0.0       0.0      0
9     4295    16578      0.0   16578.0      0

[10 rows x 23 columns]

#describing structure of dataset
print("This Dataset has {} rows and {} columns".format(df.shape[0],
df.shape[1]))
df.info()
df.describe()

This Dataset has 104345 rows and 23 columns
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104345 entries, 0 to 104344
Data columns (total 23 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   dt           104345 non-null  int64
 1   switch       104345 non-null  int64
 2   src          104345 non-null  object
 3   dst          104345 non-null  object
 4   pktcount     104345 non-null  int64
 5   bytecount    104345 non-null  int64
```

```
 6   dur          104345 non-null  int64
 7   dur_nsec     104345 non-null  int64
 8   tot_dur      104345 non-null  float64
 9   flows        104345 non-null  int64
 10  packetins    104345 non-null  int64
 11  pktperflow   104345 non-null  int64
 12  byteperflow  104345 non-null  int64
 13  pktrate      104345 non-null  int64
 14  Pairflow     104345 non-null  int64
 15  Protocol     104345 non-null  object
 16  port_no      104345 non-null  int64
 17  tx_bytes     104345 non-null  int64
 18  rx_bytes     104345 non-null  int64
 19  tx_kbps      104345 non-null  int64
 20  rx_kbps      103839 non-null  float64
 21  tot_kbps     103839 non-null  float64
 22  label        104345 non-null  int64
dtypes: float64(3), int64(17), object(3)
memory usage: 18.3+ MB

                  dt         switch       pktcount     bytecount  \
count  104345.000000  104345.000000  104345.000000  1.043450e+05
mean    17927.514169       4.214260   52860.954746  3.818660e+07
std     11977.642655       1.956327   52023.241460  4.877748e+07
min      2488.000000       1.000000       0.000000  0.000000e+00
25%      7098.000000       3.000000     808.000000  7.957600e+04
50%     11905.000000       4.000000   42828.000000  6.471930e+06
75%     29952.000000       5.000000   94796.000000  7.620354e+07
max     42935.000000      10.000000  260006.000000  1.471280e+08

                 dur      dur_nsec       tot_dur          flows  \
count  104345.000000  1.043450e+05  1.043450e+05  104345.000000
mean      321.497398  4.613880e+08  3.218865e+11       5.654234
std       283.518232  2.770019e+08  2.834029e+11       2.950036
min         0.000000  0.000000e+00  0.000000e+00       2.000000
25%       127.000000  2.340000e+08  1.270000e+11       3.000000
50%       251.000000  4.180000e+08  2.520000e+11       5.000000
75%       412.000000  7.030000e+08  4.130000e+11       7.000000
max      1881.000000  9.990000e+08  1.880000e+12      17.000000

           packetins     pktperflow   byteperflow        pktrate  \
count  104345.000000  104345.000000  1.043450e+05  104345.000000
mean     5200.383468    6381.715291  4.716150e+06     212.210676
std      5257.001450    7404.777808  7.560116e+06     246.855123
min         4.000000 -130933.000000 -1.464426e+08   -4365.000000
25%      1943.000000      29.000000  2.842000e+03       0.000000
50%      3024.000000    8305.000000  5.521680e+05     276.000000
75%      7462.000000   10017.000000  9.728112e+06     333.000000
max     25224.000000   19190.000000  1.495387e+07     639.000000
```

```
             Pairflow        port_no      tx_bytes      rx_bytes  \
count  104345.000000  104345.000000  1.043450e+05  1.043450e+05
mean        0.600987       2.331094  9.325264e+07  9.328039e+07
std         0.489698       1.084333  1.519380e+08  1.330004e+08
min         0.000000       1.000000  2.527000e+03  8.560000e+02
25%         0.000000       1.000000  4.743000e+03  3.539000e+03
50%         1.000000       2.000000  4.219610e+06  1.338339e+07
75%         1.000000       3.000000  1.356398e+08  1.439277e+08
max         1.000000       5.000000  1.269982e+09  9.905962e+08

             tx_kbps        rx_kbps       tot_kbps          label
count  104345.000000  103839.000000  103839.000000  104345.000000
mean      998.899756    1003.811420    2007.578742       0.390857
std      2423.471618    2054.887034    3144.437173       0.487945
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       4.000000       0.000000
75%       251.000000     557.000000    3838.000000       1.000000
max     20580.000000   16577.000000   20580.000000       1.000000
```

```
#data preprocessing
#seeing null values , counting them and then removing them from
dataset
df.isnull().sum()
(df.isnull().sum()/df.isnull().count())*100
df.dropna(inplace=True)
print(df.isnull().sum())
print("This Dataframe has {} rows and {} columns after removing null
values".format(df.shape[0], df.shape[1]))
```

```
dt             0
switch         0
src            0
dst            0
pktcount       0
bytecount      0
dur            0
dur_nsec       0
tot_dur        0
flows          0
packetins      0
pktperflow     0
byteperflow    0
pktrate        0
Pairflow       0
Protocol       0
port_no        0
tx_bytes       0
rx_bytes       0
tx_kbps        0
```

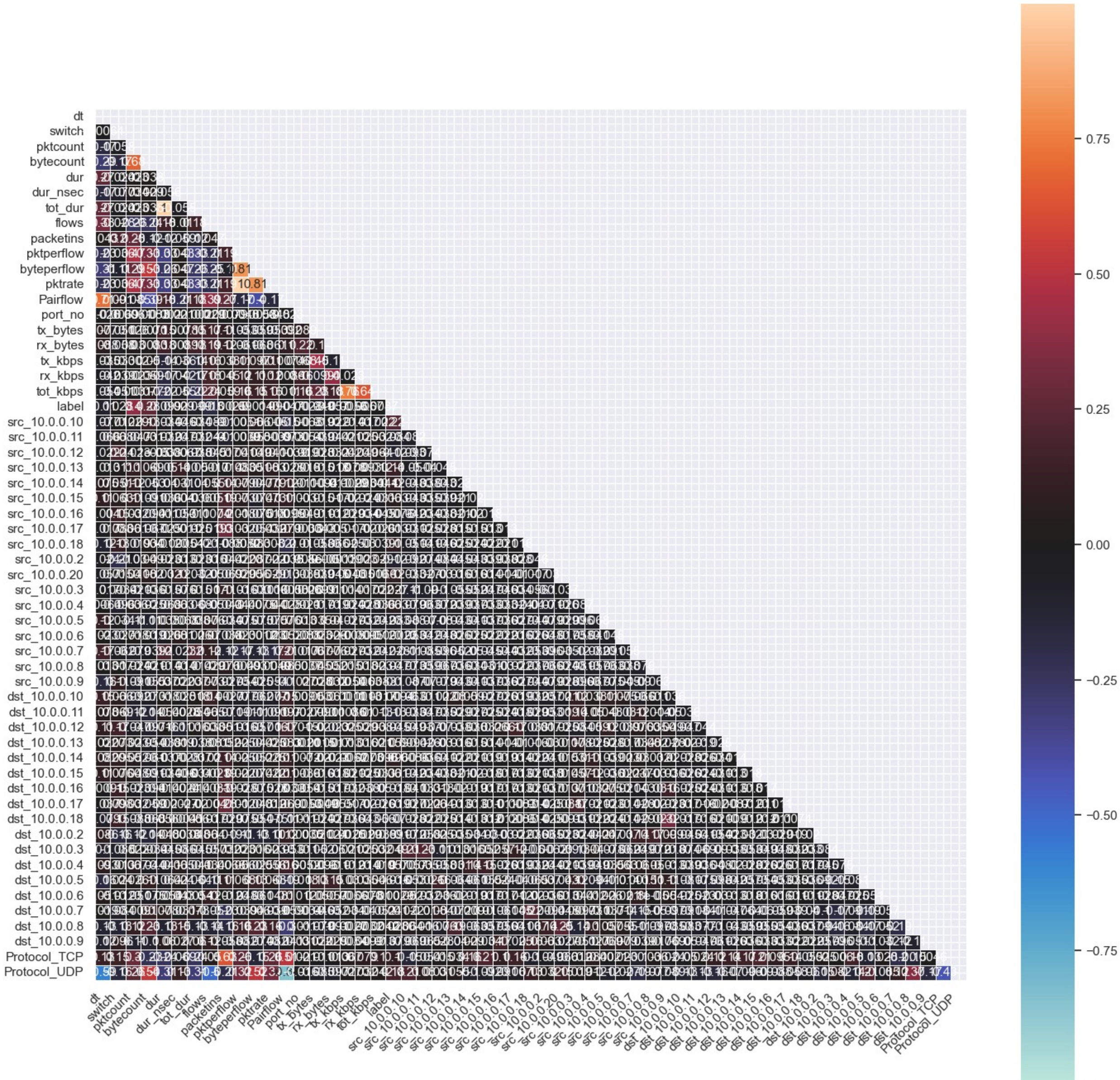
```
rx_kbps          0
tot_kbps         0
label            0
dtype: int64
This Dataframe has 103839 rows and 23 columns after removing null
values
```

```
#malign means under attack and benign is opposite of malign
malign = df[df['label'] == 1]
benign = df[df['label'] == 0]
labels = ['benign','malign']
print('Number of DDOS attacks that has
occured :',round((len(malign)/df.shape[0])*100,2),'%')
print('Number of DDOS attacks that has not
occured :',round((len(benign)/df.shape[0])*100,2),'%')
```

```
Number of DDOS attacks that has occured : 39.01 %
Number of DDOS attacks that has not occured : 60.99 %
```

```
correlation_matrix = df.corr()
fig = plt.figure(figsize=(17,17))
mask = np.zeros_like(correlation_matrix, dtype=np.bool)
mask[np.triu_indices_from(mask)]= True
sns.set_theme(style="darkgrid")
ax = sns.heatmap(correlation_matrix,square =
True,annot=True,center=0,vmin=-1,linewidths = .5,annot_kws = {"size":
11},mask = mask)
ax.set_xticklabels(ax.get_xticklabels(),rotation=45,
horizontalalignment='right');
plt.show()
```

```
C:\Users\GAUTAM\AppData\Local\Temp\ipykernel_17504\1692882402.py:3:
DeprecationWarning: `np.bool` is a deprecated alias for the builtin
`bool`. To silence this warning, use `bool` by itself. Doing this will
not modify any behavior and is safe. If you specifically wanted the
numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  mask = np.zeros_like(correlation_matrix, dtype=np.bool)
```




```
categorical_features = [feature for feature in df.columns if
df[feature].dtypes == 'O']
df = pd.get_dummies(df, columns=categorical_features,drop_first=True)
print("This Dataframe has {} rows and {} columns after
encoding".format(df.shape[0], df.shape[1]))

This Dataframe has 103839 rows and 57 columns after encoding

#dataframe after encoding
df.head(10)

      dt  switch  pktcount  bytecount  dur   dur_nsec       tot_dur
flows  \
```

```
0  11425       1     45304   48294064  100  716000000  1.010000e+11
3
1  11605       1    126395  134737070  280  734000000  2.810000e+11
2
2  11425       1     90333   96294978  200  744000000  2.010000e+11
3
3  11425       1     90333   96294978  200  744000000  2.010000e+11
3
4  11425       1     90333   96294978  200  744000000  2.010000e+11
3
5  11425       1     90333   96294978  200  744000000  2.010000e+11
3
6  11425       1     45304   48294064  100  716000000  1.010000e+11
3
7  11425       1     45304   48294064  100  716000000  1.010000e+11
3
8  11425       1     45304   48294064  100  716000000  1.010000e+11
3
9  11425       1     90333   96294978  200  744000000  2.010000e+11
3

   packetins  pktperflow  ...  dst_10.0.0.2  dst_10.0.0.3
dst_10.0.0.4  \
0       1943       13535  ...         False         False
False
1       1943       13531  ...         False         False
False
2       1943       13534  ...         False         False
False
3       1943       13534  ...         False         False
False
4       1943       13534  ...         False         False
False
5       1943       13534  ...         False         False
False
6       1943       13535  ...         False         False
False
7       1943       13535  ...         False         False
False
8       1943       13535  ...         False         False
False
9       1943       13534  ...         False         False
False

   dst_10.0.0.5  dst_10.0.0.6  dst_10.0.0.7  dst_10.0.0.8
dst_10.0.0.9  \
0         False         False         False          True
False
1         False         False         False          True
```

```
False
2          False         False         False            True
False
3          False         False         False            True
False
4          False         False         False            True
False
5          False         False         False            True
False
6          False         False         False            True
False
7          False         False         False            True
False
8          False         False         False            True
False
9          False         False         False            True
False

   Protocol_TCP  Protocol_UDP
0         False          True
1         False          True
2         False          True
3         False          True
4         False          True
5         False          True
6         False          True
7         False          True
8         False          True
9         False          True

[10 rows x 57 columns]
```

```
df.dtypes
```

```
dt                     int64
switch                 int64
pktcount               int64
bytecount              int64
dur                    int64
dur_nsec               int64
tot_dur              float64
flows                  int64
packetins              int64
pktperflow             int64
byteperflow            int64
pktrate                int64
Pairflow               int64
port_no                int64
tx_bytes               int64
rx_bytes               int64
```

```
tx_kbps              int64
rx_kbps            float64
tot_kbps           float64
label                int64
src_10.0.0.10         bool
src_10.0.0.11         bool
src_10.0.0.12         bool
src_10.0.0.13         bool
src_10.0.0.14         bool
src_10.0.0.15         bool
src_10.0.0.16         bool
src_10.0.0.17         bool
src_10.0.0.18         bool
src_10.0.0.2          bool
src_10.0.0.20         bool
src_10.0.0.3          bool
src_10.0.0.4          bool
src_10.0.0.5          bool
src_10.0.0.6          bool
src_10.0.0.7          bool
src_10.0.0.8          bool
src_10.0.0.9          bool
dst_10.0.0.10         bool
dst_10.0.0.11         bool
dst_10.0.0.12         bool
dst_10.0.0.13         bool
dst_10.0.0.14         bool
dst_10.0.0.15         bool
dst_10.0.0.16         bool
dst_10.0.0.17         bool
dst_10.0.0.18         bool
dst_10.0.0.2          bool
dst_10.0.0.3          bool
dst_10.0.0.4          bool
dst_10.0.0.5          bool
dst_10.0.0.6          bool
dst_10.0.0.7          bool
dst_10.0.0.8          bool
dst_10.0.0.9          bool
Protocol_TCP          bool
Protocol_UDP          bool
dtype: object
```

```python
#separating input and output attributes
x = df.drop(['label'], axis=1)
y = df['label']

#normalizing
ms = MinMaxScaler()
x = ms.fit_transform(x)
```

```
X_train, X_test, y_train, y_test = train_test_split(x,
y,test_size=0.3)
print(X_train.shape, X_test.shape)
```

```
(72687, 56) (31152, 56)
```

```
#defining deep neural network(DNN)
Classifier_accuracy = []

# Define and compile model
model = keras.Sequential()
model.add(Dense(28 , input_shape=(56,) , activation="relu" ,
name="Hidden_Layer_1"))
model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2"))
model.add(Dense(1 , activation="sigmoid" , name="Output_Layer"))
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile( optimizer=opt, loss="binary_crossentropy",
metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 Hidden_Layer_1 (Dense)      (None, 28)                1596      
                                                                 
 Hidden_Layer_2 (Dense)      (None, 10)                290       
                                                                 
 Output_Layer (Dense)        (None, 1)                 11        
                                                                 
=================================================================
Total params: 1,897
Trainable params: 1,897
Non-trainable params: 0
_________________________________________________________________
```

```
# fit model
X_train = np.array(X_train).astype('float32')
y_train = np.array(y_train).astype('float32')
X_test = np.array(X_test).astype('float32')
y_test = np.array(y_test).astype('float32')
history_org = model.fit(
    X_train,
    y_train,
    batch_size=32,
    epochs=100, verbose=2,
    callbacks=None,
    validation_data=(X_test,y_test),
    shuffle=True,
    class_weight=None,
```

```
    sample_weight=None,
    initial_epoch=0)
```

```
Epoch 1/100
2272/2272 - 4s - loss: 0.1700 - accuracy: 0.9275 - val_loss: 0.1109 -
val_accuracy: 0.9535 - 4s/epoch - 2ms/step
Epoch 2/100
2272/2272 - 3s - loss: 0.0977 - accuracy: 0.9602 - val_loss: 0.1031 -
val_accuracy: 0.9588 - 3s/epoch - 2ms/step
Epoch 3/100
2272/2272 - 3s - loss: 0.0808 - accuracy: 0.9668 - val_loss: 0.0752 -
val_accuracy: 0.9692 - 3s/epoch - 1ms/step
Epoch 4/100
2272/2272 - 3s - loss: 0.0697 - accuracy: 0.9721 - val_loss: 0.0631 -
val_accuracy: 0.9754 - 3s/epoch - 1ms/step
Epoch 5/100
2272/2272 - 3s - loss: 0.0610 - accuracy: 0.9762 - val_loss: 0.0532 -
val_accuracy: 0.9801 - 3s/epoch - 1ms/step
Epoch 6/100
2272/2272 - 3s - loss: 0.0591 - accuracy: 0.9766 - val_loss: 0.0514 -
val_accuracy: 0.9804 - 3s/epoch - 1ms/step
Epoch 7/100
2272/2272 - 3s - loss: 0.0526 - accuracy: 0.9790 - val_loss: 0.0520 -
val_accuracy: 0.9793 - 3s/epoch - 1ms/step
Epoch 8/100
2272/2272 - 3s - loss: 0.0492 - accuracy: 0.9798 - val_loss: 0.0598 -
val_accuracy: 0.9762 - 3s/epoch - 1ms/step
Epoch 9/100
2272/2272 - 3s - loss: 0.0463 - accuracy: 0.9803 - val_loss: 0.0449 -
val_accuracy: 0.9805 - 3s/epoch - 1ms/step
Epoch 10/100
2272/2272 - 3s - loss: 0.0461 - accuracy: 0.9801 - val_loss: 0.0419 -
val_accuracy: 0.9833 - 3s/epoch - 1ms/step
Epoch 11/100
2272/2272 - 3s - loss: 0.0440 - accuracy: 0.9822 - val_loss: 0.0548 -
val_accuracy: 0.9761 - 3s/epoch - 1ms/step
Epoch 12/100
2272/2272 - 3s - loss: 0.0422 - accuracy: 0.9824 - val_loss: 0.0414 -
val_accuracy: 0.9817 - 3s/epoch - 1ms/step
Epoch 13/100
2272/2272 - 3s - loss: 0.0401 - accuracy: 0.9830 - val_loss: 0.0385 -
val_accuracy: 0.9834 - 3s/epoch - 1ms/step
Epoch 14/100
2272/2272 - 3s - loss: 0.0392 - accuracy: 0.9838 - val_loss: 0.0390 -
val_accuracy: 0.9834 - 3s/epoch - 1ms/step
Epoch 15/100
2272/2272 - 3s - loss: 0.0384 - accuracy: 0.9839 - val_loss: 0.0405 -
val_accuracy: 0.9843 - 3s/epoch - 1ms/step
Epoch 16/100
2272/2272 - 3s - loss: 0.0380 - accuracy: 0.9843 - val_loss: 0.0342 -
```

```
val_accuracy: 0.9849 - 3s/epoch - 1ms/step
Epoch 17/100
2272/2272 - 3s - loss: 0.0364 - accuracy: 0.9849 - val_loss: 0.0424 -
val_accuracy: 0.9822 - 3s/epoch - 1ms/step
Epoch 18/100
2272/2272 - 3s - loss: 0.0350 - accuracy: 0.9849 - val_loss: 0.0344 -
val_accuracy: 0.9847 - 3s/epoch - 1ms/step
Epoch 19/100
2272/2272 - 3s - loss: 0.0346 - accuracy: 0.9852 - val_loss: 0.0354 -
val_accuracy: 0.9855 - 3s/epoch - 1ms/step
Epoch 20/100
2272/2272 - 3s - loss: 0.0338 - accuracy: 0.9853 - val_loss: 0.0341 -
val_accuracy: 0.9850 - 3s/epoch - 1ms/step
Epoch 21/100
2272/2272 - 3s - loss: 0.0341 - accuracy: 0.9855 - val_loss: 0.0411 -
val_accuracy: 0.9807 - 3s/epoch - 1ms/step
Epoch 22/100
2272/2272 - 3s - loss: 0.0338 - accuracy: 0.9852 - val_loss: 0.0339 -
val_accuracy: 0.9851 - 3s/epoch - 1ms/step
Epoch 23/100
2272/2272 - 3s - loss: 0.0314 - accuracy: 0.9864 - val_loss: 0.0323 -
val_accuracy: 0.9865 - 3s/epoch - 1ms/step
Epoch 24/100
2272/2272 - 3s - loss: 0.0326 - accuracy: 0.9861 - val_loss: 0.0370 -
val_accuracy: 0.9860 - 3s/epoch - 1ms/step
Epoch 25/100
2272/2272 - 3s - loss: 0.0317 - accuracy: 0.9866 - val_loss: 0.0327 -
val_accuracy: 0.9857 - 3s/epoch - 1ms/step
Epoch 26/100
2272/2272 - 3s - loss: 0.0306 - accuracy: 0.9867 - val_loss: 0.0366 -
val_accuracy: 0.9857 - 3s/epoch - 1ms/step
Epoch 27/100
2272/2272 - 3s - loss: 0.0319 - accuracy: 0.9864 - val_loss: 0.0433 -
val_accuracy: 0.9839 - 3s/epoch - 2ms/step
Epoch 28/100
2272/2272 - 3s - loss: 0.0297 - accuracy: 0.9869 - val_loss: 0.0378 -
val_accuracy: 0.9847 - 3s/epoch - 1ms/step
Epoch 29/100
2272/2272 - 3s - loss: 0.0289 - accuracy: 0.9872 - val_loss: 0.0319 -
val_accuracy: 0.9869 - 3s/epoch - 1ms/step
Epoch 30/100
2272/2272 - 3s - loss: 0.0282 - accuracy: 0.9876 - val_loss: 0.0346 -
val_accuracy: 0.9855 - 3s/epoch - 1ms/step
Epoch 31/100
2272/2272 - 3s - loss: 0.0284 - accuracy: 0.9882 - val_loss: 0.0273 -
val_accuracy: 0.9879 - 3s/epoch - 1ms/step
Epoch 32/100
2272/2272 - 3s - loss: 0.0289 - accuracy: 0.9881 - val_loss: 0.0281 -
val_accuracy: 0.9876 - 3s/epoch - 1ms/step
```

```
Epoch 33/100
2272/2272 - 3s - loss: 0.0258 - accuracy: 0.9892 - val_loss: 0.0343 -
val_accuracy: 0.9851 - 3s/epoch - 1ms/step
Epoch 34/100
2272/2272 - 3s - loss: 0.0279 - accuracy: 0.9880 - val_loss: 0.0316 -
val_accuracy: 0.9875 - 3s/epoch - 1ms/step
Epoch 35/100
2272/2272 - 3s - loss: 0.0275 - accuracy: 0.9882 - val_loss: 0.0382 -
val_accuracy: 0.9843 - 3s/epoch - 1ms/step
Epoch 36/100
2272/2272 - 3s - loss: 0.0279 - accuracy: 0.9880 - val_loss: 0.0360 -
val_accuracy: 0.9863 - 3s/epoch - 1ms/step
Epoch 37/100
2272/2272 - 3s - loss: 0.0255 - accuracy: 0.9887 - val_loss: 0.0312 -
val_accuracy: 0.9864 - 3s/epoch - 1ms/step
Epoch 38/100
2272/2272 - 3s - loss: 0.0269 - accuracy: 0.9882 - val_loss: 0.0271 -
val_accuracy: 0.9882 - 3s/epoch - 1ms/step
Epoch 39/100
2272/2272 - 3s - loss: 0.0250 - accuracy: 0.9890 - val_loss: 0.0408 -
val_accuracy: 0.9835 - 3s/epoch - 1ms/step
Epoch 40/100
2272/2272 - 3s - loss: 0.0262 - accuracy: 0.9884 - val_loss: 0.0318 -
val_accuracy: 0.9848 - 3s/epoch - 1ms/step
Epoch 41/100
2272/2272 - 3s - loss: 0.0258 - accuracy: 0.9889 - val_loss: 0.0240 -
val_accuracy: 0.9903 - 3s/epoch - 1ms/step
Epoch 42/100
2272/2272 - 3s - loss: 0.0259 - accuracy: 0.9886 - val_loss: 0.0296 -
val_accuracy: 0.9868 - 3s/epoch - 1ms/step
Epoch 43/100
2272/2272 - 3s - loss: 0.0236 - accuracy: 0.9896 - val_loss: 0.0330 -
val_accuracy: 0.9868 - 3s/epoch - 1ms/step
Epoch 44/100
2272/2272 - 3s - loss: 0.0255 - accuracy: 0.9889 - val_loss: 0.0278 -
val_accuracy: 0.9881 - 3s/epoch - 1ms/step
Epoch 45/100
2272/2272 - 3s - loss: 0.0250 - accuracy: 0.9891 - val_loss: 0.0300 -
val_accuracy: 0.9867 - 3s/epoch - 1ms/step
Epoch 46/100
2272/2272 - 3s - loss: 0.0249 - accuracy: 0.9894 - val_loss: 0.0251 -
val_accuracy: 0.9889 - 3s/epoch - 1ms/step
Epoch 47/100
2272/2272 - 3s - loss: 0.0241 - accuracy: 0.9892 - val_loss: 0.0245 -
val_accuracy: 0.9894 - 3s/epoch - 1ms/step
Epoch 48/100
2272/2272 - 3s - loss: 0.0250 - accuracy: 0.9889 - val_loss: 0.0203 -
val_accuracy: 0.9906 - 3s/epoch - 1ms/step
Epoch 49/100
```

```
2272/2272 - 3s - loss: 0.0234 - accuracy: 0.9896 - val_loss: 0.0360 -
val_accuracy: 0.9855 - 3s/epoch - 1ms/step
Epoch 50/100
2272/2272 - 3s - loss: 0.0236 - accuracy: 0.9897 - val_loss: 0.0222 -
val_accuracy: 0.9904 - 3s/epoch - 1ms/step
Epoch 51/100
2272/2272 - 3s - loss: 0.0234 - accuracy: 0.9897 - val_loss: 0.0271 -
val_accuracy: 0.9886 - 3s/epoch - 1ms/step
Epoch 52/100
2272/2272 - 3s - loss: 0.0230 - accuracy: 0.9903 - val_loss: 0.0215 -
val_accuracy: 0.9910 - 3s/epoch - 1ms/step
Epoch 53/100
2272/2272 - 3s - loss: 0.0223 - accuracy: 0.9904 - val_loss: 0.0269 -
val_accuracy: 0.9896 - 3s/epoch - 1ms/step
Epoch 54/100
2272/2272 - 3s - loss: 0.0242 - accuracy: 0.9895 - val_loss: 0.0259 -
val_accuracy: 0.9898 - 3s/epoch - 1ms/step
Epoch 55/100
2272/2272 - 3s - loss: 0.0226 - accuracy: 0.9899 - val_loss: 0.0323 -
val_accuracy: 0.9872 - 3s/epoch - 1ms/step
Epoch 56/100
2272/2272 - 3s - loss: 0.0214 - accuracy: 0.9905 - val_loss: 0.0216 -
val_accuracy: 0.9907 - 3s/epoch - 1ms/step
Epoch 57/100
2272/2272 - 3s - loss: 0.0239 - accuracy: 0.9900 - val_loss: 0.0258 -
val_accuracy: 0.9880 - 3s/epoch - 1ms/step
Epoch 58/100
2272/2272 - 3s - loss: 0.0227 - accuracy: 0.9901 - val_loss: 0.0267 -
val_accuracy: 0.9893 - 3s/epoch - 1ms/step
Epoch 59/100
2272/2272 - 3s - loss: 0.0216 - accuracy: 0.9904 - val_loss: 0.0306 -
val_accuracy: 0.9856 - 3s/epoch - 2ms/step
Epoch 60/100
2272/2272 - 3s - loss: 0.0227 - accuracy: 0.9899 - val_loss: 0.0209 -
val_accuracy: 0.9908 - 3s/epoch - 1ms/step
Epoch 61/100
2272/2272 - 3s - loss: 0.0224 - accuracy: 0.9904 - val_loss: 0.0222 -
val_accuracy: 0.9895 - 3s/epoch - 1ms/step
Epoch 62/100
2272/2272 - 3s - loss: 0.0210 - accuracy: 0.9908 - val_loss: 0.0209 -
val_accuracy: 0.9909 - 3s/epoch - 2ms/step
Epoch 63/100
2272/2272 - 3s - loss: 0.0217 - accuracy: 0.9907 - val_loss: 0.0268 -
val_accuracy: 0.9889 - 3s/epoch - 1ms/step
Epoch 64/100
2272/2272 - 3s - loss: 0.0216 - accuracy: 0.9909 - val_loss: 0.0240 -
val_accuracy: 0.9899 - 3s/epoch - 1ms/step
Epoch 65/100
2272/2272 - 3s - loss: 0.0221 - accuracy: 0.9908 - val_loss: 0.0213 -
```

```
val_accuracy: 0.9911 - 3s/epoch - 1ms/step
Epoch 66/100
2272/2272 - 3s - loss: 0.0208 - accuracy: 0.9910 - val_loss: 0.0232 -
val_accuracy: 0.9914 - 3s/epoch - 1ms/step
Epoch 67/100
2272/2272 - 3s - loss: 0.0197 - accuracy: 0.9915 - val_loss: 0.0297 -
val_accuracy: 0.9891 - 3s/epoch - 1ms/step
Epoch 68/100
2272/2272 - 3s - loss: 0.0215 - accuracy: 0.9907 - val_loss: 0.0189 -
val_accuracy: 0.9912 - 3s/epoch - 1ms/step
Epoch 69/100
2272/2272 - 3s - loss: 0.0213 - accuracy: 0.9908 - val_loss: 0.0220 -
val_accuracy: 0.9890 - 3s/epoch - 2ms/step
Epoch 70/100
2272/2272 - 3s - loss: 0.0220 - accuracy: 0.9903 - val_loss: 0.0234 -
val_accuracy: 0.9910 - 3s/epoch - 1ms/step
Epoch 71/100
2272/2272 - 3s - loss: 0.0197 - accuracy: 0.9914 - val_loss: 0.0286 -
val_accuracy: 0.9886 - 3s/epoch - 1ms/step
Epoch 72/100
2272/2272 - 3s - loss: 0.0202 - accuracy: 0.9914 - val_loss: 0.0201 -
val_accuracy: 0.9913 - 3s/epoch - 1ms/step
Epoch 73/100
2272/2272 - 3s - loss: 0.0205 - accuracy: 0.9912 - val_loss: 0.0231 -
val_accuracy: 0.9906 - 3s/epoch - 1ms/step
Epoch 74/100
2272/2272 - 3s - loss: 0.0199 - accuracy: 0.9916 - val_loss: 0.0213 -
val_accuracy: 0.9913 - 3s/epoch - 1ms/step
Epoch 75/100
2272/2272 - 3s - loss: 0.0205 - accuracy: 0.9911 - val_loss: 0.0260 -
val_accuracy: 0.9902 - 3s/epoch - 1ms/step
Epoch 76/100
2272/2272 - 3s - loss: 0.0204 - accuracy: 0.9911 - val_loss: 0.0235 -
val_accuracy: 0.9904 - 3s/epoch - 1ms/step
Epoch 77/100
2272/2272 - 3s - loss: 0.0203 - accuracy: 0.9911 - val_loss: 0.0207 -
val_accuracy: 0.9914 - 3s/epoch - 1ms/step
Epoch 78/100
2272/2272 - 3s - loss: 0.0188 - accuracy: 0.9918 - val_loss: 0.0245 -
val_accuracy: 0.9904 - 3s/epoch - 1ms/step
Epoch 79/100
2272/2272 - 3s - loss: 0.0209 - accuracy: 0.9912 - val_loss: 0.0196 -
val_accuracy: 0.9905 - 3s/epoch - 1ms/step
Epoch 80/100
2272/2272 - 3s - loss: 0.0208 - accuracy: 0.9915 - val_loss: 0.0250 -
val_accuracy: 0.9885 - 3s/epoch - 1ms/step
Epoch 81/100
2272/2272 - 3s - loss: 0.0198 - accuracy: 0.9914 - val_loss: 0.0190 -
val_accuracy: 0.9917 - 3s/epoch - 1ms/step
```

```
Epoch 82/100
2272/2272 - 3s - loss: 0.0205 - accuracy: 0.9914 - val_loss: 0.0218 -
val_accuracy: 0.9906 - 3s/epoch - 1ms/step
Epoch 83/100
2272/2272 - 3s - loss: 0.0193 - accuracy: 0.9916 - val_loss: 0.0218 -
val_accuracy: 0.9909 - 3s/epoch - 1ms/step
Epoch 84/100
2272/2272 - 3s - loss: 0.0194 - accuracy: 0.9916 - val_loss: 0.0200 -
val_accuracy: 0.9920 - 3s/epoch - 1ms/step
Epoch 85/100
2272/2272 - 3s - loss: 0.0200 - accuracy: 0.9913 - val_loss: 0.0207 -
val_accuracy: 0.9913 - 3s/epoch - 1ms/step
Epoch 86/100
2272/2272 - 3s - loss: 0.0181 - accuracy: 0.9919 - val_loss: 0.0213 -
val_accuracy: 0.9916 - 3s/epoch - 1ms/step
Epoch 87/100
2272/2272 - 3s - loss: 0.0204 - accuracy: 0.9910 - val_loss: 0.0213 -
val_accuracy: 0.9922 - 3s/epoch - 1ms/step
Epoch 88/100
2272/2272 - 3s - loss: 0.0180 - accuracy: 0.9927 - val_loss: 0.0200 -
val_accuracy: 0.9910 - 3s/epoch - 1ms/step
Epoch 89/100
2272/2272 - 3s - loss: 0.0209 - accuracy: 0.9915 - val_loss: 0.0196 -
val_accuracy: 0.9918 - 3s/epoch - 1ms/step
Epoch 90/100
2272/2272 - 3s - loss: 0.0179 - accuracy: 0.9924 - val_loss: 0.0175 -
val_accuracy: 0.9924 - 3s/epoch - 1ms/step
Epoch 91/100
2272/2272 - 3s - loss: 0.0193 - accuracy: 0.9915 - val_loss: 0.0170 -
val_accuracy: 0.9928 - 3s/epoch - 1ms/step
Epoch 92/100
2272/2272 - 3s - loss: 0.0181 - accuracy: 0.9920 - val_loss: 0.0251 -
val_accuracy: 0.9908 - 3s/epoch - 2ms/step
Epoch 93/100
2272/2272 - 3s - loss: 0.0191 - accuracy: 0.9916 - val_loss: 0.0208 -
val_accuracy: 0.9911 - 3s/epoch - 1ms/step
Epoch 94/100
2272/2272 - 3s - loss: 0.0179 - accuracy: 0.9923 - val_loss: 0.0196 -
val_accuracy: 0.9926 - 3s/epoch - 1ms/step
Epoch 95/100
2272/2272 - 3s - loss: 0.0181 - accuracy: 0.9922 - val_loss: 0.0272 -
val_accuracy: 0.9909 - 3s/epoch - 1ms/step
Epoch 96/100
2272/2272 - 3s - loss: 0.0190 - accuracy: 0.9921 - val_loss: 0.0219 -
val_accuracy: 0.9918 - 3s/epoch - 1ms/step
Epoch 97/100
2272/2272 - 3s - loss: 0.0191 - accuracy: 0.9919 - val_loss: 0.0256 -
val_accuracy: 0.9919 - 3s/epoch - 1ms/step
Epoch 98/100
```

```
2272/2272 - 3s - loss: 0.0182 - accuracy: 0.9925 - val_loss: 0.0245 -
val_accuracy: 0.9894 - 3s/epoch - 1ms/step
Epoch 99/100
2272/2272 - 3s - loss: 0.0192 - accuracy: 0.9917 - val_loss: 0.0218 -
val_accuracy: 0.9921 - 3s/epoch - 1ms/step
Epoch 100/100
2272/2272 - 3s - loss: 0.0189 - accuracy: 0.9919 - val_loss: 0.0174 -
val_accuracy: 0.9921 - 3s/epoch - 1ms/step
```

```
#model evaluation
loss, accuracy = model.evaluate(X_test, y_test)
print('Accuracy of Deep neural Network : %.2f' % (accuracy*100))
Classifier_accuracy.append(accuracy*100)
```

```
974/974 [==============================] - 1s 635us/step - loss:
0.0174 - accuracy: 0.9921
Accuracy of Deep neural Network : 99.21
```

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)
y_pred = knn_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of KNN Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of KNN Classifier : 96.62
```

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)
y_pred = knn_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of KNN Classifier : %.2f" % (accuracy*100))
```


```
Accuracy of KNN Classifier : 96.62
```


```
sgd_clf=SGDClassifier(loss="hinge", penalty="l2")
sgd_clf.fit(X_train,y_train)
y_pred=sgd_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of SGD Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of SGD Classifier : 84.00
```

```
Classifier_names = ["DNN", "KNN", "Decision Tree","SGD"]
```

```
df_clf = pd.DataFrame()
df_clf['name'] = Classifier_names
df_clf['Accuracy'] = Classifier_accuracy
```

```
df_clf = df_clf.sort_values(by=['Accuracy'], ascending=False)
df_clf.head(10)
```

```
            name   Accuracy
0            DNN  99.210322
1            KNN  96.623010
2  Decision Tree  96.623010
3            SGD  84.001027
```

```
print(f"The best baseline Classifier is {df_clf.name[0]} with an
accuracy of {df_clf.Accuracy[0]}.")
```

```
The best baseline Classifier is DNN with an accuracy of
99.21032190322876.
```

```
#sample predictions
classes = model.predict(X_test)
print(classes)

y_pred = []
for i in classes:
    if i > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)


y_pred[:20]

y_test[:20]
```

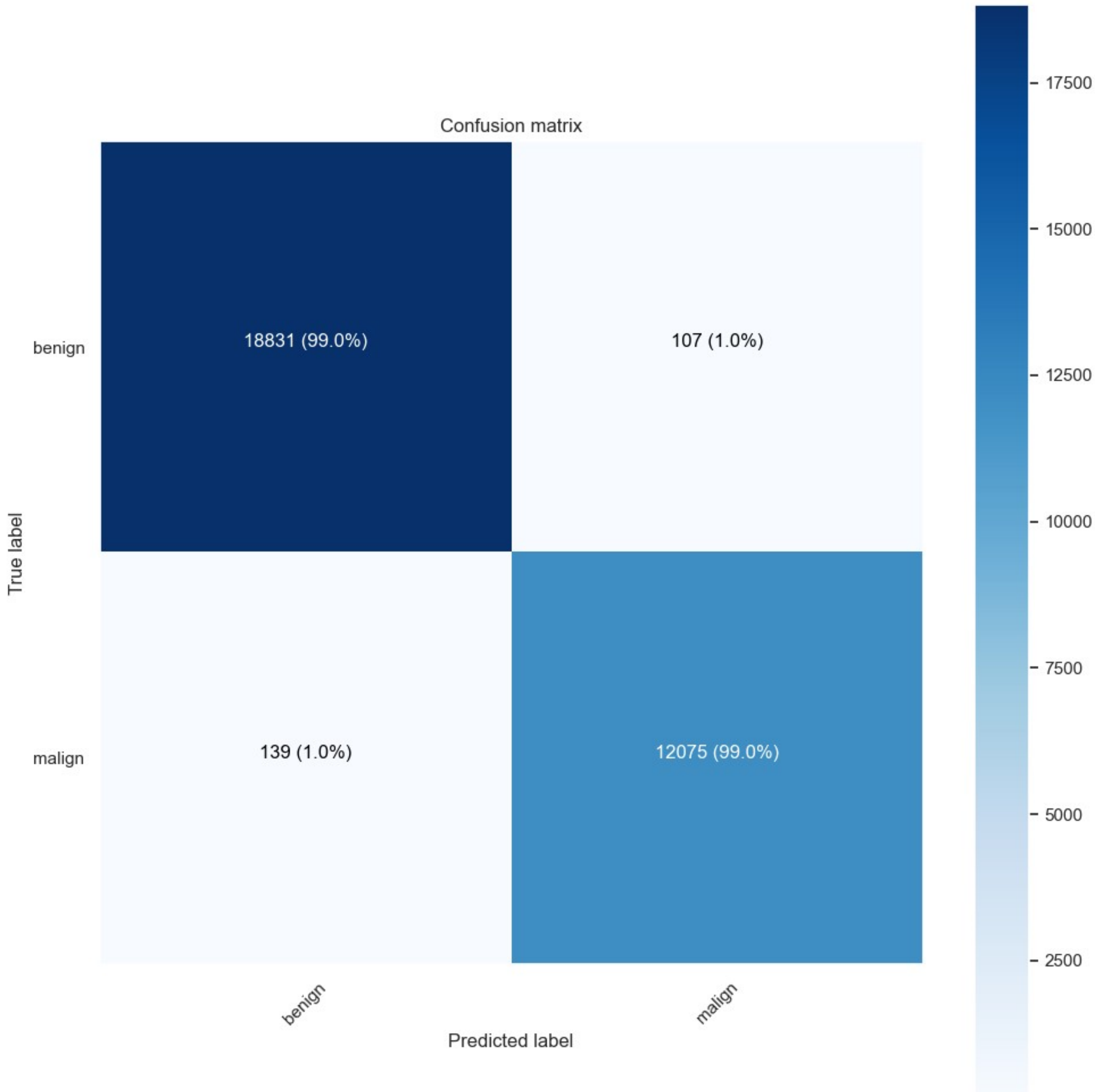
```
974/974 [==============================] - 1s 600us/step
[[1.000000e+00]
 [1.000000e+00]
 [3.465164e-22]
 ...
 [1.000000e+00]
 [4.526446e-03]
 [1.000000e+00]]
```

```
array([1., 1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 0., 0.,
0.,
       0., 1., 0.], dtype=float32)
```

```
#prints the f1 score, recall, precision of sample predictions
print(classification_report(y_test, y_pred, target_names = labels))
```

```
              precision    recall  f1-score   support

      benign       0.99      0.99      0.99     18938
      malign       0.99      0.99      0.99     12214
```

```
    accuracy                           0.99     31152
   macro avg       0.99      0.99      0.99     31152
weighted avg       0.99      0.99      0.99     31152


#plotting confusion matrix
from itertools import product
def plot_confusion_matrix(cm, classes, normalize=True,
title='Confusion matrix', cmap=plt.cm.Blues):
    plt.figure(figsize=(10,10))
    plt.grid(False)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    cm1 = cm
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.around(cm, decimals=2)
        cm[np.isnan(cm)]
        thresh = cm.max() / 2.
    for i, j in product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, str(cm1[i, j])+ " ("+ str(cm[i, j]*100)+"%)",
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')




confusion_mtx = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(confusion_mtx, classes = labels)
```


Confusion matrix
benign
malign
True label
18831 (99.0%)
107 (1.0%)
139 (1.0%)
12075 (99.0%)
benign
malign
Predicted label
17500
15000
12500
10000
7500
5000
2500

Bibliography

Deep neural network for ddos attacks article

https://ijai.iaescore.com/index.php/IJAI/article/view/20884

Machine learning for ddos attacks article

https://www.researchgate.net/publication/356465007_Detection_of_Different_DDoS_Attacks_Using_Machine_Learning_Classification_Algorithms