AMITY UNIVERSITY UTTAR PRADESH
NOIDA

AIML(203)
Deep Neural Networks Lab File

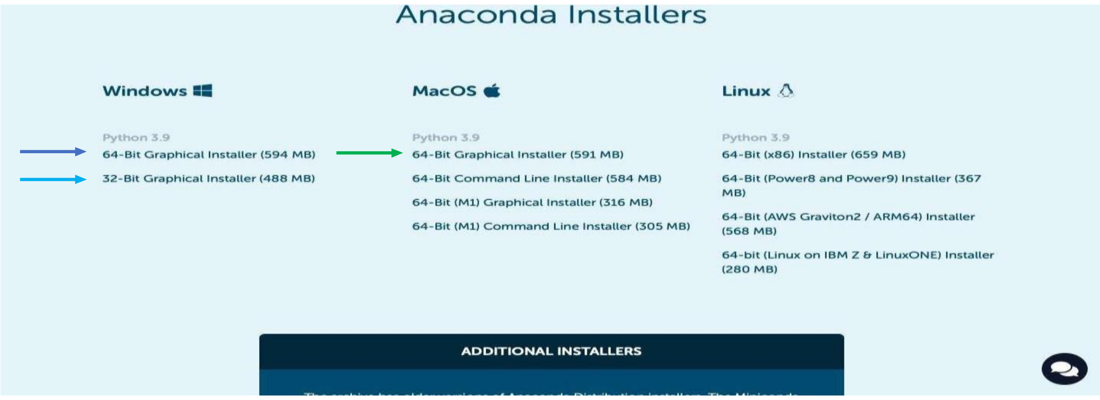
Submitted to:
Prof. M.K. Dutta
Submitted By:
Naman Sharma
A023119823058

S. No.	Practical	Page no.	Date	Teacher's Signature
1.	Anaconda Installation	1-2	16/12/24	
2.	Bank Churn ANN Algorithm	3-15	23/12/24	
3.	ANN algorithm using bird data	16-37	13/1/25	
4.	Creating CNN model from scratch	38-46	20/1/25	
5.	Deep Learning Training and Architecture	47-69	27/1/25	
6.	Text data handling with RNN for sentiment analysis	70-78	3/2/25	
7.	Sentiment analysis using RNN-LSTM on tweets data	79-91	10/2/25	
8.	Auto encoders using MNIST data	92-97	17/2/25	
9.	Variational Autoencoders	98-106	24/2/25	
10.	Implementing GAN Architecture on MINST dataset	107-116	10/3/25	

Anaconda Installation Guide

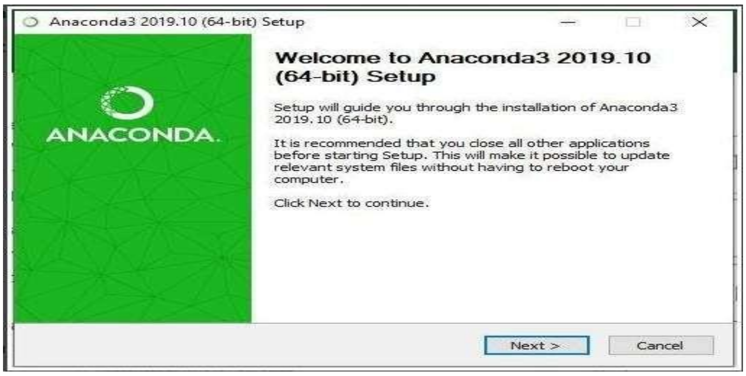
1. **Download Anaconda**

- Visit the official website: Anaconda Distribution.
- Select the appropriate version based on your system (Windows, macOS, Linux).



2. **Start Installation**

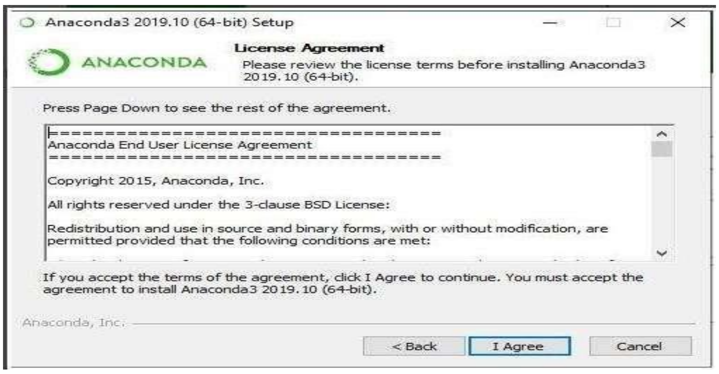
- Run the downloaded installer.



- Click **Next** to begin the installation process.

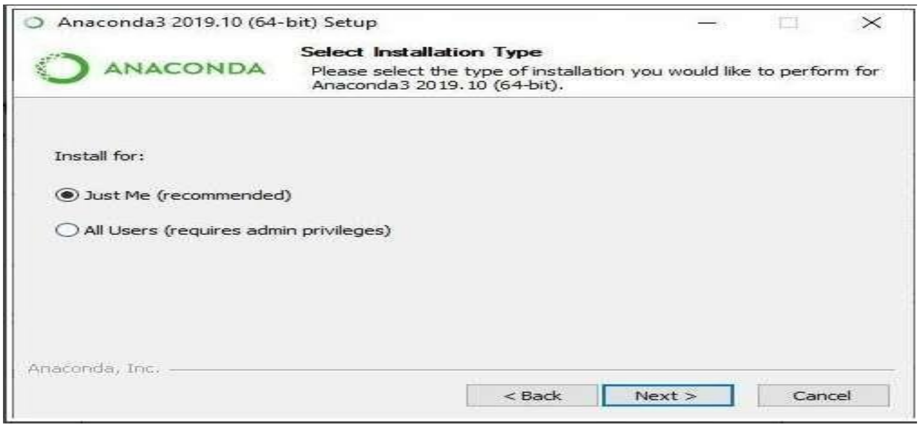
3. **Accept License Agreement**

- Read and accept the license agreement to proceed.



4. **Select Installation Type**

- Choose "**Just Me**" if you want to install for a single user.



5. **Choose Installation Location**

- Select the directory where Anaconda should be installed.

6. **Configure Advanced Options (Optional)**

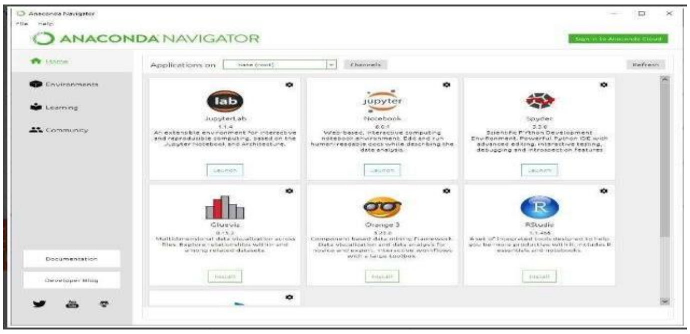
- Add Anaconda to your system **PATH** (not recommended).
- Register Anaconda as the default Python environment.

7. **Complete Installation**

- Click **Install** and wait for the process to finish.
- Click **Finish** once done.

8. **Launch Anaconda**

- Open **Anaconda Navigator** from the Start Menu.
- You can now start using tools like **Jupyter Notebook** for Python coding.



Experiment - 1

Bank Churn ANN

Importing Necessary Libraries

```
import numpy as np
import pandas as pd
```

Loading the Churn Dataset

```
churn_data = pd.read_csv('/content/Churn_Modelling.csv', delimiter = ',')
churn_data.head(5)
```

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

Accessing the Column Names in the Dataset

```
churn_data.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

Setting Column as a Index

```
churn_data = churn_data.set_index('RowNumber')
churn_data.head()
```

```
           CustomerId   Surname  CreditScore Geography  Gender  Age  Tenure  \
RowNumber
1            15634602  Hargrave          619    France  Female   42       2
2            15647311      Hill          608     Spain  Female   41       1
3            15619304      Onio          502    France  Female   42       8
4            15701354      Boni          699    France  Female   39       1
5            15737888  Mitchell          850     Spain  Female   43       2

             Balance  NumOfProducts  HasCrCard  IsActiveMember  \
RowNumber
1               0.00              1          1               1
2           83807.86              1          0               1
3          159660.80              3          1               0
4               0.00              2          0               0
5          125510.82              1          1               1

           EstimatedSalary  Exited
RowNumber
1                101348.88       1
2                112542.58       0
3                113931.57       1
4                 93826.63       0
5                 79084.10       0
```

Finding the Shape of the Dataset

```
churn_data.shape
```

```
(10000, 13)
```

```
churn_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CustomerId       10000 non-null  int64
 1   Surname          10000 non-null  object
 2   CreditScore      10000 non-null  int64
 3   Geography        10000 non-null  object
 4   Gender           10000 non-null  object
 5   Age              10000 non-null  int64
 6   Tenure           10000 non-null  int64
 7   Balance          10000 non-null  float64
 8   NumOfProducts    10000 non-null  int64
 9   HasCrCard        10000 non-null  int64
 10  IsActiveMember   10000 non-null  int64
 11  EstimatedSalary  10000 non-null  float64
 12  Exited           10000 non-null  int64
```

```
dtypes: float64(2), int64(8), object(3)
memory usage: 1.1+ MB
```

Checking Missing Values

```
churn_data.isna().sum()
```

```
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
churn_data.nunique()
```

```
CustomerId         10000
Surname             2932
CreditScore          460
Geography              3
Gender                 2
Age                   70
Tenure                11
Balance             6382
NumOfProducts          4
HasCrCard              2
IsActiveMember         2
EstimatedSalary     9999
Exited                 2
dtype: int64
```

```
churn_data.drop(['CustomerId','Surname'],axis=1,inplace=True)
```

```
churn_data.head()
```

```
           CreditScore Geography  Gender  Age  Tenure    Balance  \
RowNumber
1                  619    France  Female   42       2       0.00
2                  608     Spain  Female   41       1   83807.86
3                  502    France  Female   42       8  159660.80
4                  699    France  Female   39       1       0.00
5                  850     Spain  Female   43       2  125510.82

           NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary  Exited
RowNumber
1                      1          1               1        101348.88       1
2                      1          0               1        112542.58       0
3                      3          1               0        113931.57       1
4                      2          0               0         93826.63       0
5                      1          1               1         79084.10       0
```

```
churn_data.shape
```

```
(10000, 11)
```

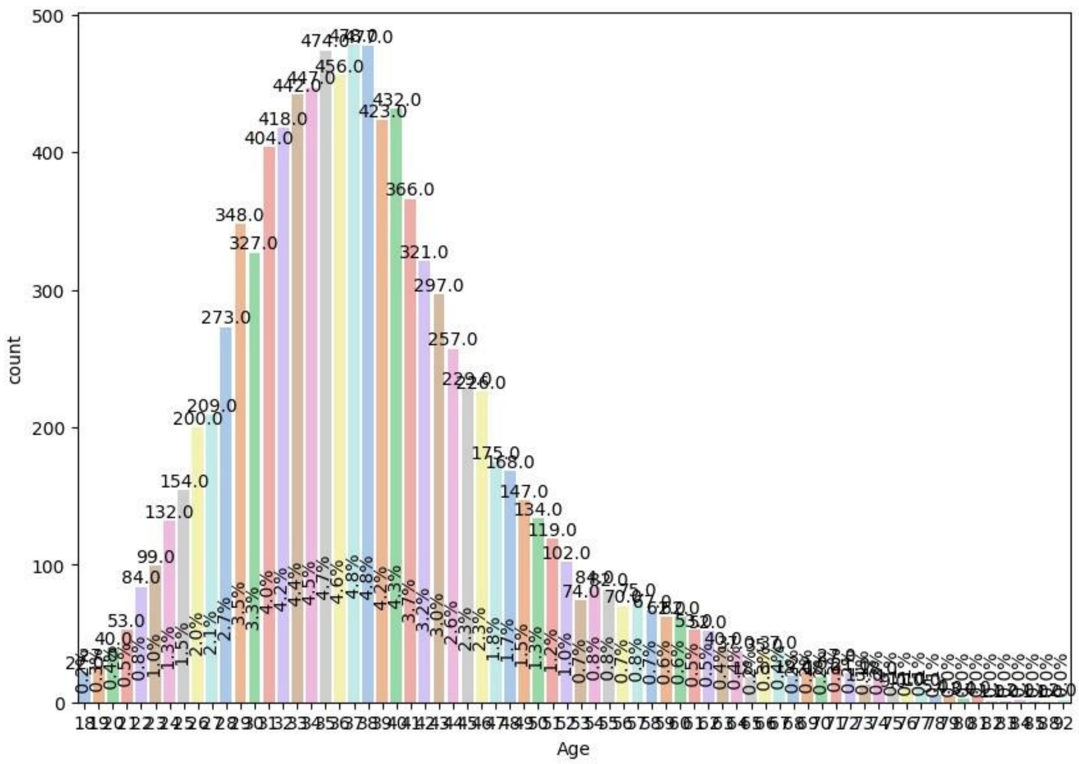
Some Visualizations

```
from matplotlib import pyplot as plt
import seaborn as sns
from scipy import stats
df = churn_data.copy()

def plot_univariate(col):
    if(df[col].nunique()>2):
        plt.figure(figsize=(10,7))
        h = 0.15
        rot=90
    else:
        plt.figure(figsize=(6,6))
        h = 0.5
        rot=0
    plot = sns.countplot(x = df[col], palette='pastel')

    for bars in plot.containers:
        for p in bars:
            plot.annotate(format(p.get_height()), (p.get_x() +
p.get_width()*0.5, p.get_height()),
                    ha = 'center', va = 'bottom')
            plot.annotate(f'{p.get_height()*100/df[col].shape[0] : .1f}%',
(p.get_x() + p.get_width()*0.5, h*p.get_height()),
                    ha = 'center', va = 'bottom', rotation=rot)

def spearman(df,hue):
    feature = []
```

```
    correlation = []
    result = []
    for col in df.columns:
        corr, p = stats.spearmanr(df[col], df[hue])
        feature.append(col)
        correlation.append(corr)
        alpha = 0.05
        if p > alpha:
            result.append('No correlation (fail to reject H0)')
        else:
            result.append('Some correlation (reject H0)')
    c = pd.DataFrame({'Feature Name':feature,'correlation
coefficient':correlation, 'Inference':result})
    display(c)
```

```
plot_univariate('Age')
```

```
/tmp/ipykernel_6944/4200767248.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  plot = sns.countplot(x = df[col], palette='pastel')
```




```
spearman(churn_data,'Age')
```

```
        Feature Name  correlation coefficient  \
0        CreditScore                -0.007974   
1          Geography                 0.035351   
2             Gender                -0.029785   
3                Age                 1.000000   
4             Tenure                -0.010405   
5            Balance                 0.033304   
6      NumOfProducts                -0.058566   
7          HasCrCard                -0.015278   
8     IsActiveMember                 0.039839   
9    EstimatedSalary                -0.002431   
10            Exited                 0.323968   

                            Inference  
0   No correlation (fail to reject H0)  
1         Some correlation (reject H0)  
2         Some correlation (reject H0)  
3         Some correlation (reject H0)  
4   No correlation (fail to reject H0)  
5         Some correlation (reject H0)  
6         Some correlation (reject H0)  
7   No correlation (fail to reject H0)  
8         Some correlation (reject H0)  
9   No correlation (fail to reject H0)  
10        Some correlation (reject H0)  
```

```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
churn_data[['Geography', 'Gender']] = churn_data[['Geography',
'Gender']].apply(le.fit_transform)

churn_data.head()
```

```
           CreditScore  Geography  Gender  Age  Tenure    Balance  \
RowNumber                                                           
1                  619          0       0   42       2       0.00   
2                  608          2       0   41       1   83807.86   
3                  502          0       0   42       8  159660.80   
4                  699          0       0   39       1       0.00   
5                  850          2       0   43       2  125510.82   

           NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary  Exited  
RowNumber                                                                     
1                      1          1               1        101348.88       1  
2                      1          0               1        112542.58       0  
```

```
3                        3          1                0         113931.57        1
4                        2          0                0          93826.63        0
5                        1          1                1          79084.10        0
```

Seperating Label from Data

```
y = churn_data.Exited
X = churn_data.drop(['Exited'],axis=1)
```

```
X.columns
```

```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
      'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary'],
      dtype='object')
```

```
y
```

```
RowNumber
1        1
2        0
3        1
4        0
5        0
        ..
9996     0
9997     0
9998     1
9999     1
10000    0
Name: Exited, Length: 10000, dtype: int64
```

Splitting the Data into Training and Testing

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
random_state = 2)
```

```
print("Shape of the X_train", X_train.shape)
print("Shape of the X_test", X_test.shape)
print("Shape of the y_train", y_train.shape)
print("Shape of the y_test", y_test.shape)
```

```
Shape of the X_train (7000, 10)
Shape of the X_test (3000, 10)
Shape of the y_train (7000,)
Shape of the y_test (3000,)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Building the ANN Model

```
from keras.models import Sequential
from keras.layers import Dense
```

```
2024-05-21 08:34:06.981674: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9373] Unable to
register cuDNN factory: Attempting to register factory for plugin cuDNN when
one has already been registered
2024-05-21 08:34:06.981725: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to
register cuFFT factory: Attempting to register factory for plugin cuFFT when
one has already been registered
2024-05-21 08:34:06.982807: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1534] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS
when one has already been registered
2024-05-21 08:34:06.989509: I
tensorflow/core/platform/cpu_feature_guard.cc:183] This TensorFlow binary is
optimized to use available CPU instructions in performance-critical
operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

```
classifier = Sequential()
# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation =
'relu', input_dim = 10))

# Adding the second hidden layer
classifier.add(Dense(units = 16, kernel_initializer = 'uniform', activation =
'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation =
'sigmoid'))
```

```
2024-05-21 08:34:10.706731: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1926] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 17947 MB memory:  ->
device: 0, name: NVIDIA A100-SXM4-40GB MIG 3g.20gb, pci bus id: 0000:bd:00.0,
compute capability: 8.0
```

Compiling and Fitting the Model

```
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics
= ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100, verbose = 1)
```

```
Epoch 1/100

2024-05-21 08:34:30.051165: I external/local_xla/xla/service/service.cc:168]
XLA service 0x7fc261d536a0 initialized for platform CUDA (this does not
guarantee that XLA will be used). Devices:
2024-05-21 08:34:30.051210: I external/local_xla/xla/service/service.cc:176]
StreamExecutor device (0): NVIDIA A100-SXM4-40GB MIG 3g.20gb, Compute
Capability 8.0
2024-05-21 08:34:30.056612: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling
MLIR crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to
enable.
2024-05-21 08:34:30.108404: I
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:467] Loaded cuDNN
version 90000
WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
I0000 00:00:1716280470.197288     7745 device_compiler.h:186] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.

700/700 [==============================] - 3s 1ms/step - loss: 0.4773 -
accuracy: 0.7924
Epoch 2/100
700/700 [==============================] - 1s 1ms/step - loss: 0.4296 -
accuracy: 0.7926
Epoch 3/100
700/700 [==============================] - 1s 1ms/step - loss: 0.4231 -
accuracy: 0.8116
Epoch 4/100
700/700 [==============================] - 1s 1ms/step - loss: 0.4188 -
accuracy: 0.8240
Epoch 5/100
700/700 [==============================] - 1s 1ms/step - loss: 0.4158 -
accuracy: 0.8297
Epoch 6/100
700/700 [==============================] - 1s 1ms/step - loss: 0.4134 -
accuracy: 0.8314
Epoch 7/100
700/700 [==============================] - 1s 1ms/step - loss: 0.4120 -
accuracy: 0.8291
```

```
700/700 [==============================] - 1s 1ms/step - loss: 0.3837 -
accuracy: 0.8449
Epoch 92/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3820 -
accuracy: 0.8456
Epoch 93/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3771 -
accuracy: 0.8496
Epoch 94/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3705 -
accuracy: 0.8497
Epoch 95/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3610 -
accuracy: 0.8530
Epoch 96/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3520 -
accuracy: 0.8581
Epoch 97/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3464 -
accuracy: 0.8581
Epoch 98/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3428 -
accuracy: 0.8616
Epoch 99/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3409 -
accuracy: 0.8613
Epoch 100/100
700/700 [==============================] - 1s 1ms/step - loss: 0.3404 -
accuracy: 0.8590
```

```
<keras.src.callbacks.History at 0x7fcbe2df92d0>
```

Testing the Model

```
score, acc = classifier.evaluate(X_train, y_train,
                             batch_size=10)
print('Train score:', score)
print('Train accuracy:', acc)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

print('*'*20)
score, acc = classifier.evaluate(X_test, y_test,
                             batch_size=10)
print('Test score:', score)
print('Test accuracy:', acc)
```

```
700/700 [==============================] - 1s 884us/step - loss: 0.3388 -
accuracy: 0.8609
```

```
Train score: 0.33875057101249695
Train accuracy: 0.8608571290969849
94/94 [==============================] - 0s 667us/step
********************
300/300 [==============================] - 0s 910us/step - loss: 0.3567 -
accuracy: 0.8577
Test score: 0.3566587269306183
Test accuracy: 0.8576666712760925
```

Confusion Matrix

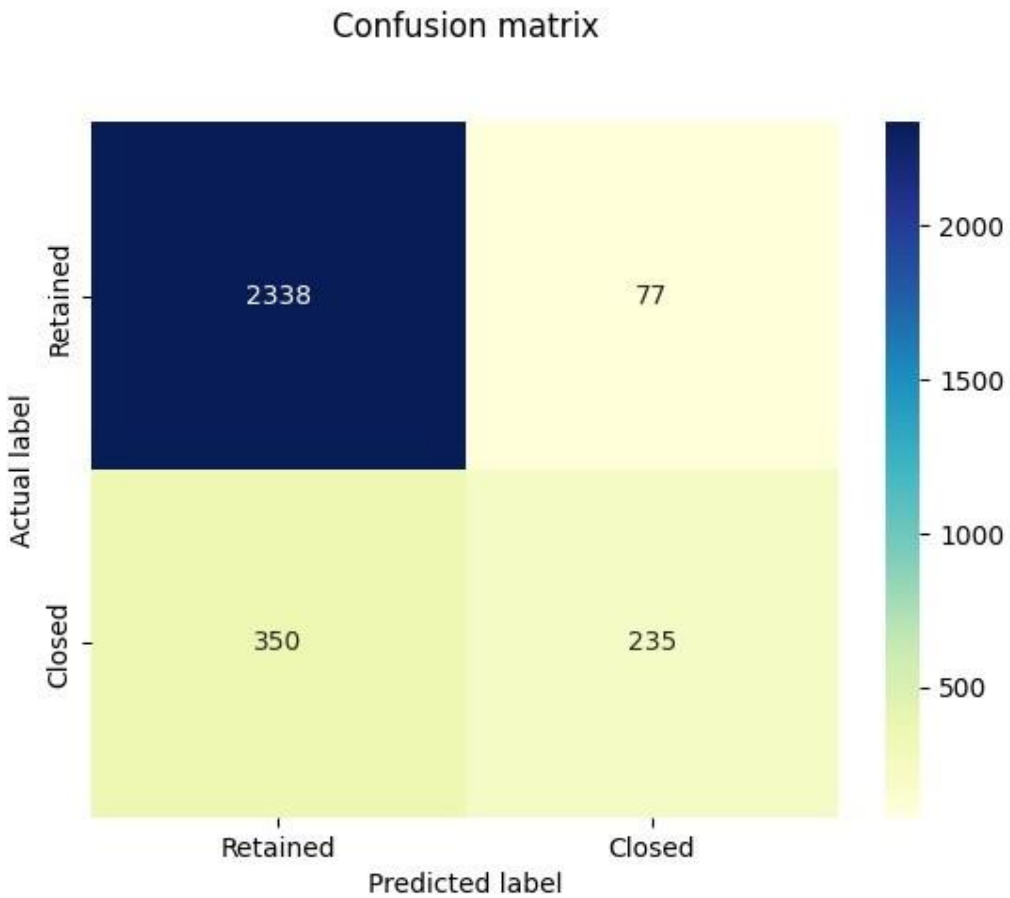
```
from sklearn.metrics import confusion_matrix
target_names = ['Retained', 'Closed']
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[2338   77]
 [ 350  235]]
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
p = sns.heatmap(pd.DataFrame(cm), annot=True, xticklabels=target_names,
yticklabels=target_names, cmap="YlGnBu" ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 23.52222222222222, 'Predicted label')
```



Classification

```
#import classification_report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred, target_names=target_names))
```

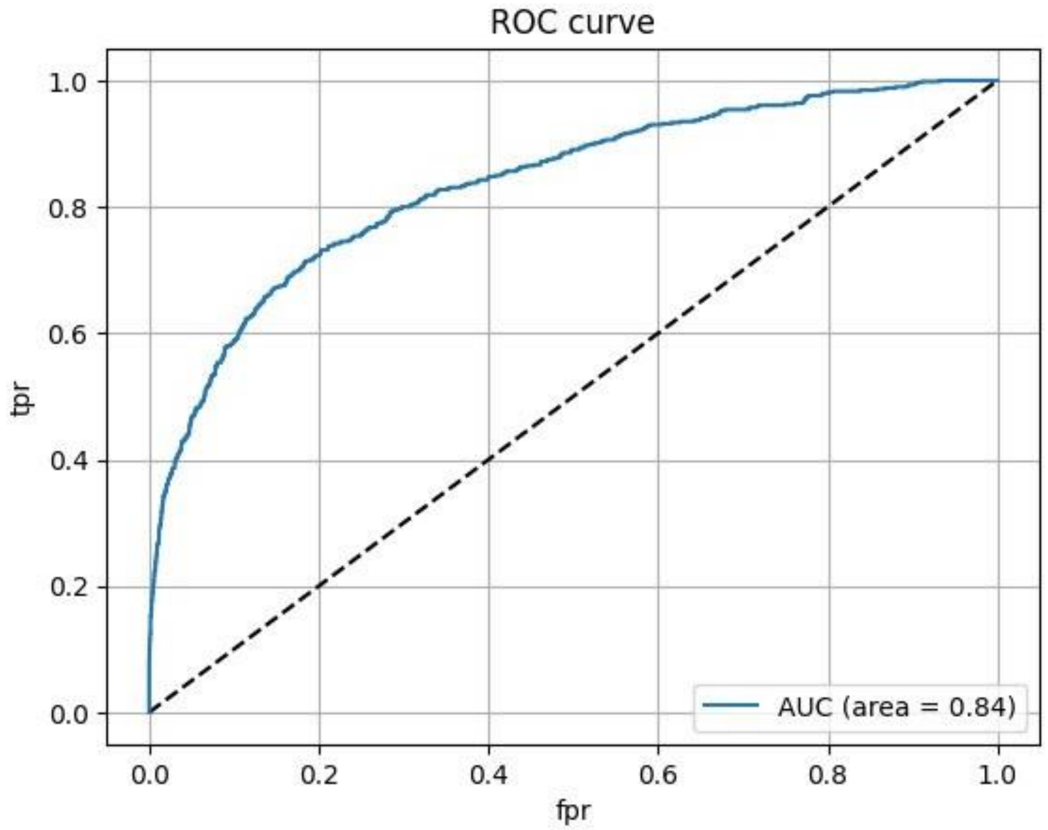
```
              precision    recall  f1-score   support

    Retained       0.87      0.97      0.92      2415
      Closed       0.75      0.40      0.52       585

    accuracy                           0.86      3000
   macro avg       0.81      0.68      0.72      3000
weighted avg       0.85      0.86      0.84      3000
```

#ROC curve

```
from sklearn.metrics import roc_curve, auc
y_pred_proba = classifier.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
```

```
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='AUC (area = %0.2f)' % roc_auc)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.grid()
plt.legend(loc="lower right")
plt.title('ROC curve')
plt.show()
```

```
94/94 [==============================] - 0s 667us/step
```




```
#Area under ROC curve
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

```
0.8357169400647662
```

Experiment - 2

ANN using Bird data

#Importing Necessary Libraries

```
import numpy as np
import pandas as pd
```

#Loading the Birds Dataset

```
bird_data = pd.read_csv('/content/bird.csv', delimiter = ',')
bird_data.head(5)
```

```
         id                             huml humw ulnal ulnaw      feml
femw     tibl  tibw                     tarl tarw type
0    0   80.78   6.68   72.01    4.88   41.81   3.70    5.50   4.03   38.70
3.84     SW
1    1   88.91   6.63   80.53    5.59   47.04   4.30   80.22   4.51   41.50
4.01     SW
2    2   79.97   6.37   69.26    5.28   43.07   3.90   75.35   4.04   38.31
3.34     SW
3    3   77.65   5.70   65.76    4.77   40.04   3.52   69.17   3.40   35.78
3.41     SW
4    4   62.80   4.84   52.09    3.73   33.95   2.72   56.27   2.96   31.88
3.13     SW
```

#Accessing the Column Names in the Dataset

```
bird_data.columns

Index(['id', 'huml', 'humw', 'ulnal', 'ulnaw', 'feml', 'femw', 'tibl',
'tibw',
        'tarl', 'tarw', 'type'],
       dtype='object')

bird_data = bird_data.set_index('id')
bird_data.head()

     huml  humw  ulnal  ulnaw      feml  femw     tibl  tibw     tarl  tarw
type
id

0    80.78   6.68   72.01    4.88   41.81   3.70    5.50   4.03   38.70   3.84
SW
1    88.91   6.63   80.53    5.59   47.04   4.30   80.22   4.51   41.50   4.01
SW
2    79.97   6.37   69.26    5.28   43.07   3.90   75.35   4.04   38.31   3.34
SW
```

```
3    77.65   5.70   65.76    4.77   40.04   3.52   69.17   3.40   35.78   3.41
SW
4    62.80   4.84   52.09    3.73   33.95   2.72   56.27   2.96   31.88   3.13
SW
```

#Finding the Shape of the Dataset

```
bird_data.shape
```

```
(420, 11)
```

```
bird_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 420 entries, 0 to 419
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
                             -----
 0   huml    419 non-null    float64
 1   humw    419 non-null    float64
 2   ulnal   417 non-null    float64
 3   ulnaw   418 non-null    float64
 4   feml    418 non-null    float64
 5   femw    419 non-null    float64
 6   tibl    418 non-null    float64
 7   tibw    419 non-null    float64
 8   tarl    419 non-null    float64
 9   tarw    419 non-null    float64
 10 type     420 non-null    object
dtypes: float64(10), object(1)
memory usage: 39.4+ KB
```

#Checking Missing Values

```
bird_data.isna().sum()
```

```
huml     1
humw     1
ulnal    3
ulnaw    2
feml     2
femw     1
tibl     2
tibw     1
tarl     1
tarw     1
type     0
dtype: int64
```

```
bird_data.dropna(how='any', inplace=True)
```

```
bird_data.isna().sum()
```

```
huml     0
humw     0
ulnal    0
ulnaw    0
feml     0
femw     0
tibl     0
tibw     0
```

```
tarl      0
tarw      0
type      0
dtype: int64
```

```
bird_data.shape
```

```
(413, 11)
```

Unique Values in the Data

```
bird_data.nunique()
```

```
huml      403
humw      319
ulnal     394
ulnaw     305
feml      397
femw      287
tibl      401
tibw      283
tarl      403
tarw      277
type        6
dtype:  int64
```

```
bird_data['type'].unique()
```

```
array(['SW', 'W', 'T', 'R', 'P', 'SO'], dtype=object)
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
bird_data[['type']] = bird_data[['type']].apply(le.fit_transform)
```

```
bird_data.head()
```

```
                                    huml  humw  ulnal  ulnaw      feml  femw  \
    tibl  tibw                      tarl  tarw  type
id

0    80.78   6.68   72.01    4.88   41.81   3.70    5.50   4.03   38.70   3.84
3
1    88.91   6.63   80.53    5.59   47.04   4.30   80.22   4.51   41.50   4.01
3
2    79.97   6.37   69.26    5.28   43.07   3.90   75.35   4.04   38.31   3.34
3
3    77.65   5.70   65.76    4.77   40.04   3.52   69.17   3.40   35.78   3.41
3
4    62.80   4.84   52.09    3.73   33.95   2.72   56.27   2.96   31.88   3.13
3
```

#Seperating Label from Data

```
y = bird_data['type']
X = bird_data.drop(['type'],axis=1)

X.columns

Index(['huml', 'humw', 'ulnal', 'ulnaw', 'feml', 'femw', 'tibl',
'tibw',
       'tarl', 'tarw'],
      dtype='object')

y

id
0       3
1       3
2       3
3       3
4       3
       ..
415     2
416     2
417     2
418     2
419     2
Name: type, Length: 413, dtype: int64

y.shape

(413,)
```

#Splitting the Data into Training and Testing

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 2)

print("Shape of the X_train", X_train.shape)
print("Shape of the X_test", X_test.shape)
print("Shape of the y_train", y_train.shape)
print("Shape of the y_test", y_test.shape)
```

```
Shape of the X_train (330, 10)
Shape of the X_test (83, 10)
Shape of the y_train (330, 6)
Shape of the y_test (83, 6)
```

#Building the ANN Model

```
# sequential model to initialise our ann and dense module to build the
layers
from keras.models import Sequential
from keras.layers import Dense

classifier = Sequential()
# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 8, kernel_initializer = 'uniform',
activation = 'relu', input_dim = 10))

# Adding the second hidden layer
classifier.add(Dense(units = 16, kernel_initializer = 'uniform',
activation = 'relu'))

# Adding the third hidden layer
classifier.add(Dense(units = 32, kernel_initializer = 'uniform',
activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform',
activation = 'softmax'))
```

```
2024-05-21 08:54:03.283163: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1926] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 17947 MB
memory:  -> device: 0, name: NVIDIA A100-SXM4-40GB MIG 3g.20gb, pci
bus id: 0000:bd:00.0, compute capability: 8.0
```

Compiling and Fitting the Model

```
classifier.compile(optimizer = 'adam', loss =
'categorical_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 16, epochs = 800,
verbose = 1)
```

```
Epoch 1/800
```

```
2024-05-21 08:54:07.013668: I
external/local_xla/xla/service/service.cc:168] XLA service
0x7fd5fd374770 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
2024-05-21 08:54:07.013710: I
external/local_xla/xla/service/service.cc:176]     StreamExecutor device
(0): NVIDIA A100-SXM4-40GB MIG 3g.20gb, Compute Capability 8.0
2024-05-21 08:54:07.019479: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269]
disabling MLIR crash reproducer, set env var
```

```
`MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
2024-05-21 08:54:07.057513: I
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:467] Loaded
cuDNN version 90000
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1716281647.144364  13834 device_compiler.h:186] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

21/21 [==============================] - 2s 2ms/step - loss: 1.7886 -
accuracy: 0.2364
Epoch 2/800
21/21 [==============================] - 0s 2ms/step - loss: 1.7772 -
accuracy: 0.3364
Epoch 3/800
21/21 [==============================] - 0s 2ms/step - loss: 1.7505 -
accuracy: 0.4364
Epoch 4/800
21/21 [==============================] - 0s 2ms/step - loss: 1.6777 -
accuracy: 0.4818
Epoch 5/800
21/21 [==============================] - 0s 2ms/step - loss: 1.5472 -
accuracy: 0.4909
Epoch 6/800
21/21 [==============================] - 0s 2ms/step - loss: 1.4310 -
accuracy: 0.4970
Epoch 7/800
21/21 [==============================] - 0s 2ms/step - loss: 1.3724 -
accuracy: 0.5000
Epoch 8/800
21/21 [==============================] - 0s 2ms/step - loss: 1.3375 -
accuracy: 0.5030
Epoch 9/800
21/21 [==============================] - 0s 2ms/step - loss: 1.3134 -
accuracy: 0.5030
Epoch 10/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2927 -
accuracy: 0.5061
Epoch 11/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2780 -
accuracy: 0.5091
Epoch 12/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2687 -
accuracy: 0.5091
Epoch 13/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2571 -
accuracy: 0.5121
Epoch 14/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2445 -
```

```
accuracy: 0.5212
Epoch 15/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2328 -
accuracy: 0.5242
Epoch 16/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2212 -
accuracy: 0.5273
Epoch 17/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2172 -
accuracy: 0.5273
Epoch 18/800
21/21 [==============================] - 0s 2ms/step - loss: 1.2037 -
accuracy: 0.5303
Epoch 19/800
21/21 [==============================] - 0s 2ms/step - loss: 1.1877 -
accuracy: 0.5303
Epoch 20/800
21/21 [==============================] - 0s 2ms/step - loss: 1.1763 -
accuracy: 0.5333
Epoch 21/800
21/21 [==============================] - 0s 2ms/step - loss: 1.1673 -
accuracy: 0.5364
Epoch 22/800
21/21 [==============================] - 0s 2ms/step - loss: 1.1575 -
accuracy: 0.5394
Epoch 23/800
21/21 [==============================] - 0s 2ms/step - loss: 1.1437 -
accuracy: 0.5424
Epoch 24/800
21/21 [==============================] - 0s 2ms/step - loss: 1.1276 -
accuracy: 0.5424
Epoch 25/800
21/21 [==============================] - 0s 2ms/step - loss: 1.1143 -
accuracy: 0.5455
Epoch 26/800
21/21 [==============================] - 0s 2ms/step - loss: 1.1014 -
accuracy: 0.5455
Epoch 27/800
21/21 [==============================] - 0s 2ms/step - loss: 1.0870 -
accuracy: 0.5455
Epoch 28/800
21/21 [==============================] - 0s 2ms/step - loss: 1.0758 -
accuracy: 0.5455
Epoch 29/800
21/21 [==============================] - 0s 2ms/step - loss: 1.0680 -
accuracy: 0.5455
Epoch 30/800
21/21 [==============================] - 0s 2ms/step - loss: 1.0548 -
accuracy: 0.5485
```

```
accuracy: 0.9606
Epoch 784/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1375 -
accuracy: 0.9545
Epoch 785/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1342 -
accuracy: 0.9485
Epoch 786/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1369 -
accuracy: 0.9545
Epoch 787/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1419 -
accuracy: 0.9636
Epoch 788/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1409 -
accuracy: 0.9545
Epoch 789/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1464 -
accuracy: 0.9485
Epoch 790/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1528 -
accuracy: 0.9515
Epoch 791/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1446 -
accuracy: 0.9455
Epoch 792/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1383 -
accuracy: 0.9576
Epoch 793/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1366 -
accuracy: 0.9576
Epoch 794/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1394 -
accuracy: 0.9576
Epoch 795/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1360 -
accuracy: 0.9545
Epoch 796/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1376 -
accuracy: 0.9576
Epoch 797/800
21/21 [==============================] - 0s 2ms/step - loss: 0.1394 -
accuracy: 0.9576
Epoch 798/800
21/21 [==============================] - 0s 2ms/step - loss: 0.2885 -
accuracy: 0.9212
Epoch 799/800
21/21 [==============================] - 0s 2ms/step - loss: 0.2031 -
accuracy: 0.9273
Epoch 800/800
```

```
21/21 [==============================] - 0s 2ms/step - loss: 0.1498 -
accuracy: 0.9485

<keras.src.callbacks.History at 0x7fdf783122c0>
```

#Testing the Model

```
score, acc = classifier.evaluate(X_train, y_train,
                            batch_size=10)
print('Train score:', score)
print('Train accuracy:', acc)

print('*'*20)
score, acc = classifier.evaluate(X_test, y_test,
                            batch_size=10)
print('Test score:', score)
print('Test accuracy:', acc)

33/33 [==============================] - 0s 999us/step - loss: 0.1421
- accuracy: 0.9545
Train score: 0.14205241203308105
Train accuracy: 0.9545454382896423
********************
9/9 [==============================] - 0s 1ms/step - loss: 0.4386 -
accuracy: 0.8916
Test score: 0.4385785162448883
Test accuracy: 0.891566276550293
```

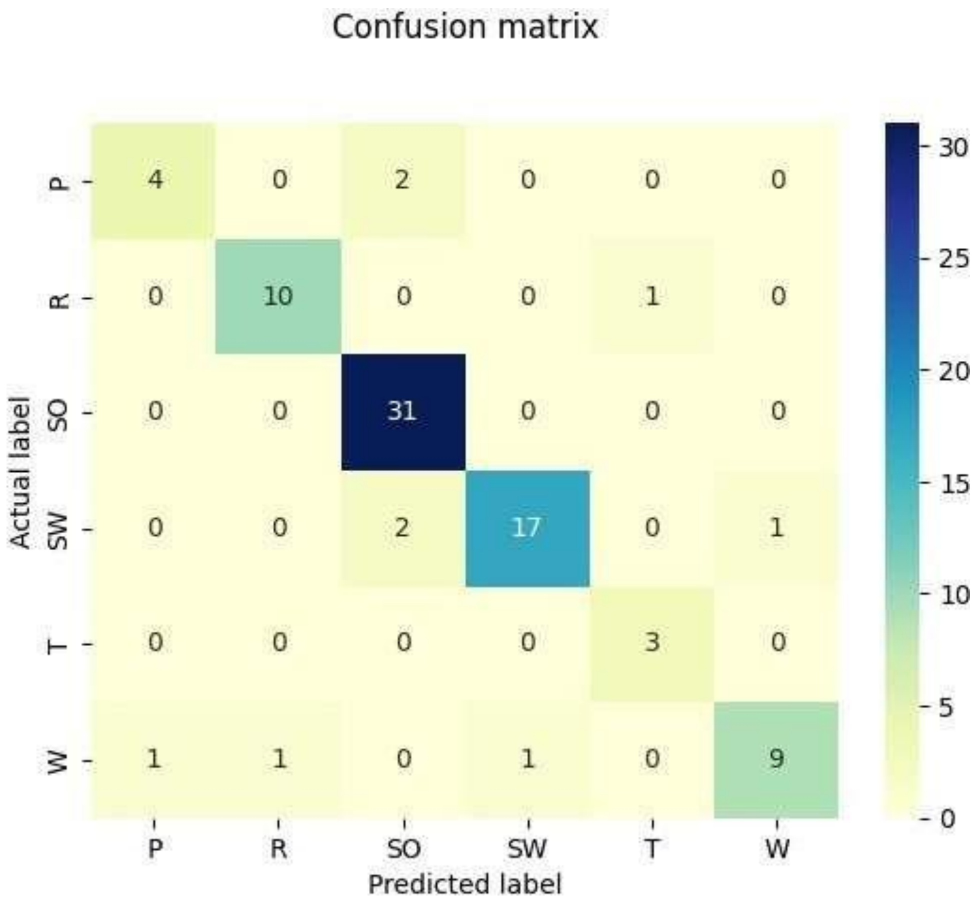
```python
# Predicting the Test set results
pred = classifier.predict(X_test)
print("Y_pred:", pred)
print("*****************")
y_pred = np.argmax(pred, axis = 1)
print("Y_pred:", y_pred)
print("*****************")
print("Y_test:", y_test)
y_true = np.argmax(y_test, axis = 1)
print("*****************")
print("Y_test:", y_true)
```

```
3/3 [==============================] - 0s 1ms/step
Y pred: [[1.13437502e-10 1.58537  41e-13 9.99999762e-01 1.9958  027e-07
   1.19342267e-13 7.11663781e-17]
  [2.60476186e-03 9.92207229e-01  1.64550258e-07 8.11149876e-05
   1.85898723e-04 4.92081419e-03]
  [2.14676633e-02 2.31342129e-02  3.83366924e-03 7.04811055e-06
   9.51290905e-01 2.66507734e-04]
  [5.19312546e-02 6.41205178e-09  9.46038127e-01 2.02322891e-03
   1.39814540e-06 5.98970792e-06]
  [9.30234570e-11 5.98462997e-14  4.77789929e-30 3.15114677e-01
   0.00000000e+00 6.84885323e-01]
  [1.88400315e-06 6.03930058e-21  4.66691310e-15 9.15600002e-01
   3.42472671e-25 8.43981877e-02]
  [1.35823920e-01 5.13227701e-01  2.56901711e-01 3.06145989e-06
   9.39941630e-02 4.94873129e-05]
  [3.26984525e-01 3.14950244e-09  6.71750968e-05 1.57738656e-01
   3.85546102e-08 5.15209615e-01]
  [3.84205058e-02 7.30247200e-01  5.26678741e-06 1.89626201e-06
   1.04787380e-01 1.26537830e-01]
  [3.14090314e-04 8.32000315e-01  4.38185356e-11 1.05659328e-01
   1.07772495e-13 6.20262362e-02]
  [8.62169103e-12 1.65840709e-11  1.00000000e+00 3.86799854e-08
   1.72668073e-13 8.60259400e-18]
  [2.11188064e-11 4.37313232e-20  7.18141360e-29 9.77322459e-01
   0.00000000e+00 2.26775333e-02]
  [4.72837513e-07 3.80029655e-06  9.99995351e-01 3.83343348e-07
   5.30217150e-08 9.50168676e-12]
  [9.80835333e-02 2.61182129e-01  6.15853071e-01 8.95882567e-06
   2.48215068e-02 5.08308731e-05]
  [8.58899879e-15 2.80578638e-17  1.00000000e+00 3.12439283e-08
   6.21814207e-18 7.50081058e-22]
  [1.29032907e-09 3.42193189e-08  1.00000000e+00 5.48537127e-09
   2.02274794e-10 1.47560022e-15]
  [7.71484792e-01 3.33889749e-10  3.85360676e-04 7.93045461e-02
   1.71934516e-06 1.48823529e-01]
  [3.79146299e-38 0.00000000e+00  0.00000000e+00 1.00000000e+00
   0.00000000e+00 2.19223724e-17]
  [1.42191124e-15 1.41321845e-34  3.30129911e-33 9.99977827e-01
```

```
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]]
****************
Y_test: [2 1 4 0 5 3 1 5 1 1 2 3 2 0 2 2 5 3 3 4 2 3 2 3 2 3 2 2 2 5 1
2 3 5 5 3 3
 2 2 2 0 3 3 1 2 3 3 1 2 0 3 2 3 1 5 2 2 2 2 2 4 2 5 1 0 2 2 0 5 3 2 1
5 3
 1 3 5 5 2 3 2 2 2]
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
target_names = ['P', 'R', 'SO', 'SW', 'T', 'W']

import matplotlib.pyplot as plt
import seaborn as sns

p = sns.heatmap(pd.DataFrame(cm), annot=True,xticklabels=target_names,
yticklabels=target_names,  cmap="YlGnBu"  ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 23.52222222222222, 'Predicted label')
```




```
#import classification_report
from sklearn.metrics import classification_report
print(classification_report(y_true,y_pred, target_names =
target_names))
```

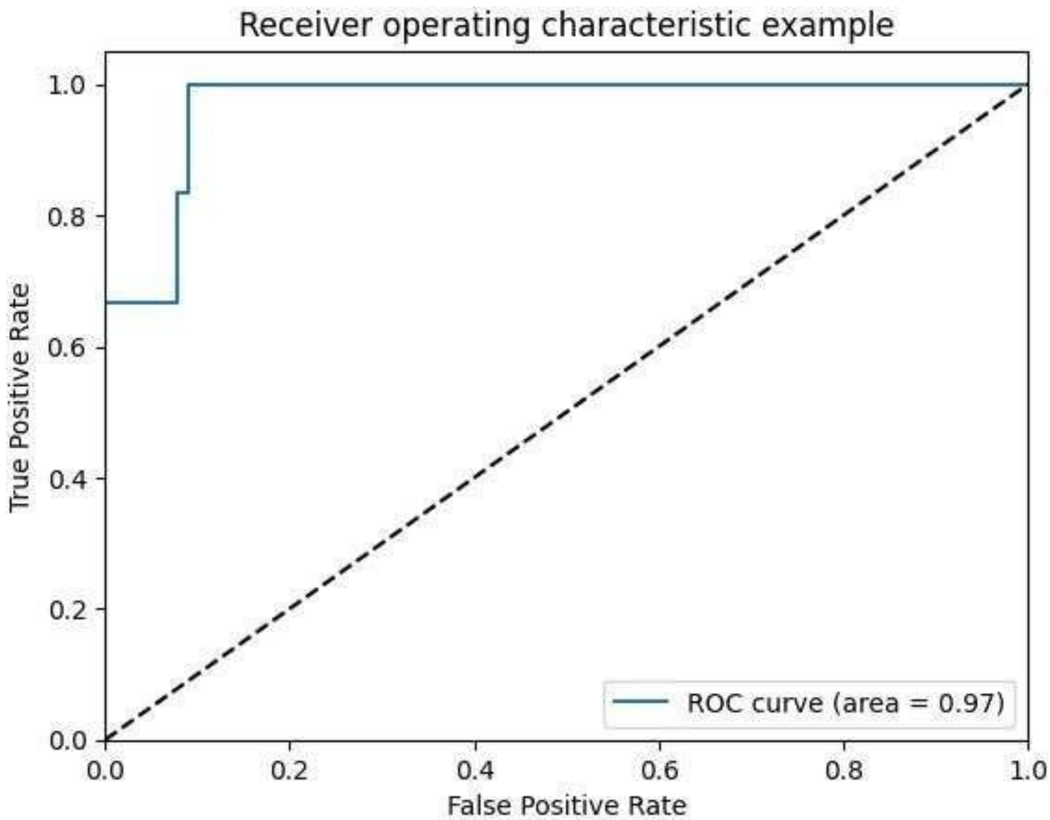
```
              precision    recall  f1-score   support

           P       0.80      0.67      0.73         6
           R       0.91      0.91      0.91        11
          SO       0.89      1.00      0.94        31
          SW       0.94      0.85      0.89        20
           T       0.75      1.00      0.86         3
           W       0.90      0.75      0.82        12

    accuracy                           0.89        83
   macro avg       0.86      0.86      0.86        83
weighted avg       0.89      0.89      0.89        83
```

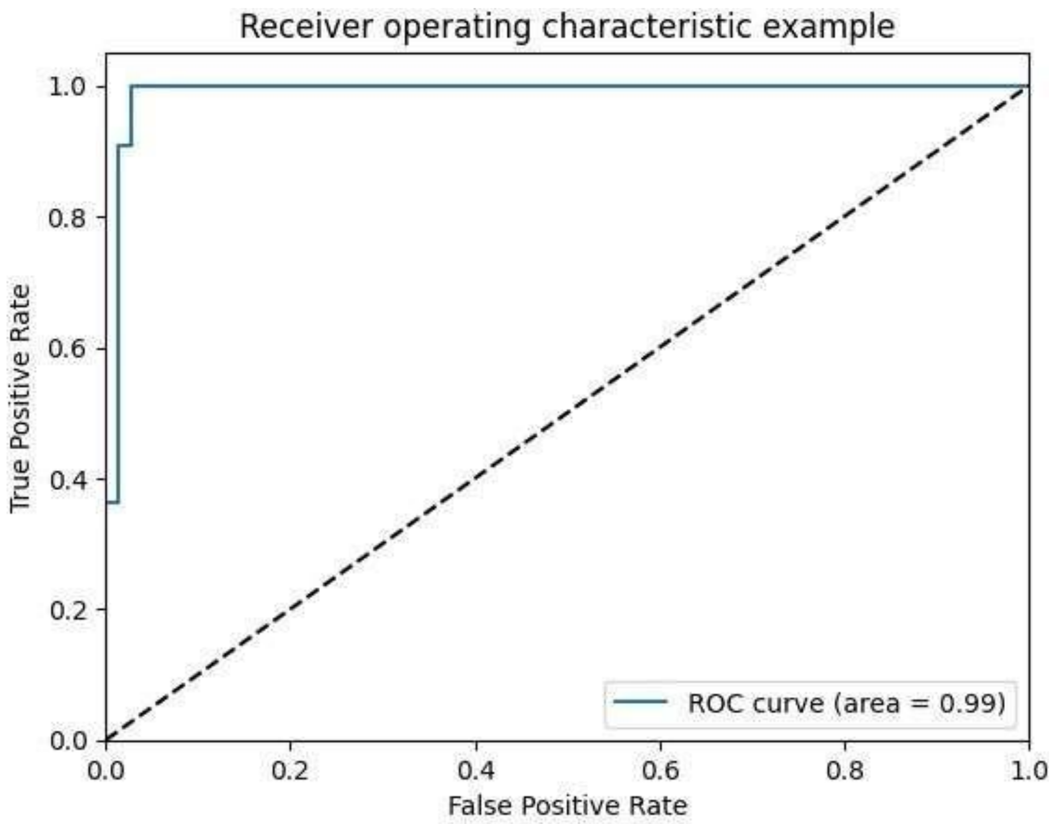
#

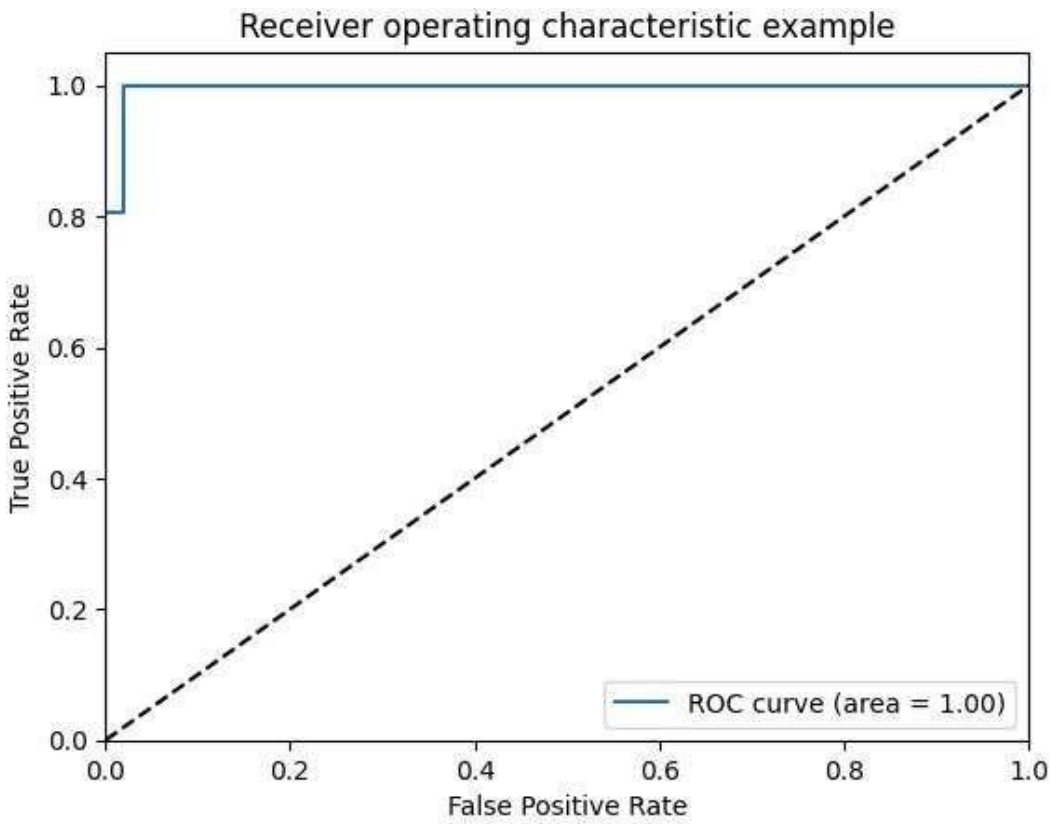
ROC curve

```
from sklearn.metrics import roc_curve, auc
from itertools import cycle
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(6):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

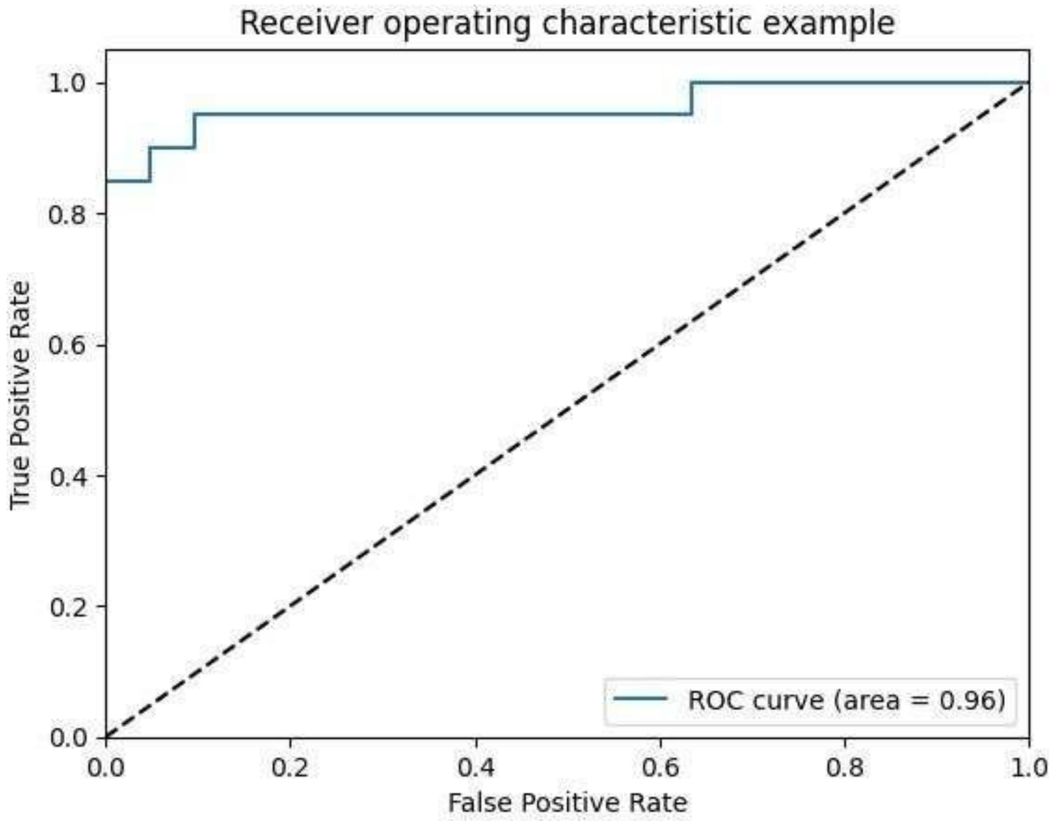
```python
# Plot of a ROC curve for a specific class
for i in range(6):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % 
roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

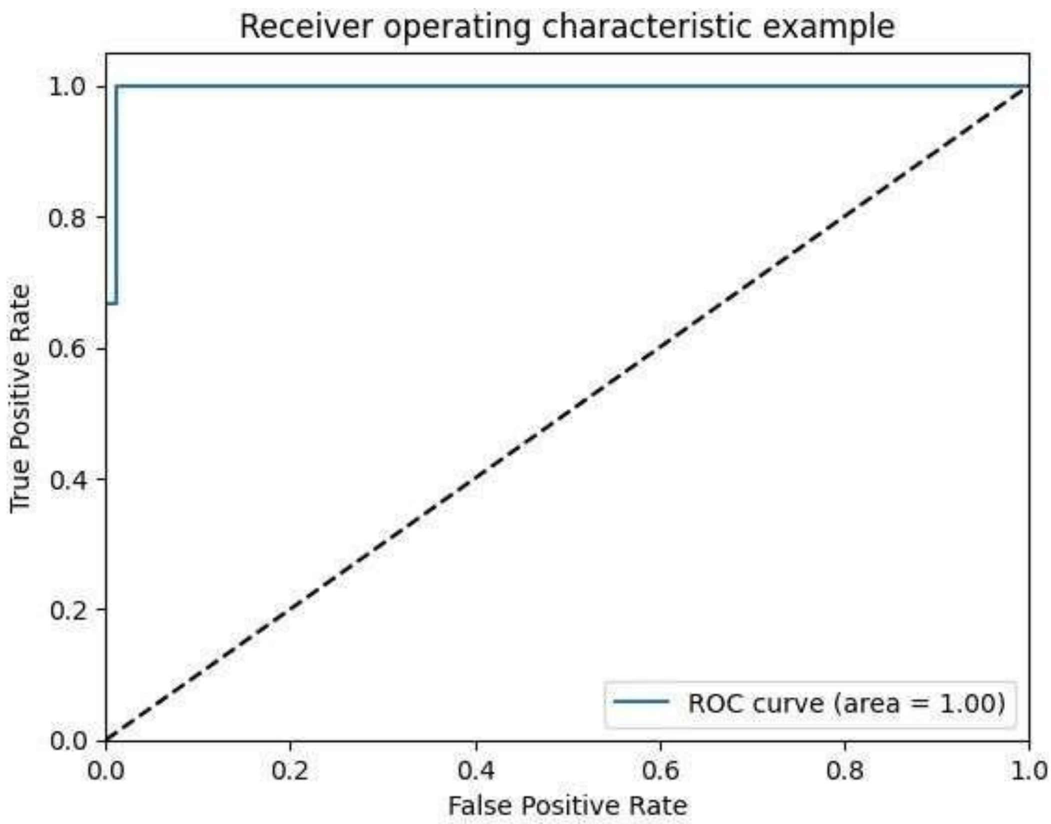



Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0

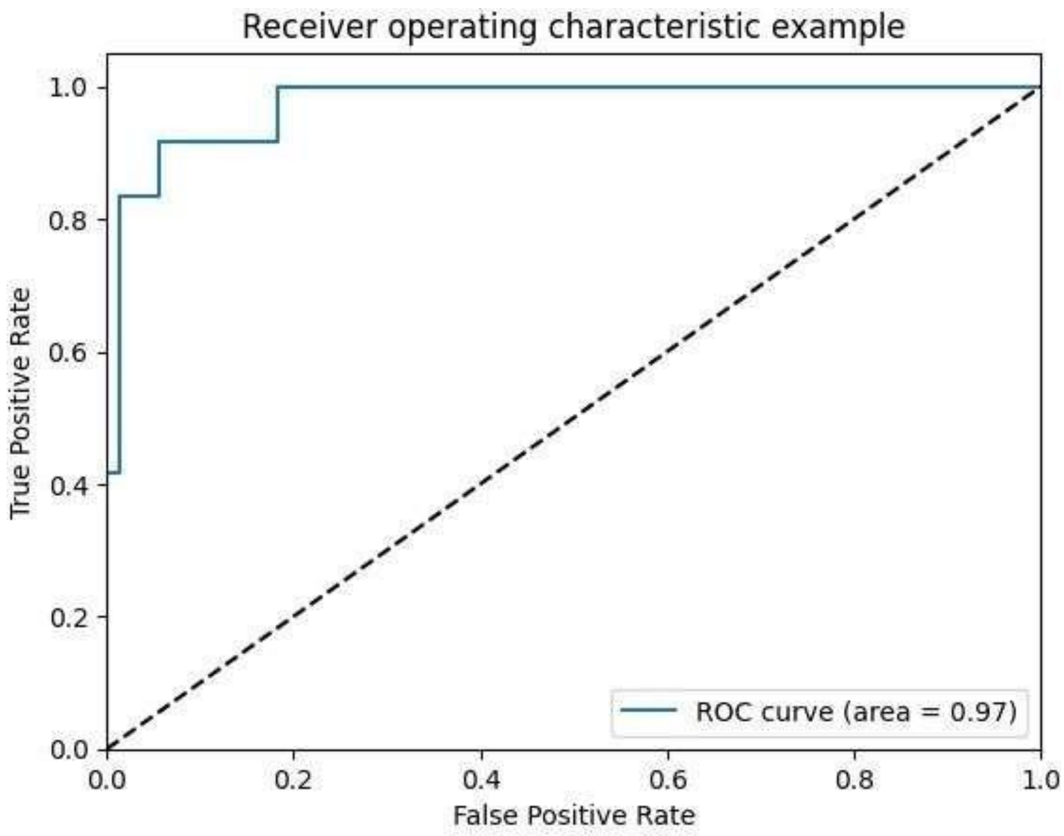


Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

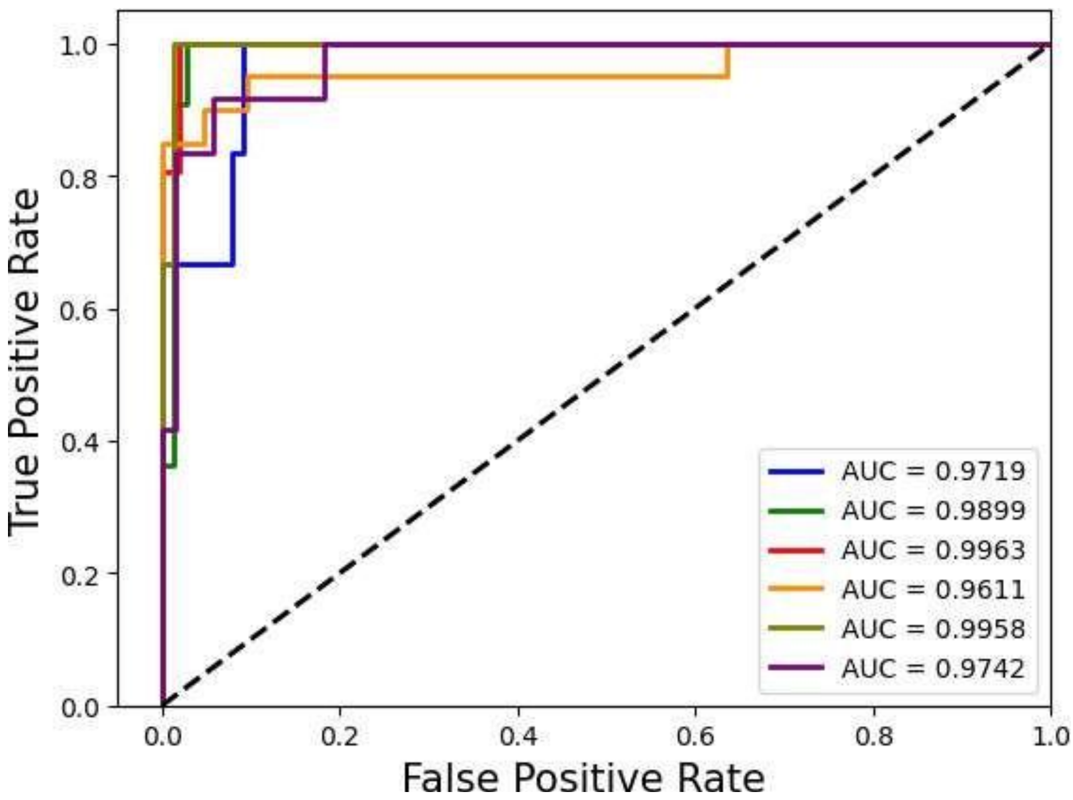

Receiver operating characteristic example
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.96)



Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0




```
fpr = dict()
tpr = dict()
roc_auc = dict()
lw=2
for i in range(6):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
colors =cycle(['blue', 'green', 'red','darkorange','olive','purple'])
for i, color in zip(range(6), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='AUC = {1:0.4f}'
             ''.format(i, roc_auc[i]))
plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate',fontsize=15)
plt.ylabel('True Positive Rate',fontsize=15)
# plt.title('Receiver operating characteristic for multi-class data')
plt.legend(loc="lower right")
plt.show()
```



True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.9719
AUC = 0.9899
AUC = 0.9963
AUC = 0.9611
AUC = 0.9958
AUC = 0.9742

Experiment - 3

Creating CNN Models from Scratch, Compiling of CNN Models, Training and Testing, Plotting of Curves

Importing Necessary libraries

```
from tensorflow.keras import applications
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dropout, Flatten, Dense,
GlobalAveragePooling2D
from tensorflow.keras import backend as k
from tensorflow.keras.callbacks import ModelCheckpoint,
LearningRateScheduler, TensorBoard, EarlyStopping

import numpy as np
from tensorflow.keras import models
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras.layers import  Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras import Input
```

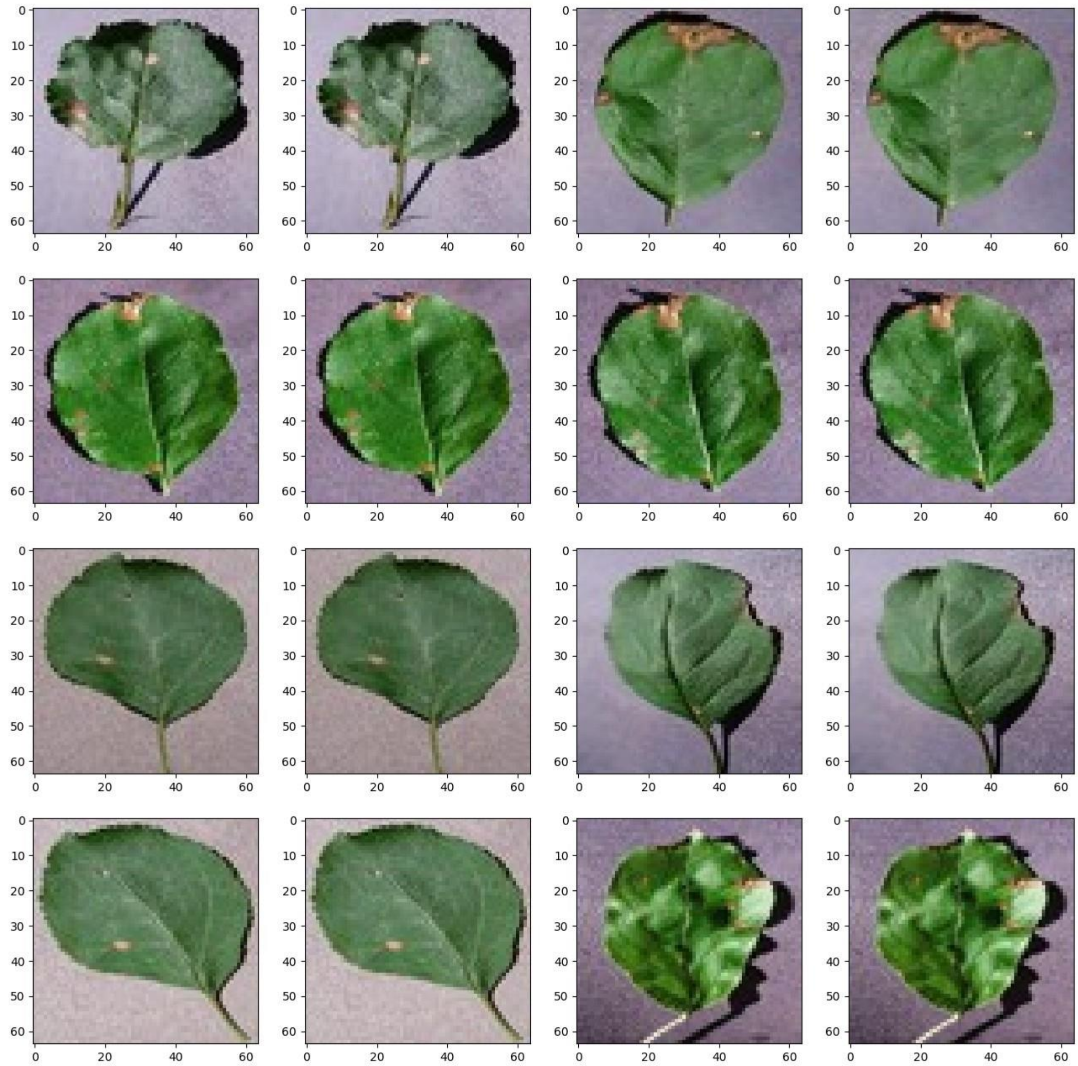
Loading the training and testing data and defining the basic parameters

```
train_datagen = ImageDataGenerator(rescale=1./255) # vertical_flip=True,
                                                    #  horizontal_flip=True,
                                                    # height_shift_range=0.1,
                                                    # width_shift_range=0.1
validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Read the training sample and set the batch size
train_generator = train_datagen.flow_from_directory(
        'C://Users//abhia//Downloads//plant_village
(1)//plant_village/t/rain',
        target_size=(64, 64),
        batch_size=16,
        class_mode='categorical')

# Read Validation data from directory and define target size with batch size
validation_generator = validation_datagen.flow_from_directory(
        '/workspace/Bootcamp/Data/plant_village/val/',
        target_size=(64, 64),
        batch_size=16,
```

```
        class_mode='categorical',
        shuffle=False)

test_generator = test_datagen.flow_from_directory(
        '/workspace/Bootcamp/Data/plant_village/test/',
        target_size=(64, 64),
        batch_size=1,
        class_mode='categorical',
        shuffle=False)
```

Visualization of few images

```
plt.figure(figsize=(16, 16))
for i in range(1, 17):
  plt.subplot(4, 4, i)
  img, label = test_generator.next()
  # print(img.shape)
  # print(label)
  plt.imshow(img[0])
plt.show()
```



```
img, label = test_generator.next()
img[0].shape
```

```
(64, 64, 3)
```

```
# Create the model
model = models.Sequential()
# Add new layers
model.add(Conv2D(128, kernel_size=(3,3), activation = 'relu',
input_shape=(64,64,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, kernel_size=(3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, kernel_size=(3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
```

```
model.add(layers.Dense(4, activation='softmax'))
model.summary()
```

Compiling and Training the Model

```
model.compile(optimizer = optimizers.Adam(learning_rate = 0.0001),
loss='categorical_crossentropy', metrics=['acc'])
# Train the model
history = model.fit(train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size,
      epochs=30,
      validation_data=validation_generator,

validation_steps=validation_generator.samples/validation_generator.batch_size
,
      verbose=2)
```

```
Epoch 1/30
Epoch 1/30
188/187 - 2s - loss: 0.0354 - acc: 0.9893 - val_loss: 0.2115 - val_acc:
0.9402
Epoch 2/30
Epoch 1/30
188/187 - 2s - loss: 0.0313 - acc: 0.9890 - val_loss: 0.2183 - val_acc:
0.9339
Epoch 3/30
Epoch 1/30
188/187 - 2s - loss: 0.0292 - acc: 0.9903 - val_loss: 0.2038 - val_acc:
0.9465
Epoch 4/30
Epoch 1/30
188/187 - 2s - loss: 0.0314 - acc: 0.9907 - val_loss: 0.2843 - val_acc:
0.9181
Epoch 5/30
Epoch 1/30
188/187 - 2s - loss: 0.0213 - acc: 0.9940 - val_loss: 0.3160 - val_acc:
0.9181
Epoch 6/30
Epoch 1/30
188/187 - 2s - loss: 0.0276 - acc: 0.9923 - val_loss: 0.2019 - val_acc:
0.9449
Epoch 7/30
Epoch 1/30
188/187 - 2s - loss: 0.0184 - acc: 0.9960 - val_loss: 0.2326 - val_acc:
0.9307
Epoch 8/30
Epoch 1/30
188/187 - 2s - loss: 0.0161 - acc: 0.9970 - val_loss: 0.2079 - val_acc:
0.9386
Epoch 9/30
```

```
Epoch 1/30
188/187 - 2s - loss: 0.0145 - acc: 0.9980 - val_loss: 0.2025 - val_acc:
0.9465
Epoch 10/30
Epoch 1/30
188/187 - 2s - loss: 0.0180 - acc: 0.9953 - val_loss: 0.2236 - val_acc:
0.9386
Epoch 11/30
Epoch 1/30
188/187 - 2s - loss: 0.0152 - acc: 0.9970 - val_loss: 0.2413 - val_acc:
0.9339
Epoch 12/30
Epoch 1/30
188/187 - 2s - loss: 0.0245 - acc: 0.9913 - val_loss: 0.1965 - val_acc:
0.9465
Epoch 13/30
Epoch 1/30
188/187 - 2s - loss: 0.0202 - acc: 0.9940 - val_loss: 0.3189 - val_acc:
0.9228
Epoch 14/30
Epoch 1/30
188/187 - 2s - loss: 0.0136 - acc: 0.9970 - val_loss: 0.1991 - val_acc:
0.9449
Epoch 15/30
Epoch 1/30
188/187 - 2s - loss: 0.0083 - acc: 0.9983 - val_loss: 0.2098 - val_acc:
0.9402
Epoch 16/30
Epoch 1/30
188/187 - 2s - loss: 0.0081 - acc: 0.9993 - val_loss: 0.2170 - val_acc:
0.9496
Epoch 17/30
Epoch 1/30
188/187 - 2s - loss: 0.0104 - acc: 0.9980 - val_loss: 0.2084 - val_acc:
0.9480
Epoch 18/30
Epoch 1/30
188/187 - 2s - loss: 0.0070 - acc: 0.9993 - val_loss: 0.1953 - val_acc:
0.9480
Epoch 19/30
Epoch 1/30
188/187 - 2s - loss: 0.0578 - acc: 0.9804 - val_loss: 0.4641 - val_acc:
0.8898
Epoch 20/30
Epoch 1/30
188/187 - 2s - loss: 0.0172 - acc: 0.9950 - val_loss: 0.2292 - val_acc:
0.9417
Epoch 21/30
Epoch 1/30
188/187 - 2s - loss: 0.0063 - acc: 0.9993 - val_loss: 0.2090 - val_acc:
```

```
0.9449
Epoch 22/30
Epoch 1/30
188/187 - 2s - loss: 0.0047 - acc: 1.0000 - val_loss: 0.1988 - val_acc:
0.9449
Epoch 23/30
Epoch 1/30
188/187 - 2s - loss: 0.0035 - acc: 1.0000 - val_loss: 0.2330 - val_acc:
0.9449
Epoch 24/30
Epoch 1/30
188/187 - 2s - loss: 0.0031 - acc: 0.9997 - val_loss: 0.2031 - val_acc:
0.9496
Epoch 25/30
Epoch 1/30
188/187 - 2s - loss: 0.0037 - acc: 1.0000 - val_loss: 0.2201 - val_acc:
0.9480
Epoch 26/30
Epoch 1/30
188/187 - 2s - loss: 0.0041 - acc: 0.9990 - val_loss: 0.2835 - val_acc:
0.9276
Epoch 27/30
Epoch 1/30
188/187 - 2s - loss: 0.0510 - acc: 0.9827 - val_loss: 0.2389 - val_acc:
0.9386
Epoch 28/30
Epoch 1/30
188/187 - 2s - loss: 0.0063 - acc: 0.9993 - val_loss: 0.2344 - val_acc:
0.9465
Epoch 29/30
Epoch 1/30
188/187 - 2s - loss: 0.0030 - acc: 1.0000 - val_loss: 0.2219 - val_acc:
0.9480
Epoch 30/30
Epoch 1/30
188/187 - 2s - loss: 0.0029 - acc: 1.0000 - val_loss: 0.2166 - val_acc:
0.9449
```

```
model.save("CONV_plant_deseas.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

```
model = models.load_model('CONV_plant_deseas.h5')
```

Visualization of Accuracy and Loss Curves

```
train_acc = history.history['acc']
val_acc = history.history['val_acc']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(train_acc))
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'g', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.grid()
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'g', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.grid()
plt.legend()
plt.show()
```

Prediction

```
fnames = test_generator.filenames
ground_truth = test_generator.classes
label2index = test_generator.class_indices

idx2label = dict((v,k) for k,v in label2index.items())

# Get the predictions from the model using the generator
predictions = model.predict_generator(test_generator,
steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes = np.argmax(predictions,axis=1)

errors = np.where(predicted_classes != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples))
```

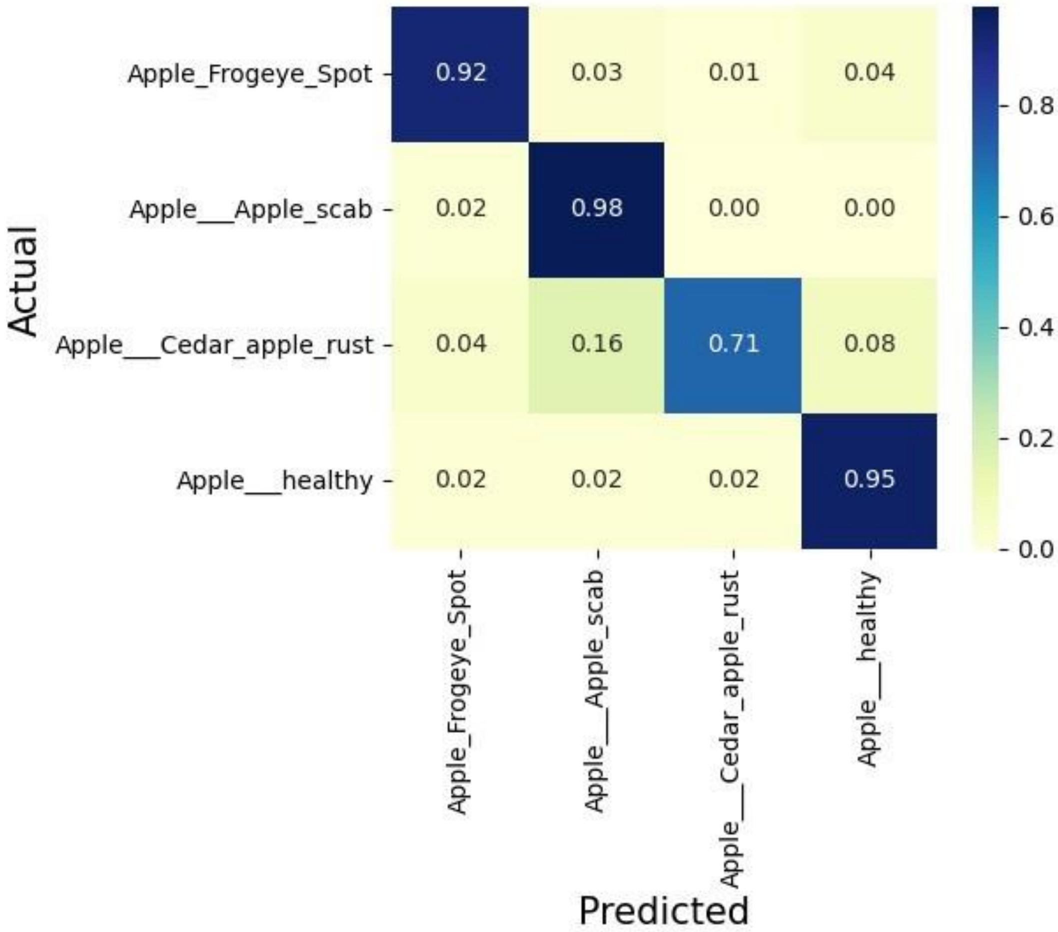
```
546/546 [==============================] - 1s 1ms/step
No of errors = 38/546
```

```
accuracy = ((test_generator.samples-len(errors))/test_generator.samples) *
100
accuracy
```

```
93.04029304029304
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
cm = confusion_matrix(y_true=ground_truth, y_pred=predicted_classes)
cm = np.array(cm)
# Normalise
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(5,4))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=label2index,
yticklabels=label2index, cmap="YlGnBu")
plt.ylabel('Actual', fontsize=15)
plt.xlabel('Predicted', fontsize=15)
plt.show(block=False)
```



Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(ground_truth, predicted_classes,
target_names=label2index))
```

```
                          precision    recall  f1-score   support

      Apple_Frogeye_Spot       0.91      0.92      0.92       103
      Apple___Apple_scab       0.90      0.98      0.94       134
Apple___Cedar_apple_rust       0.85      0.71      0.78        49
         Apple___healthy       0.97      0.95      0.96       260

                accuracy                           0.93       546
               macro avg       0.91      0.89      0.90       546
            weighted avg       0.93      0.93      0.93       546
```

Experiment - 4

Deep Learning Training and Architecture, Feature Extraction, Models training with some pretrained models.

Importing Necessary libraries

```
import numpy as np
from tensorflow.keras import Input
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras.models import Model
from tensorflow.keras import applications
from tensorflow.keras import backend as k
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.layers import  Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dropout, Flatten, Dense,
GlobalAveragePooling2D
from tensorflow.keras.callbacks import ModelCheckpoint,
LearningRateScheduler, TensorBoard, EarlyStopping
```

Loading the Training and Testing Data and Defining the Basic Parameters

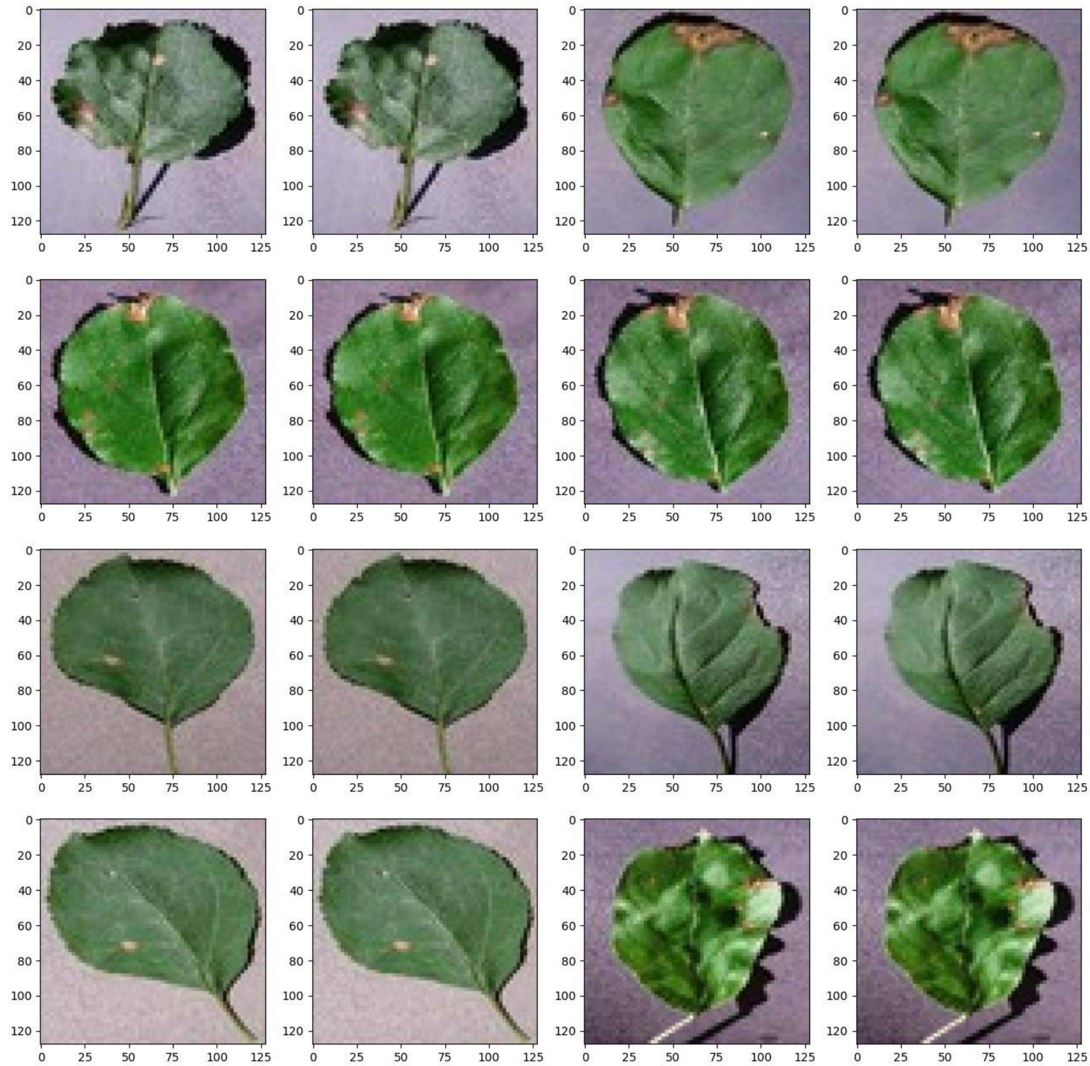
```
train_datagen = ImageDataGenerator(rescale=1./255) # vertical_flip=True,
                                                    #  horizontal_flip=True,
                                                    # height_shift_range=0.1,
                                                    # width_shift_range=0.1
validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Read the training sample and set the batch size
train_generator = train_datagen.flow_from_directory(
        '/workspace/Bootcamp/Data/plant_village/train/',
        target_size=(128, 128),
        batch_size=16,
        class_mode='categorical')

# Read Validation data from directory and define target size with batch size
validation_generator = validation_datagen.flow_from_directory(
        '/content/plant_village/val/',
```

```
        target_size=(128, 128),
        batch_size=16,
        class_mode='categorical',
        shuffle=False)

test_generator = test_datagen.flow_from_directory(
        '/content/plant_village/test/',
        target_size=(128, 128),
        batch_size=1,
        class_mode='categorical',
        shuffle=False)
```

```
Found 3004 images belonging to 4 classes.
Found 635 images belonging to 4 classes.
Found 547 images belonging to 4 classes.
```

Visualization of Few Images

```
plt.figure(figsize=(16, 16))
for i in range(1, 17):
  plt.subplot(4, 4, i)
  img, label = test_generator.next()
  # print(img.shape)
  # print(label)
  plt.imshow(img[0])
plt.show()
```



```
img, label = test_generator.next()
img[0].shape
```

```
(128, 128, 3)
```

Exploring Keras Applications for Transfer Learning

```
from tensorflow.keras.applications.vgg16 import VGG16

## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape= (128,
128, 3))
# Include_top = False means excluding the model fully connected layers
base_model.trainable = False ## Not trainable weights,
```

```
#weights of the VGG16 model will not be updated during training
base_model.summary()

Model: "vgg16"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_2 (InputLayer)        [(None, 128, 128, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

Adding top layers according to number of classes in our data

```
flatten_layer = layers.GlobalAveragePooling2D()
# dense_layer_1 = layers.Dense(64, activation='relu')
# dense_layer_2 = layers.Dense(32, activation='relu')
prediction_layer = layers.Dense(4, activation='softmax')

model = models.Sequential([
    base_model,
    flatten_layer,
    prediction_layer
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 4)                 2052
=================================================================
Total params: 14,716,740
Trainable params: 2,052
Non-trainable params: 14,714,688
_________________________________________________________________
```

Training

```
# sgd = SGD(lr=0.001,decay=1e-6, momentum=0.9, nesterov=True)
# We are going to use accuracy metrics and cross entropy loss as performance parameters
model.compile(optimizer = Adam(learning_rate = 0.0001),
loss='categorical_crossentropy', metrics=['acc'])
# Train the model
history = model.fit(train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size,
      epochs=30,
      validation_data=validation_generator,

validation_steps=validation_generator.samples/validation_generator.batch_size
,
      verbose=1)
```

```
Epoch 1/30
186/187 [============================>.] - ETA: 0s - loss: 0.1192 - acc:
0.9707Epoch 1/30
188/187 [==============================] - 6s 34ms/step - loss: 0.1190 - acc:
```

```
0.9707 - val_loss: 0.1386 - val_acc: 0.9559
Epoch 2/30
186/187 [============================>.] - ETA: 0s - loss: 0.1190 - acc:
0.9701Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1184 - acc:
0.9704 - val_loss: 0.1364 - val_acc: 0.9559
Epoch 3/30
186/187 [============================>.] - ETA: 0s - loss: 0.1184 - acc:
0.9711Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1180 - acc:
0.9714 - val_loss: 0.1360 - val_acc: 0.9559
Epoch 4/30
186/187 [============================>.] - ETA: 0s - loss: 0.1170 - acc:
0.9704Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1175 - acc:
0.9704 - val_loss: 0.1361 - val_acc: 0.9559
Epoch 5/30
186/187 [============================>.] - ETA: 0s - loss: 0.1168 - acc:
0.9694Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1172 - acc:
0.9690 - val_loss: 0.1367 - val_acc: 0.9575
Epoch 6/30
186/187 [============================>.] - ETA: 0s - loss: 0.1169 - acc:
0.9704Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1169 - acc:
0.9704 - val_loss: 0.1355 - val_acc: 0.9575
Epoch 7/30
186/187 [============================>.] - ETA: 0s - loss: 0.1162 - acc:
0.9704Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1164 - acc:
0.9704 - val_loss: 0.1348 - val_acc: 0.9559
Epoch 8/30
186/187 [============================>.] - ETA: 0s - loss: 0.1162 - acc:
0.9707Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1164 - acc:
0.9707 - val_loss: 0.1354 - val_acc: 0.9559
Epoch 9/30
186/187 [============================>.] - ETA: 0s - loss: 0.1163 - acc:
0.9711Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1162 - acc:
0.9710 - val_loss: 0.1349 - val_acc: 0.9575
Epoch 10/30
186/187 [============================>.] - ETA: 0s - loss: 0.1155 - acc:
0.9704Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1157 - acc:
0.9704 - val_loss: 0.1349 - val_acc: 0.9575
Epoch 11/30
186/187 [============================>.] - ETA: 0s - loss: 0.1157 - acc:
0.9711Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1155 - acc:
```

```
0.9710 - val_loss: 0.1325 - val_acc: 0.9559
Epoch 22/30
186/187 [============================>.] - ETA: 0s - loss: 0.1127 - acc:
0.9714Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1123 - acc:
0.9717 - val_loss: 0.1333 - val_acc: 0.9591
Epoch 23/30
186/187 [============================>.] - ETA: 0s - loss: 0.1108 - acc:
0.9714Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1120 - acc:
0.9707 - val_loss: 0.1331 - val_acc: 0.9591
Epoch 24/30
186/187 [============================>.] - ETA: 0s - loss: 0.1117 - acc:
0.9717Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1117 - acc:
0.9714 - val_loss: 0.1324 - val_acc: 0.9559
Epoch 25/30
186/187 [============================>.] - ETA: 0s - loss: 0.1111 - acc:
0.9711Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1114 - acc:
0.9710 - val_loss: 0.1318 - val_acc: 0.9559
Epoch 26/30
186/187 [============================>.] - ETA: 0s - loss: 0.1114 - acc:
0.9707Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1113 - acc:
0.9710 - val_loss: 0.1314 - val_acc: 0.9559
Epoch 27/30
186/187 [============================>.] - ETA: 0s - loss: 0.1093 - acc:
0.9727Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1109 - acc:
0.9717 - val_loss: 0.1318 - val_acc: 0.9559
Epoch 28/30
186/187 [============================>.] - ETA: 0s - loss: 0.1108 - acc:
0.9721Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1106 - acc:
0.9720 - val_loss: 0.1307 - val_acc: 0.9575
Epoch 29/30
186/187 [============================>.] - ETA: 0s - loss: 0.1104 - acc:
0.9711Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1104 - acc:
0.9710 - val_loss: 0.1322 - val_acc: 0.9575
Epoch 30/30
186/187 [============================>.] - ETA: 0s - loss: 0.1096 - acc:
0.9738Epoch 1/30
188/187 [==============================] - 6s 32ms/step - loss: 0.1101 - acc:
0.9734 - val_loss: 0.1305 - val_acc: 0.9559
```

Saving the model

```
model.save("VGG16_plant_deseas.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

Loading the model

```
model = models.load_model('VGG16_plant_deseas.h5')
print("Model is loaded")
```

```
Model is loaded
```

Saving the Weights

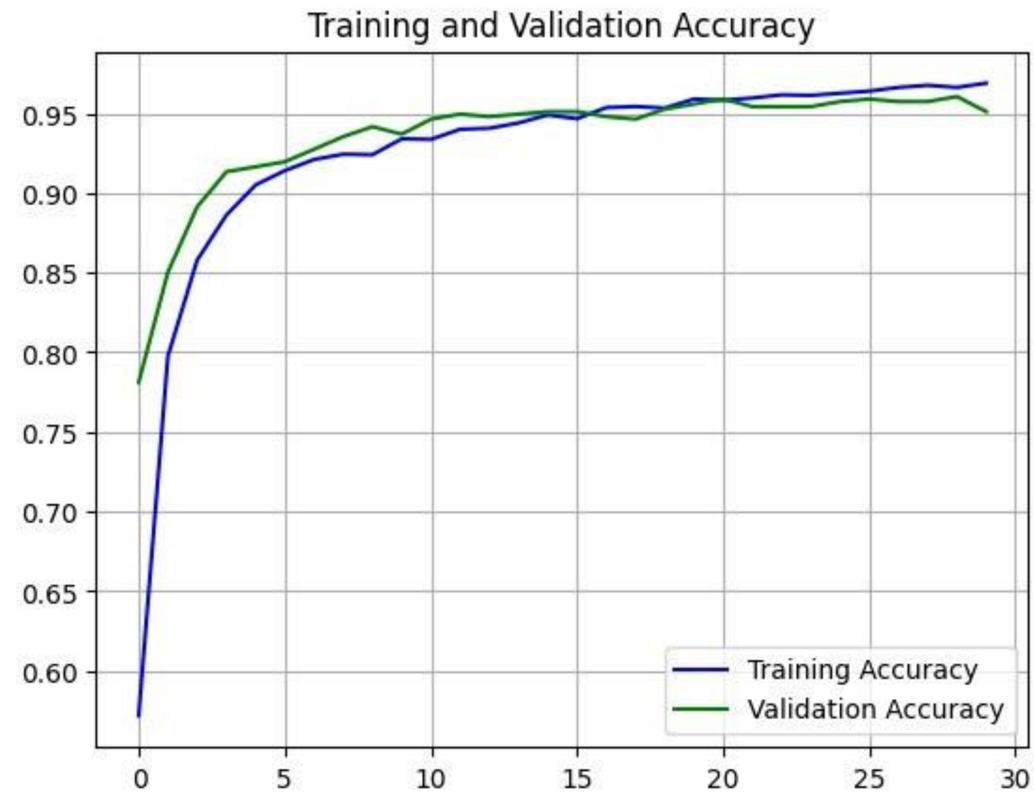
```
model.save_weights('cnn_classification.h5')
```

Loading the weights

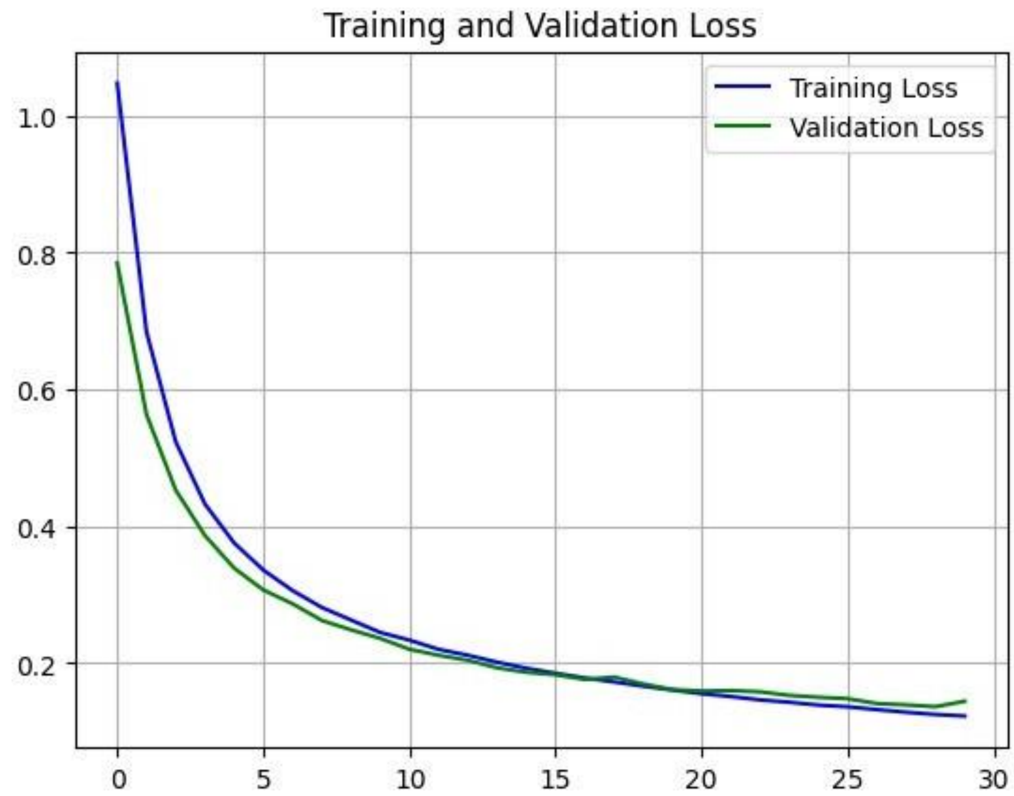
```
model.load_weights('cnn_classification.h5')
```

Visualization of training over epoch

```
train_acc = history.history['acc']
val_acc = history.history['val_acc']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(train_acc))
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'g', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.grid()
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'g', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.grid()
plt.legend()
plt.show()
```



<Figure size 640x480 with 0 Axes>



Performance measure

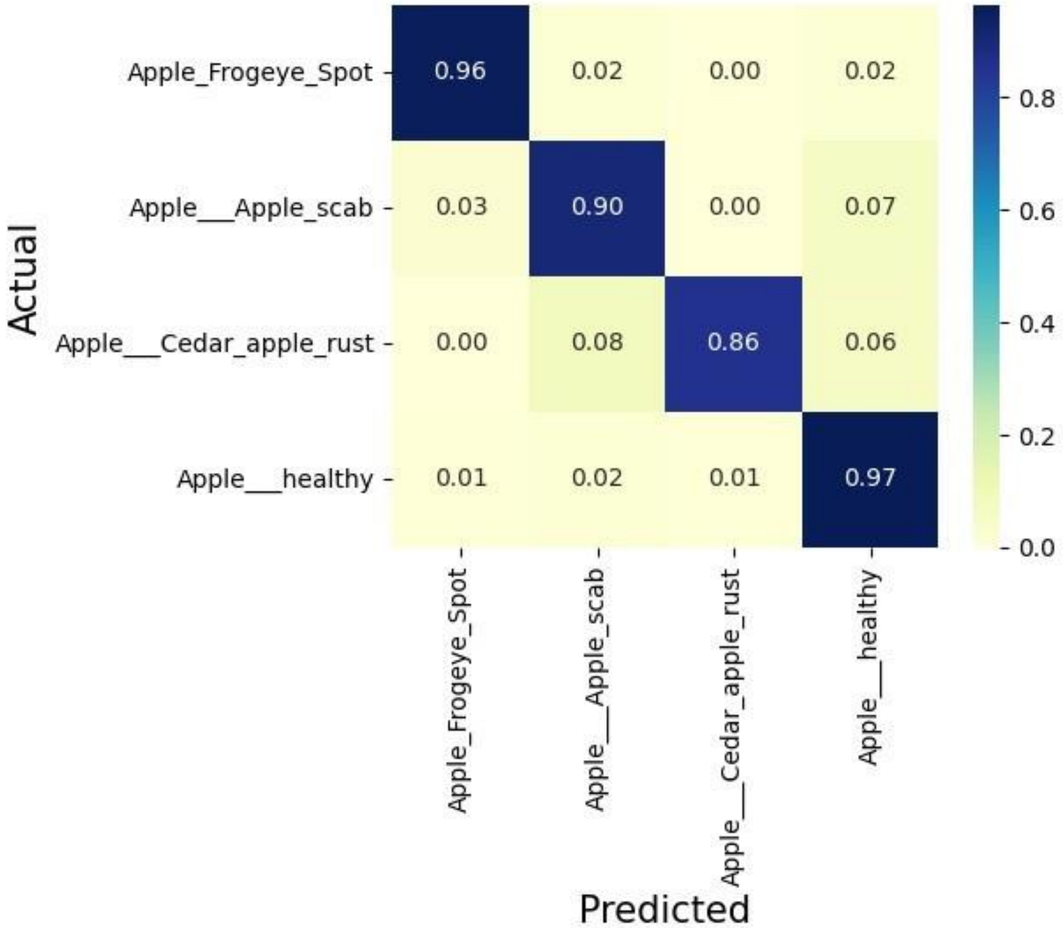
```
# Get the filenames from the generator
fnames = test_generator.filenames

# Get the ground truth from generator
ground_truth = test_generator.classes

# Get the label to class mapping from the generator
label2index = test_generator.class_indices

# Getting the mapping from class index to class label
idx2label = dict((v,k) for k,v in label2index.items())

# Get the predictions from the model using the generator
predictions = model.predict_generator(test_generator,
steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes = np.argmax(predictions,axis=1)

errors = np.where(predicted_classes != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples))
```

```
547/547 [==============================] - 3s 6ms/step
No of errors = 33/547
```

```
accuracy = ((test_generator.samples-len(errors))/test_generator.samples) *
100
accuracy
```

```
93.96709323583181
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
cm = confusion_matrix(y_true=ground_truth, y_pred=predicted_classes)
cm = np.array(cm)
# Normalise
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(5,4))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=label2index,
yticklabels=label2index, cmap="YlGnBu")
plt.ylabel('Actual', fontsize=15)
plt.xlabel('Predicted', fontsize=15)
plt.show(block=False)
```



```
from sklearn.metrics import classification_report
print(classification_report(ground_truth, predicted_classes,
target_names=label2index))
```

```
                            precision    recall  f1-score   support

         Apple_Frogeye_Spot       0.94      0.96      0.95       104
         Apple___Apple_scab       0.92      0.90      0.91       134
   Apple___Cedar_apple_rust       0.95      0.86      0.90        49
            Apple___healthy       0.95      0.97      0.96       260

                   accuracy                           0.94       547
                  macro avg       0.94      0.92      0.93       547
               weighted avg       0.94      0.94      0.94       547
```

InceptionNet

```
from tensorflow.keras import applications

## Loading InceptionV3 model
base_model = applications.InceptionV3(weights="imagenet", include_top=False,
input_shape= (128, 128, 3))
base_model.trainable = False ## Not trainable weights

base_model.summary()
```

```
Downloading data from https://github.com/fchollet/deep-learning-
models/releases/download/v0.5/inception_v3_weights_tf_dim_ordering_tf_kernels
_notop.h5
87916544/87910968 [==============================] - 15s 0us/step
Model: "inception_v3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 63, 63, 32)   864
input_3[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 63, 63, 32)   96          conv2d[0][0]
__________________________________________________________________________________________________
```

```
________________________________
activation (Activation)          (None, 63, 63, 32)   0
batch_normalization[0][0]
________________________________________________________________________________
________________________________
conv2d_1 (Conv2D)                (None, 61, 61, 32)   9216
activation[0][0]
________________________________________________________________________________
________________________________
batch_normalization_1 (BatchNor  (None, 61, 61, 32)   96
conv2d_1[0][0]
________________________________________________________________________________
________________________________
activation_1 (Activation)        (None, 61, 61, 32)   0
batch_normalization_1[0][0]
________________________________________________________________________________
________________________________
conv2d_2 (Conv2D)                (None, 61, 61, 64)   18432
activation_1[0][0]
________________________________________________________________________________
________________________________
batch_normalization_2 (BatchNor  (None, 61, 61, 64)   192
conv2d_2[0][0]
________________________________________________________________________________
________________________________
activation_2 (Activation)        (None, 61, 61, 64)   0
batch_normalization_2[0][0]
________________________________________________________________________________
________________________________
max_pooling2d (MaxPooling2D)     (None, 30, 30, 64)   0
activation_2[0][0]
________________________________________________________________________________
________________________________
conv2d_3 (Conv2D)                (None, 30, 30, 80)   5120
max_pooling2d[0][0]
________________________________________________________________________________
________________________________
batch_normalization_3 (BatchNor  (None, 30, 30, 80)   240
conv2d_3[0][0]
________________________________________________________________________________
________________________________
activation_3 (Activation)        (None, 30, 30, 80)   0
batch_normalization_3[0][0]
________________________________________________________________________________
________________________________
conv2d_4 (Conv2D)                (None, 28, 28, 192)  138240
activation_3[0][0]
________________________________________________________________________________
________________________________
batch_normalization_4 (BatchNor  (None, 28, 28, 192)  576
```

```
conv2d_91[0][0]
__________________________________________________________________________________________________
batch_normalization_92 (BatchNo  (None, 2, 2, 384)    1152
conv2d_92[0][0]
__________________________________________________________________________________________________
conv2d_93 (Conv2D)               (None, 2, 2, 192)    393216
average_pooling2d_8[0][0]
__________________________________________________________________________________________________
batch_normalization_85 (BatchNo  (None, 2, 2, 320)    960
conv2d_85[0][0]
__________________________________________________________________________________________________
activation_87 (Activation)       (None, 2, 2, 384)    0
batch_normalization_87[0][0]
__________________________________________________________________________________________________
activation_88 (Activation)       (None, 2, 2, 384)    0
batch_normalization_88[0][0]
__________________________________________________________________________________________________
activation_91 (Activation)       (None, 2, 2, 384)    0
batch_normalization_91[0][0]
__________________________________________________________________________________________________
activation_92 (Activation)       (None, 2, 2, 384)    0
batch_normalization_92[0][0]
__________________________________________________________________________________________________
batch_normalization_93 (BatchNo  (None, 2, 2, 192)    576
conv2d_93[0][0]
__________________________________________________________________________________________________
activation_85 (Activation)       (None, 2, 2, 320)    0
batch_normalization_85[0][0]
__________________________________________________________________________________________________
mixed9_1 (Concatenate)           (None, 2, 2, 768)    0
activation_87[0][0]

activation_88[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)      (None, 2, 2, 768)    0
activation_91[0][0]

activation_92[0][0]
__________________________________________________________________________________________________
```

```
__________________________________
activation_93 (Activation)      (None, 2, 2, 192)    0
batch_normalization_93[0][0]
______________________________________________________________________________
__________________________________
mixed10 (Concatenate)           (None, 2, 2, 2048)   0
activation_85[0][0]

mixed9_1[0][0]

concatenate_1[0][0]

activation_93[0][0]
==============================================================================
====================
Total params: 21,802,784
Trainable params: 0
Non-trainable params: 21,802,784
______________________________________________________________________________
__________________________________
```

```
flatten_layer = layers.GlobalAveragePooling2D()
dense_layer_1 = layers.Dense(64, activation='relu')
dense_layer_2 = layers.Dense(32, activation='relu')
prediction_layer = layers.Dense(4, activation='softmax')

model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    prediction_layer
])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Model)         (None, 2, 2, 2048)        21802784
_________________________________________________________________
global_average_pooling2d_2 ( (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                131136
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
```

```
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 132
=================================================================
Total params: 21,936,132
Trainable params: 133,348
Non-trainable params: 21,802,784
_________________________________________________________________
```

```python
model.compile(optimizer = Adam(learning_rate = 0.001),
loss='categorical_crossentropy', metrics=['acc'])
# Train the model
history = model.fit(train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size,
      epochs=10,
      validation_data=validation_generator,

validation_steps=validation_generator.samples/validation_generator.batch_size
,
      verbose=2)
```

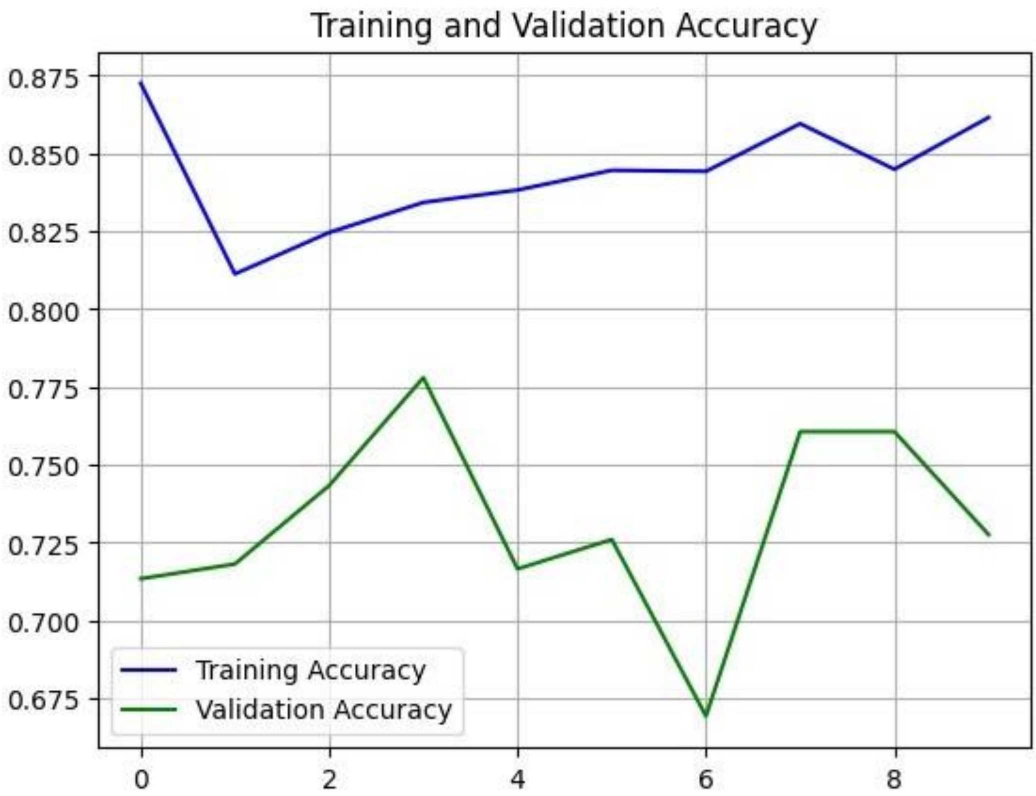
```
Epoch 1/10
Epoch 1/10
188/187 - 9s  - loss: 0.3544 - acc: 0.8725 - val_loss: 1.8910 - val_acc:
0.7134
Epoch 2/10
Epoch 1/10
188/187 - 5s  - loss: 0.4988 - acc: 0.8113 - val_loss: 1.5254 - val_acc:
0.7181
Epoch 3/10
Epoch 1/10
188/187 - 5s  - loss: 0.4549 - acc: 0.8246 - val_loss: 1.2376 - val_acc:
0.7433
Epoch 4/10
Epoch 1/10
188/187 - 5s  - loss: 0.4242 - acc: 0.8342 - val_loss: 1.2466 - val_acc:
0.7780
Epoch 5/10
Epoch 1/10
188/187 - 5s  - loss: 0.4462 - acc: 0.8382 - val_loss: 1.6309 - val_acc:
0.7165
Epoch 6/10
Epoch 1/10
188/187 - 5s  - loss: 0.4180 - acc: 0.8445 - val_loss: 1.3079 - val_acc:
0.7260
Epoch 7/10
Epoch 1/10
188/187 - 5s  - loss: 0.4094 - acc: 0.8442 - val_loss: 2.0786 - val_acc:
0.6693
Epoch 8/10
Epoch 1/10
```

```
188/187 - 5s - loss: 0.3816 - acc: 0.8595 - val_loss: 1.5253 - val_acc:
0.7606
Epoch 9/10
Epoch 1/10
188/187 - 5s - loss: 0.4081 - acc: 0.8449 - val_loss: 1.1693 - val_acc:
0.7606
Epoch 10/10
Epoch 1/10
188/187 - 5s - loss: 0.3675 - acc: 0.8615 - val_loss: 1.4619 - val_acc:
0.7276
```

```
model.save("InceptionNet_plant_deseas.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

```
train_acc = history.history['acc']
val_acc = history.history['val_acc']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(train_acc))
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'g', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.grid()
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'g', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.grid()
plt.legend()
plt.show()
```



<Figure size 640x480 with 0 Axes>

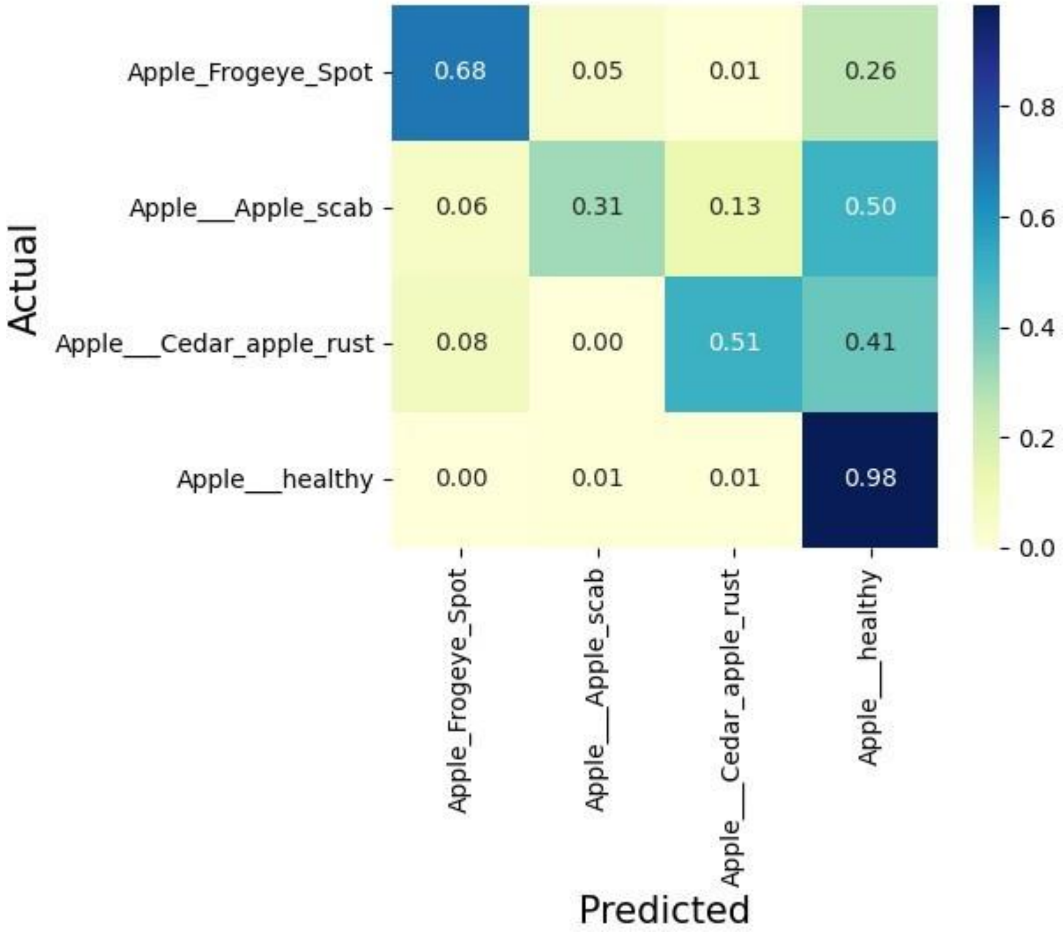

```
# Get the filenames from the generator
fnames = test_generator.filenames

# Get the ground truth from generator
ground_truth = test_generator.classes

# Get the label to class mapping from the generator
label2index = test_generator.class_indices

# Getting the mapping from class index to class label
idx2label = dict((v,k) for k,v in label2index.items())

# Get the predictions from the model using the generator
predictions = model.predict_generator(test_generator,
steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes = np.argmax(predictions,axis=1)

errors = np.where(predicted_classes != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples))
```

```
547/547 [==============================] - 9s 17ms/step
No of errors = 153/547
```

```
accuracy = ((test_generator.samples-len(errors))/test_generator.samples) *
100
accuracy
```

```
72.0292504570384
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
cm = confusion_matrix(y_true=ground_truth, y_pred=predicted_classes)
cm = np.array(cm)
# Normalise
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(5,4))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=label2index,
yticklabels=label2index, cmap="YlGnBu")
plt.ylabel('Actual', fontsize=15)
plt.xlabel('Predicted', fontsize=15)
plt.show(block=False)
```



```
from sklearn.metrics import classification_report
print(classification_report(ground_truth, predicted_classes,
target_names=label2index))
```

```
                          precision    recall  f1-score   support

        Apple_Frogeye_Spot       0.86      0.68      0.76       104
        Apple___Apple_scab       0.86      0.31      0.46       134
  Apple___Cedar_apple_rust       0.56      0.51      0.53        49
            Apple___healthy      0.69      0.98      0.81       260

                  accuracy                           0.72       547
                 macro avg       0.74      0.62      0.64       547
              weighted avg       0.75      0.72      0.69       547
```

ResNet

```
from keras import applications

## Loading VGG16 model
base_model = applications.ResNet50(weights="imagenet", include_top=False,
input_shape= (128, 128, 3))
base_model.trainable = False ## Not trainable weights

base_model.summary()

flatten_layer = layers.GlobalAveragePooling2D()
# dense_layer_1 = layers.Dense(63, activation='relu')
# dense_layer_2 = layers.Dense(32, activation='relu')
prediction_layer = layers.Dense(4, activation='softmax')

model = models.Sequential([
    base_model,
    flatten_layer,
    # dense_layer_1,
    # dense_layer_2,
    prediction_layer
])

model.summary()

model.compile(optimizer = Adam(learning_rate = 0.001),
loss='categorical_crossentropy', metrics=['acc'])
# Train the model
history = model.fit(train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size,
      epochs=30,
```

```
      validation_data=validation_generator,

validation_steps=validation_generator.samples/validation_generator.batch_size
,
      verbose=1)

model.save("ResNet_plant_deseas.h5")
print("Saved model to disk")

train_acc = history.history['acc']
val_acc = history.history['val_acc']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(train_acc))
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'g', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.grid()
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'g', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.grid()
plt.legend()
plt.show()

# Get the filenames from the generator
fnames = test_generator.filenames

# Get the ground truth from generator
ground_truth = test_generator.classes

# Get the label to class mapping from the generator
label2index = test_generator.class_indices

# Getting the mapping from class index to class label
idx2label = dict((v,k) for k,v in label2index.items())

# Get the predictions from the model using the generator
predictions = model.predict_generator(test_generator,
steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes = np.argmax(predictions,axis=1)

errors = np.where(predicted_classes != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples))
```

```
accuracy = ((test_generator.samples-len(errors))/test_generator.samples) *
100
accuracy
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
cm = confusion_matrix(y_true=ground_truth, y_pred=predicted_classes)
cm = np.array(cm)
# Normalise
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(5,4))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=label2index,
yticklabels=label2index, cmap="YlGnBu")
plt.ylabel('Actual', fontsize=15)
plt.xlabel('Predicted', fontsize=15)
plt.show(block=False)
```

```
from sklearn.metrics import classification_report
print(classification_report(ground_truth, predicted_classes,
target_names=label2index))
```

Experiment - 5

Text data handling with RNN for sentiment analysis

Importing Necessary libraries

```
import pandas as pd     # to load dataset
import numpy as np      # for mathematic equation
from nltk.corpus import stopwords    # to get collection of stopwords
from sklearn.model_selection import train_test_split        # for splitting
dataset
from tensorflow.keras.preprocessing.text import Tokenizer  # to encode text
to int
from tensorflow.keras.preprocessing.sequence import pad_sequences     # to do
padding or truncating
from tensorflow.keras.models import Sequential      # the model
from tensorflow.keras.layers import Embedding, LSTM, Dense # layers of the
architecture
from tensorflow.keras.callbacks import ModelCheckpoint     # save model
from tensorflow.keras.models import load_model     # load saved model
import re
from keras.layers import SimpleRNN
```

Preparing the data named IMDB

```
data = pd.read_csv('/content/ IMDB Dataset.csv')
print(data)
```

```
                                                  review sentiment
0      One of the other reviewers has mentioned that ...  positive
1      A wonderful little production. <br /><br />The...  positive
2      I thought this was a wonderful way to spend ti...  positive
3      Basically there's a family where a little boy ...  negative
4      Petter Mattei's "Love in the Time of Money" is...  positive
...                                                  ...       ...
49995  I thought this movie did a down right good job...  positive
49996  Bad plot, bad dialogue, bad acting, idiotic di...  negative
49997  I am a Catholic taught in parochial elementary...  negative
49998  I'm going to have to disagree with the previou...  negative
49999  No one expects the Star Trek movies to be high...  negative

[50000 rows x 2 columns]
```

```
import nltk
nltk.download("stopwords")
english_stops = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
def load_dataset():
    df = pd.read_csv('/content/ IMDB Dataset.csv')
    x_data = df['review']       # Reviews/Input
    y_data = df['sentiment']    # Sentiment/Output

    # PRE-PROCESS REVIEW
    x_data = x_data.replace({'<.*?>': ''}, regex = True)          # remove
html tag
    x_data = x_data.replace({'[^A-Za-z]': ' '}, regex = True)     # remove
non alphabet
    x_data = x_data.apply(lambda review: [w for w in review.split() if w not
in english_stops])  # remove stop words
    x_data = x_data.apply(lambda review: [w.lower() for w in review])    #
lower case

    # ENCODE SENTIMENT -> 0 & 1
    y_data = y_data.replace('positive', 1)
    y_data =  y_data.replace('negative', 0)

    return x_data, y_data

x_data, y_data = load_dataset()

print('Reviews')
print(x_data, '\n')
print('Sentiment')
print(y_data)
```

```
Reviews
0        [one, reviewers, mentioned, watching, oz, epis...
1        [a, wonderful, little, production, the, filmin...
2        [i, thought, wonderful, way, spend, time, hot,...
3        [basically, family, little, boy, jake, thinks,...
4        [petter, mattei, love, time, money, visually, ...
                                ...
49995   [i, thought, movie, right, good, job, it, crea...
49996   [bad, plot, bad, dialogue, bad, acting, idioti...
49997   [i, catholic, taught, parochial, elementary, s...
49998   [i, going, disagree, previous, comment, side, ...
49999   [no, one, expects, star, trek, movies, high, a...
Name: review, Length: 50000, dtype: object
```

```
Sentiment
0        1
1        1
2        1
3        0
4        1
        ..
49995    1
49996    0
49997    0
49998    0
49999    0
Name: sentiment, Length: 50000, dtype: int64
```

Split Dataset

```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size
= 0.2)

print('Train Set')
print(x_train, '\n')
print(x_test, '\n')
print('Test Set')
print(y_train, '\n')
print(y_test)
```

```
Train Set
22596    [boring, badly, written, italian, exploitation...
5353     [the, other, supposed, horror, movie, made, it...
42152    [a, tough, life, gets, tougher, three, childre...
15434    [why, earth, colin, firth, pointless, film, ha...
7280     [this, far, worst, movie, i, ever, seen, cinem...
                                ...
39945    [this, show, lasted, moments, plots, usually, ...
13858    [i, rented, thinking, would, pretty, good, cov...
25266    [having, pleasantly, surprised, sandra, bulloc...
10659    [the, difficulty, i, musical, version, les, mi...
39372    [this, movie, proof, film, noire, enduring, st...
Name: review, Length: 40000, dtype: object

2006     [this, movie, time, favorite, you, really, see...
33575    [this, british, film, version, stage, play, i,...
6808     [alexander, nevsky, brilliant, piece, cinemati...
32330    [found, old, vhs, version, film, parents, hous...
3777     [i, went, see, movie, daughter, i, insisted, g...
                                ...
40255    [what, heck, people, expect, horror, films, da...
```

```
5864     [especially, time, much, science, fiction, fil...
44604    [nicole, eggert, listed, star, despite, michea...
42481     [a, thief, night, got, best, end, times, thril...
31671    [i, enjoy, national, anthem, i, enjoy, nationa...
Name: review, Length: 10000, dtype: object

Test Set
22596     0
5353      0
42152     1
15434     0
7280      0
         ..
39945     0
13858     0
25266     0
10659     0
39372     1
Name: sentiment, Length: 40000, dtype: int64

2006      1
33575     1
6808      1
32330     0
3777      0
         ..
40255     1
5864      1
44604     0
42481     1
31671     1
Name: sentiment, Length: 10000, dtype: int64
```

```python
def get_max_length():
    review_length = []
    for review in x_train:
        review_length.append(len(review))

    return int(np.ceil(np.mean(review_length)))

# ENCODE REVIEW
token = Tokenizer(lower=False)    # no need lower, because already lowered the data in load_data()
token.fit_on_texts(x_train)
x_train = token.texts_to_sequences(x_train)
x_test = token.texts_to_sequences(x_test)

max_length = get_max_length()
```

```python
x_train = pad_sequences(x_train, maxlen=max_length, padding='post',
truncating='post')
x_test = pad_sequences(x_test, maxlen=max_length, padding='post',
truncating='post')

total_words = len(token.word_index) + 1   # add 1 because of 0 padding
print('Total Words:', total_words)

print('Encoded X Train\n', x_train, '\n')
print('Encoded X Test\n', x_test, '\n')
print('Maximum review length: ', max_length)
```

```
Total Words: 92636
Encoded X Train
 [[  257   863   310 ...     0     0     0]
 [    2  1340   350 ...    28   282   409]
 [   39  1138    40 ...     0     0     0]
 ...
 [ 1587  3903   660 ...    62 14457  1006]
 [    2  6090     1 ...  4973  5675   406]
 [    8     3  2912 ...     0     0     0]]

Encoded X Test
 [[    8     3    10 ...     0     0     0]
 [    8   603     4 ...   278 10278  2289]
 [ 3551 11276   417 ...     0     0     0]
 ...
 [ 3908 20405  3718 ...     0     0     0]
 [   39  2984   218 ...  3947     3   765]
 [    1   260  1833 ...     0     0     0]]

Maximum review length:  130
```

Build Architecture/Model Embedding Layer

```python
rnn = Sequential()

rnn.add(Embedding(total_words,32,input_length =max_length))
rnn.add(SimpleRNN(64,input_shape = (total_words, max_length),
return_sequences=False,activation="relu"))
rnn.add(Dense(1, activation = 'sigmoid')) #flatten

print(rnn.summary())
rnn.compile(loss="binary_crossentropy",optimizer='adam',metrics=["accuracy"])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 130, 32)           2964352

 simple_rnn (SimpleRNN)      (None, 64)                6208

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 2,970,625
Trainable params: 2,970,625
Non-trainable params: 0
_________________________________________________________________
None
```

#Trainin the Model

```
history = rnn.fit(x_train,y_train,epochs = 20,batch_size=128,verbose = 1)
```

```
Epoch 1/20
313/313 [==============================] - 96s 286ms/step - loss: 0.6915 -
accuracy: 0.5184
Epoch 2/20
313/313 [==============================] - 68s 218ms/step - loss: 0.6616 -
accuracy: 0.5879
Epoch 3/20
313/313 [==============================] - 63s 202ms/step - loss: 0.6626 -
accuracy: 0.5762
Epoch 4/20
313/313 [==============================] - 56s 180ms/step - loss: 0.5900 -
accuracy: 0.6328
Epoch 5/20
313/313 [==============================] - 51s 162ms/step - loss: 0.4166 -
accuracy: 0.8135
Epoch 6/20
313/313 [==============================] - 51s 161ms/step - loss: 0.3511 -
accuracy: 0.8806
Epoch 7/20
313/313 [==============================] - 47s 149ms/step - loss: 0.2362 -
accuracy: 0.9194
Epoch 8/20
313/313 [==============================] - 46s 148ms/step - loss: 0.1676 -
accuracy: 0.9421
Epoch 9/20
313/313 [==============================] - 45s 144ms/step - loss: 0.3168 -
accuracy: 0.8689
Epoch 10/20
313/313 [==============================] - 46s 148ms/step - loss: 0.5571 -
```

```
accuracy: 0.6458
Epoch 11/20
313/313 [==============================] - 44s 141ms/step - loss: 0.4791 -
accuracy: 0.7574
Epoch 12/20
313/313 [==============================] - 45s 144ms/step - loss: 0.2817 -
accuracy: 0.9088
Epoch 13/20
313/313 [==============================] - 43s 139ms/step - loss: 0.4030 -
accuracy: 0.8431
Epoch 14/20
313/313 [==============================] - 45s 142ms/step - loss: 0.2630 -
accuracy: 0.9154
Epoch 15/20
313/313 [==============================] - 43s 138ms/step - loss: 0.2351 -
accuracy: 0.9260
Epoch 16/20
313/313 [==============================] - 43s 138ms/step - loss: 0.2000 -
accuracy: 0.9385
Epoch 17/20
313/313 [==============================] - 45s 143ms/step - loss: 0.1599 -
accuracy: 0.9498
Epoch 18/20
313/313 [==============================] - 44s 140ms/step - loss: 0.1374 -
accuracy: 0.9577
Epoch 19/20
313/313 [==============================] - 44s 141ms/step - loss: 0.1331 -
accuracy: 0.9611
Epoch 20/20
313/313 [==============================] - 43s 136ms/step - loss: 0.2814 -
accuracy: 0.8869
```

Saving The Model

```
model = rnn.save('rnn.h5')
loaded_model = load_model('rnn.h5')
```

Evaluation

```
y_pred = rnn.predict(x_test, batch_size = 128)
print(y_pred)
print(y_test)
for i in range(len(y_pred)):
  if y_pred[i]>0.5:
    y_pred[i] = 1
  else:
    y_pred[i] = 0

true = 0
for i, y in enumerate(y_test):
    if y == y_pred[i]:
```

```
        true += 1

print('Correct Prediction: {}'.format(true))
print('Wrong Prediction: {}'.format(len(y_pred) - true))
print('Accuracy: {}'.format(true/len(y_pred)*100))
```

```
79/79 [==============================] - 1s 12ms/step
[[0.78446704]
 [0.02569966]
 [0.78301245]
 ...
 [0.2700789 ]
 [0.72713566]
 [0.78446704]]
2006     1
33575    1
6808     1
32330    0
3777     0
        ..
40255    1
5864     1
44604    0
42481    1
31671    1
Name: sentiment, Length: 10000, dtype: int64
Correct Prediction: 6918
Wrong Prediction: 3082
Accuracy: 69.17999999999999
```

Message: Nothing was typical about this. Everything was beautifully done in this movie, the story, the flow, the scenario, everything. I highly recommend it for mystery lovers, for anyone who wants to watch a good movie!

Example review

```
review = str(input('Movie Review: '))
```

```
Movie Review: Nothing was typical about this. Everything was beautifully done
in this movie, the story, the flow, the scenario, everything. I highly
recommend it for mystery lovers, for anyone who wants to watch a good movie!
```

#Pre-processing of entered review

```
# Pre-process input
regex = re.compile(r'[^a-zA-Z\s]')
review = regex.sub('', review)
print('Cleaned: ', review)
```

```
words = review.split(' ')
filtered = [w for w in words if w not in english_stops]
filtered = ' '.join(filtered)
filtered = [filtered.lower()]

print('Filtered: ', filtered)
```

```
Cleaned:  Nothing was typical about this Everything was beautifully done in
this movie the story the flow the scenario everything I highly recommend it
for mystery lovers for anyone who wants to watch a good movie
Filtered: ['nothing typical everything beautifully done movie story flow
scenario everything i highly recommend mystery lovers anyone wants watch good
movie']
```

```
tokenize_words  =  token.texts_to_sequences(filtered)
tokenize_words = pad_sequences(tokenize_words, maxlen=max_length,
padding='post', truncating='post')
print(tokenize_words)
```

```
[[  76  705  174 1210  126    3   13 2692 2596  174    1  442  280  701
  1771  155  400   33    9    3    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0]]
```

#Prediction

```
result = rnn.predict(tokenize_words)
print(result)
```

```
1/1 [==============================] - 0s 40ms/step
[[0.78446704]]
```

```
if result >= 0.7:
    print('positive')
else:
    print('negative')
```

```
positive
```

Experiment - 6

Sentiment analysis using RNN-LSTM on tweets data

Importing Necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Preparing the data named IMDB

```
df = pd.read_csv('/content/data.csv')
df.head()
```

```
   Unnamed: 0  count  hate_speech  offensive_language  neither  class  \
0           0      3            0                   0        3      2
1           1      3            0                   3        0      1
2           2      3            0                   3        0      1
3           3      3            0                   2        1      1
4           4      6            0                   6        0      1

                                               tweet
0  !!! RT @mayasolovely: As a woman you shouldn't...
1  !!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2  !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3  !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4  !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
```

```
classes = ['Hate Speech','Offensive Language','None']
```

```
df.drop(['count','hate_speech','offensive_language','neither','Unnamed: 
0'],axis=1,inplace=True)
```

```
df.head()
```

```
   class                                              tweet
0      2  !!! RT @mayasolovely: As a woman you shouldn't...
1      1  !!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2      1  !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3      1  !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4      1 !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
```

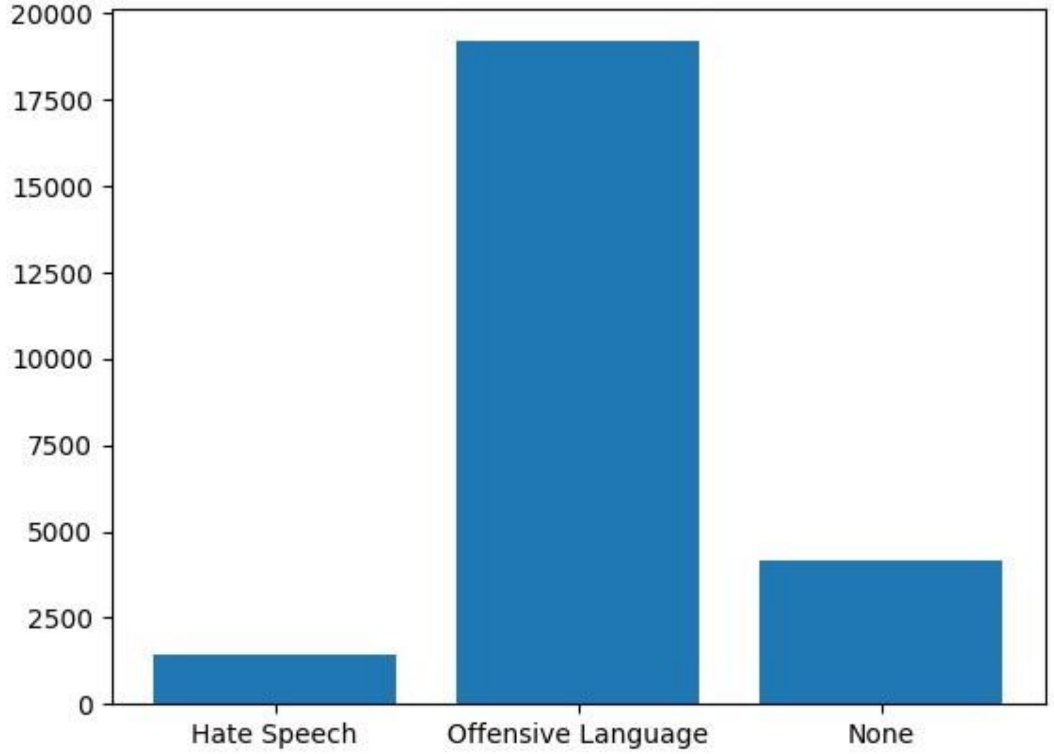
```
df.shape
```

```
(24783, 2)
```

Statistics of the Data

```
labels = df['class']
unique, counts = np.unique(labels, return_counts=True)
values = list(zip(unique, counts))
plt.bar(classes,counts)
for i in values:
    print(classes[i[0]],' : ',i[1])
plt.show()
```

```
Hate Speech  :  1430
Offensive Language  :  19190
None  :  4163
```




```
hate_tweets = df[df['class']==0]
offensive_tweets = df[df['class']==1]
neither = df[df['class']==2]
print(hate_tweets.shape)
print(offensive_tweets.shape)
print(neither.shape)
```

```
(1430, 2)
(19190, 2)
(4163, 2)
```

```
for i in range(3):
    hate_tweets = pd.concat([hate_tweets,hate_tweets],ignore_index = True)
```

```python
neither = pd.concat([neither,neither,neither], ignore_index = True)
offensive_tweets = offensive_tweets.iloc[0:12000,:]
print(hate_tweets.shape)
print(offensive_tweets.shape)
print(neither.shape)
```

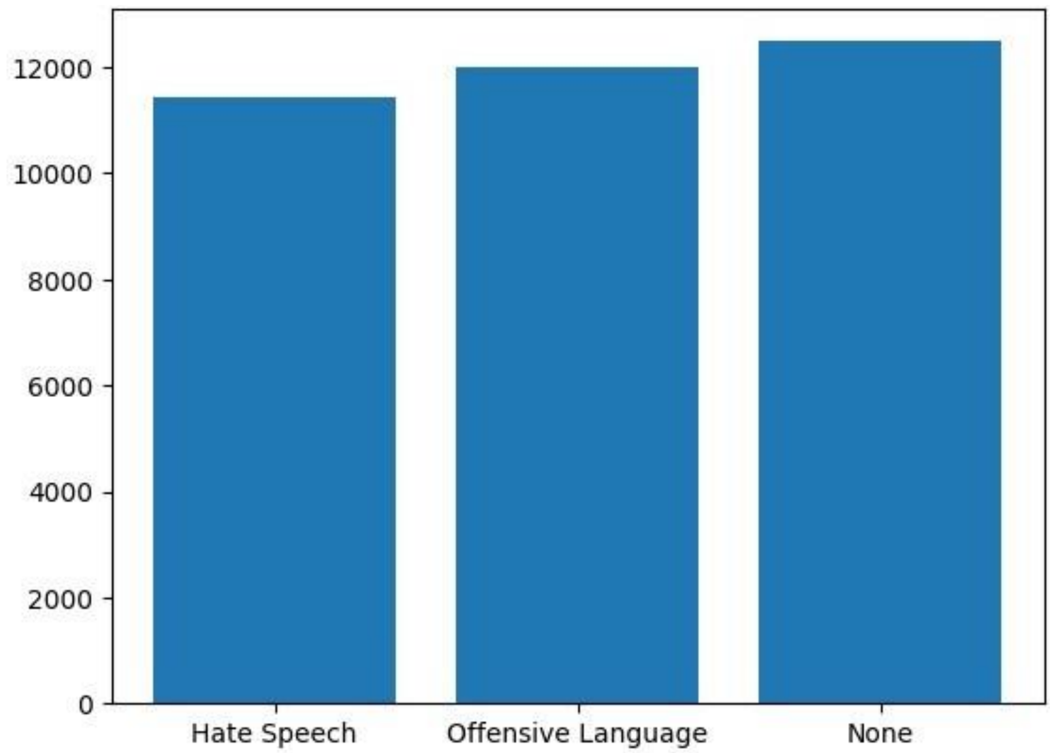
```
(11440, 2)
(12000, 2)
(12489, 2)
```

```python
df = pd.concat([hate_tweets,offensive_tweets,neither],ignore_index = True)
df.shape
```

```
(35929, 2)
```

```python
labels = df['class']
unique, counts = np.unique(labels, return_counts=True)
values = list(zip(unique, counts))
plt.bar(classes,counts)
for i in values:
    print(classes[i[0]],' : ',i[1])
plt.show()
```

```
Hate Speech  :  11440
Offensive Language  :  12000
None  :  12489
```



```
df.head()
```

```
   class                                              tweet
0      0  "@Blackman38Tide: @WhaleLookyHere @HowdyDowdy1...
1      0  "@CB_Baby24: @white_thunduh alsarabsss" hes a ...
2      0  "@DevilGrimz: @VigxRArts you're fucking gay, b...
3      0  "@MarkRoundtreeJr: LMFAOOOO I HATE BLACK PEOPL...
4      0  "@NoChillPaz: "At least I'm not a nigger" http...
```

Preprocessing

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import re
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
True
```

```
# dealing with Slangs
d = {'luv':'love','wud':'would','lyk':'like','wateva':'whatever','ttyl':'talk to you later',
               'kul':'cool','fyn':'fine','omg':'oh my god!','fam':'family','bruh':'brother',
               'cud':'could','fud':'food', 'u': 'you',
     'ur':'your', 'bday' : 'birthday', 'bihday' : 'birthday'}

stop_words = set(stopwords.words("english"))
stop_words.add('rt')
stop_words.remove('not')
lemmatizer = WordNetLemmatizer()
giant_url_regex =  ('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|'
'[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
mention_regex = '@[\w\-]+'


def clean_text(text):
    text = re.sub('"', "", text)
    text = re.sub(mention_regex, ' ',text) #removing all user names
    text = re.sub(giant_url_regex, ' ', text)  #remocing the urls
    text = text.lower()
    text = re.sub("hm+", "", text) #removing variants of hmmm
    text = re.sub("[^a-z]+", " ", text) #removing all numbers, special chars like @,#,? etc
    text = text.split()
    text = [word for word in text if not word in stop_words]
    text = [d[word] if word in d else word for word in text]  #replacing some
```

```
slangs
    text = [lemmatizer.lemmatize(token) for token in text]
    text = [lemmatizer.lemmatize(token, "v") for token in text]
    text = " ".join(text)
    return text

df['processed_tweets'] = df.tweet.apply(lambda x: clean_text(x))     #
df.review.map(clean_text) Also can be used
df.head()
```

```
   class                                              tweet  \
0      0  "@Blackman38Tide: @WhaleLookyHere @HowdyDowdy1...
1      0  "@CB_Baby24: @white_thunduh alsarabsss" hes a ...
2      0  "@DevilGrimz: @VigxRArts you're fucking gay, b...
3      0  "@MarkRoundtreeJr: LMFAOOOO I HATE BLACK PEOPL...
4      0  "@NoChillPaz: "At least I'm not a nigger" http...

                                processed_tweets
0                                   queer gaywad
1         alsarabsss he beaner smh tell he mexican
2   fuck gay blacklist hoe hold tehgodclan anyway
3  lmfaoooo hate black people black people nigger
4                           least not nigger lmfao
```

```
x = df.processed_tweets
y = df['class']
print(x.shape)
print(y.shape)
```

```
(35929,)
(35929,)
```

```
# finding unique words
word_unique = []
for i in x:
    for j in i.split():
        word_unique.append(j)
unique, counts = np.unique(word_unique, return_counts=True)
print("The total words in the tweets are : ", len(word_unique))
print("The total UNIQUE words in the tweets are : ", len(unique))
```

```
The total words in the tweets are :  275540
The total UNIQUE words in the tweets are :  14146
```

```
# finding length of tweets
tweets_length = []
for i in x:
    tweets_length.append(len(i.split()))
print("The Average Length tweets are : ",np.mean(tweets_length))
print("The max length of tweets is : ", np.max(tweets_length))
print("The min length of tweets is : ", np.min(tweets_length))
```

```
The Average Length tweets are :  7.669013888502324
The max length of tweets is :   28
The min length of tweets is :   0
```

```
tweets_length = pd.DataFrame(tweets_length)
# tweets_length.describe()
```

	0
count	35929.000000
mean	7.669014
std	3.989625
min	0.000000
25%	4.000000
50%	7.000000
75%	11.000000
max	28.000000

```
#Sorting the Unique words based on their Frequency
col = list(zip(unique, counts))
col = sorted(col, key = lambda x: x[1],reverse=True)
col=pd.DataFrame(col)
print("Top 20 Occuring Words with their frequency are:")
col.iloc[:20,:]
```

Top 20 Occuring Words with their frequency are:

	0	1
0	bitch	9066
1	like	3817
2	get	3636
3	hoe	3426
4	trash	3217
5	fuck	3103
6	nigga	2819
7	faggot	2239
8	as	2073
9	you	1851
10	pussy	1847
11	go	1773
12	not	1764
13	bird	1515
14	lol	1494
15	nigger	1459
16	say	1456
17	make	1373
18	amp	1329
19	white	1328

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
vectorizer = TfidfVectorizer(max_features = 8000 )
# tokenize and build vocab

vectorizer.fit(x)
# summarize

print(len(vectorizer.vocabulary_))
print(vectorizer.idf_.shape)
```

```
8000
(8000,)
```

```
x_tfidf = vectorizer.transform(x).toarray()
print(x_tfidf.shape)
```

```
(35929, 8000)
```

```
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
num_words = 8000
embed_dim = 32
tokenizer = Tokenizer(num_words=num_words,oov_token = "<oov>" )
tokenizer.fit_on_texts(x)
word_index=tokenizer.word_index
sequences = tokenizer.texts_to_sequences(x)
length=[]
for i in sequences:
    length.append(len(i))
print(len(length))
print("Mean is: ",np.mean(length))
print("Max is: ",np.max(length))
print("Min is: ",np.min(length))
```

```
35929
Mean is:  7.669013888502324
Max is:  28
Min is:  0
```

```
pad_length = 24
sequences = pad_sequences(sequences, maxlen = pad_length, truncating = 'pre',
padding = 'post')
sequences.shape
```

```
(35929, 24)
```

#**Splitting the Data**

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(sequences,y,test_size =
0.05)
print(x_train.shape)
```

```
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(34132, 24)
(1797, 24)
(34132,)
(1797,)
```

#RNN Model

```
from keras.layers import Dense, Embedding, Dropout , Activation, Flatten,
SimpleRNN
from keras.layers import GlobalMaxPool1D
from keras.models import Model, Sequential
import tensorflow as tf

recall = tf.keras.metrics.Recall()
precision = tf.keras.metrics.Precision()


model = Sequential([Embedding(num_words, embed_dim, input_length =
pad_length),
                    SimpleRNN(8, return_sequences = True),
                    GlobalMaxPool1D(),
                    Dense(20,activation =
'relu',kernel_initializer='he_uniform'),
                    Dropout(0.25),
                    Dense(3,activation = 'softmax')])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# model.name = 'Twitter Hate Text Classification'
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 24, 32)            256000

 simple_rnn_1 (SimpleRNN)    (None, 24, 8)             328

 global_max_pooling1d_1 (Glo  (None, 8)                0
 balMaxPooling1D)

 dense_2 (Dense)             (None, 20)                180

 dropout_1 (Dropout)         (None, 20)                0

 dense_3 (Dense)             (None, 3)                 63
```

```
=================================================================
Total params: 256,571
Trainable params: 256,571
Non-trainable params: 0
_________________________________________________________________
```

```
#Training the Model
history = model.fit(x = x_train, y = y_train, epochs = 5,validation_split = 0.05)
```

```
Epoch 1/5
1014/1014 [==============================] - 12s 12ms/step - loss: 0.2029 - accuracy: 0.9373 - val_loss: 0.1583 - val_accuracy: 0.9473
Epoch 2/5
1014/1014 [==============================] - 11s 11ms/step - loss: 0.1271 - accuracy: 0.9624 - val_loss: 0.1521 - val_accuracy: 0.9467
Epoch 3/5
1014/1014 [==============================] - 12s 11ms/step - loss: 0.0906 - accuracy: 0.9754 - val_loss: 0.1479 - val_accuracy: 0.9537
Epoch 4/5
1014/1014 [==============================] - 12s 12ms/step - loss: 0.0714 - accuracy: 0.9801 - val_loss: 0.1644 - val_accuracy: 0.9525
Epoch 5/5
1014/1014 [==============================] - 12s 12ms/step - loss: 0.0588 - accuracy: 0.9839 - val_loss: 0.1541 - val_accuracy: 0.9584
```

```
#Evaluation
evaluate = model.evaluate(x_test,y_test)
```

```
57/57 [==============================] - 0s 3ms/step - loss: 0.1112 - accuracy: 0.9688
```

```
print("Test Acuracy is : {:.2f} %".format(evaluate[1]*100))
print("Test Loss is : {:.4f}".format(evaluate[0]))
```

```
Test Acuracy is : 96.88 %
Test Loss is : 0.1112
```

```
predictions = model.predict(x_test)
```

```
57/57 [==============================] - 0s 2ms/step
```

```
predict = []
for i in predictions:
    predict.append(np.argmax(i))
```

Confusion Matrix

```
from sklearn import metrics
cm = metrics.confusion_matrix(predict,y_test)
acc = metrics.accuracy_score(predict,y_test)

print("The Confusion matrix is: \n",cm)
```

```
The Confusion matrix is:
 [[548  22   1]
 [  6 572   8]
 [  0  19 621]]
```

```
#Accuracy

print(acc*100)
```

```
96.88369504730106
```

```
#Classification Report

from sklearn import metrics
print(metrics.classification_report(y_test, predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       554
           1       0.98      0.93      0.95       613
           2       0.97      0.99      0.98       630

    accuracy                           0.97      1797
   macro avg       0.97      0.97      0.97      1797
weighted avg       0.97      0.97      0.97      1797
```

```
#LSTM

from tensorflow.keras.layers import Embedding, LSTM, Dense
# ARCHITECTURE
EMBED_DIM = 32
LSTM_OUT = 64

model = Sequential()
model.add(Embedding(num_words, EMBED_DIM, input_length = pad_length))
model.add(LSTM(LSTM_OUT))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy',
metrics = ['accuracy'])
```

```
print(model.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_4 (Embedding)     (None, 24, 32)            256000    
                                                                 
 lstm_1 (LSTM)               (None, 64)                24832     
                                                                 
 dense_5 (Dense)             (None, 3)                 195       
                                                                 
=================================================================
Total params: 281,027
Trainable params: 281,027
Non-trainable params: 0
_________________________________________________________________
None
```

#**Training Model**

```
history = model.fit(x = x_train, y = y_train, epochs = 10,validation_split = 
0.05)
```

```
Epoch 1/10
1014/1014 [==============================] - 26s 24ms/step - loss: 0.4328 - 
accuracy: 0.8246 - val_loss: 0.2531 - val_accuracy: 0.9127
Epoch 2/10
1014/1014 [==============================] - 22s 22ms/step - loss: 0.1874 - 
accuracy: 0.9405 - val_loss: 0.1876 - val_accuracy: 0.9367
Epoch 3/10
1014/1014 [==============================] - 23s 23ms/step - loss: 0.1426 - 
accuracy: 0.9554 - val_loss: 0.1991 - val_accuracy: 0.9391
Epoch 4/10
1014/1014 [==============================] - 24s 23ms/step - loss: 0.1188 - 
accuracy: 0.9626 - val_loss: 0.1991 - val_accuracy: 0.9361
Epoch 5/10
1014/1014 [==============================] - 23s 23ms/step - loss: 0.0999 - 
accuracy: 0.9688 - val_loss: 0.2142 - val_accuracy: 0.9408
Epoch 6/10
1014/1014 [==============================] - 23s 22ms/step - loss: 0.0835 - 
accuracy: 0.9730 - val_loss: 0.1820 - val_accuracy: 0.9508
Epoch 7/10
1014/1014 [==============================] - 24s 23ms/step - loss: 0.0735 - 
accuracy: 0.9767 - val_loss: 0.2202 - val_accuracy: 0.9473
Epoch 8/10
1014/1014 [==============================] - 24s 23ms/step - loss: 0.0670 - 
accuracy: 0.9783 - val_loss: 0.2666 - val_accuracy: 0.9461
Epoch 9/10
```

```
1014/1014 [==============================] - 22s 22ms/step - loss: 0.0577 -
accuracy: 0.9815 - val_loss: 0.1988 - val_accuracy: 0.9490
Epoch 10/10
1014/1014 [==============================] - 24s 23ms/step - loss: 0.0537 -
accuracy: 0.9823 - val_loss: 0.2313 - val_accuracy: 0.9490
```

#**Evaluation**

```
evaluate = model.evaluate(x_test,y_test)
```

```
57/57 [==============================] - 0s 8ms/step - loss: 0.1719 -
accuracy: 0.9594
```

```
print("Test Acuracy is : {:.2f} %".format(evaluate[1]*100))
print("Test Loss is : {:.4f}".format(evaluate[0]))
```

```
Test Acuracy is : 95.94 %
Test Loss is : 0.1719
```

```
predictions = model.predict(x_test)
```

```
57/57 [==============================] - 1s 8ms/step
```

```
predict = []
for i in predictions:
    predict.append(np.argmax(i))
```

#**Confusion Matrix**

```
from sklearn import metrics
cm = metrics.confusion_matrix(predict,y_test)
acc = metrics.accuracy_score(predict,y_test)
```

```
print("The Confusion matrix is: \n",cm)
```

```
The Confusion matrix is:
 [[552  42   1]
 [  0 543   0]
 [  2  28 629]]
```

#**Accuracy**

```
print(acc*100)
```

```
95.93767390094602
```

#**Classification Report**

```
from sklearn import metrics
print(metrics.classification_report(y_test,  predict))
```

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96       554
```

1	1.00	0.89	0.94	613
2	0.95	1.00	0.98	630
accuracy			0.96	1797
macro avg	0.96	0.96	0.96	1797
weighted avg	0.96	0.96	0.96	1797

Experiment - 7

Auto encoders using MNIST data

Importing Necessary libraries

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
```

Pre-processing

```
def preprocess(array):
    array = array.astype("float32") / 255.0
    array = np.reshape(array, (len(array), 28, 28, 1))
    return array
```

Adding noise to the original images

```
def noise(array):
    noise_factor = 0.4 #amount of noise to add
    noisy_array = array + noise_factor * np.random.normal(
        loc=0.0, scale=1.0, size=array.shape
    )

    return np.clip(noisy_array, 0.0, 1.0)
```

Visualizing the images

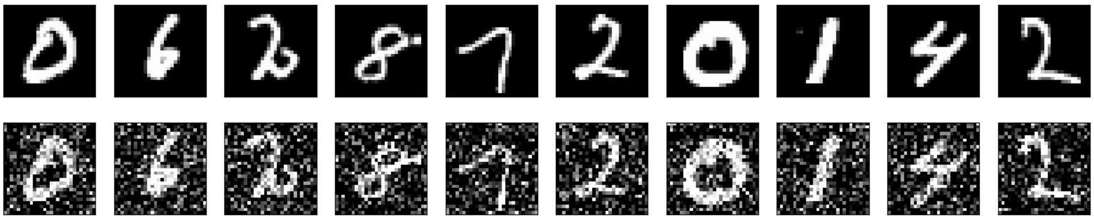
```
def display(array1, array2):
    n = 10

    indices = np.random.randint(len(array1), size=n)
    images1 = array1[indices, :]
    images2 = array2[indices, :]

    plt.figure(figsize=(20, 4))
    for i, (image1, image2) in enumerate(zip(images1, images2)):
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(image1.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
```

```
        ax.get_yaxis().set_visible(False)

        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(image2.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

    plt.show()
```

Preparing the data

```
 (train_data, _), (test_data, _) = mnist.load_data()

# Normalize and reshape the data
train_data = preprocess(train_data)
test_data = preprocess(test_data)

# Create a copy of the data with added noise
noisy_train_data = noise(train_data)
noisy_test_data = noise(test_data)

# Display the train data and a version of it with added noise
display(train_data, noisy_train_data)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```



Building the Autoencoder

```
input = layers.Input(shape=(28, 28, 1))

# Encoder
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(input)
x = layers.MaxPooling2D((2, 2), padding="same")(x)
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2, 2), padding="same")(x)

# Decoder
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
```

```
padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)

# Autoencoder
autoencoder = Model(input, x)
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")
autoencoder.summary()
```

```
Model: "functional"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━
━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━
━━━━━━━━━┩
│ input_layer (InputLayer)              │ (None, 28, 28, 1)           │
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────
─────────┤
│ conv2d (Conv2D)                       │ (None, 28, 28, 32)          │
320 │
├──────────────────────────────────────┼─────────────────────────────┼──────
─────────┤
│ max_pooling2d (MaxPooling2D)          │ (None, 14, 14, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────
─────────┤
│ conv2d_1 (Conv2D)                     │ (None, 14, 14, 32)          │
9,248 │
├──────────────────────────────────────┼─────────────────────────────┼──────
─────────┤
│ max_pooling2d_1 (MaxPooling2D)        │ (None, 7, 7, 32)            │
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────
─────────┤
│ conv2d_transpose (Conv2DTranspose)    │ (None, 14, 14, 32)          │
9,248 │
├──────────────────────────────────────┼─────────────────────────────┼──────
─────────┤
│ conv2d_transpose_1 (Conv2DTranspose) │ (None, 28, 28, 32)          │
9,248 │
├──────────────────────────────────────┼─────────────────────────────┼──────
─────────┤
│ conv2d_2 (Conv2D)                     │ (None, 28, 28, 1)           │
289 │
└──────────────────────────────────────┴─────────────────────────────┴──────
─────────┘
```

```
 Total params: 28,353 (110.75 KB)

 Trainable params: 28,353 (110.75 KB)

 Non-trainable params: 0 (0.00 B)
```

Training the model

```
autoencoder.fit(
    x=train_data,
    y=train_data,
    epochs=50,
    batch_size=128,
    shuffle=True,
    validation_data=(test_data, test_data),
)
```

```
Epoch 1/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 0.2439 - val_loss: 0.0743
Epoch 2/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0736 - val_loss: 0.0701
Epoch 3/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0704 - val_loss: 0.0685
Epoch 4/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0689 - val_loss: 0.0676
Epoch 5/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0679 - val_loss: 0.0669
Epoch 6/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0672 - val_loss: 0.0663
Epoch 7/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0667 - val_loss: 0.0659
Epoch 8/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0663 - val_loss: 0.0656
Epoch 9/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0659 - val_loss: 0.0652
Epoch 10/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0656 - val_loss: 0.0649
Epoch 11/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0654 - val_loss: 0.0647
Epoch 12/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0650 - val_loss: 0.0644
Epoch 13/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0648 - val_loss: 0.0644
Epoch 14/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0645 - val_loss: 0.0641
Epoch 15/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 0.0643 - val_loss: 0.0639
Epoch 16/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0642 - val_loss: 0.0637
Epoch 17/50
```

```
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0641 - val_loss: 0.0636
Epoch 18/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0640 - val_loss: 0.0634
Epoch 19/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0638 - val_loss: 0.0634
Epoch 20/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0637 - val_loss: 0.0633
Epoch 21/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 0.0636 - val_loss: 0.0632
Epoch 22/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0634 - val_loss: 0.0631
Epoch 23/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0635 - val_loss: 0.0630
Epoch 24/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 0.0634 - val_loss: 0.0629
Epoch 25/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0634 - val_loss: 0.0629
Epoch 26/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0633 - val_loss: 0.0629
Epoch 27/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 0.0632 - val_loss: 0.0628
Epoch 28/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0630 - val_loss: 0.0627
Epoch 29/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0630 - val_loss: 0.0626
Epoch 30/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0630 - val_loss: 0.0626
Epoch 31/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 0.0631 - val_loss: 0.0626
Epoch 32/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0628 - val_loss: 0.0626
Epoch 33/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0628 - val_loss: 0.0625
Epoch 34/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 0.0628 - val_loss: 0.0625
Epoch 35/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0628 - val_loss: 0.0625
Epoch 36/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0629 - val_loss: 0.0624
Epoch 37/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 0.0628 - val_loss: 0.0625
Epoch 38/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0628 - val_loss: 0.0625
Epoch 39/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0627 - val_loss: 0.0624
Epoch 40/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 0.0626 - val_loss: 0.0623
Epoch 41/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0627 - val_loss: 0.0622
Epoch 42/50
```

```
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0626 - val_loss: 0.0622
Epoch 43/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0625 - val_loss: 0.0622
Epoch 44/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0626 - val_loss: 0.0622
Epoch 45/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0625 - val_loss: 0.0621
Epoch 46/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0625 - val_loss: 0.0622
Epoch 47/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0625 - val_loss: 0.0622
Epoch 48/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0625 - val_loss: 0.0621
Epoch 49/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0624 - val_loss: 0.0621
Epoch 50/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 0.0624 - val_loss: 0.0620

<keras.src.callbacks.history.History at 0x79e5e6633490>
```

Prediction

```
predictions = autoencoder.predict(test_data)
display(test_data, predictions)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

Experiment - 8

Variational Autoencoders

Imports

```
# For working with and visualizing the data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# For training the VAE
import tensorflow as tf

# For creating interactive widgets
import ipywidgets as widgets
from IPython.display import display
```

```
2024-05-22 06:28:22.848294: I
tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is
optimized to use available CPU instructions in performance-critical
operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

```
# Load the data from a .csv file
pixel_data = pd.read_csv('/content/age_gender.csv')['pixels']

# Shuffle the data
pixel_data = pixel_data.sample(frac=1.0, random_state=1)
# Convert the data into a NumPy array
pixel_data = pixel_data.apply(lambda x: np.array(x.split(" "), dtype=np.int))
pixel_data = np.stack(np.array(pixel_data), axis=0)
# Rescale pixel values to be between 0 and 1
pixel_data = pixel_data * (1./255)
```

```
/tmp/ipykernel_92174/3099532490.py:4: DeprecationWarning: `np.int` is a
deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note
link for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

```
  pixel_data = pixel_data.apply(lambda x: np.array(x.split(" "),
dtype=np.int))

# The data is now a NumPy array of 23705 images.
# we are working with 48x48x1 images)
pixel_data.shape
```

```
(23705, 2304)
```

Building the VAE

```
class Sampling(tf.keras.layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs  # Unpack the inputs into mean and log-
variance
        batch = tf.shape(z_mean)[0]  # Get the batch size
        dim = tf.shape(z_mean)[1]  # Get the dimensionality of the latent
space
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))  #
Sample from standard normal distribution
        return epsilon * tf.exp(z_log_var * 0.5) + z_mean  # Apply the
reparameterization trick

def build_vae(num_pixels, num_latent_vars=3):

    # Encoder
    encoder_inputs = tf.keras.Input(shape=(num_pixels,)) # Input layer for
the encoder
    x = tf.keras.layers.Dense(512, activation='relu')(encoder_inputs)  #
First dense layer with 512 units and ReLU activation
    x = tf.keras.layers.Dense(128, activation='relu')(x)  # Second dense
layer with 128 units and ReLU activation
    x = tf.keras.layers.Dense(32, activation='relu')(x)  # Third dense layer
with 32 units and ReLU activation
    z_mean = tf.keras.layers.Dense(num_latent_vars)(x)  # Dense layer for the
mean of the latent variables
    z_log_var = tf.keras.layers.Dense(num_latent_vars)(z_mean)  # Dense layer
for the log-variance of the latent variables
    z = Sampling()([z_mean, z_log_var])  # Sampling layer to sample the
latent variables using the reparameterization trick

    encoder = tf.keras.Model(inputs=encoder_inputs, outputs=z)  # Define the
encoder model

    # Decoder
    decoder_inputs = tf.keras.Input(shape=(num_latent_vars,))  # Input layer
for the decoder
    x = tf.keras.layers.Dense(32, activation='relu')(decoder_inputs)  # First
dense layer with 32 units and ReLU activation
    x = tf.keras.layers.Dense(128, activation='relu')(x)   # Second dense
```

```
layer with 128 units and ReLU activation
    x = tf.keras.layers.Dense(512, activation='relu')(x)  # Third dense layer
with 512 units and ReLU activation
    reconstruction = tf.keras.layers.Dense(num_pixels,
activation='linear')(x)  # Output dense layer with 'num_pixels' units and
linear activation

    decoder = tf.keras.Model(inputs=decoder_inputs, outputs=reconstruction)
# Define the decoder model

    # Full model
    model_inputs = encoder.input  # Inputs of the full VAE model are the
inputs of the encoder
    model_outputs = decoder(encoder.output)  # Outputs of the full VAE model
are the outputs of the decoder, given the encoder's output

    model = tf.keras.Model(inputs=model_inputs, outputs=model_outputs)  #
Define the full VAE model

    # Compile model for training
    model.compile(
        optimizer='adam',  # Adam optimizer
        loss='mse'  # Mean Squared Error (MSE) loss function
    )

    # Return all three models
    return encoder, decoder, model  # Return the encoder, decoder, and full
VAE models

face_encoder, face_decoder, face_model = build_vae(num_pixels=2304,
num_latent_vars=3)
```

```
2024-05-22 06:28:31.067844: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:1960] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are
installed properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices...
```

```
print(face_encoder.summary())
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected
to
==================================================================================================
 input_1 (InputLayer)        [(None, 2304)]               0         []
```

```
 dense (Dense)                (None, 512)                  1180160
['input_1[0][0]']

 dense_1 (Dense)              (None, 128)                  65664
['dense[0][0]']

 dense_2 (Dense)              (None, 32)                   4128
['dense_1[0][0]']

 dense_3 (Dense)              (None, 3)                    99
['dense_2[0][0]']

 dense_4 (Dense)              (None, 3)                    12
['dense_3[0][0]']

 sampling (Sampling)          (None, 3)                    0
['dense_3[0][0]',

'dense_4[0][0]']

=============================================================================
=====================
Total params: 1250063 (4.77 MB)
Trainable params: 1250063 (4.77 MB)
Non-trainable params: 0 (0.00 Byte)
_____________________________________________________________________________
_____________________
None
```

```
print(face_decoder.summary())
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 3)]               0

 dense_5 (Dense)             (None, 32)                128

 dense_6 (Dense)             (None, 128)               4224

 dense_7 (Dense)             (None, 512)               66048

 dense_8 (Dense)             (None, 2304)              1181952

=================================================================
Total params: 1252352 (4.78 MB)
Trainable params: 1252352 (4.78 MB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Train the VAE

```
history = face_model.fit(
    pixel_data,
    pixel_data,
    validation_split=0.2,
    batch_size=32,
    epochs=100,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=10,
            restore_best_weights=True
        )
    ]
)
```

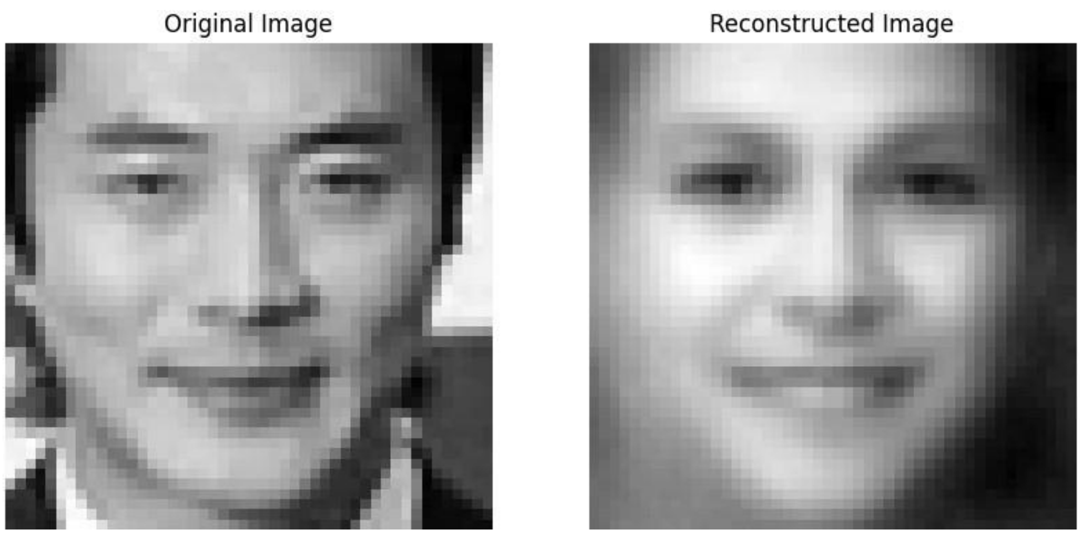
```
Epoch 1/100
593/593 [==============================] - 8s 12ms/step - loss: 0.0256 -
val_loss: 0.0216
Epoch 2/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0216 -
val_loss: 0.0213
Epoch 3/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0214 -
val_loss: 0.0212
Epoch 4/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0212 -
val_loss: 0.0212
Epoch 5/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0212 -
val_loss: 0.0210
Epoch 6/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0211 -
val_loss: 0.0213
Epoch 7/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0210 -
val_loss: 0.0209
Epoch 8/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0209 -
val_loss: 0.0209
Epoch 9/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0209 -
val_loss: 0.0210
Epoch 10/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0208 -
```

```
val_loss: 0.0206
Epoch 11/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0206 -
val_loss: 0.0206
Epoch 12/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0206 -
val_loss: 0.0208
Epoch 13/100
593/593 [==============================] - 6s 9ms/step - loss: 0.0206 -
val_loss: 0.0207
Epoch 14/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0206 -
val_loss: 0.0205
Epoch 15/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0207 -
val_loss: 0.0210
Epoch 16/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0207 -
val_loss: 0.0204
Epoch 17/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0208 -
val_loss: 0.0208
Epoch 18/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0206 -
val_loss: 0.0206
Epoch 19/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0205 -
val_loss: 0.0207
Epoch 20/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0207 -
val_loss: 0.0205
Epoch 21/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0207 -
val_loss: 0.0206
Epoch 22/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0207 -
val_loss: 0.0209
Epoch 23/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0207 -
val_loss: 0.0207
Epoch 24/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0206 -
val_loss: 0.0210
Epoch 25/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0205 -
val_loss: 0.0205
Epoch 26/100
593/593 [==============================] - 5s 9ms/step - loss: 0.0205 -
val_loss: 0.0205
```

Image Reconstruction

```
i = 6

sample = np.array(pixel_data)[i].copy()
sample = sample.reshape(48, 48, 1)

reconstruction = face_model.predict(pixel_data)[i].copy()
reconstruction = reconstruction.reshape(48, 48, 1)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(sample, cmap='gray')
plt.axis('off')
plt.title("Original Image")

plt.subplot(1, 2, 2)
plt.imshow(reconstruction, cmap='gray')
plt.axis('off')
plt.title("Reconstructed Image")

plt.show()
```

```
741/741 [==============================] - 1s 2ms/step
```



Specify our own latent variable values

```
def generate_face_image(latent1, latent2, latent3):
    latent_vars = np.array([[latent1, latent2, latent3]])
    reconstruction = np.array(face_decoder(latent_vars))
```

```python
    reconstruction = reconstruction.reshape(48, 48, 1)
    plt.figure()
    plt.imshow(reconstruction, cmap='gray')
    plt.axis('off')
    plt.show()

# Let's get the min and max for each slider on the interactive widget
latent1_min = np.min(face_encoder(pixel_data).numpy()[:, 0])
latent1_max = np.max(face_encoder(pixel_data).numpy()[:, 0])

latent2_min = np.min(face_encoder(pixel_data).numpy()[:, 1])
latent2_max  =  np.max(face_encoder(pixel_data).numpy()[:,  1])

latent3_min = np.min(face_encoder(pixel_data).numpy()[:, 2])
latent3_max  =  np.max(face_encoder(pixel_data).numpy()[:,  2])

import tensorflow as tf
print(tf.__version__)
```

```
2.13.1
```

```python
# Create the interactive widget
face_image_generator = widgets.interact(
    generate_face_image,
    latent1=(latent1_min, latent1_max),
    latent2=(latent2_min, latent2_max),
    latent3=(latent3_min, latent3_max),
)

# Display the widget
display(face_image_generator)
```

{"model_id":"94a81f02f9594f228f804939dcececfe","version_major":2,"version_minor":0}

<function __main__.generate_face_image(latent1, latent2, latent3)>

Experiment - 9

Implementing GAN Architecture on MINST dataset

Importing Necessary libraries

```
from tensorflow.keras.layers import (Dense,
                                     BatchNormalization,
                                     LeakyReLU,
                                     Reshape,
                                     Conv2DTranspose,
                                     Conv2D,
                                     Dropout,
                                     Flatten)
import tensorflow as tf
import matplotlib.pyplot as plt
```

Preparing the data

```
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28,
1).astype('float32')
train_images = (train_images - 127.5) / 127.5

BUFFER_SIZE = 60000
BATCH_SIZE = 256

# Batch and shuffle the data
train_dataset =
tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(B
ATCH_SIZE)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

Dimension of the Noise

```
z_dim = 100
```

Functional Code Generator

```
nch = 200
g_input = Input(shape=[100])
```

```
H = Dense(nch*14*14, kernel_initializer='glorot_normal')(g_input)
H = BatchNormalization()(H)
H = Activation('relu')(H)
H = Reshape( [nch, 14, 14] )(H)
H = UpSampling2D(size=(2, 2))(H)
H = Convolution2D(int(nch/2), 3, 3, padding='same',
kernel_initializer='glorot_uniform')(H)
H = BatchNormalization()(H)
H = Activation('relu')(H)
H = Convolution2D(int(nch/4), 3, 3, padding='same',
kernel_initializer='glorot_uniform')(H)
H = BatchNormalization()(H)
H = Activation('relu')(H)
H = Convolution2D(1, 1, 1, padding='same',
kernel_initializer='glorot_uniform')(H)
g_V = Activation('sigmoid')(H)
generator = Model(g_input,g_V)
generator.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002,
beta_1=0.5))
generator.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 100)]             0

 dense_2 (Dense)             (None, 39200)             3959200

 batch_normalization_1  (Batc  (None, 39200)           156800
 hNormalization)

 activation_1 (Activation)   (None, 39200)             0

 reshape_1 (Reshape)         (None, 200, 14, 14)       0

 up_sampling2d_1 (UpSampling  (None, 400, 28, 14)      0
 2D)

 conv2d (Conv2D)             (None, 134, 10, 100)      12700

 batch_normalization_2 (Batc  (None, 134, 10, 100)     400
 hNormalization)

 activation_2 (Activation)   (None, 134, 10, 100)      0

 conv2d_1 (Conv2D)           (None, 45, 4, 50)         45050

 batch_normalization_3  (Batc  (None, 45, 4, 50)       200
```

```
 hNormalization)

 activation_3 (Activation)    (None, 45, 4, 50)         0

 conv2d_2 (Conv2D)            (None, 45, 4, 1)          51

 activation_4 (Activation)    (None, 45, 4, 1)          0

=================================================================
Total params: 4,174,401
Trainable params: 4,095,701
Non-trainable params: 78,700
_________________________________________________________________

/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/adam.py:117:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

Building Generator

```
def generator_model():
    model = tf.keras.Sequential()
    model.add(Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256) # Note: None is the batch
size

    model.add(Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same',
use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',
use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same',
use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    print(model.summary())

    return model
```

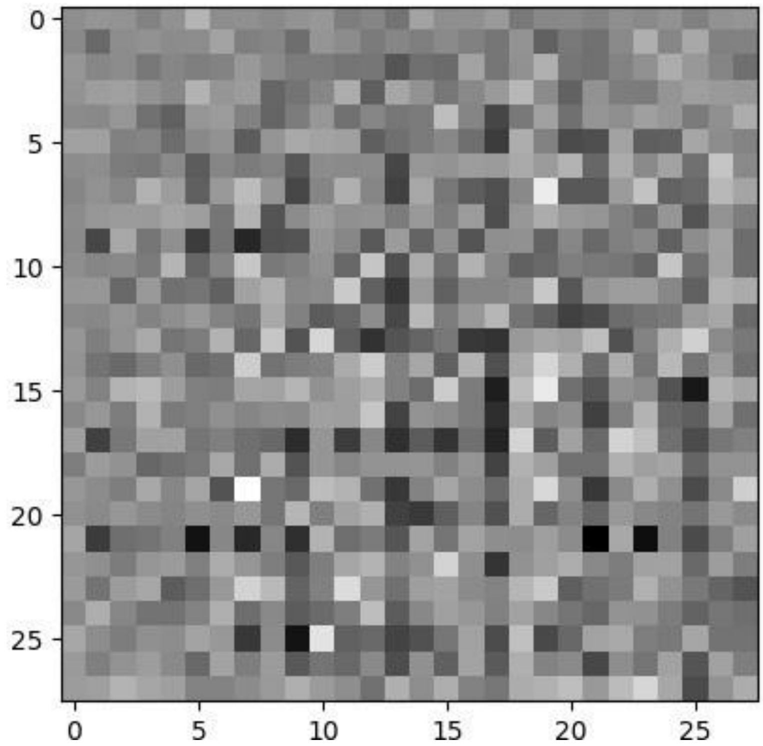
```
generator = generator_model()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12544)             1254400

 batch_normalization (BatchN  (None, 12544)            50176
 ormalization)

 leaky_re_lu (LeakyReLU)     (None, 12544)             0

 reshape (Reshape)           (None, 7, 7, 256)         0

 conv2d_transpose (Conv2DTra  (None, 7, 7, 128)        819200
 nspose)

 batch_normalization_1 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 7, 7, 128)         0

 conv2d_transpose_1 (Conv2DT  (None, 14, 14, 64)       204800
 ranspose)

 batch_normalization_2 (Batc  (None, 14, 14, 64)       256
 hNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 14, 14, 64)        0

 conv2d_transpose_2 (Conv2DT  (None, 28, 28, 1)        1600
 ranspose)

=================================================================
Total params: 2,330,944
Trainable params: 2,305,472
Non-trainable params: 25,472
_________________________________________________________________
None
```

Generate a Sample Image

```
# Create a random noise and generate a sample
noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)
# Visualize the generated sample
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7fa67e014b50>
```



#Discriminative Model

```python
def discriminator_model():
    model = tf.keras.Sequential()

    model.add(Conv2D(64, (5, 5), strides=(2, 2), padding='same',
input_shape=[28, 28, 1]))
    model.add(LeakyReLU())
    model.add(Dropout(0.3))

    model.add(Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(LeakyReLU())
    model.add(Dropout(0.3))

    model.add(Flatten())
    model.add(Dense(1))

    print(model.summary())

    return model

discriminator = discriminator_model()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_1 (Conv2D)           (None, 14, 14, 64)        1664      
                                                                 
 leaky_re_lu_4 (LeakyReLU)   (None, 14, 14, 64)        0         
                                                                 
 dropout (Dropout)           (None, 14, 14, 64)        0         
                                                                 
 conv2d_2 (Conv2D)           (None, 7, 7, 128)         204928    
                                                                 
 leaky_re_lu_5 (LeakyReLU)   (None, 7, 7, 128)         0         
                                                                 
 dropout_1 (Dropout)         (None, 7, 7, 128)         0         
                                                                 
 flatten (Flatten)           (None, 6272)              0         
                                                                 
 dense_1 (Dense)             (None, 1)                 6273      
                                                                 
=================================================================
Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0
_________________________________________________________________
None
```

Configure the Model

```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

#Training

```
import os
checkpoint_dir = '/content/GAN/'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
```

```
discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)

EPOCHS = 60

num_examples_to_generate = 16
noise_dim = 100
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

#Training Steps

```
@tf.function
def train_step(images):

  noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss,
                                               generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss,

discriminator.trainable_variables)


    generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))
```

#Image Generation Function

```
def generate_and_save_images(model, epoch, test_input):
  predictions = model(test_input, training=False)
  fig = plt.figure(figsize=(4,4))
  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')
```

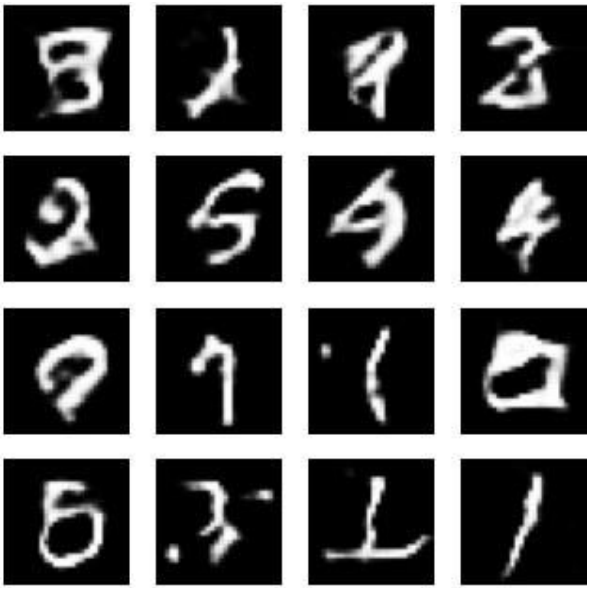
```
plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

#Train GAN

```
import time
from IPython import display
def train(dataset, epochs):

  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
            train_step(image_batch)

    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
                             seed)
if (epoch + 1) % 5 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-
start))

  display.clear_output(wait=True)
  generate_and_save_images(generator,
                           epochs,
                           seed)
```

#Start Training

```
train(train_dataset, EPOCHS)
```

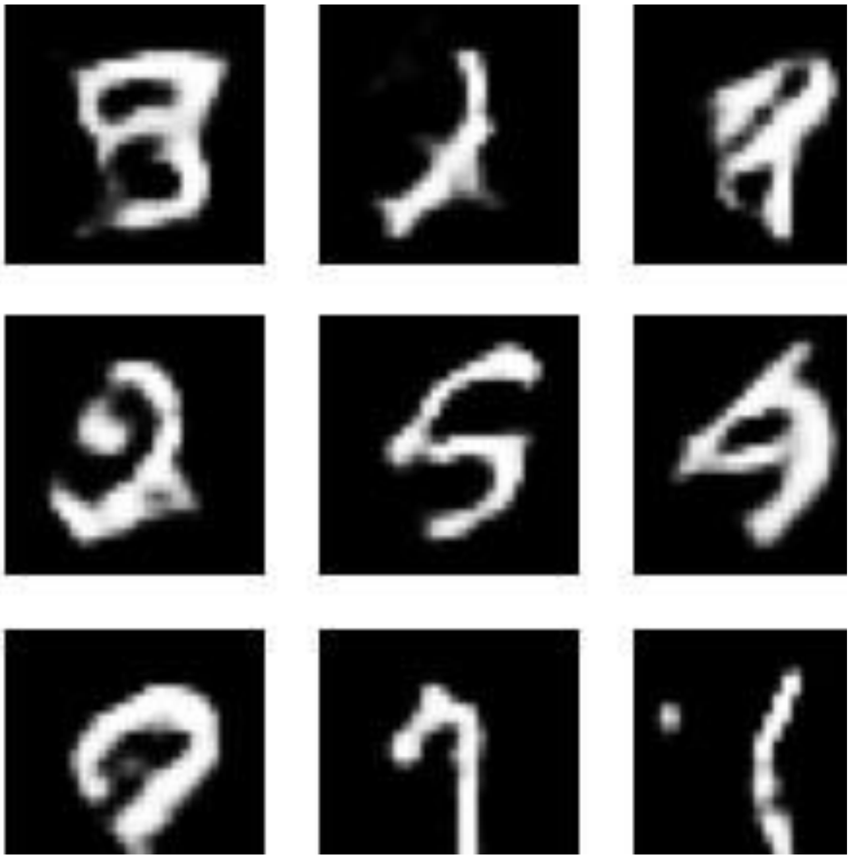


Generated Digits

```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at
0x7fa5c2d49450>
```

```
import PIL
def display_image(epoch_no):
  return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
display_image(EPOCHS)
```



```
import glob # The glob module is used for Unix style pathname pattern expansion.
import imageio # The library that provides an easy interface to read and write a wide range of image data

anim_file = 'dcgan.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
  filenames = glob.glob('image*.png')
  filenames = sorted(filenames)
  for filename in filenames:
    image = imageio.imread(filename)
    writer.append_data(image)
  # image = imageio.imread(filename)
  # writer.append_data(image)

display.Image(open('dcgan.gif','rb').read())
```