Name : Priyangana Das
Roll No : 3011001001
Class : B.E.I.T 4th year 1st semester
ML Lab Assignment : 3

Wine Dataset

Importing the Dataset :-

```
import pandas as pd
import numpy as np

df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of
ash','Magnesium','Total phenols','Flavanoids',
            'Nonflavanoid phenols','Proanthocyanins','Color
intensity','Hue','OD280/OD315 of diluted wines','Proline']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']
```

GaussianHMM Without Tuning(70-30 Split) :-

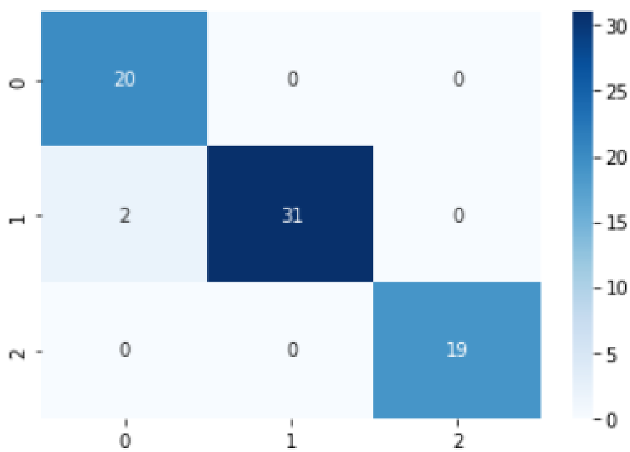
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=3, covariance_type="full")
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 0:
      strings[i] = 1
    elif y_pred[i] == 1:
      strings[i] = 2
    else:
      strings[i] = 3

strings = strings.astype(np.int)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[20  0  0]
 [ 2 31  0]
 [ 0  0 19]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.91      1.00      0.95        20
           2       1.00      0.94      0.97        33
           3       1.00      1.00      1.00        19

    accuracy                           0.97        72
   macro avg       0.97      0.98      0.97        72
weighted avg       0.97      0.97      0.97        72

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9722222222222222
```



GaussianHMM With Tuning(70-30 Split) :-

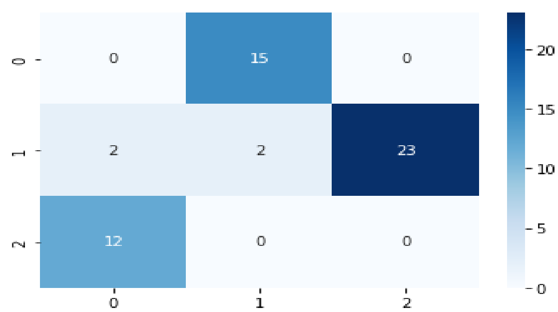
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=3, covariance_type="full",
n_iter=5,algorithm='viterbi',verbose=False
)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 0:
      strings[i] = 1
    elif y_pred[i] == 1:
      strings[i] = 2
    else:
      strings[i] = 3

strings = strings.astype(np.int)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[ 0 15  0]
 [ 2  2 23]
 [12  0  0]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        15
           2       0.12      0.07      0.09        27
           3       0.00      0.00      0.00        12

    accuracy                           0.04        54
   macro avg       0.04      0.02      0.03        54
weighted avg       0.06      0.04      0.05        54

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.037037037037037035
```



GMMHMM Without Tuning(70-30 Split) :-

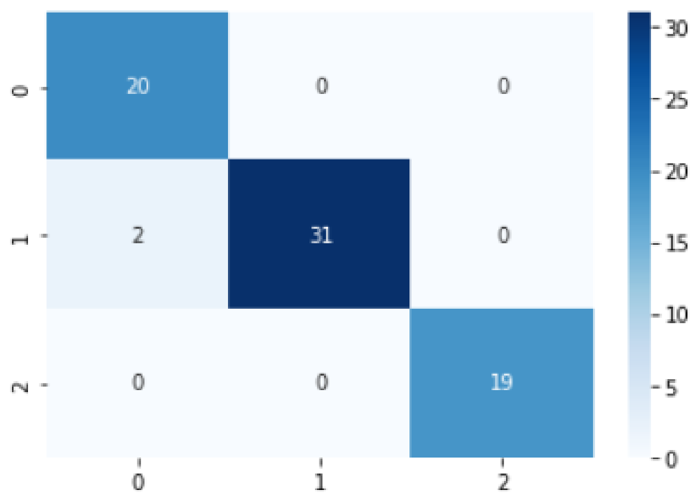
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```python
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=3,
random_state=10,covariance_type='full',algorithm='viterbi',n_iter=10)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 0:
      strings[i] = 3
    elif y_pred[i] == 1:
      strings[i] = 2
    else:
      strings[i] = 1

strings = strings.astype(np.int)
```

```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[20  0  0]
 [ 2 31  0]
 [ 0  0 19]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.91      1.00      0.95        20
           2       1.00      0.94      0.97        33
           3       1.00      1.00      1.00        19

    accuracy                           0.97        72
   macro avg       0.97      0.98      0.97        72
weighted avg       0.97      0.97      0.97        72

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9722222222222222
```



GMMHMM With Tuning(30-70) :-

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=10)
```

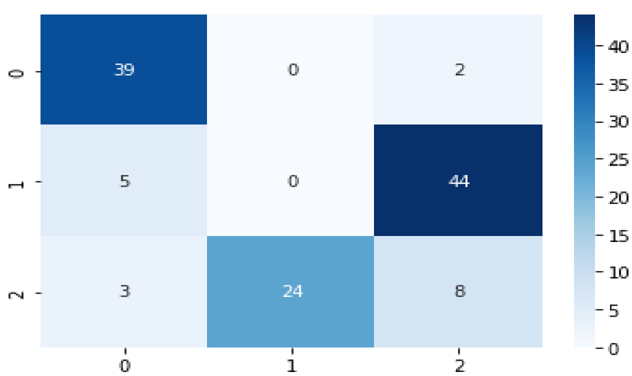
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=3, covariance_type="full",
n_iter=5,algorithm='viterbi',verbose=False
)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 0:
      strings[i] = 1
    elif y_pred[i] == 1:
      strings[i] = 2
    else:
      strings[i] = 3

strings = strings.astype(np.int)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))
```

```
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[39  0  2]
 [ 5  0 44]
 [ 3 24  8]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.83      0.95      0.89        41
           2       0.00      0.00      0.00        49
           3       0.15      0.23      0.18        35

    accuracy                           0.38       125
   macro avg       0.33      0.39      0.36       125
weighted avg       0.31      0.38      0.34       125

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.376
```



<u>MultinomialHMM With Tuning(70-30 Split)</u> **:-**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

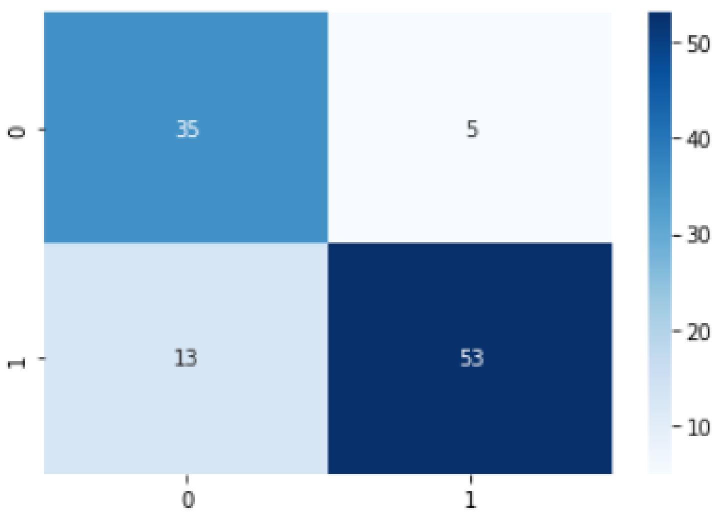
```
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4,
random_state=15,n_iter=10,algorithm='viterbi',params='ste')

import math
row = len(X_train)
col = len(X_train[0])
new = [1] * 33
for i in range(row):
    for j in range(col):
        X_train[i][j] = X_train[i][j]*10
        X_train[i][j] = math.floor(X_train[i][j])
    x = X_train[i].astype(np.int)
    new = np.vstack([new,x])

y = new
y = np.absolute(y)
X_train = y
import math
row = len(X_test)
col = len(X_test[0])
new
for i in range(row):
    for j in range(col):
        X_test[i][j] = X_test[i][j]*10
        X_test[i][j] = math.floor(X_test[i][j])
    x = X_test[i].astype(np.int)
    new = np.vstack([new,x])

y = new
y = np.absolute(y)
X_test = y


classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)
```

```
for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("b")
    else:
      strings[i] = ("g")

strings
strings = strings[0:246]
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("---------------------------------------------------")
print("---------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("---------------------------------------------------")
print("---------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
```

plt.show()

```
Confusion Matrix:
[[35  5]
 [13 53]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.73      0.88      0.80        40
           g       0.91      0.80      0.85        66

    accuracy                           0.83       106
   macro avg       0.82      0.84      0.83       106
weighted avg       0.84      0.83      0.83       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.8301886792452831
```



The maximum accuracy was achieved when the Train-Test split ratio was 70:30, which was achieved by using the Gaussian Model. The maximum range of accuracies was achieved by the Gaussian Model, followed by the GMMHMM model, which is followed by the MultinomialHMM model.

2) Ionosphere Dataset

Importing the dataset

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name =
['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','1
9'

,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Class']


df.columns = col_name


X = df.drop(['1','2','Class'], axis=1)
y = df['Class']
```

GaussianHMM Without Tuning(70-30 Split) :-

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

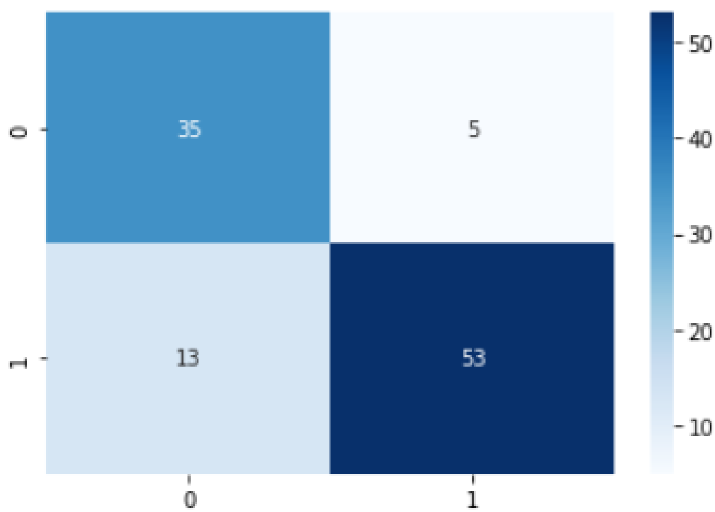
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full")
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
```

```python
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("g")
    else:
      strings[i] = ("b")

strings
```

```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[35  5]
 [13 53]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.73      0.88      0.80        40
           g       0.91      0.80      0.85        66

    accuracy                           0.83       106
   macro avg       0.82      0.84      0.83       106
weighted avg       0.84      0.83      0.83       106

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.8301886792452831
```



GaussianHMM With Tuning(70-30 Split) :-

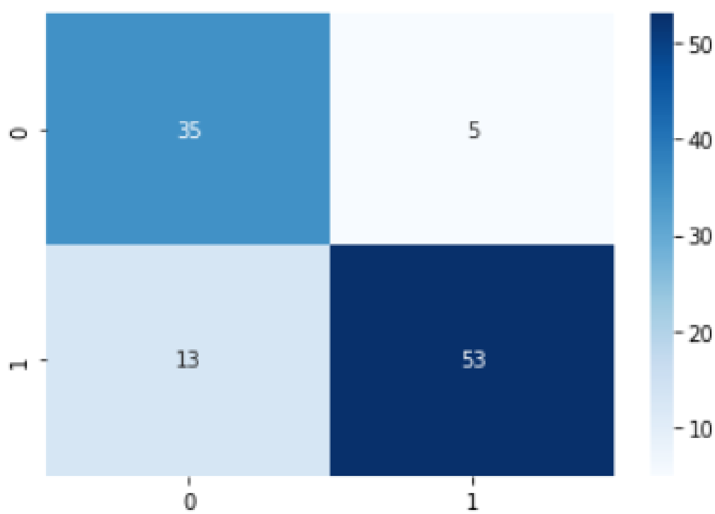
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2,
covariance_type="full",n_iter=5,algorithm='viterbi',verbose=False)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("g")
    else:
      strings[i] = ("b")

strings
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[35  5]
 [13 53]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.73      0.88      0.80        40
           g       0.91      0.80      0.85        66

    accuracy                           0.83       106
   macro avg       0.82      0.84      0.83       106
weighted avg       0.84      0.83      0.83       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.8301886792452831
```



GMMHMM Without Tuning(70-30 Split) **:-**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

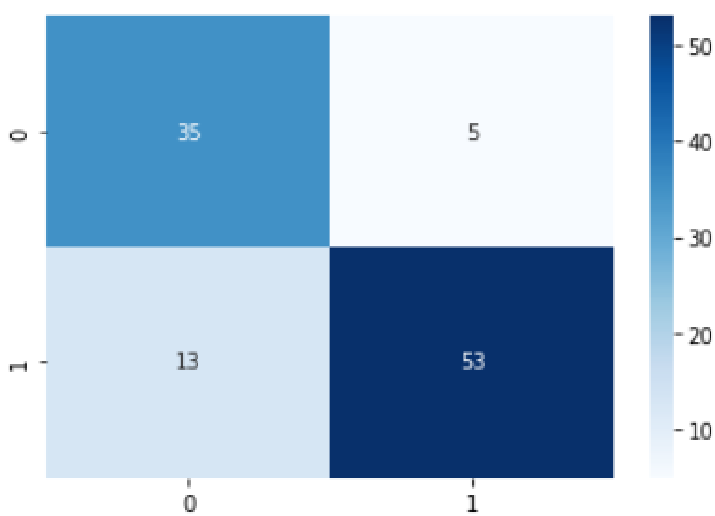
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2,
random_state=10,covariance_type='full',algorithm='viterbi',n_iter=10)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("g")
    else:
      strings[i] = ("b")

strings
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
```

```
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[35  5]
 [13 53]]
---------------------------------------------------
---------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.73      0.88      0.80        40
           g       0.91      0.80      0.85        66

    accuracy                           0.83       106
   macro avg       0.82      0.84      0.83       106
weighted avg       0.84      0.83      0.83       106

---------------------------------------------------
---------------------------------------------------
Accuracy:
0.8301886792452831
```



GMMHMM With Tuning(70-30 Split) **:-**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

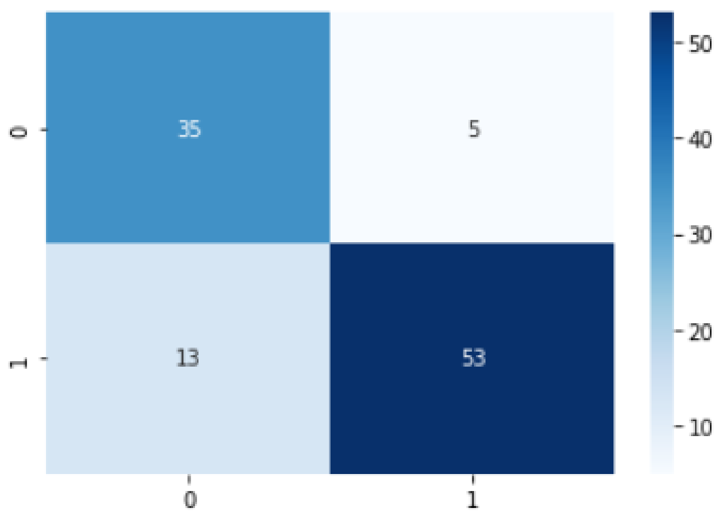
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2,
random_state=10,covariance_type='full',algorithm='viterbi',n_iter=10)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("g")
    else:
      strings[i] = ("b")

strings
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")
```

```python
print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[35  5]
 [13 53]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.73      0.88      0.80        40
           g       0.91      0.80      0.85        66

    accuracy                           0.83       106
   macro avg       0.82      0.84      0.83       106
weighted avg       0.84      0.83      0.83       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.8301886792452831
```



MultinomialHMM With Tuning(70-30 split) :-

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4,
random_state=15,n_iter=10,algorithm='viterbi',params='ste')

import math
row = len(X_train)
col = len(X_train[0])
new = [1] * 33
for i in range(row):
    for j in range(col):
        X_train[i][j] = X_train[i][j]*10
        X_train[i][j] = math.floor(X_train[i][j])
    x = X_train[i].astype(np.int)
    new = np.vstack([new,x])

y = new
y = np.absolute(y)
X_train = y
```

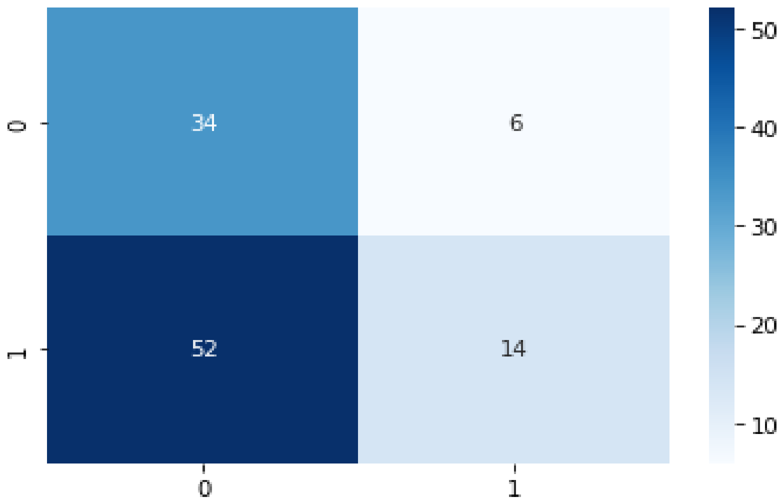
```
import math
row = len(X_test)
col = len(X_test[0])
new
for i in range(row):
    for j in range(col):
        X_test[i][j] = X_test[i][j]*10
        X_test[i][j] = math.floor(X_test[i][j])
    x = X_test[i].astype(np.int)
    new = np.vstack([new,x])

y = new
y = np.absolute(y)
X_test = y


classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)
```

```python
for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("g")
    else:
      strings[i] = ("b")

strings
strings = strings[0:106]
```

```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[34  6]
 [52 14]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.40      0.85      0.54        40
           g       0.70      0.21      0.33        66

    accuracy                           0.45       106
   macro avg       0.55      0.53      0.43       106
weighted avg       0.59      0.45      0.41       106

--------------------------------------------------
--------------------------------------------------
```

```
Accuracy:
0.4528301886792453
```



MultinomialHMM With Tuning(30-70 Split) :-

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=10)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4,
random_state=15,n_iter=10,algorithm='viterbi',params='ste')

import math
row = len(X_train)
col = len(X_train[0])
new = [1] * 33
for i in range(row):
    for j in range(col):
```

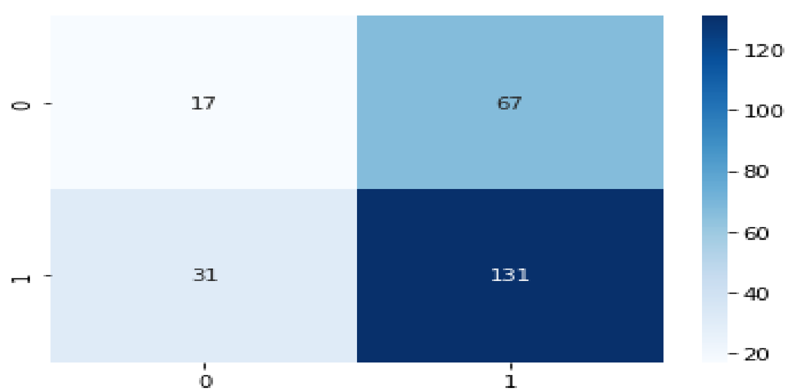
```
        X_train[i][j] = X_train[i][j]*10
        X_train[i][j] = math.floor(X_train[i][j])
    x = X_train[i].astype(np.int)
    new = np.vstack([new,x])

y = new
y = np.absolute(y)
X_train = y
```

```
import math
row = len(X_test)
col = len(X_test[0])
new
for i in range(row):
    for j in range(col):
        X_test[i][j] = X_test[i][j]*10
        X_test[i][j] = math.floor(X_test[i][j])
    x = X_test[i].astype(np.int)
    new = np.vstack([new,x])

y = new
y = np.absolute(y)
X_test = y


classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("g")
    else:
      strings[i] = ("b")

strings
strings = strings[0:106]
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))
```

```
print("------------------------------------------------------")
print("------------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[ 17  67]
 [ 31 131]]
------------------------------------------------------
------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.35      0.20      0.26        84
           g       0.66      0.81      0.73       162

    accuracy                           0.60       246
   macro avg       0.51      0.51      0.49       246
weighted avg       0.56      0.60      0.57       246

------------------------------------------------------
------------------------------------------------------
Accuracy:
0.6016260162601627
```



The maximum accuracy was achieved when the Train-Test split ratio was 70:30, which was achieved by using the Gaussian Model. The maximum range of accuracies was achieved by the Gaussian Model, followed by the GMMHMM model, which is followed by the MultinomialHMM model.

3) Breast Cancer Dataset

Importing the Dataset

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name =
['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18
','19'
           ,'20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name


X = df.drop(['1','Class'], axis=1)
y = df['Class']
```

GaussianHMM Without Tuning(70-30 Split) :-

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

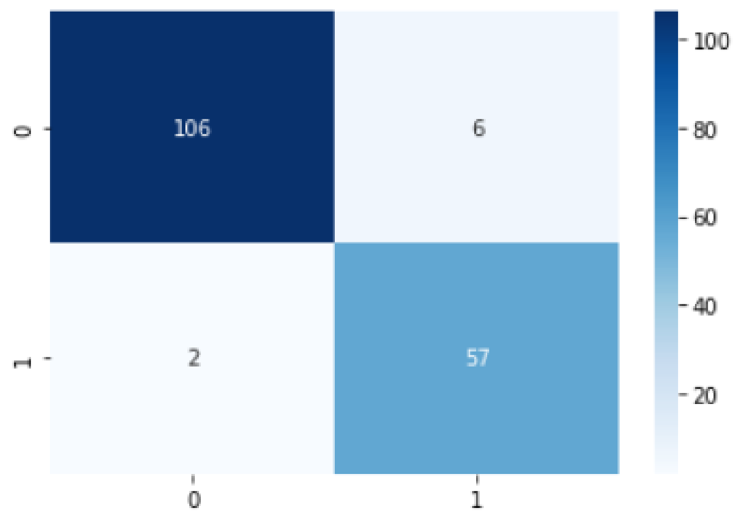
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full")
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
```

```
      strings[i] = ("M")
    else:
      strings[i] = ("B")

strings
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[106   6]
 [  2  57]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.98      0.95      0.96       112
           M       0.90      0.97      0.93        59

    accuracy                           0.95       171
   macro avg       0.94      0.96      0.95       171
weighted avg       0.96      0.95      0.95       171

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9532163742690059
```



GaussianHMM With Tuning(70-30 Split) :-

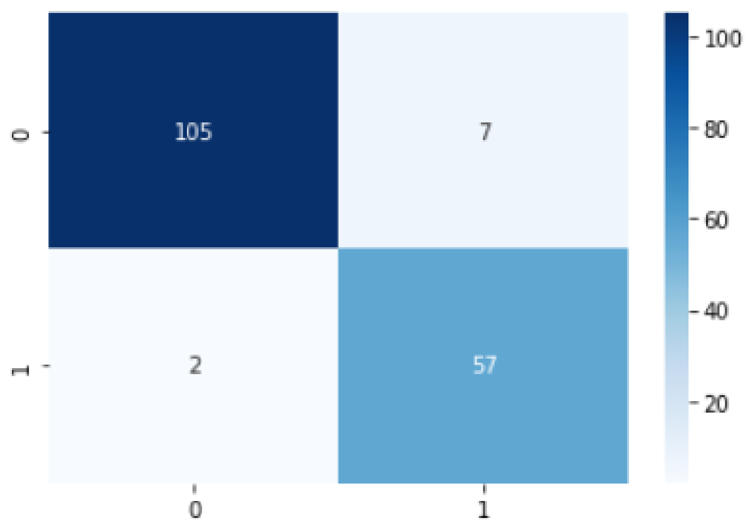
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

```python
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```python
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2,
covariance_type="full",n_iter=10,algorithm='viterbi',verbose=False)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("M")
    else:
      strings[i] = ("B")

strings
```

```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[105   7]
 [  2  57]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.98      0.94      0.96       112
           M       0.89      0.97      0.93        59

    accuracy                           0.95       171
   macro avg       0.94      0.95      0.94       171
weighted avg       0.95      0.95      0.95       171

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9473684210526315
```



__GMMHMM Without Tuning(70-30 Split)__ **:-**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

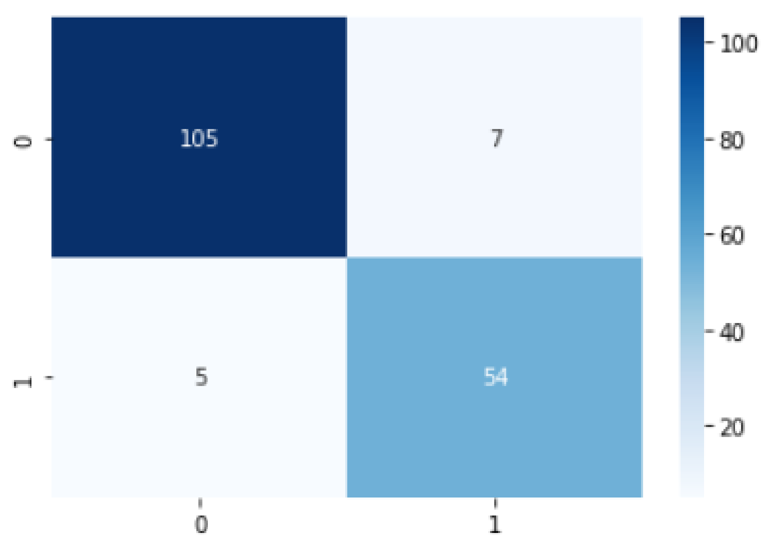
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2, random_state=10)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("M")
    else:
      strings[i] = ("B")

strings
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
```

```python
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[105   7]
 [  5  54]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.95      0.94      0.95       112
           M       0.89      0.92      0.90        59

    accuracy                           0.93       171
   macro avg       0.92      0.93      0.92       171
weighted avg       0.93      0.93      0.93       171

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9298245614035088
```



GMMHMM With Tuning(70-30 Split) **:-**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

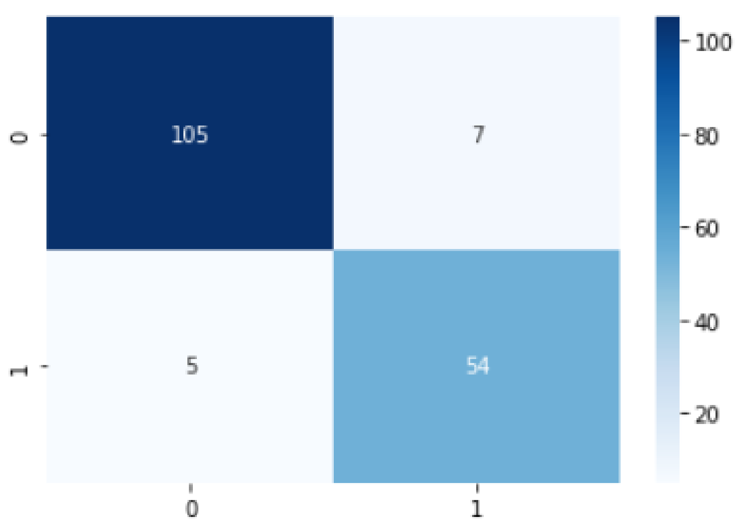
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2,
random_state=10,covariance_type='diag',algorithm='viterbi',n_iter=10)
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("M")
    else:
      strings[i] = ("B")

strings
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))
```

```
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[105   7]
 [  5  54]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.95      0.94      0.95       112
           M       0.89      0.92      0.90        59

    accuracy                           0.93       171
   macro avg       0.92      0.93      0.92       171
weighted avg       0.93      0.93      0.93       171

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9298245614035088
```



MultinomialHMM With Tuning(70-30 Split) **:-**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=10)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4,
random_state=15,n_iter=10,algorithm='viterbi',params='ste')

import math
row = len(X_train)
col = len(X_train[0])
new
for i in range(row):
    for j in range(col):
        X_train[i][j] = X_train[i][j]*10
        X_train[i][j] = math.floor(X_train[i][j])
    x = X_train[i].astype(np.int)
    new = np.vstack([new,x])

y = new
y = np.absolute(y)
X_train = y
```

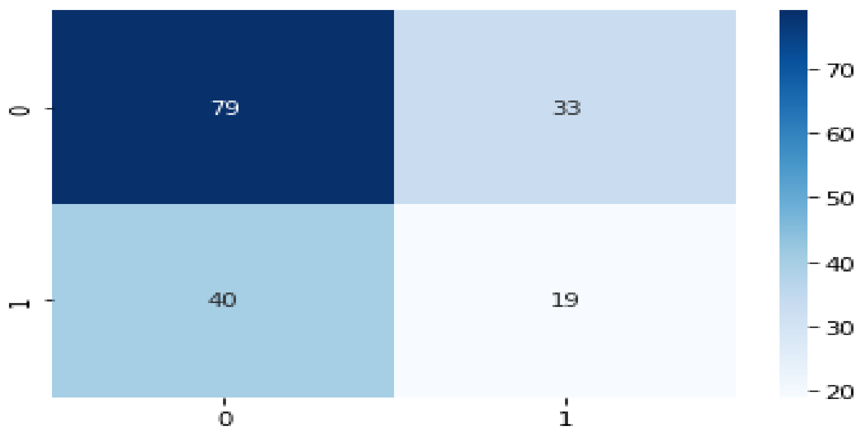
```
import math
row = len(X_test)
col = len(X_test[0])
new
for i in range(row):
    for j in range(col):
        X_test[i][j] = X_test[i][j]*10
        X_test[i][j] = math.floor(X_test[i][j])
    x = X_test[i].astype(np.int)
    new = np.vstack([new,x])

y = new
y = np.absolute(y)
X_test = y
```

```python
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

size = len(y_pred)
strings = np.empty(size, np.unicode_)

for i in range (size):
    if y_pred[i] == 1:
      strings[i] = ("M")
    else:
      strings[i] = ("B")

strings
strings = strings[0:171]
```

```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, strings))

print("--------------------------------------------------")
print("--------------------------------------------------")


print("Performance Evaluation")
print(classification_report(y_test, strings))


print("--------------------------------------------------")
print("--------------------------------------------------")

print("Accuracy:")
print(accuracy_score(y_test, strings))

import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, strings)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion Matrix:
[[79 33]
 [40 19]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support
```

```
           B       0.66      0.71      0.68       112
           M       0.37      0.32      0.34        59

    accuracy                           0.57       171
   macro avg       0.51      0.51      0.51       171
weighted avg       0.56      0.57      0.57       171

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.5730994152046783
```



The maximum accuracy was achieved when the Train-Test split ratio was 70:30, which was achieved by using the Gaussian Model. The maximum range of accuracies was achieved by the Gaussian Model, followed by the GMMHMM model, which is followed by the MultinomialHMM model.

1) CIFAR-10

Importing dataset **:-**

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

# dataset preparation
```

```python
from tensorflow.keras import datasets,layers,models

(train_images,train_labels) , (test_images,test_labels) =
datasets.cifar10.load_data()

# Normalize pixel values to be within 0 , 1
train_images , test_images = train_images/255.0 , test_images/255.0
```

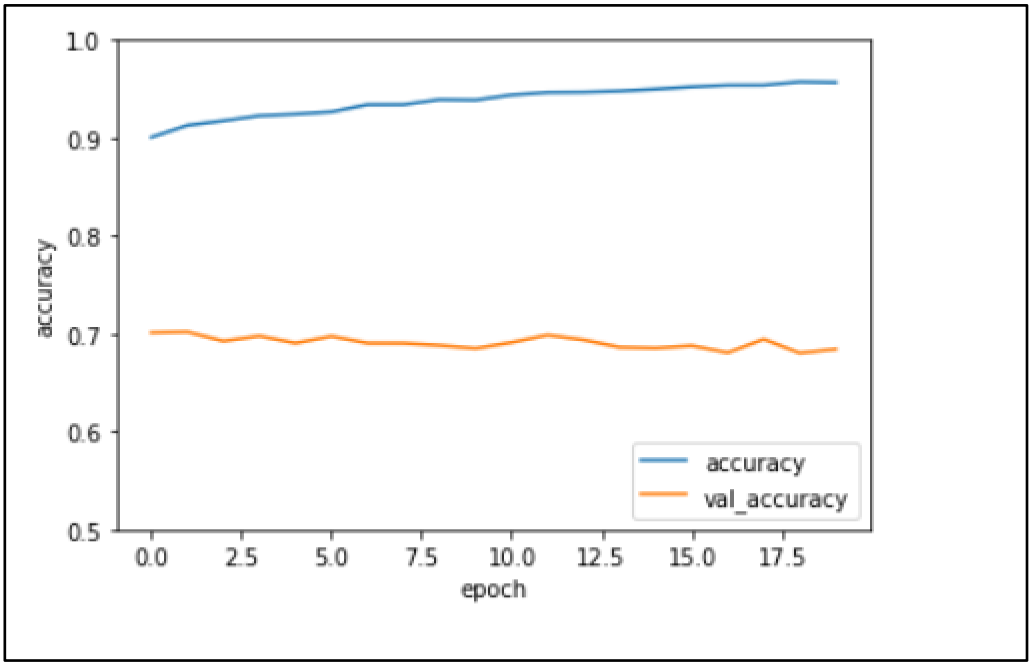
```python
input_shape = train_images[0].shape

model = models.Sequential()
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=input_shape))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10))


model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(fr
om_logits=True),metrics=['accuracy'])

history =
model.fit(train_images,train_labels,epochs=20,validation_data=(test_images,test_labe
ls))
```

```
Epoch 11/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1561 - accuracy: 0.9437 - val_loss: 1.8716 - val_accuracy: 0.6910
Epoch 12/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1498 - accuracy: 0.9463 - val_loss: 1.9775 - val_accuracy: 0.6986
Epoch 13/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1524 - accuracy: 0.9466 - val_loss: 2.0503 - val_accuracy: 0.6936
Epoch 14/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1490 - accuracy: 0.9477 - val_loss: 2.0715 - val_accuracy: 0.6861
Epoch 15/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1429 - accuracy: 0.9497 - val_loss: 2.1616 - val_accuracy: 0.6852
Epoch 16/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1356 - accuracy: 0.9520 - val_loss: 2.2363 - val_accuracy: 0.6877
Epoch 17/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1327 - accuracy: 0.9537 - val_loss: 2.2239 - val_accuracy: 0.6806
Epoch 18/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1348 - accuracy: 0.9538 - val_loss: 2.2102 - val_accuracy: 0.6943
Epoch 19/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1250 - accuracy: 0.9571 - val_loss: 2.3900 - val_accuracy: 0.6802
Epoch 20/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1245 - accuracy: 0.9565 - val_loss: 2.3173 - val_accuracy: 0.6842
```

```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label='val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.ylim([0.5,1])
plt.legend(loc='lower right')

plt.show()
```




```
test_loss , test_acc = model.evaluate(test_images,test_labels,verbose=2)
```

```
313/313 - 3s - loss: 2.3173 - accuracy: 0.6842
```

2) MNIST

Importing the Dataset :-

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

# dataset preparation

from tensorflow.keras import datasets,layers,models
```

```python
(train_images,train_labels) , (test_images,test_labels) = datasets.mnist.load_data()

# Normalize pixel values to be within 0 , 1
train_images , test_images = train_images/255.0 , test_images/255.0
train_images = np.reshape(train_images, train_images.shape + (1,))
test_images = np.reshape(test_images, test_images.shape + (1,))

train_images[0].shape
```

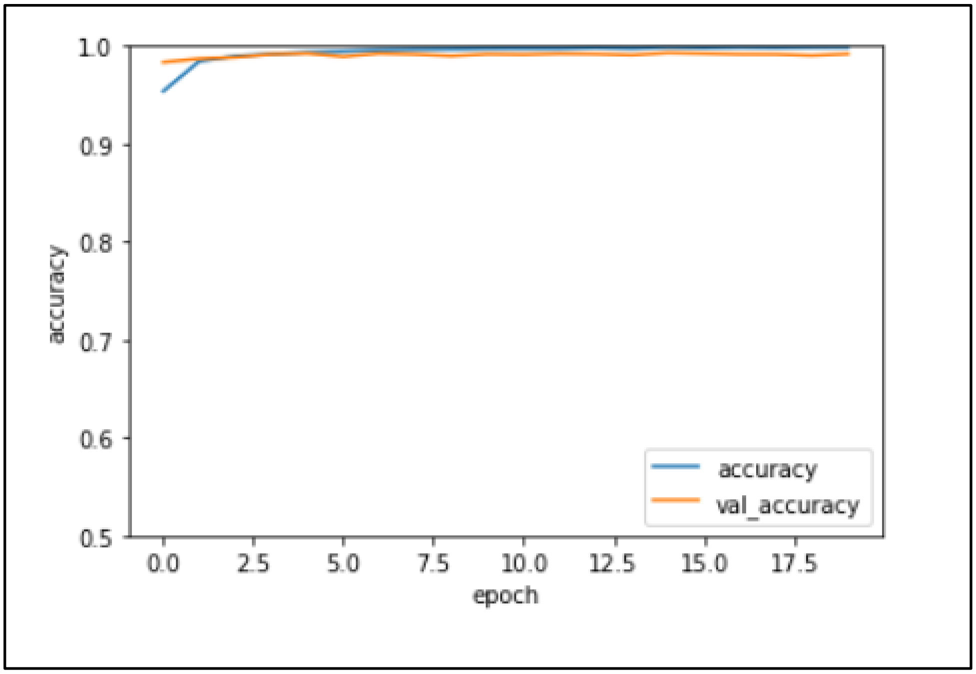
```python
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10))


model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 3, 3, 64)          36928     
_________________________________________________________________
flatten_3 (Flatten)          (None, 576)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 64)                36928     
_________________________________________________________________
dense_7 (Dense)              (None, 10)                650       
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

```python
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])

history =
model.fit(train_images,train_labels,epochs=20,validation_data=(test_images,test_labels))
```

```
1875/1875 [==============================] - 57s 30ms/step - loss: 0.0079 - accuracy: 0.9973 - val_loss: 0.0319 - val_accuracy: 0.9913
Epoch 12/20
1875/1875 [==============================] - 57s 31ms/step - loss: 0.0084 - accuracy: 0.9973 - val_loss: 0.0329 - val_accuracy: 0.9921
Epoch 13/20
1875/1875 [==============================] - 57s 31ms/step - loss: 0.0067 - accuracy: 0.9980 - val_loss: 0.0343 - val_accuracy: 0.9918
Epoch 14/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0078 - accuracy: 0.9973 - val_loss: 0.0390 - val_accuracy: 0.9908
Epoch 15/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0048 - accuracy: 0.9985 - val_loss: 0.0374 - val_accuracy: 0.9930
Epoch 16/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0069 - accuracy: 0.9980 - val_loss: 0.0336 - val_accuracy: 0.9923
Epoch 17/20
1875/1875 [==============================] - 57s 31ms/step - loss: 0.0049 - accuracy: 0.9985 - val_loss: 0.0430 - val_accuracy: 0.9916
Epoch 18/20
1875/1875 [==============================] - 57s 31ms/step - loss: 0.0053 - accuracy: 0.9982 - val_loss: 0.0397 - val_accuracy: 0.9915
Epoch 19/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0048 - accuracy: 0.9986 - val_loss: 0.0540 - val_accuracy: 0.9903
Epoch 20/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0043 - accuracy: 0.9987 - val_loss: 0.0419 - val_accuracy: 0.9919
```

```python
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label='val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.ylim([0.5,1])
plt.legend(loc='lower right')

plt.show()
```




```
test_loss , test_acc = model.evaluate(test_images,test_labels,verbose=2)
```

```
313/313 - 3s - loss: 0.0419 - accuracy: 0.9919
```

3) SAVEE

Importing the Dataset :-

```
!unzip "/content/drive/MyDrive/SaveeDataset.zip"
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels=
n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T
```

```
def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np

rootDirectory = "/content/AudioData/"
personNames = ["DC","JE","JK","KL"]

classes = ["a" , "d" , "f", "h", "n", "sa" , "su" ]

X = list()
y = list()

for person in personNames:
  directory = os.path.join(rootDirectory,person)
  for filename in os.listdir(directory):
    filePath = os.path.join(directory, filename)
    data = load_audio_file(file_path=filePath)
    data = np.reshape(data, data.shape + (1,))
    if(filename[0:1] in classes):
      X.append(data)
      y.append(classes.index(filename[0:1]))
    elif(filename[0:2] in classes):
      X.append(data)
      y.append(classes.index(filename[0:2]))
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

```python
# dataset preparation

from tensorflow.keras import datasets,layers,models
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, train_size=
0.7 ,random_state=10)
```

```python
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(157,320,1)))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10))


model.summary()
```

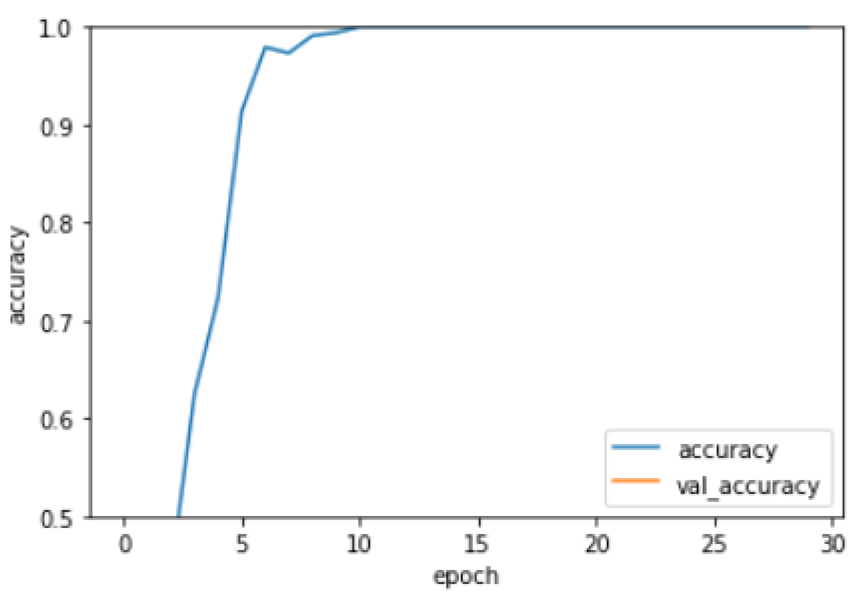
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 155, 318, 32)      320
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 77, 159, 32)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 75, 157, 64)       18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 37, 78, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 35, 76, 64)        36928
_________________________________________________________________
flatten_3 (Flatten)          (None, 170240)            0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                10895424
_________________________________________________________________
dense_7 (Dense)              (None, 10)                650
=================================================================
Total params: 10,951,818
Trainable params: 10,951,818
Non-trainable params: 0
_________________________________________________________________
```

```python
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(fr
om_logits=True),metrics=['accuracy'])
```

```
history = model.fit(X_train,y_train,epochs=30,validation_data=(X_test,y_test))
```

```
11/11 [==============================] - 27s 2s/step - loss: 2.2025e-05 - accuracy: 1.0000 - val_loss: 7.0087 - val_accuracy: 0.3056
Epoch 25/30
11/11 [==============================] - 27s 2s/step - loss: 1.9328e-05 - accuracy: 1.0000 - val_loss: 7.0391 - val_accuracy: 0.2986
Epoch 26/30
11/11 [==============================] - 27s 2s/step - loss: 1.7196e-05 - accuracy: 1.0000 - val_loss: 7.0967 - val_accuracy: 0.2986
Epoch 27/30
11/11 [==============================] - 27s 2s/step - loss: 1.5431e-05 - accuracy: 1.0000 - val_loss: 7.1239 - val_accuracy: 0.3056
Epoch 28/30
11/11 [==============================] - 27s 2s/step - loss: 1.3852e-05 - accuracy: 1.0000 - val_loss: 7.1493 - val_accuracy: 0.2986
Epoch 29/30
11/11 [==============================] - 27s 2s/step - loss: 1.2641e-05 - accuracy: 1.0000 - val_loss: 7.2041 - val_accuracy: 0.2986
Epoch 30/30
11/11 [==============================] - 27s 2s/step - loss: 1.1668e-05 - accuracy: 1.0000 - val_loss: 7.2112 - val_accuracy: 0.2986
```

```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label='val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.ylim([0.5,1])
plt.legend(loc='lower right')

plt.show()
```




```
test_loss , test_acc = model.evaluate(X_test,y_test,verbose=2)
```

```
5/5 - 3s - loss: 7.2112 - accuracy: 0.2986
```

4) EmoDB

Importing the Dataset :-

```
!unzip "/content/drive/MyDrive/EmoDB.zip"
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels=
n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
# Preprocessing the dataset
import os
```

```
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np

directory = "/content/wav/"

classes = ["W" ,"L" ,"E" ,"A" , "F" ,"T" ,"N" ]

X = list()
y = list()

for filename in os.listdir(directory):
  filePath = os.path.join(directory, filename)
  data = load_audio_file(file_path=filePath)
  data = np.reshape(data, data.shape + (1,))
  if(filename[5:6] in classes):
    X.append(data)
    y.append(classes.index(filename[5:6]))

X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models
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, train_size=
0.7 ,random_state=10)
```

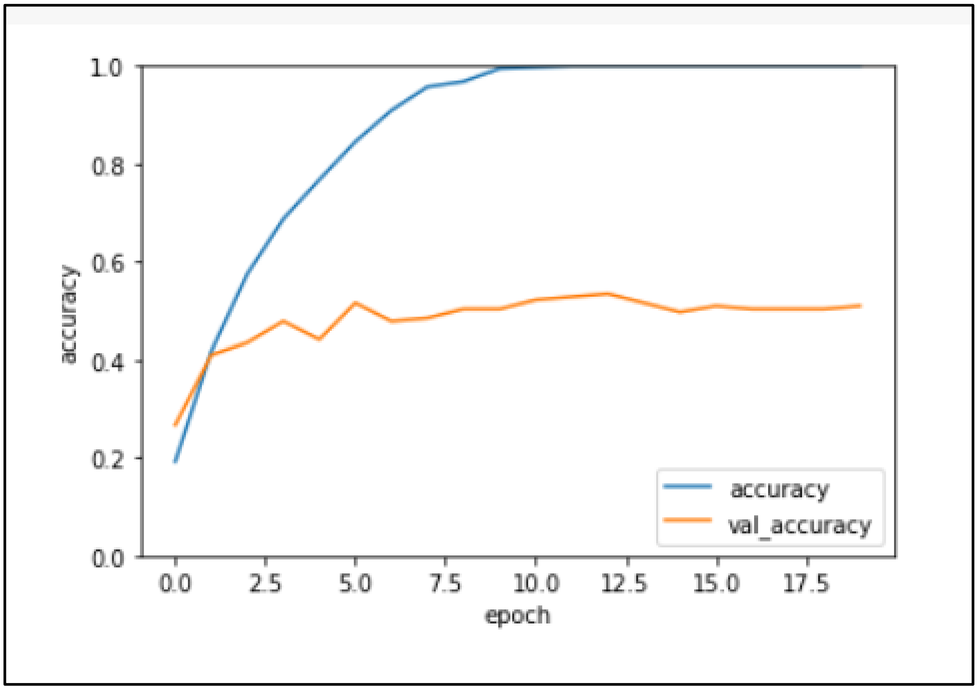
```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(157,320,1)))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10))


model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 155, 318, 32)      320
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 77, 159, 32)       0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 75, 157, 64)       18496
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 37, 78, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 35, 76, 64)        36928
_________________________________________________________________
flatten_4 (Flatten)          (None, 170240)            0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                10895424
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 10,951,818
Trainable params: 10,951,818
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(fr
om_logits=True),metrics=['accuracy']) history =
model.fit(X_train,y_train,epochs=20,validation_data=(X_test,y_test))
```

```
Epoch 14/20
12/12 [==============================] - 30s 2s/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 3.9037 - val_accuracy: 0.5155
Epoch 15/20
12/12 [==============================] - 30s 2s/step - loss: 7.0827e-04 - accuracy: 1.0000 - val_loss: 4.0446 - val_accuracy: 0.4969
Epoch 16/20
12/12 [==============================] - 30s 2s/step - loss: 4.9740e-04 - accuracy: 1.0000 - val_loss: 4.1150 - val_accuracy: 0.5093
Epoch 17/20
12/12 [==============================] - 30s 3s/step - loss: 3.8747e-04 - accuracy: 1.0000 - val_loss: 4.1542 - val_accuracy: 0.5031
Epoch 18/20
12/12 [==============================] - 30s 2s/step - loss: 3.0542e-04 - accuracy: 1.0000 - val_loss: 4.2023 - val_accuracy: 0.5031
Epoch 19/20
12/12 [==============================] - 31s 3s/step - loss: 2.5256e-04 - accuracy: 1.0000 - val_loss: 4.2239 - val_accuracy: 0.5031
Epoch 20/20
12/12 [==============================] - 30s 2s/step - loss: 2.1154e-04 - accuracy: 1.0000 - val_loss: 4.2753 - val_accuracy: 0.5093
```

```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label='val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.ylim([0,1])
plt.legend(loc='lower right')

plt.show()
```




```
test_loss , test_acc = model.evaluate(X_test,y_test,verbose=2)
```

```
6/6 - 3s - loss: 4.2753 - accuracy: 0.5093
```

It was observed that the more layers we add the higher accuracy we can achieve. At the same time, if we keep on adding more layers, the final accuracy will saturate. Also, the number of convolution and the pooling layers play an important role in training the model.

1) VGG-16

```
from google.colab import drive drive.mount('/content/drive')
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import skimage.transform
from __future__ import print_function
```

```
!pip install keras_applications

import numpy as np
import warnings

from keras.models import Model
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Input
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import GlobalMaxPooling2D
from keras.layers import GlobalAveragePooling2D
from keras.preprocessing import image
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
from keras import backend as K
from keras.applications.imagenet_utils import decode_predictions
from keras.applications.imagenet_utils import preprocess_input
from keras_applications.imagenet_utils import _obtain_input_shape
from keras.utils.layer_utils import get_source_inputs
```

```
def load_preprocess_training_batch(X_train):

    new = []

    for item in X_train:
        tmpFeature = skimage.transform.resize(item, (224, 224), mode='constant')
        new.append(tmpFeature)

    return new
```

```
def preprocess_data(X_train):

    for item in X_train:
      item = np.expand_dims(item, axis=0)
      item = preprocess_input(item)

    return X_train
```

And def VGG16 is also declared(which code is too long and can be available online) .

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
```

```
%matplotlib inline
import numpy as np
import skimage.transform
```

1.1) CIFAR-10

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.cifar10.load_data()

X_train = X_train[0:2000]
y_train = y_train[0:2000]
X_test = X_test[0:2000]
y_test = y_test[0:2000]
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)
```

```
X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)
```

```
X_train_resized = X_train_resized / 255
X_test_resized = X_test_resized / 255
```

```
X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

```
model = VGG16(include_top=True, weights='imagenet')

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_resized, y_train, epochs=5)
```

```
Epoch 1/5
63/63 [==============================] - 97s 855ms/step - loss: nan - accuracy: 0.0985
Epoch 2/5
63/63 [==============================] - 47s 743ms/step - loss: nan - accuracy: 0.1010
Epoch 3/5
63/63 [==============================] - 47s 745ms/step - loss: nan - accuracy: 0.1010
Epoch 4/5
63/63 [==============================] - 47s 744ms/step - loss: nan - accuracy: 0.1010
Epoch 5/5
63/63 [==============================] - 47s 744ms/step - loss: nan - accuracy: 0.1010

[ ] model.evaluate(X_test_resized, y_test)

63/63 [==============================] - 14s 225ms/step - loss: nan - accuracy: 0.0980
[nan, 0.09799999743700027]
```

```
model.evaluate(X_test_resized, y_test)
```

```
[nan, 0.09799999743700027]
```

1.2) MNIST

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()

X_train = X_train[0:2000]
y_train = y_train[0:2000]
X_test = X_test[0:2000]
y_test = y_test[0:2000]
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)

X_train_resized = X_train_resized / 255.0
X_test_resized = X_test_resized / 255.0

X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

```
import cv2

X_train_new = list()
```

```
for i in range(len(X_train_resized)):
  g  = X_train_resized[i]
  X_train_new.append(cv2.merge([g,g,g]))

X_train_new = np.asarray(X_train_new,dtype=np.float32)

X_test_new = list()

for i in range(len(X_test_resized)):
  g  = X_test_resized[i]
  X_test_new.append(cv2.merge([g,g,g]))

X_test_new = np.asarray(X_test_new,dtype=np.float32)
```

```
model = VGG16(include_top=True, weights='imagenet')

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_new, y_train, epochs=5)
```

```
Epoch 1/5
63/63 [==============================] - 71s 877ms/step - loss: 3.6193 - accuracy: 0.0885
Epoch 2/5
63/63 [==============================] - 48s 764ms/step - loss: 2.5311 - accuracy: 0.0990
Epoch 3/5
63/63 [==============================] - 48s 763ms/step - loss: 2.5246 - accuracy: 0.1105
Epoch 4/5
63/63 [==============================] - 48s 763ms/step - loss: 2.5036 - accuracy: 0.1040
Epoch 5/5
63/63 [==============================] - 48s 763ms/step - loss: 2.4806 - accuracy: 0.0965
```

```
model.evaluate(X_test_new, y_test)
```

```
63/63 [==============================] - 15s 233ms/step - loss: 2.6352 - accuracy: 0.1095
[2.6351511478424072, 0.10949999839067459]
```


```
model.evaluate(X_test_new, y_test)
```



```
[2.6351511478424072, 0.10949999839067459]
```

1.3) SAVEE

```
!unzip "/content/drive/MyDrive/SaveeDataset.zip"
```

```
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels=
n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2

rootDirectory = "/content/AudioData/"
```

```python
personNames = ["DC","JE","JK","KL"]

classes = ["a" , "d" , "f", "h", "n", "sa" , "su" ]

X = list()
y = list()

for person in personNames:
  directory = os.path.join(rootDirectory,person)
  for filename in os.listdir(directory):
    filePath = os.path.join(directory, filename)
    a = load_audio_file(file_path=filePath)
    data = cv2.merge([a,a,a])
    # data = np.reshape(data, data.shape + (1,))
    if(filename[0:1] in classes):
      X.append(data)
      y.append(classes.index(filename[0:1]))
    elif(filename[0:2] in classes):
      X.append(data)
      y.append(classes.index(filename[0:2]))
```

```python
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, train_size=
0.5 ,random_state=10)
```

```python
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)

X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

```python
model = VGG16(include_top=True, weights='imagenet')

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(X_train_resized, y_train, epochs=50)
```

```
8/8 [==============================] - 6s 708ms/step - loss: nan - accuracy: 0.1208
Epoch 45/50
8/8 [==============================] - 6s 706ms/step - loss: nan - accuracy: 0.1208
Epoch 46/50
8/8 [==============================] - 6s 705ms/step - loss: nan - accuracy: 0.1208
Epoch 47/50
8/8 [==============================] - 6s 706ms/step - loss: nan - accuracy: 0.1208
Epoch 48/50
8/8 [==============================] - 6s 709ms/step - loss: nan - accuracy: 0.1208
Epoch 49/50
8/8 [==============================] - 6s 706ms/step - loss: nan - accuracy: 0.1208
Epoch 50/50
8/8 [==============================] - 6s 706ms/step - loss: nan - accuracy: 0.1208
```

```
model.evaluate(X_test_resized, y_test)
```

```
8/8 [==============================] - 2s 215ms/step - loss: nan - accuracy: 0.1292
[nan, 0.12916666269302368]
```

```
model.evaluate(X_test_resized, y_test)
```

```
[nan, 0.12916666269302368]
```

1.4) EmoDB

```
!unzip "/content/drive/MyDrive/EmoDB.zip"
```

```
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):
```

```
    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels=
n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2

directory = "/content/wav/"

classes = ["W" ,"L" ,"E" ,"A" , "F" ,"T" ,"N" ]

X = list()
y = list()

for filename in os.listdir(directory):
  filePath = os.path.join(directory, filename)
  a = load_audio_file(file_path=filePath)
  data = cv2.merge([a,a,a])
  if(filename[5:6] in classes):
    X.append(data)
    y.append(classes.index(filename[5:6]))
```

```
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, train_size=
0.5 ,random_state=10)
```

```python
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)

X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

```python
model = VGG16(include_top=True, weights='imagenet')

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_resized, y_train, epochs=20)
```

```
9/9 [==============================] - 6s 711ms/step - loss: nan - accuracy: 0.2247
Epoch 14/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
Epoch 15/20
9/9 [==============================] - 6s 713ms/step - loss: nan - accuracy: 0.2247
Epoch 16/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
Epoch 17/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
Epoch 18/20
9/9 [==============================] - 6s 711ms/step - loss: nan - accuracy: 0.2247
Epoch 19/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
Epoch 20/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
```

```
model.evaluate(X_test_resized, y_test)
```

```
9/9 [==============================] - 6s 718ms/step - loss: nan - accuracy: 0.2500
[nan, 0.25]
```

```
model.evaluate(X_test_resized, y_test)
```

```
[nan, 0.25]
```

The entire model can be broken down into 5 blocks, where each block contains 3 convolution and 1 max-pooling layers.

Looking at the complexity of the model and the limitations of google colab, I have reduced the input size for the model,i.e., i have taken 2000 training data points and 2000 testing data points.

2) ResNet-50

```
from google.colab import drive
drive.mount('/content/drive')
from __future__ import print_function

```

```
import numpy as np
import warnings

!pip install keras_applications

from keras.layers import Input
from keras import layers
from keras.layers import Dense
from keras.layers import Activation
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import GlobalMaxPooling2D
from keras.layers import ZeroPadding2D
from keras.layers import AveragePooling2D
from keras.layers import GlobalAveragePooling2D
from keras.layers import BatchNormalization
from keras.models import Model
from keras.preprocessing import image
import keras.backend as K
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
from keras.applications.imagenet_utils import decode_predictions
from keras.applications.imagenet_utils import preprocess_input
from keras_applications.imagenet_utils import _obtain_input_shape
from keras.utils.layer_utils import get_source_inputs

import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import skimage.transform
```

And def ResNet50 is also declared(which code is too long and can be available online) .

```
def load_preprocess_training_batch(X_train):

    new = []

    for item in X_train:
```

```
        tmpFeature = skimage.transform.resize(item, (224, 224), mode='constant')
        new.append(tmpFeature)

    return new
```

```
def preprocess_data(X_train):

    for item in X_train:
      item = np.expand_dims(item, axis=0)
      item = preprocess_input(item)

    return X_train
```

2.1) CIFAR-10

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.cifar10.load_data()

X_train = X_train[0:2000]
y_train = y_train[0:2000]
X_test = X_test[0:2000]
y_test = y_test[0:2000]

X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)
```

```
X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)
```

```
X_train_resized = X_train_resized / 255
X_test_resized = X_test_resized / 255
```

```
X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

```
model = ResNet50(include_top=True, weights='imagenet')

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(X_train_resized, y_train, epochs=5)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet5
102858752/102853048 [==============================] - 1s 0us/step
102866944/102853048 [==============================] - 1s 0us/step
Epoch 1/5
63/63 [==============================] - 81s 703ms/step - loss: 2.9229 - accuracy: 0.0975
Epoch 2/5
63/63 [==============================] - 42s 673ms/step - loss: 2.4506 - accuracy: 0.1040
Epoch 3/5
63/63 [==============================] - 42s 673ms/step - loss: 2.2995 - accuracy: 0.1755
Epoch 4/5
63/63 [==============================] - 42s 672ms/step - loss: 2.1401 - accuracy: 0.2325
Epoch 5/5
63/63 [==============================] - 42s 672ms/step - loss: 2.0272 - accuracy: 0.2715
```

```
model.evaluate(X_test_resized, y_test)
```

```
63/63 [==============================] - 15s 217ms/step - loss: 16.8393 - accuracy: 0.0000e+00
[16.839269638061523, 0.0]
```

2.2) MNIST

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()

X_train = X_train[0:2000]
y_train = y_train[0:2000]
X_test = X_test[0:2000]
y_test = y_test[0:2000]
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)

X_train_resized = X_train_resized / 255.0
X_test_resized = X_test_resized / 255.0

X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

```python
import cv2

X_train_new = list()

for i in range(len(X_train_resized)):
  g  = X_train_resized[i]
  X_train_new.append(cv2.merge([g,g,g]))

X_train_new = np.asarray(X_train_new,dtype=np.float32)

X_test_new = list()

for i in range(len(X_test_resized)):
  g  = X_test_resized[i]
  X_test_new.append(cv2.merge([g,g,g]))

X_test_new = np.asarray(X_test_new,dtype=np.float32)
```

```python
model = ResNet50(include_top=True, weights='imagenet')

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_new, y_train, epochs=10)
```

```
Epoch 6/10
63/63 [==============================] - 44s 705ms/step - loss: 0.0647 - accuracy: 0.9825
Epoch 7/10
63/63 [==============================] - 44s 705ms/step - loss: 0.0655 - accuracy: 0.9815
Epoch 8/10
63/63 [==============================] - 44s 705ms/step - loss: 0.0429 - accuracy: 0.9890
Epoch 9/10
63/63 [==============================] - 44s 705ms/step - loss: 0.0272 - accuracy: 0.9950
Epoch 10/10
63/63 [==============================] - 44s 704ms/step - loss: 0.0121 - accuracy: 0.9985
```

```python
model.evaluate(X_test_new, y_test)
```

```
63/63 [==============================] - 14s 227ms/step - loss: 13.1569 - accuracy: 0.0000e+00
[13.156914710998535, 0.0]
```

2.3) SAVEE

```
!unzip "/content/drive/MyDrive/SaveeDataset.zip"
```

```
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels= n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
```

```python
  return data

# Preprocessing the dataset
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2

rootDirectory = "/content/AudioData/"
personNames = ["DC","JE","JK","KL"]

classes = ["a" , "d" , "f", "h", "n", "sa" , "su" ]

X = list()
y = list()

for person in personNames:
  directory = os.path.join(rootDirectory,person)
  for filename in os.listdir(directory):
    filePath = os.path.join(directory, filename)
    a = load_audio_file(file_path=filePath)
    data = cv2.merge([a,a,a])
    # data = np.reshape(data, data.shape + (1,))
    if(filename[0:1] in classes):
      X.append(data)
      y.append(classes.index(filename[0:1]))
    elif(filename[0:2] in classes):
      X.append(data)
      y.append(classes.index(filename[0:2]))
```

```python
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, train_size= 0.5 ,r
andom_state=10)
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)

X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

```
model = ResNet50(include_top=True, weights='imagenet')

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_resized, y_train, epochs=10)
```

```
Epoch 4/10
8/8 [==============================] - 5s 669ms/step - loss: 1.0197 - accuracy: 0.6833
Epoch 5/10
8/8 [==============================] - 5s 667ms/step - loss: 0.5054 - accuracy: 0.9167
Epoch 6/10
8/8 [==============================] - 5s 669ms/step - loss: 0.2390 - accuracy: 0.9833
Epoch 7/10
8/8 [==============================] - 5s 671ms/step - loss: 0.0966 - accuracy: 1.0000
Epoch 8/10
8/8 [==============================] - 5s 668ms/step - loss: 0.0691 - accuracy: 1.0000
Epoch 9/10
8/8 [==============================] - 5s 669ms/step - loss: 0.0550 - accuracy: 1.0000
Epoch 10/10
8/8 [==============================] - 5s 666ms/step - loss: 0.0281 - accuracy: 1.0000
```

```
model.evaluate(X_test_resized, y_test)
```

```
8/8 [==============================] - 3s 215ms/step - loss: 8.7594 - accuracy: 0.0000e+00
[8.759380340576172, 0.0]
```

2.4) EmoDB

```
!unzip "/content/drive/MyDrive/EmoDB.zip"
```

```python
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels= n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```python
# Preprocessing the dataset
import os
from scipy.io import wavfile
import librosa
```

```
import matplotlib.pyplot as plt
import numpy as np
import cv2

directory = "/content/wav/"

classes = ["W" ,"L" ,"E" ,"A" , "F" ,"T" ,"N" ]

X = list()
y = list()

for filename in os.listdir(directory):
  filePath = os.path.join(directory, filename)
  a = load_audio_file(file_path=filePath)
  data = cv2.merge([a,a,a])
  if(filename[5:6] in classes):
    X.append(data)
    y.append(classes.index(filename[5:6]))
```

```
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
```

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

# dataset preparation

from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, train_size= 0.5 ,r
andom_state=10)
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)

X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

```
model = ResNet50(include_top=True, weights='imagenet')

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_resized, y_train, epochs=10)
```

```
Epoch 3/10
9/9 [==============================] - 6s 663ms/step - loss: 1.1062 - accuracy: 0.6367
Epoch 4/10
9/9 [==============================] - 6s 661ms/step - loss: 0.6534 - accuracy: 0.7678
Epoch 5/10
9/9 [==============================] - 6s 662ms/step - loss: 0.3835 - accuracy: 0.8914
Epoch 6/10
9/9 [==============================] - 6s 662ms/step - loss: 0.3716 - accuracy: 0.8689
Epoch 7/10
9/9 [==============================] - 6s 662ms/step - loss: 0.2297 - accuracy: 0.9213
Epoch 8/10
9/9 [==============================] - 6s 661ms/step - loss: 0.1096 - accuracy: 0.9775
Epoch 9/10
9/9 [==============================] - 6s 664ms/step - loss: 0.1170 - accuracy: 0.9850
Epoch 10/10
9/9 [==============================] - 6s 659ms/step - loss: 0.2414 - accuracy: 0.9251

model.evaluate(X_test_resized, y_test)

9/9 [==============================] - 4s 304ms/step - loss: 7.2902 - accuracy: 0.0000e+00
[7.290168285369873, 0.0]
```

Looking at the complexity of the model and the limitations of google colab, I have reduced the input size for the model,i.e., I have taken 2000 training data points and 2000 testing data points.

2) Recurrent Neural Networks (RNN)

```
from google.colab import drive
drive.mount('/content/drive')
```

3.1) CIFAR-10

```
import os
import tensorflow as tf
import keras
from tensorflow.keras import layers
from tensorflow.keras import Model
from os import getcwd
```

```
cifar10 = tf.keras.datasets.cifar10
(training_images, training_labels), (test_images, test_labels) = cifar10.load_data()
```

```
training_images = training_images.reshape(50000, 1024, 3)
training_images = training_images[0:10000]
training_labels = training_labels[0:10000]
training_images = training_images/255.0
test_images = test_images.reshape(10000, 1024, 3)
test_images = test_images[0:5000]
test_labels = test_labels[0:5000]
test_images = test_images/255.0
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input_shape=(1024,3), return_seq
uences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
    ])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(training_images, training_labels, batch_size = 50, epochs=10)
```

```
Epoch 3/10
200/200 [==============================] - 111s 557ms/step - loss: 2.0085 - accuracy: 0.2645
Epoch 4/10
200/200 [==============================] - 112s 558ms/step - loss: 1.9649 - accuracy: 0.2771
Epoch 5/10
200/200 [==============================] - 111s 557ms/step - loss: 1.9583 - accuracy: 0.2816
Epoch 6/10
200/200 [==============================] - 111s 557ms/step - loss: 1.9388 - accuracy: 0.2896
Epoch 7/10
200/200 [==============================] - 111s 557ms/step - loss: 1.9371 - accuracy: 0.2899
Epoch 8/10
200/200 [==============================] - 111s 556ms/step - loss: 1.9254 - accuracy: 0.2989
Epoch 9/10
200/200 [==============================] - 111s 557ms/step - loss: 1.9188 - accuracy: 0.2966
Epoch 10/10
200/200 [==============================] - 111s 556ms/step - loss: 1.9341 - accuracy: 0.2930
```

```
model.evaluate(test_images, test_labels)
```

```
157/157 [==============================] - 38s 225ms/step - loss: 1.9601 - accuracy: 0.2912
[1.9600898027420044, 0.29120001196861267]
```

3.2) MNIST

```
import torch
```

```
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

```
from torchvision import datasets
from torchvision.transforms import ToTensor
train_data = datasets.MNIST(
    root = 'data',
    train = True,
    transform = ToTensor(),
    download = True,
)
test_data = datasets.MNIST(
    root = 'data',
    train = False,
    transform = ToTensor()
)
```

```
import matplotlib.pyplot as plt
plt.imshow(train_data.data[0], cmap='gray')
plt.title('%i' % train_data.targets[0])
plt.show()
```

```
figure = plt.figure(figsize=(10, 8))
cols, rows = 5, 5
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(train_data), size=(1,)).item()
    img, label = train_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(label)
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```

```
from torch.utils.data import DataLoader
loaders = {
    'train' : torch.utils.data.DataLoader(train_data,
                                          batch_size=100,
                                          shuffle=True,
                                          num_workers=1),

    'test'  : torch.utils.data.DataLoader(test_data,
                                          batch_size=100,
                                          shuffle=True,
                                          num_workers=1),
}
loaders
```

```
from torch import nn
import torch.nn.functional as F
```

```
sequence_length = 28
input_size = 28
hidden_size = 128
num_layers = 2
num_classes = 10
batch_size = 100
num_epochs = 2
```

```
learning_rate = 0.01
```

```
class RNN(nn.Module):
    pass
model = RNN().to(device)
print(model)
```

```
class RNN(nn.Module):

    def __init__(self, input_size, hidden_size, num_layers, num_classes):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)
        pass

    def forward(self, x):
        # Set initial hidden and cell states
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
        # Passing in the input and hidden state into the model and  obtaining outputs
        out, hidden = self.lstm(x, (h0, c0))  # out: tensor of shape (batch_size, seq_lengt
h, hidden_size)

        #Reshaping the outputs such that it can be fit into the fully connected layer
        out = self.fc(out[:, -1, :])
        return out

        pass
pass
model = RNN(input_size, hidden_size, num_layers, num_classes).to(device)
print(model)
```

```
loss_func = nn.CrossEntropyLoss()
loss_func
```

```
from torch import optim
optimizer = optim.Adam(model.parameters(), lr = 0.01)
optimizer
```

```python
def train(num_epochs, model, loaders):

    # Train the model
    total_step = len(loaders['train'])

    for epoch in range(num_epochs):
        for i, (images, labels) in enumerate(loaders['train']):

            images = images.reshape(-1, sequence_length, input_size).to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = loss_func(outputs, labels)
            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if (i+1) % 100 == 0:
                print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                       .format(epoch + 1, num_epochs, i + 1, total_step, loss.item()))
                pass

        pass
    pass
train(num_epochs, model, loaders)
```

```python
# Test the model
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in loaders['test']:
        images = images.reshape(-1, sequence_length, input_size).to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total = total + labels.size(0)
        correct = correct + (predicted == labels).sum().item()
print('Test Accuracy of the model on the 10000 test images: {} %'.format(100 * correct / to
tal))
```

```
print('Test Accuracy of the model on the 10000 test images: {} %'.format(100 * correct / total))

Test Accuracy of the model on the 10000 test images: 97.77 %
```

3.3) SAVEE

```
!unzip "/content/drive/MyDrive/SaveeDataset.zip"
```

```
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels= n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
```

```
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```
# Preprocessing the dataset
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2

rootDirectory = "/content/AudioData/"
personNames = ["DC","JE","JK","KL"]

classes = ["a" , "d" , "f", "h", "n", "sa" , "su" ]

X = list()
y = list()

for person in personNames:
  directory = os.path.join(rootDirectory,person)
  for filename in os.listdir(directory):
    filePath = os.path.join(directory, filename)
    data = load_audio_file(file_path=filePath)
    # data = cv2.merge([a,a,a])
    if(filename[0:1] in classes):
      X.append(data)
      y.append(classes.index(filename[0:1]))
    elif(filename[0:2] in classes):
      X.append(data)
      y.append(classes.index(filename[0:2]))
```

```
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
```

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

# dataset preparation
```

```
from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, train_size= 0.6 ,r
andom_state=10)
```

```
import os
import tensorflow as tf
import keras
from tensorflow.keras import layers
from tensorflow.keras import Model
from os import getcwd
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input_shape=(157,320), return_se
quences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
    ])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train,y_train, batch_size = 50, epochs=50)
```

```
[ ] model.evaluate(X_test, y_test)

    6/6 [==============================] - 2s 57ms/step - loss: 1.4087 - accuracy: 0.4323
    [1.408677577972412, 0.4322916567325592]
```

3.4) EmoDB

```
!unzip "/content/drive/MyDrive/EmoDB.zip"
```

```
import librosa
import numpy as np
```

```python
input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels= n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```python
# Preprocessing the dataset
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

```
directory = "/content/wav/"

classes = ["W" ,"L" ,"E" ,"A" , "F" ,"T" ,"N" ]

X = list()
y = list()

for filename in os.listdir(directory):
  filePath = os.path.join(directory, filename)
  data = load_audio_file(file_path=filePath)
  if(filename[5:6] in classes):
    X.append(data)
    y.append(classes.index(filename[5:6]))
```

```
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
```

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

# dataset preparation

from tensorflow.keras import datasets,layers,models
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, train_size= 0.7 ,r
andom_state=10)
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input_shape=(157,320), return_se
quences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
    ])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train,y_train, batch_size = 50, epochs=50)
```

```
8/8 [==============================] - 1s 130ms/step - loss: 0.2239 - accuracy: 0.9144
Epoch 50/50
8/8 [==============================] - 1s 127ms/step - loss: 0.2858 - accuracy: 0.8984

[ ] model.evaluate(X_test, y_test)

6/6 [==============================] - 2s 54ms/step - loss: 1.7593 - accuracy: 0.5590
[1.7592660188674927, 0.5590062141418457]
```

Looking at the complexity of the model and the limitations of google colab, I have reduced the input size for the model,i.e., I have taken 2000 training data points and 2000 testing data points.

3) AlexNet

```
from google.colab import drive
drive.mount('/content/drive')
```

```
# Import necessary packages
import argparse

# Import necessary components to build LeNet
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers import BatchNormalization
from keras.regularizers import l2

import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import skimage.transform
```

```python
def alexnet_model(img_shape=(224, 224, 3), n_classes=10, l2_reg=0.,
    weights=None):

    # Initialize model
    alexnet = Sequential()

    # Layer 1
    alexnet.add(Conv2D(30, (11, 11), input_shape=img_shape,
        padding='same', kernel_regularizer=l2(l2_reg)))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 2
    alexnet.add(Conv2D(30, (5, 5), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 3
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(30, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 4
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(30, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))

    # Layer 5
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(30, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 6
    alexnet.add(Flatten())
    alexnet.add(Dense(30))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))
```

```python
    # Layer 7
    alexnet.add(Dense(30))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 8
    alexnet.add(Dense(n_classes))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('softmax'))

    if weights is not None:
        alexnet.load_weights(weights)

    return alexnet

def parse_args():
    """
    Parse command line arguments.
    Parameters:
        None
    Returns:
        parser arguments
    """
    parser = argparse.ArgumentParser(description='AlexNet model')
    optional = parser._action_groups.pop()
    required = parser.add_argument_group('required arguments')
    optional.add_argument('--print_model',
        dest='print_model',
        help='Print AlexNet model',
        action='store_true')
    parser._action_groups.append(optional)
    return parser.parse_args()
```

```python
def load_preprocess_training_batch(X_train):

    new = []

    for item in X_train:
        tmpFeature = skimage.transform.resize(item, (224, 224), mode='constant')
        new.append(tmpFeature)

    return new
```

4.1) CIFAR-10

```
model = alexnet_model()
(X_train, y_train) , (X_test, y_test) = keras.datasets.cifar10.load_data()

X_train = X_train[0:500]
y_train = y_train[0:500]
X_test = X_test[0:200]
y_test = y_test[0:200]
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)
```

```

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)
```

```
X_train_resized = X_train_resized / 255
X_test_resized = X_test_resized / 255
```

```
model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_resized, y_train, epochs=5)
```

```
Epoch 1/5
16/16 [==============================] - 61s 4s/step - loss: 2.6788 - accuracy: 0.1240
Epoch 2/5
16/16 [==============================] - 59s 4s/step - loss: 2.6332 - accuracy: 0.0960
Epoch 3/5
16/16 [==============================] - 60s 4s/step - loss: 2.5360 - accuracy: 0.1220
Epoch 4/5
16/16 [==============================] - 59s 4s/step - loss: 2.4222 - accuracy: 0.1400
Epoch 5/5
16/16 [==============================] - 60s 4s/step - loss: 2.3684 - accuracy: 0.1400
```

+ Code + Text

```
[ ] model.evaluate(X_test_resized, y_test)

7/7 [==============================] - 6s 753ms/step - loss: 2.3611 - accuracy: 0.0750
[2.3610661029815674, 0.07500000298023224]
```

4.2) MNIST

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()

X_train = X_train[0:2000]
y_train = y_train[0:2000]
X_test = X_test[0:2000]
y_test = y_test[0:2000]
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)

X_train_resized = X_train_resized / 255.0
X_test_resized = X_test_resized / 255.0
```

```
import cv2

X_train_new = list()

for i in range(len(X_train_resized)):
  g  = X_train_resized[i]
  X_train_new.append(cv2.merge([g,g,g]))

X_train_new = np.asarray(X_train_new,dtype=np.float32)

X_test_new = list()

for i in range(len(X_test_resized)):
  g  = X_test_resized[i]
  X_test_new.append(cv2.merge([g,g,g]))

X_test_new = np.asarray(X_test_new,dtype=np.float32)
```

```
model = alexnet_model()

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_new, y_train, epochs=5)
```

```
Epoch 1/5
63/63 [==============================] - 474s 8s/step - loss: 2.0734 - accuracy: 0.2610
Epoch 2/5
63/63 [==============================] - 476s 8s/step - loss: 1.7821 - accuracy: 0.3680
Epoch 3/5
63/63 [==============================] - 468s 7s/step - loss: 1.6773 - accuracy: 0.4395
Epoch 4/5
63/63 [==============================] - 469s 7s/step - loss: 1.5820 - accuracy: 0.4810
Epoch 5/5
63/63 [==============================] - 472s 7s/step - loss: 1.5318 - accuracy: 0.5040

[ ] model.evaluate(X_test_new, y_test)

63/63 [==============================] - 107s 2s/step - loss: 2.3417 - accuracy: 0.1170
[2.3417277336120605, 0.11699999868869781]
```

4.3) SAVEE

```
!unzip "/content/drive/MyDrive/SaveeDataset.zip"
```

```
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels= n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
```

```
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```
# Preprocessing the dataset
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2

rootDirectory = "/content/AudioData/"
personNames = ["DC","JE","JK","KL"]

classes = ["a" , "d" , "f", "h", "n", "sa" , "su" ]

X = list()
y = list()

for person in personNames:
  directory = os.path.join(rootDirectory,person)
  for filename in os.listdir(directory):
    filePath = os.path.join(directory, filename)
    a = load_audio_file(file_path=filePath)
    data = cv2.merge([a,a,a])
    if(filename[0:1] in classes):
      X.append(data)
      y.append(classes.index(filename[0:1]))
    elif(filename[0:2] in classes):
      X.append(data)
```

```
      y.append(classes.index(filename[0:2]))
```

```
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
```

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

# dataset preparation

from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, train_size= 0.5 ,r
andom_state=10)
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)
```

```
model = alexnet_model()

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_resized, y_train, epochs=10)
```

```
Epoch 4/10
8/8 [==============================] - 56s 7s/step - loss: 2.4215 - accuracy: 0.1583
Epoch 5/10
8/8 [==============================] - 56s 7s/step - loss: 2.2042 - accuracy: 0.2333
Epoch 6/10
8/8 [==============================] - 57s 7s/step - loss: 2.2080 - accuracy: 0.2042
Epoch 7/10
8/8 [==============================] - 56s 7s/step - loss: 2.1114 - accuracy: 0.2792
Epoch 8/10
8/8 [==============================] - 57s 7s/step - loss: 2.1120 - accuracy: 0.2542
Epoch 9/10
8/8 [==============================] - 56s 7s/step - loss: 2.0292 - accuracy: 0.2583
Epoch 10/10
8/8 [==============================] - 57s 7s/step - loss: 2.1150 - accuracy: 0.2417
```

```
model.evaluate(X_test_resized, y_test)
```

```
8/8 [==============================] - 13s 2s/step - loss: 2.2758 - accuracy: 0.2375
[2.275780200958252, 0.23749999701976776]
```

4.4) EmoDB

```
!unzip "/content/drive/MyDrive/EmoDB.zip"
```

```
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels= n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
```

```
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```
# Preprocessing the dataset
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2

directory = "/content/wav/"

classes = ["W" ,"L" ,"E" ,"A" , "F" ,"T" ,"N" ]

X = list()
y = list()

for filename in os.listdir(directory):
  filePath = os.path.join(directory, filename)
  a = load_audio_file(file_path=filePath)
  data = cv2.merge([a,a,a])
  if(filename[5:6] in classes):
    X.append(data)
    y.append(classes.index(filename[5:6]))
```

```
X = np.asarray(X, dtype=np.float32)
```

```
y = np.asarray(y, dtype=np.float32)
```

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

# dataset preparation

from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, train_size= 0.6 ,r
andom_state=10)
```

```
X_train_resized = load_preprocess_training_batch(X_train)
X_test_resized = load_preprocess_training_batch(X_test)

X_train_resized = np.array(X_train_resized)
X_test_resized = np.array(X_test_resized)
```

```
model = alexnet_model()

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_resized, y_train, epochs=10)
```

```
11/11 [==============================] - 75s 7s/step - loss: 2.2610 - accuracy: 0.1838
Epoch 6/10
11/11 [==============================] - 75s 7s/step - loss: 2.1741 - accuracy: 0.2025
Epoch 7/10
11/11 [==============================] - 77s 7s/step - loss: 2.0738 - accuracy: 0.2336
Epoch 8/10
11/11 [==============================] - 76s 7s/step - loss: 2.0492 - accuracy: 0.2679
Epoch 9/10
11/11 [==============================] - 76s 7s/step - loss: 2.0359 - accuracy: 0.2679
Epoch 10/10
11/11 [==============================] - 76s 7s/step - loss: 1.9726 - accuracy: 0.3115
```

```
model.evaluate(X_test_resized, y_test)
```

```
7/7 [==============================] - 12s 2s/step - loss: 2.1748 - accuracy: 0.2336
[2.1748006343841553, 0.23364485800266266]
```

Looking at the complexity of the model and the limitations of google colab, I have reduced the input size for the model,i.e., I have taken 2000 training data points and 2000 testing data points.

4) GoogLeNet

```
from google.colab import drive
drive.mount('/content/drive')
```

5.1) CIFAR-10

```
def inception_module(x,
                     filters_1x1,
                     filters_3x3_reduce,
                     filters_3x3,
                     filters_5x5_reduce,
                     filters_5x5,
                     filters_pool_proj,
```

```
                     name=None):

    conv_1x1 = Conv2D(filters_1x1, (1, 1), padding='same', activation='relu', kernel_initia
lizer=kernel_init, bias_initializer=bias_init)(x)

    conv_3x3 = Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu', kernel
_initializer=kernel_init, bias_initializer=bias_init)(x)
    conv_3x3 = Conv2D(filters_3x3, (3, 3), padding='same', activation='relu', kernel_initia
lizer=kernel_init, bias_initializer=bias_init)(conv_3x3)

    conv_5x5 = Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu', kernel
_initializer=kernel_init, bias_initializer=bias_init)(x)
    conv_5x5 = Conv2D(filters_5x5, (5, 5), padding='same', activation='relu', kernel_initia
lizer=kernel_init, bias_initializer=bias_init)(conv_5x5)

    pool_proj = MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
    pool_proj = Conv2D(filters_pool_proj, (1, 1), padding='same', activation='relu', kernel
_initializer=kernel_init, bias_initializer=bias_init)(pool_proj)

    output = concatenate([conv_1x1, conv_3x3, conv_5x5, pool_proj], axis=3, name=name)

    return output
```

```
kernel_init = keras.initializers.glorot_uniform()
bias_init = keras.initializers.Constant(value=0.2)
```

```
input_layer = Input(shape=(224, 224, 3))

x = Conv2D(64, (7, 7), padding='same', strides=(2, 2), activation='relu', name='conv_1_7x7/
2', kernel_initializer=kernel_init, bias_initializer=bias_init)(input_layer)
x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max_pool_1_3x3/2')(x)
x = Conv2D(64, (1, 1), padding='same', strides=(1, 1), activation='relu', name='conv_2a_3x3
/1')(x)
x = Conv2D(192, (3, 3), padding='same', strides=(1, 1), activation='relu', name='conv_2b_3x
3/1')(x)
x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max_pool_2_3x3/2')(x)

x = inception_module(x,
                     filters_1x1=64,
                     filters_3x3_reduce=96,
                     filters_3x3=128,
                     filters_5x5_reduce=16,
                     filters_5x5=32,
                     filters_pool_proj=32,
```

```python
                     name='inception_3a')

x = inception_module(x,
                     filters_1x1=128,
                     filters_3x3_reduce=128,
                     filters_3x3=192,
                     filters_5x5_reduce=32,
                     filters_5x5=96,
                     filters_pool_proj=64,
                     name='inception_3b')

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max_pool_3_3x3/2')(x)

x = inception_module(x,
                     filters_1x1=192,
                     filters_3x3_reduce=96,
                     filters_3x3=208,
                     filters_5x5_reduce=16,
                     filters_5x5=48,
                     filters_pool_proj=64,
                     name='inception_4a')


x1 = AveragePooling2D((5, 5), strides=3)(x)
x1 = Conv2D(128, (1, 1), padding='same', activation='relu')(x1)
x1 = Flatten()(x1)
x1 = Dense(1024, activation='relu')(x1)
x1 = Dropout(0.7)(x1)
x1 = Dense(10, activation='softmax', name='auxilliary_output_1')(x1)

x = inception_module(x,
                     filters_1x1=160,
                     filters_3x3_reduce=112,
                     filters_3x3=224,
                     filters_5x5_reduce=24,
                     filters_5x5=64,
                     filters_pool_proj=64,
                     name='inception_4b')

x = inception_module(x,
                     filters_1x1=128,
                     filters_3x3_reduce=128,
                     filters_3x3=256,
                     filters_5x5_reduce=24,
                     filters_5x5=64,
```

```
                     filters_pool_proj=64,
                     name='inception_4c')

x = inception_module(x,
                     filters_1x1=112,
                     filters_3x3_reduce=144,
                     filters_3x3=288,
                     filters_5x5_reduce=32,
                     filters_5x5=64,
                     filters_pool_proj=64,
                     name='inception_4d')


x2 = AveragePooling2D((5, 5), strides=3)(x)
x2 = Conv2D(128, (1, 1), padding='same', activation='relu')(x2)
x2 = Flatten()(x2)
x2 = Dense(1024, activation='relu')(x2)
x2 = Dropout(0.7)(x2)
x2 = Dense(10, activation='softmax', name='auxilliary_output_2')(x2)

x = inception_module(x,
                     filters_1x1=256,
                     filters_3x3_reduce=160,
                     filters_3x3=320,
                     filters_5x5_reduce=32,
                     filters_5x5=128,
                     filters_pool_proj=128,
                     name='inception_4e')

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max_pool_4_3x3/2')(x)

x = inception_module(x,
                     filters_1x1=256,
                     filters_3x3_reduce=160,
                     filters_3x3=320,
                     filters_5x5_reduce=32,
                     filters_5x5=128,
                     filters_pool_proj=128,
                     name='inception_5a')

x = inception_module(x,
                     filters_1x1=384,
                     filters_3x3_reduce=192,
                     filters_3x3=384,
                     filters_5x5_reduce=48,
```

```
                     filters_5x5=128,
                     filters_pool_proj=128,
                     name='inception_5b')

x = GlobalAveragePooling2D(name='avg_pool_5_3x3/1')(x)

x = Dropout(0.4)(x)

x = Dense(10, activation='softmax', name='output')(x)
```

```
import keras
from keras.layers.core import Layer
import keras.backend as K
import tensorflow as tf
from keras.datasets import cifar10
```

```
from keras.models import Model

from keras.layers import Conv2D, MaxPool2D,  \
    Dropout, Dense, Input, concatenate,      \
    GlobalAveragePooling2D, AveragePooling2D,\
    Flatten

import cv2
import numpy as np
from keras.datasets import cifar10
from keras import backend as K
from keras.utils import np_utils

import math
from tensorflow.keras.optimizers import SGD
from keras.callbacks import LearningRateScheduler
```

```
num_classes = 10

def load_cifar10_data(img_rows, img_cols):

    # Load cifar10 training and validation sets
    (X_train, Y_train), (X_valid, Y_valid) = cifar10.load_data()

    X_train = X_train[0:5000]
    Y_train = Y_train[0:5000]
    X_valid = X_valid[0:2000]
```

```
    Y_valid = Y_valid[0:2000]

    # Resize training images
    X_train = np.array([cv2.resize(img, (img_rows,img_cols)) for img in X_train[:,:,:,:]])
    X_valid = np.array([cv2.resize(img, (img_rows,img_cols)) for img in X_valid[:,:,:,:]])

    # Transform targets to keras compatible format
    Y_train = np_utils.to_categorical(Y_train, num_classes)
    Y_valid = np_utils.to_categorical(Y_valid, num_classes)

    X_train = X_train.astype('float32')
    X_valid = X_valid.astype('float32')

    # preprocess data
    X_train = X_train / 255.0
    X_valid = X_valid / 255.0

    return X_train, Y_train, X_valid, Y_valid
```

```
X_train, y_train, X_test, y_test = load_cifar10_data(224, 224)
model = Model(input_layer, [x, x1, x2], name='inception_v1')
model.summary()
epochs = 10
initial_lrate = 0.01

def decay(epoch, steps=100):
    initial_lrate = 0.01
    drop = 0.96
    epochs_drop = 8
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
    return lrate

sgd = SGD(learning_rate=initial_lrate, momentum=0.9, nesterov=False)

lr_sc = LearningRateScheduler(decay, verbose=1)

model.compile(loss=['categorical_crossentropy', 'categorical_crossentropy', 'categorical_cr
ossentropy'], loss_weights=[1, 0.3, 0.3], optimizer=sgd, metrics=['accuracy'])
```

```
history = model.fit(X_train, [y_train, y_train, y_train], validation_data=(X_test, [y_test,
 y_test, y_test]), epochs=epochs, batch_size=256, callbacks=[lr_sc])
```

```
output_2_loss: 2.1621 - val_output_accuracy: 0.1675 - val_auxilliary_output_1_accuracy: 0.2250 - val_auxilliary_output_2_accuracy: 0.1965

output_2_loss: 2.0650 - val_output_accuracy: 0.2305 - val_auxilliary_output_1_accuracy: 0.2400 - val_auxilliary_output_2_accuracy: 0.2240

output_2_loss: 2.0244 - val_output_accuracy: 0.2470 - val_auxilliary_output_1_accuracy: 0.2630 - val_auxilliary_output_2_accuracy: 0.2585

output_2_loss: 2.0076 - val_output_accuracy: 0.2355 - val_auxilliary_output_1_accuracy: 0.2735 - val_auxilliary_output_2_accuracy: 0.2660
```

5.2) MNIST

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models, losses, Model
```

```
(x_train, y_train), (x_test, y_test)=tf.keras.datasets.mnist.load_data()
x_train = tf.pad(x_train, [[0, 0], [2,2], [2,2]])/255
x_test = tf.pad(x_test, [[0, 0], [2,2], [2,2]])/255
x_train = tf.expand_dims(x_train, axis=3, name=None)
x_test = tf.expand_dims(x_test, axis=3, name=None)
x_train = tf.repeat(x_train, 3, axis=3)
x_test = tf.repeat(x_test, 3, axis=3)
x_val = x_train[-2000:,:,:,:]
y_val = y_train[-2000:]
x_train = x_train[:-2000,:,:,:]
y_train = y_train[:-2000]
```

```
def inception(x,
              filters_1x1,
              filters_3x3_reduce,
              filters_3x3,
              filters_5x5_reduce,
              filters_5x5,
              filters_pool):
  path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu')(x)
  path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same', activation='relu')(path2)
```

```
  path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu')(x)
  path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same', activation='relu')(path3)

  path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
  path4 = layers.Conv2D(filters_pool, (1, 1), padding='same', activation='relu')(path4)

  return tf.concat([path1, path2, path3, path4], axis=3)
```

```
inp = layers.Input(shape=(32, 32, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="bilinear
", input_shape=x_train.shape[1:])(inp)

x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input_tensor)
x = layers.MaxPooling2D(3, strides=2)(x)

x = layers.Conv2D(64, 1, strides=1, padding='same', activation='relu')(x)
x = layers.Conv2D(192, 3, strides=1, padding='same', activation='relu')(x)

x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=64,
              filters_3x3_reduce=96,
              filters_3x3=128,
              filters_5x5_reduce=16,
              filters_5x5=32,
              filters_pool=32)

x = inception(x,
              filters_1x1=128,
              filters_3x3_reduce=128,
              filters_3x3=192,
              filters_5x5_reduce=32,
              filters_5x5=96,
              filters_pool=64)

x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=192,
              filters_3x3_reduce=96,
              filters_3x3=208,
              filters_5x5_reduce=16,
              filters_5x5=48,
```

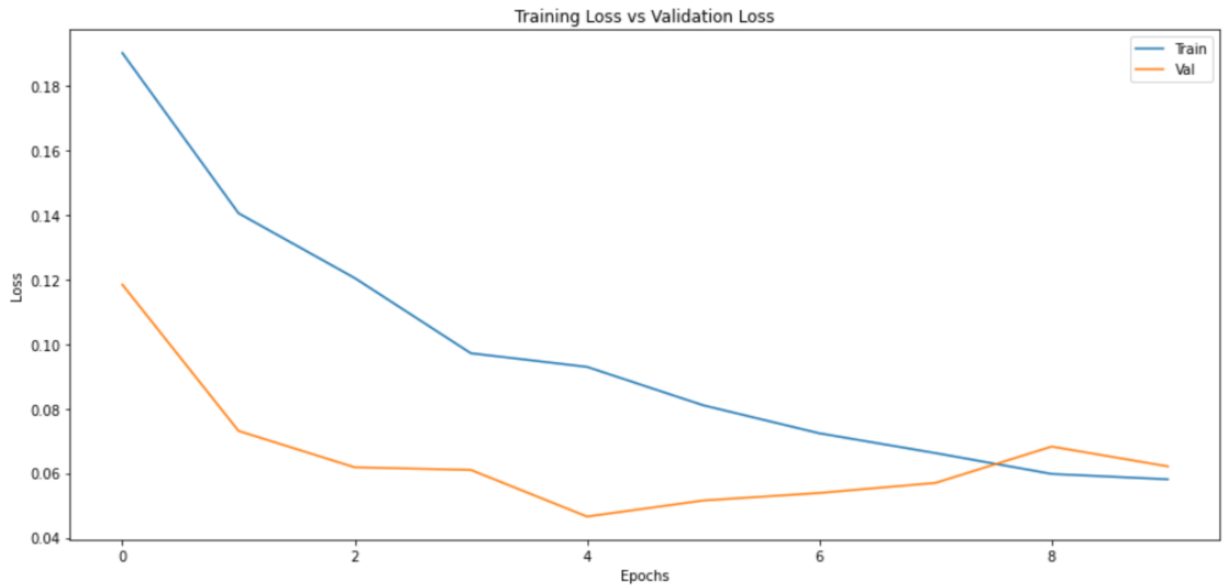
```
              filters_pool=64)

aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux1 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)
aux1 = layers.Flatten()(aux1)
aux1 = layers.Dense(1024, activation='relu')(aux1)
aux1 = layers.Dropout(0.7)(aux1)
aux1 = layers.Dense(10, activation='softmax')(aux1)

x = inception(x,
              filters_1x1=160,
              filters_3x3_reduce=112,
              filters_3x3=224,
              filters_5x5_reduce=24,
              filters_5x5=64,
              filters_pool=64)

x = inception(x,
              filters_1x1=128,
              filters_3x3_reduce=128,
              filters_3x3=256,
              filters_5x5_reduce=24,
              filters_5x5=64,
              filters_pool=64)

x = inception(x,
              filters_1x1=112,
              filters_3x3_reduce=144,
              filters_3x3=288,
              filters_5x5_reduce=32,
              filters_5x5=64,
              filters_pool=64)

aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux2 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)
aux2 = layers.Flatten()(aux2)
aux2 = layers.Dense(1024, activation='relu')(aux2)
aux2 = layers.Dropout(0.7)(aux2)
aux2 = layers.Dense(10, activation='softmax')(aux2)

x = inception(x,
              filters_1x1=256,
              filters_3x3_reduce=160,
              filters_3x3=320,
              filters_5x5_reduce=32,
```

```
              filters_5x5=128,
              filters_pool=128)

x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=256,
              filters_3x3_reduce=160,
              filters_3x3=320,
              filters_5x5_reduce=32,
              filters_5x5=128,
              filters_pool=128)

x = inception(x,
              filters_1x1=384,
              filters_3x3_reduce=192,
              filters_3x3=384,
              filters_5x5_reduce=48,
              filters_5x5=128,
              filters_pool=128)

x = layers.GlobalAveragePooling2D()(x)

x = layers.Dropout(0.4)(x)
out = layers.Dense(10, activation='softmax')(x)
```

```
model = Model(inputs = inp, outputs = [out, aux1, aux2])
model.compile(optimizer='adam', loss=[losses.sparse_categorical_crossentropy, losses.sparse
_categorical_crossentropy, losses.sparse_categorical_crossentropy], loss_weights=[1, 0.3, 0
.3], metrics=['accuracy'])

history = model.fit(x_train, [y_train, y_train, y_train], validation_data=(x_val, [y_val, y
_val, y_val]), batch_size=64, epochs=10)
```

```
fig, axs = plt.subplots(2, 1, figsize=(15,15))

axs[0].plot(history.history['loss'])
axs[0].plot(history.history['val_loss'])
axs[0].title.set_text('Training Loss vs Validation Loss')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
axs[0].legend(['Train','Val'])
```

```
<matplotlib.legend.Legend at 0x7feaad89cf50>
```




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

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

```
313/313 [==============================] - 24s 6
[0.05425573140382767,
 0.03818700090050697,
 0.02569129876792431,
 0.027871087193489075,
 0.9873999953269958,
 0.9926000237464905,
 0.9902999997138977]
```

5.3) SAVEE

```
!unzip "/content/drive/MyDrive/SaveeDataset.zip"
```

```
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels= n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```
# Preprocessing the dataset
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2

rootDirectory = "/content/AudioData/"
personNames = ["DC","JE","JK","KL"]

classes = ["a" , "d" , "f", "h", "n", "sa" , "su" ]

X = list()
y = list()

for person in personNames:
  directory = os.path.join(rootDirectory,person)
  for filename in os.listdir(directory):
    filePath = os.path.join(directory, filename)
    a = load_audio_file(file_path=filePath)
    data = cv2.merge([a,a,a])
    if(filename[0:1] in classes):
      X.append(data)
      y.append(classes.index(filename[0:1]))
    elif(filename[0:2] in classes):
      X.append(data)
      y.append(classes.index(filename[0:2]))
```

```
X = np.asarray(X, dtype=np.float32)
y = np.asarray(y, dtype=np.float32)
```

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

# dataset preparation

from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, train_size= 0.6 ,random_state=10)
```

```python
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models, losses, Model
```

```python
def inception(x,
              filters_1x1,
              filters_3x3_reduce,
              filters_3x3,
              filters_5x5_reduce,
              filters_5x5,
              filters_pool):
  path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu')(x)
  path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same', activation='relu')(path2)

  path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu')(x)
  path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same', activation='relu')(path3)

  path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
  path4 = layers.Conv2D(filters_pool, (1, 1), padding='same', activation='relu')(path4)

  return tf.concat([path1, path2, path3, path4], axis=3)
```

```python
inp = layers.Input(shape=(157, 320, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="bilinear
", input_shape=X_train.shape[1:])(inp)

x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input_tensor)
x = layers.MaxPooling2D(3, strides=2)(x)

x = layers.Conv2D(64, 1, strides=1, padding='same', activation='relu')(x)
x = layers.Conv2D(192, 3, strides=1, padding='same', activation='relu')(x)

x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=64,
              filters_3x3_reduce=96,
              filters_3x3=128,
              filters_5x5_reduce=16,
              filters_5x5=32,
```

```
              filters_pool=32)

x = inception(x,
              filters_1x1=128,
              filters_3x3_reduce=128,
              filters_3x3=192,
              filters_5x5_reduce=32,
              filters_5x5=96,
              filters_pool=64)

x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=192,
              filters_3x3_reduce=96,
              filters_3x3=208,
              filters_5x5_reduce=16,
              filters_5x5=48,
              filters_pool=64)

aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux1 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)
aux1 = layers.Flatten()(aux1)
aux1 = layers.Dense(1024, activation='relu')(aux1)
aux1 = layers.Dropout(0.7)(aux1)
aux1 = layers.Dense(10, activation='softmax')(aux1)

x = inception(x,
              filters_1x1=160,
              filters_3x3_reduce=112,
              filters_3x3=224,
              filters_5x5_reduce=24,
              filters_5x5=64,
              filters_pool=64)

x = inception(x,
              filters_1x1=128,
              filters_3x3_reduce=128,
              filters_3x3=256,
              filters_5x5_reduce=24,
              filters_5x5=64,
              filters_pool=64)

x = inception(x,
              filters_1x1=112,
```

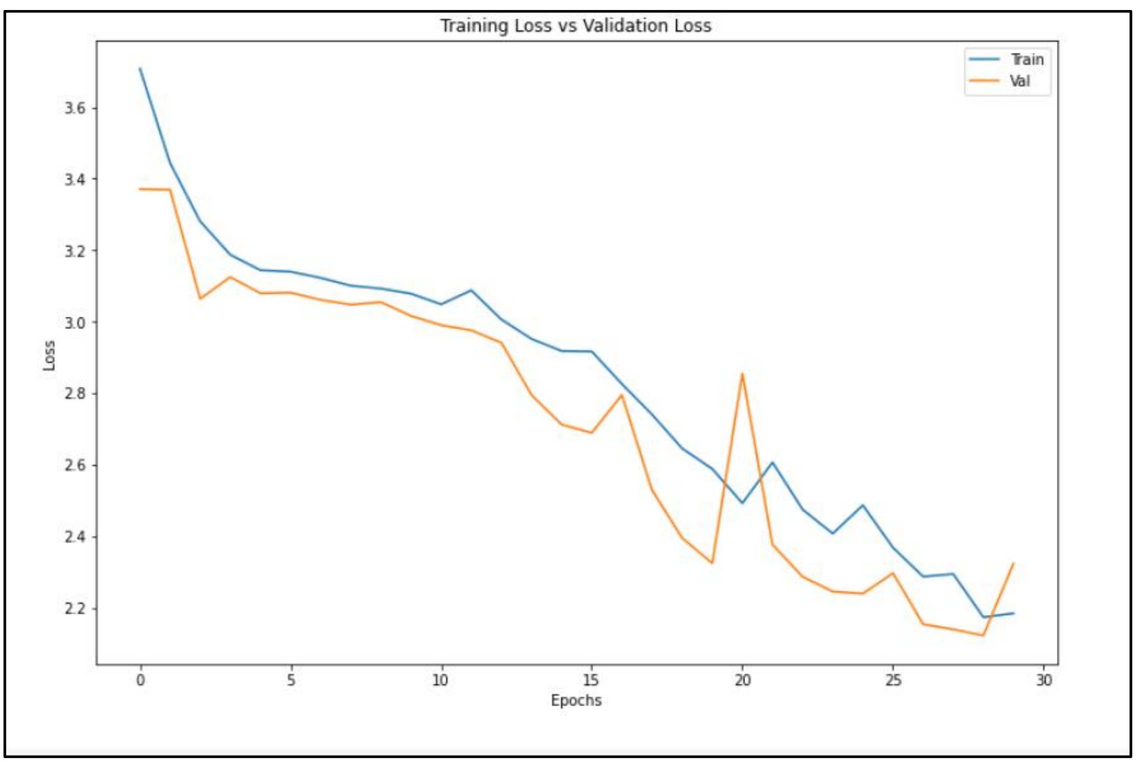
```
              filters_3x3_reduce=144,
              filters_3x3=288,
              filters_5x5_reduce=32,
              filters_5x5=64,
              filters_pool=64)

aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux2 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)
aux2 = layers.Flatten()(aux2)
aux2 = layers.Dense(1024, activation='relu')(aux2)
aux2 = layers.Dropout(0.7)(aux2)
aux2 = layers.Dense(10, activation='softmax')(aux2)

x = inception(x,
              filters_1x1=256,
              filters_3x3_reduce=160,
              filters_3x3=320,
              filters_5x5_reduce=32,
              filters_5x5=128,
              filters_pool=128)

x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=256,
              filters_3x3_reduce=160,
              filters_3x3=320,
              filters_5x5_reduce=32,
              filters_5x5=128,
              filters_pool=128)

x = inception(x,
              filters_1x1=384,
              filters_3x3_reduce=192,
              filters_3x3=384,
              filters_5x5_reduce=48,
              filters_5x5=128,
              filters_pool=128)

x = layers.GlobalAveragePooling2D()(x)

x = layers.Dropout(0.4)(x)
out = layers.Dense(10, activation='softmax')(x)
```

```
model = Model(inputs = inp, outputs = [out, aux1, aux2])
```

```
model.compile(optimizer='adam', loss=[losses.sparse_categorical_crossentropy, losses.sparse
_categorical_crossentropy, losses.sparse_categorical_crossentropy], loss_weights=[1, 0.3, 0
.3], metrics=['accuracy'])
```

```
history = model.fit(X_train, [y_train, y_train, y_train], validation_data=(X_test, [y_test,
 y_test, y_test]), batch_size=64, epochs=30)
```

```
fig, axs = plt.subplots(figsize=(12,8))

axs.plot(history.history['loss'])
axs.plot(history.history['val_loss'])
axs.title.set_text('Training Loss vs Validation Loss')
axs.set_xlabel('Epochs')
axs.set_ylabel('Loss')
axs.legend(['Train','Val'])

plt.show()
```



```
model.evaluate(X_test, y_test)
```

0 5 10

```
model.evaluate(X_test, y_test)
```

```
6/6 [==============================]
[2.324364423751831,
 1.458065390586853,
 1.4511628150939941,
 1.4365006685256958,
 0.3541666567325592,
 0.3333333432674408,
 0.3802083432674408]
```

5.4) EmoDB

```
!unzip "/content/drive/MyDrive/EmoDB.zip"
```

```
import librosa
import numpy as np

input_length = 16000*5

batch_size = 32

n_mels = 320

def preprocess_audio_mel_T(audio, sample_rate=16000, window_size=20, #log_specgram
                 step_size=10, eps=1e-10):

    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels= n_mels)
    mel_db = (librosa.power_to_db(mel_spec, ref=np.max) + 40)/40

    return mel_db.T


def load_audio_file(file_path, input_length=input_length):
```

```
  data = librosa.core.load(file_path, sr=16000)[0] #, sr=16000
  if len(data)>input_length:
    max_offset = len(data)-input_length

    offset = np.random.randint(max_offset)

    data = data[offset:(input_length+offset)]

  else:
    if input_length > len(data):
      max_offset = input_length - len(data)

      offset = np.random.randint(max_offset)
    else:
      offset = 0
    data = np.pad(data, (offset, input_length - len(data) - offset), "constant")

  data = preprocess_audio_mel_T(data)
  return data
```

```
# Preprocessing the dataset
import os
from scipy.io import wavfile
import librosa
import matplotlib.pyplot as plt
import numpy as np
import cv2

directory = "/content/wav/"

classes = ["W" ,"L" ,"E" ,"A" , "F" ,"T" ,"N" ]

X = list()
y = list()

for filename in os.listdir(directory):
  filePath = os.path.join(directory, filename)
  a = load_audio_file(file_path=filePath)
  data = cv2.merge([a,a,a])
  if(filename[5:6] in classes):
    X.append(data)
    y.append(classes.index(filename[5:6]))
```

```
X = np.asarray(X, dtype=np.float32)
```

```
y = np.asarray(y, dtype=np.float32)
```

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

# dataset preparation

from tensorflow.keras import datasets,layers,models
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, train_size= 0.6 ,r
andom_state=10)
```

```
def inception(x,
              filters_1x1,
              filters_3x3_reduce,
              filters_3x3,
              filters_5x5_reduce,
              filters_5x5,
              filters_pool):
  path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu')(x)
  path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same', activation='relu')(path2)

  path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu')(x)
  path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same', activation='relu')(path3)

  path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
  path4 = layers.Conv2D(filters_pool, (1, 1), padding='same', activation='relu')(path4)

  return tf.concat([path1, path2, path3, path4], axis=3)
```

```
inp = layers.Input(shape=(157, 320, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="bilinear
", input_shape=X_train.shape[1:])(inp)

x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input_tensor)
x = layers.MaxPooling2D(3, strides=2)(x)

x = layers.Conv2D(64, 1, strides=1, padding='same', activation='relu')(x)
x = layers.Conv2D(192, 3, strides=1, padding='same', activation='relu')(x)
```

```python
x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=64,
              filters_3x3_reduce=96,
              filters_3x3=128,
              filters_5x5_reduce=16,
              filters_5x5=32,
              filters_pool=32)

x = inception(x,
              filters_1x1=128,
              filters_3x3_reduce=128,
              filters_3x3=192,
              filters_5x5_reduce=32,
              filters_5x5=96,
              filters_pool=64)

x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=192,
              filters_3x3_reduce=96,
              filters_3x3=208,
              filters_5x5_reduce=16,
              filters_5x5=48,
              filters_pool=64)

aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux1 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)
aux1 = layers.Flatten()(aux1)
aux1 = layers.Dense(1024, activation='relu')(aux1)
aux1 = layers.Dropout(0.7)(aux1)
aux1 = layers.Dense(10, activation='softmax')(aux1)

x = inception(x,
              filters_1x1=160,
              filters_3x3_reduce=112,
              filters_3x3=224,
              filters_5x5_reduce=24,
              filters_5x5=64,
              filters_pool=64)

x = inception(x,
```

```python
              filters_1x1=128,
              filters_3x3_reduce=128,
              filters_3x3=256,
              filters_5x5_reduce=24,
              filters_5x5=64,
              filters_pool=64)

x = inception(x,
              filters_1x1=112,
              filters_3x3_reduce=144,
              filters_3x3=288,
              filters_5x5_reduce=32,
              filters_5x5=64,
              filters_pool=64)

aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux2 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)
aux2 = layers.Flatten()(aux2)
aux2 = layers.Dense(1024, activation='relu')(aux2)
aux2 = layers.Dropout(0.7)(aux2)
aux2 = layers.Dense(10, activation='softmax')(aux2)

x = inception(x,
              filters_1x1=256,
              filters_3x3_reduce=160,
              filters_3x3=320,
              filters_5x5_reduce=32,
              filters_5x5=128,
              filters_pool=128)

x = layers.MaxPooling2D(3, strides=2)(x)

x = inception(x,
              filters_1x1=256,
              filters_3x3_reduce=160,
              filters_3x3=320,
              filters_5x5_reduce=32,
              filters_5x5=128,
              filters_pool=128)

x = inception(x,
              filters_1x1=384,
              filters_3x3_reduce=192,
              filters_3x3=384,
              filters_5x5_reduce=48,
```

```python
              filters_5x5=128,
              filters_pool=128)

x = layers.GlobalAveragePooling2D()(x)

x = layers.Dropout(0.4)(x)
out = layers.Dense(10, activation='softmax')(x)
```

```python
model = Model(inputs = inp, outputs = [out, aux1, aux2])

model.compile(optimizer='adam', loss=[losses.sparse_categorical_crossentropy, losses.sparse
_categorical_crossentropy, losses.sparse_categorical_crossentropy], loss_weights=[1, 0.3, 0
.3], metrics=['accuracy'])
```

```python
history = model.fit(X_train, [y_train, y_train, y_train], validation_data=(X_test, [y_test,
 y_test, y_test]), batch_size=64, epochs=30)
```

```python
fig, axs = plt.subplots(figsize=(12,8))

axs.plot(history.history['loss'])
axs.plot(history.history['val_loss'])
axs.title.set_text('Training Loss vs Validation Loss')
axs.set_xlabel('Epochs')
axs.set_ylabel('Loss')
axs.legend(['Train','Val'])

plt.show()
```


```
model.evaluate(X_test, y_test)
```

```
model.evaluate(X_test, y_test)

7/7 [==============================]
[2.8008506298065186,
 1.7843737602233887,
 1.6508854627609253,
 1.7373706102371216,
 0.30841121077537537,
 0.38785046339035034,
 0.3644859790802002]
```

Looking at the complexity of the model and the limitations of google colab, I have reduced the input size for the model,i.e., I have taken 2000 training data points and 2000 testing data points.