BE IT - 001811001012

SHUVRASISH ROY

ML Assignment 3

Comprehensive Report

GITHUB REPO LINK

https://github.com/shuvrasish/ML-Lab/tree/main/assgn3

DATASETS USED

- Wine Dataset: https://archive.ics.uci.edu/ml/datasets/wine

- Ionosphere Dataset: https://archive.ics.uci.edu/ml/datasets/Ionosphere

- Wisconsin Breast Cancer Dataset: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

- CIFAR-10: https://www.cs.toronto.edu/~kriz/cifar.html

- MNIST: http://yann.lecun.com/exdb/mnist/

- SAVEE: http://kahlan.eps.surrey.ac.uk/savee/Download.html

- EmoDB: http://www.emodb.bilderbar.info/navi.html

QUESTION 1

- Implement Hidden Markov Model (HMM) for classification using Python for the following UCI datasets:
    1. Wine Dataset
    2. Ionosphere Dataset
    3. Wisconsin Breast Cancer Dataset

- Compare the performance the following HMM classifiers for all the three datasets and show the classification results (**Accuracy, Precision, Recall, F-score, confusion matrix**) with and without parameter tuning:
    - GaussianHMM
    - GMMHMM
    - MultinomialHMM

- Also, compare the performance results with that of a trained ANN.

Apply different values of train-test set splits and report the corresponding results for all the classifiers.

Generate the **image (heat map)** of the confusion matrix for the best case of every classifier. Also, generate the images of **training & loss generation curves**. For each dataset, generate an image illustrating **Receiver Operating Characteristic (ROC) curve** and **Area Under Curve (AUC)** for the best case of every classifier only.
Try to achieve accuracy **>=80%.**

Show the performance comparison among classifiers in a table.

WORKING WITH IONOSPHERE DATASET

Without and With Parameter Tuning TABULATION

(CODE ALONGWITH OUTPUTS ATTACHED AT THE END OF TABULATION)

CLASSIFIER	PARAMETER TUNING	TRAIN-TEST RATIO	PRECISION	RECALL	F1 SCORE	SUPPORT	ACCURACY
GAUSSIAN CLASSIFIER	No	70:30	0.38	1.00	0.55	40	0.37
	Yes		0.73	0.88	0.80	40	**0.83**
	No	60:40	0.39	1.00	0.56	55	0.39
	Yes		0.12	0.18	0.15	55	0.18
	No	50:50	0.37	1.00	0.54	65	0.36
	Yes		0.17	0.28	0.21	65	0.24
	No	40:60	0.36	1.00	0.53	76	0.36
	Yes		0.21	0.34	0.26	76	0.30
	No	30:70	0.34	1.00	0.51	84	0.34
	Yes		0.33	0.25	0.29	84	0.57

CLASSIFIER	PARAMETER TUNING	TRAIN-TEST RATIO	PRECISION	RECALL	F1 SCORE	SUPPORT	ACCURACY
GMM CLASSIFIER	No	70:30	0.38	1.00	0.55	40	0.37
	Yes		0.73	0.88	0.80	40	**0.83**
	No	60:40	0.39	1.00	0.56	55	0.39
	Yes		0.75	0.87	0.81	55	**0.83**
	No	50:50	0.37	1.00	0.54	65	0.36
	Yes		0.65	0.72	0.69	65	0.75
	No	40:60	0.36	1.00	0.53	76	0.36
	Yes		0.60	0.62	0.61	76	0.71
	No	30:70	0.34	1.00	0.51	84	0.34
	Yes		0.35	0.77	0.48	84	0.43

CLASSIFIER	PARAM E TER TUNING	TRAIN - TEST RATIO	PRECIS I ON	RECALL	F1 SCORE	SUPPOR T	ACCURA CY
MULTINOMIAL CLASSIFIER	No	70:30	0.38	1.00	0.55	40	0.37
	Yes		0.40	0.85	0.54	40	0.45
	No	60:40	0.39	1.00	0.56	55	0.39
	Yes		0.39	0.89	0.54	55	0.40
	No	50:50	1.00	0.00	0.00	65	0.63
	Yes		0.41	0.17	0.24	65	0.60
	No	40:60	1.00	0.00	0.00	76	0.63
	Yes		0.54	0.33	0.41	76	**0.65**
	No	30:70	1.00	0.00	0.00	84	**0.65**
	Yes		0.35	0.20	0.26	84	0.60

IONOSPHERE DATASET

In [93]:

```
#DATASET PREPARATION AND IMPORTS

import pandas as pd
import numpy as np

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','
,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

WITHOUT PARAMETER TUNING GAUSSIAN

HMM 70-30 SPLIT WITHOUT PARAMETER

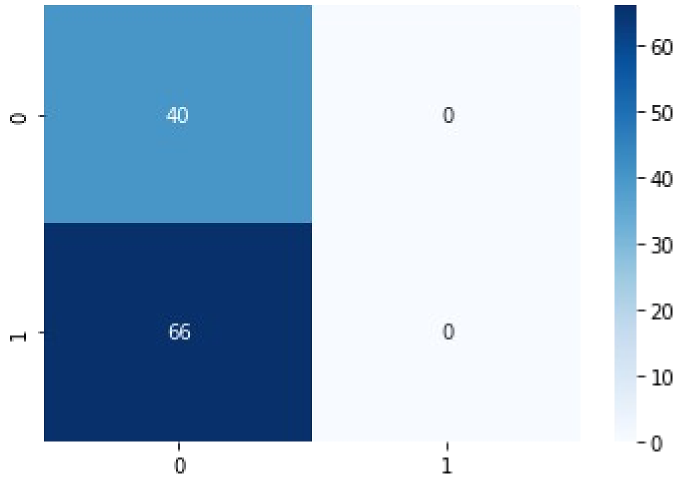
TUNING

In [94]:

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM() 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, zero_division=1))
```

```
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: [[40
         0]
 [66  0]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.38      1.00      0.55        40
           g       1.00      0.00      0.00        66

    accuracy                           0.38       106
   macro avg       0.69      0.50      0.27       106
weighted  avg      0.77      0.38      0.21       106

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.37735849056603776
```



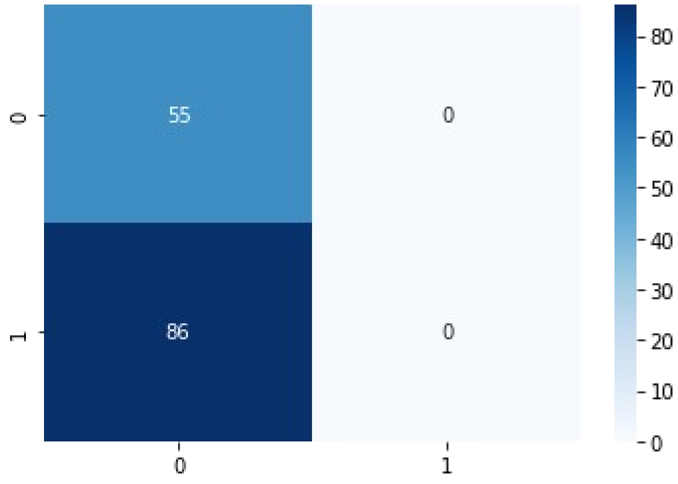
60-40 SPLIT WITHOUT PARAMETER TUNING

In [95]:

```
from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.6,test_size=0.4

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier  =  hmm.GaussianHMM() 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, zero_division=1))

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: [[55
       0]
 [86  0]]
-------------------------------------------------------
-------------------------------------------------------
Performance Evaluation
               precision    recall  f1-score   support

           b       0.39      1.00      0.56        55
           g       1.00      0.00      0.00        86

    accuracy                           0.39       141
   macro avg       0.70      0.50      0.28       141
weighted avg       0.76      0.39      0.22       141

-------------------------------------------------------
-------------------------------------------------------
Accuracy:
0.3900709219858156
```



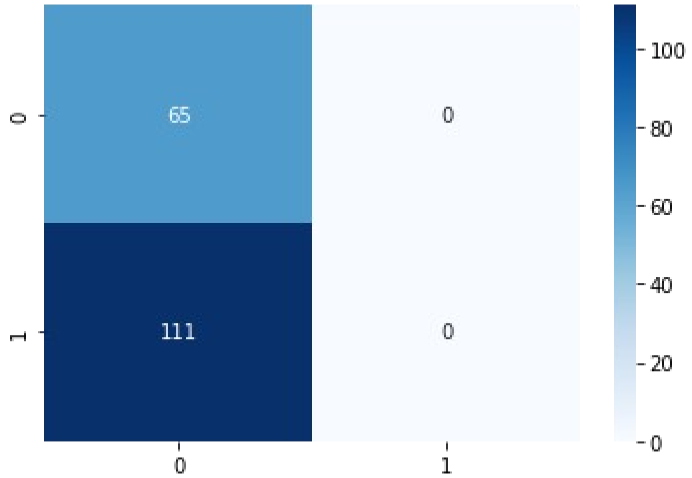
50-50 SPLIT WITHOUT PARAMETER TUNING

In [96]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM()
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, zero_division=1))

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: [[ 65
          0]
 [111   0]]
---------------------------------------------
---------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.37      1.00      0.54        65
           g       1.00      0.00      0.00       111

    accuracy                           0.37       176
   macro avg       0.68      0.50      0.27       176
weighted  avg       0.77      0.37      0.20       176

---------------------------------------------
---------------------------------------------
Accuracy:
0.3693181818181818
```



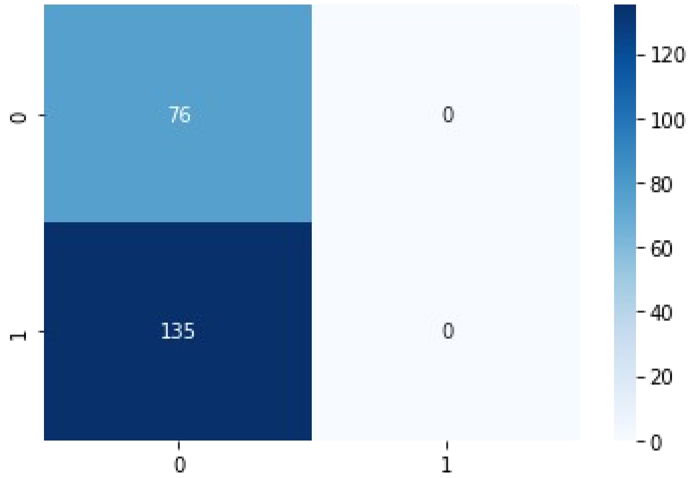
40-60 SPLIT WITHOUT PARAMETER TUNING

In [97]:

```
from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.4,test_size=0.6

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier  =  hmm.GaussianHMM() 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

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, zero_division=1))

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: [[ 76
         0]
 [135   0]]
-------------------------------------------------------
-------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.36      1.00      0.53        76
           g       1.00      0.00      0.00       135

    accuracy                           0.36       211
   macro avg       0.68      0.50      0.26       211
weighted  avg      0.77      0.36      0.19       211

-------------------------------------------------------
-------------------------------------------------------
Accuracy:
0.36018957345971564
```

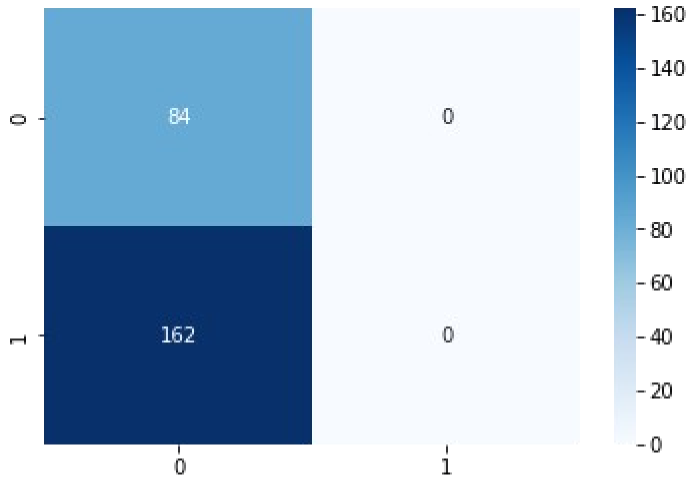


30-70 SPLIT WITHOUT PARAMETER TUNING

In [98]:
```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM()
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, zero_division=1))

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: [[ 84
          0]
 [162    0]]
---------------------------------------------
---------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.34      1.00      0.51        84
           g       1.00      0.00      0.00       162

    accuracy                           0.34       246
   macro avg       0.67      0.50      0.25       246
weighted  avg       0.78      0.34      0.17       246

---------------------------------------------
---------------------------------------------
Accuracy:
0.34146341463414637
```



WITH PARAMETER TUNING GAUSSIAN HMM

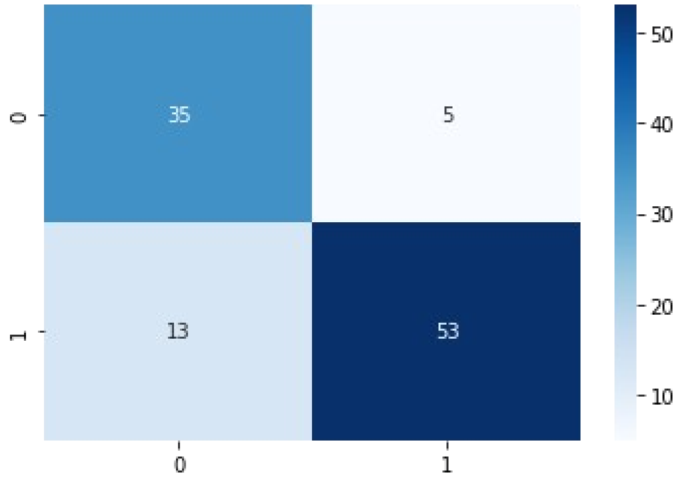
70-30 SPLIT WITH PARAMETER TUNING Algorithm, covariance type, n_iter, verbose

In [99]:

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier  =  hmm.GaussianHMM(n_components=2,  covariance_type="full",n_iter=5,algorit 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
```



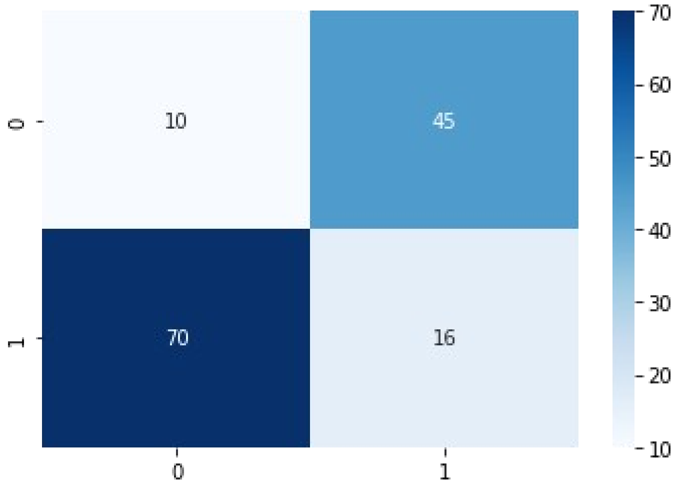
60-40 SPLIT WITH PARAMETER TUNING Algorithm, covariance type, n_iter, verbose

In [100…

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=5,algorit
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:
[[10 45]
 [70 16]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.12      0.18      0.15        55
           g       0.26      0.19      0.22        86

    accuracy                           0.18       141
   macro avg       0.19      0.18      0.18       141
weighted  avg      0.21      0.18      0.19       141

------------------------------------------------
------------------------------------------------
Accuracy:
0.18439716312056736
```



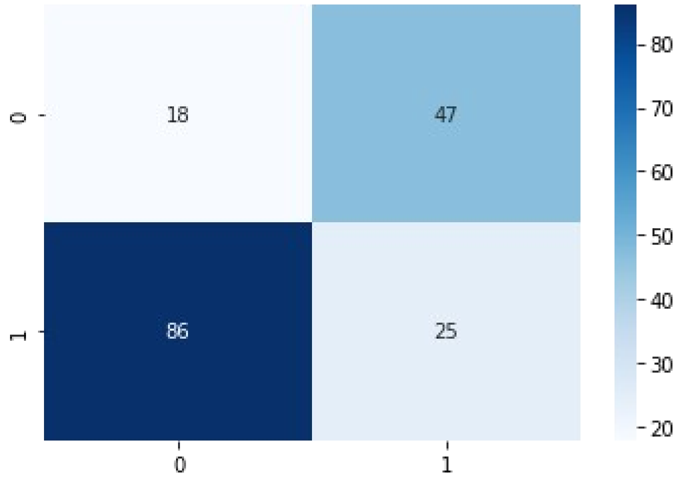
50-50 SPLIT WITH PARAMETER TUNING Algorithm, covariance type, n_iter, verbose

In [101…

```
from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.5,test_size=0.5

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier  =  hmm.GaussianHMM(n_components=2,  covariance_type="full",n_iter=5,algorit 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:
[[18 47]
 [86 25]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.17      0.28      0.21        65
           g       0.35      0.23      0.27       111

    accuracy                           0.24       176
   macro avg       0.26      0.25      0.24       176
weighted avg       0.28      0.24      0.25       176

------------------------------------------------
------------------------------------------------
Accuracy:
0.24431818181818182
```



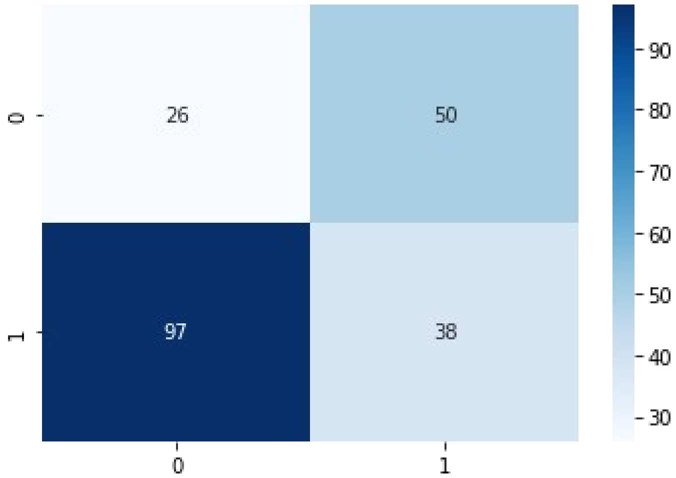
40-60 SPLIT WITH PARAMETER TUNING Algorithm, covariance type, n_iter, verbose

In [102…

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=5,algorit
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:
[[26 50]
 [97 38]]
-----------------------------------------------
-----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.21      0.34      0.26        76
           g       0.43      0.28      0.34       135

    accuracy                           0.30       211
   macro avg       0.32      0.31      0.30       211
weighted  avg      0.35      0.30      0.31       211

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.3033175355450237
```



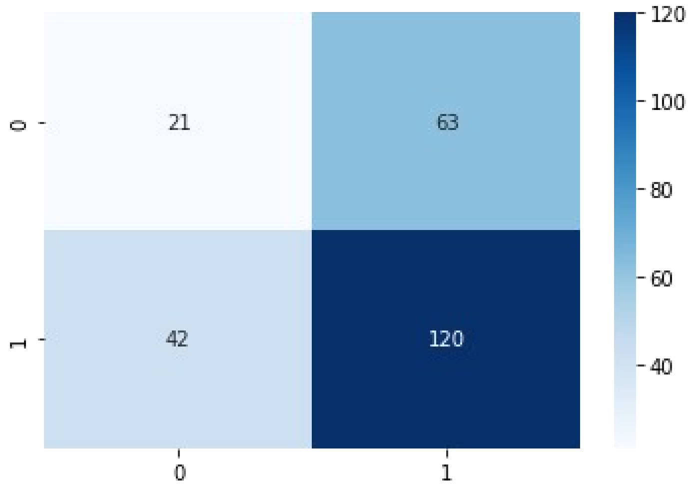
30-70 SPLIT WITH PARAMETER TUNING Algorithm, covariance type, n_iter, verbose

In [103…

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier  =  hmm.GaussianHMM(n_components=2,  covariance_type="full",n_iter=5,algorit 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

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: [[ 21
        63]
 [ 42  120]]
-----------------------------------------------
-----------------------------------------------
Performance Evaluation
               precision    recall  f1-score   support

           b       0.33      0.25      0.29        84
           g       0.66      0.74      0.70       162

    accuracy                           0.57       246
   macro avg       0.49      0.50      0.49       246
weighted  avg      0.55      0.57      0.56       246

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.573170731707317
```



WITHOUT PARAMETER TUNING GMM HMM

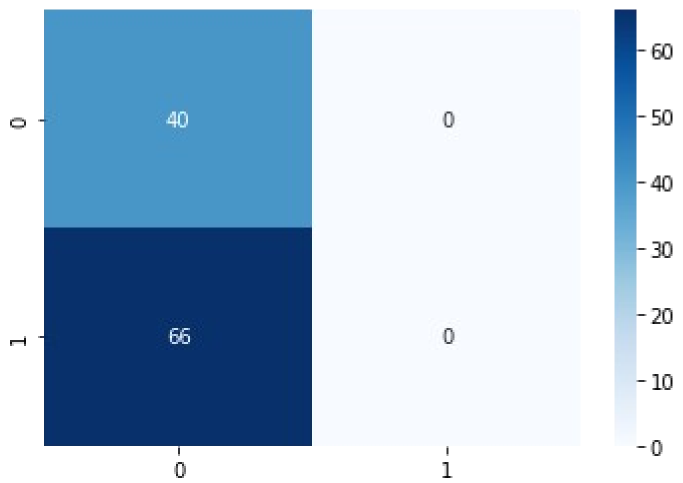
70-30 SPLIT WITHOUT PARAMETER TUNING

In [104…

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier = hmmlearn.hmm.GMMHMM()
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, zero_division=1))

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: [[40
      0]
 [66  0]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
               precision    recall  f1-score   support

           b       0.38      1.00      0.55        40
           g       1.00      0.00      0.00        66

    accuracy                           0.38       106
   macro avg       0.69      0.50      0.27       106
weighted  avg      0.77      0.38      0.21       106

------------------------------------------------
------------------------------------------------
Accuracy:
0.37735849056603776
```



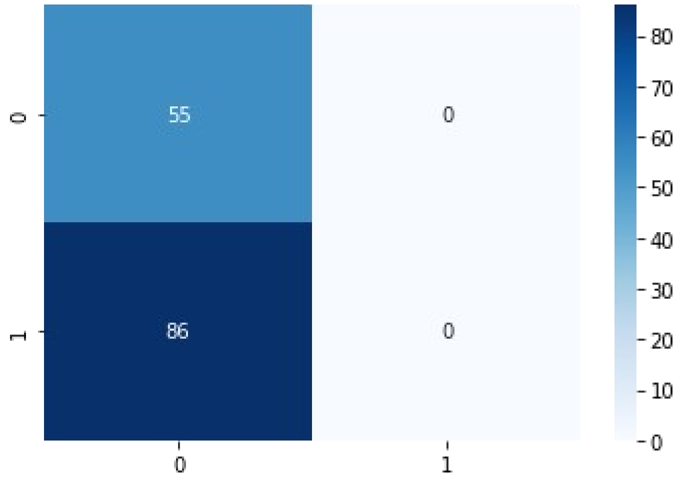
60-40 SPLIT WITHOUT PARAMETER TUNING

In [105…

```
from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.6,test_size=0.4

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.GMMHMM() 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, zero_division=1))

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: [[55
       0]
 [86  0]]
-----------------------------------------------------
-----------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.39      1.00      0.56        55
           g       1.00      0.00      0.00        86

    accuracy                           0.39       141
   macro avg       0.70      0.50      0.28       141
weighted  avg      0.76      0.39      0.22       141

-----------------------------------------------------
-----------------------------------------------------
Accuracy:
0.3900709219858156
```



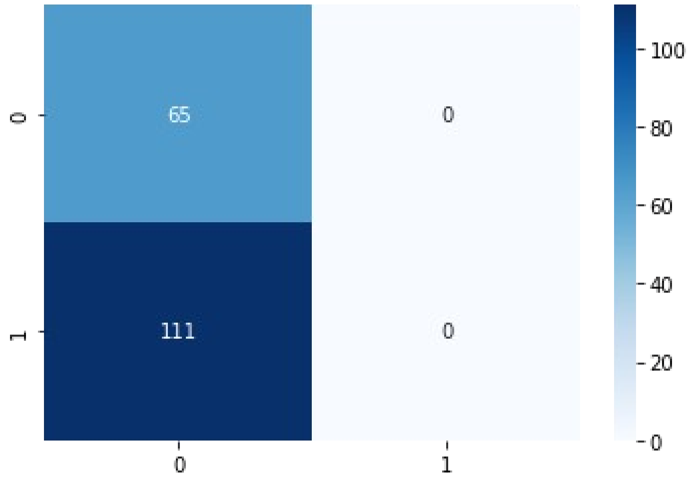
50-50 SPLIT WITHOUT PARAMETER TUNING

In [106…

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM()
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, zero_division=1))

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

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

```python
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: [[ 65
          0]
 [111    0]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.37      1.00      0.54        65
           g       1.00      0.00      0.00       111

    accuracy                           0.37       176
   macro avg       0.68      0.50      0.27       176
weighted  avg      0.77      0.37      0.20       176

------------------------------------------------
------------------------------------------------
Accuracy:
0.3693181818181818
```



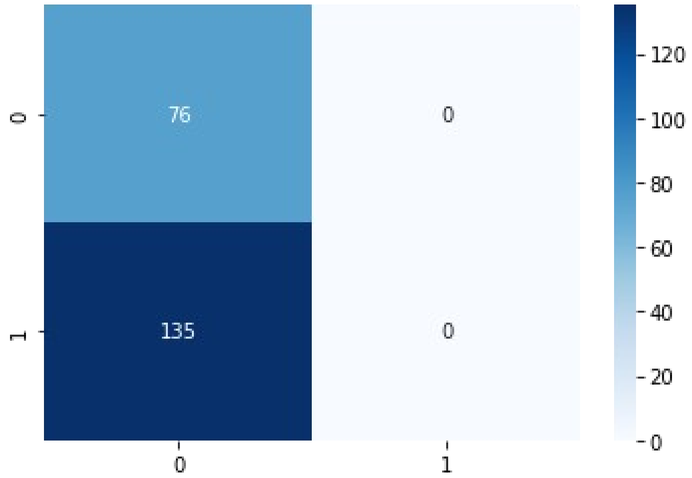
40-60 SPLIT WITHOUT PARAMETER TUNING

In [107…

```python
from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.4,test_size=0.6

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.GMMHMM() 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, zero_division=1))

 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: [[ 76
          0]
 [135   0]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score    support

           b       0.36      1.00      0.53        76
           g       1.00      0.00      0.00       135

    accuracy                           0.36       211
   macro avg       0.68      0.50      0.26       211
weighted  avg      0.77      0.36      0.19       211

------------------------------------------------
------------------------------------------------
Accuracy:
0.36018957345971564
```



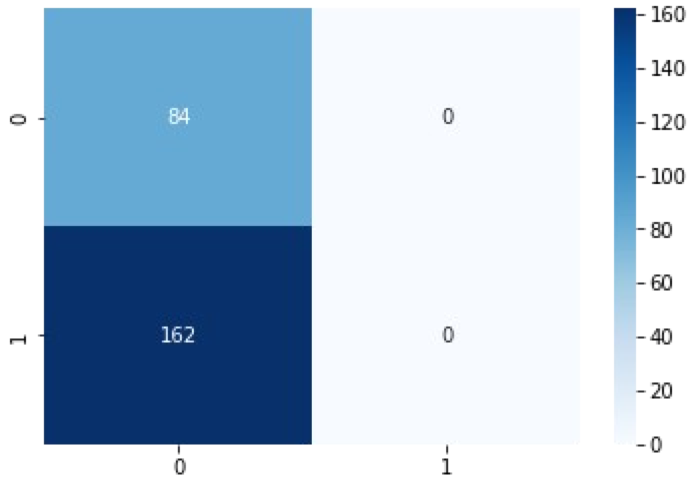
30-70 SPLIT WITHOUT PARAMETER TUNING

In [108…

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM()
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, zero_division=1))

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: [[ 84
            0]
 [162    0]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.34      1.00      0.51        84
           g       1.00      0.00      0.00       162

    accuracy                           0.34       246
   macro avg       0.67      0.50      0.25       246
weighted  avg      0.78      0.34      0.17       246

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.34146341463414637
```



WITH PARAMETER TUNING GMM HMM

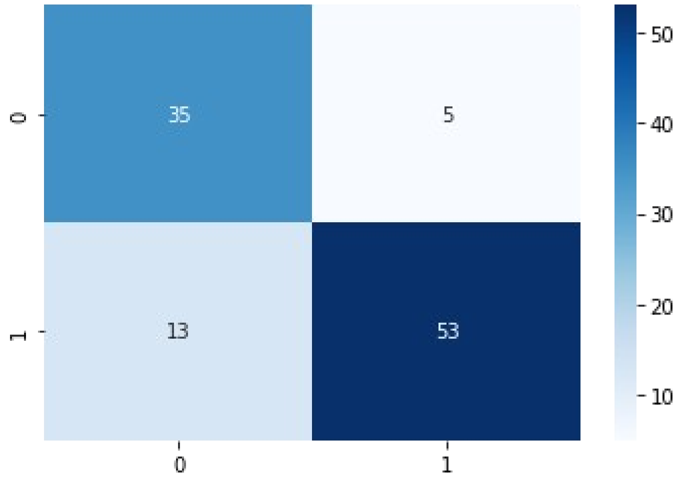
70-30 SPLIT WITH PARAMETER TUNING n_components, random_state, covariance_type, algorithm, n_iter

In [109…

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2, random_state=10,covariance_type='fu 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
```



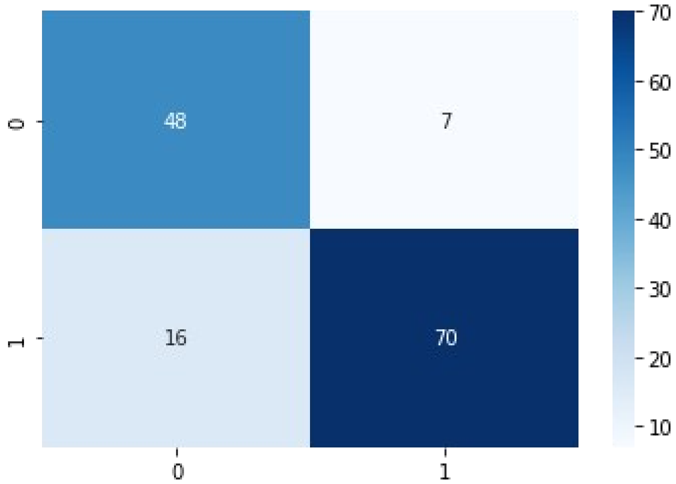
60-40 SPLIT WITH PARAMETER TUNING n_components, random_state, covariance_type, algorithm, n_iter

In [110…

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2, random_state=10,covariance_type='fu
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: [[48
        7]
 [16 70]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.75      0.87      0.81        55
           g       0.91      0.81      0.86        86

    accuracy                           0.84       141
   macro avg       0.83      0.84      0.83       141
weighted  avg      0.85      0.84      0.84       141

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.8368794326241135
```



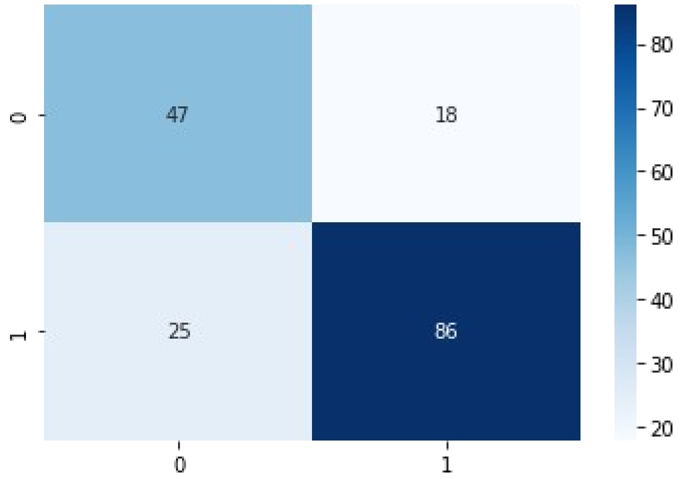
50-50 SPLIT WITH PARAMETER TUNING n_components, random_state, covariance_type, algorithm, n_iter

In [111…

```
from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.5,test_size=0.5

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.GMMHMM(n_components=2,  random_state=10,covariance_type='fu 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:
[[47 18]
 [25 86]]
-----------------------------------------------------
-----------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.65      0.72      0.69        65
           g       0.83      0.77      0.80       111

    accuracy                           0.76       176
   macro avg       0.74      0.75      0.74       176
weighted avg       0.76      0.76      0.76       176

-----------------------------------------------------
-----------------------------------------------------
Accuracy:
0.7556818181818182
```



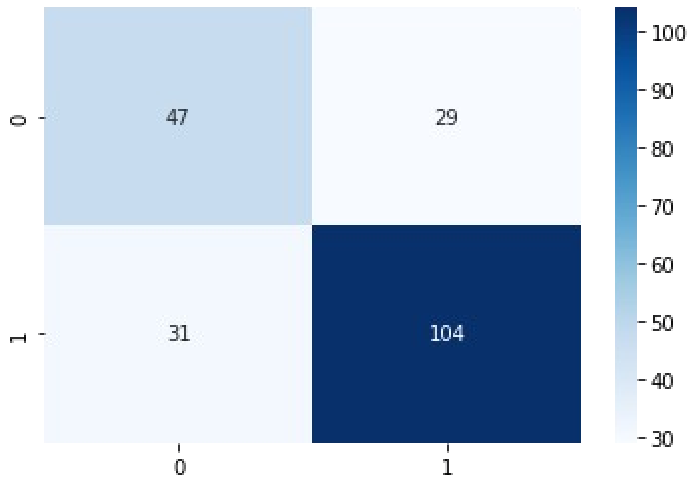
40-60 SPLIT WITH PARAMETER TUNING n_components, random_state, covariance_type, algorithm, n_iter

In [112…

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2, random_state=10,covariance_type='fu
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: [[ 47
         29]
 [ 31  104]]
-----------------------------------------------
-----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.60      0.62      0.61        76
           g       0.78      0.77      0.78       135

    accuracy                           0.72       211
   macro avg       0.69      0.69      0.69       211
weighted  avg      0.72      0.72      0.72       211

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.7156398104265402
```



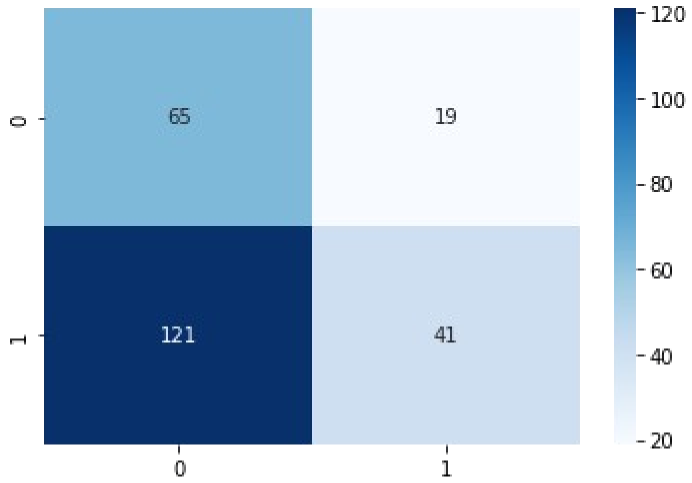
30-70 SPLIT WITH PARAMETER TUNING n_components, random_state, covariance_type, algorithm, n_iter

In [113…

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.GMMHMM(n_components=2,  random_state=10,covariance_type='fu 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: [[ 65
        19]
 [121  41]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.35      0.77      0.48        84
           g       0.68      0.25      0.37       162

    accuracy                           0.43       246
   macro avg       0.52      0.51      0.43       246
weighted  avg      0.57      0.43      0.41       246

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.43089430894308944
```



WITHOUT PARAMETER TUNING MULTINOMIAL HMM

70-30 SPLIT WITHOUT PARAMETER TUNING

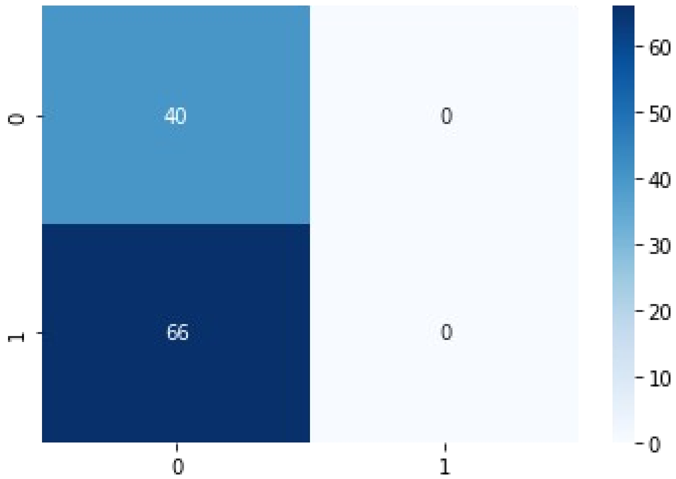
In [114…

```
#DATASET PREPARATION FOR MULTINOMIAL

                                        col_name    =    ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
                                                      ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

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




# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier  =  hmmlearn.hmm.MultinomialHMM()

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, zero_division=1))


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: [[40
        0]
 [66  0]]
-----------------------------------------------------
-----------------------------------------------------
```

```
Performance Evaluation
              precision    recall  f1-score   support

           b       0.38      1.00      0.55        40
           g       1.00      0.00      0.00        66

    accuracy                           0.38       106
   macro avg       0.69      0.50      0.27       106
weighted  avg       0.77      0.38      0.21       106

-----------------------------------------------------
-----------------------------------------------------
Accuracy:
0.37735849056603776
```



60-40 SPLIT WITHOUT PARAMETER TUNING

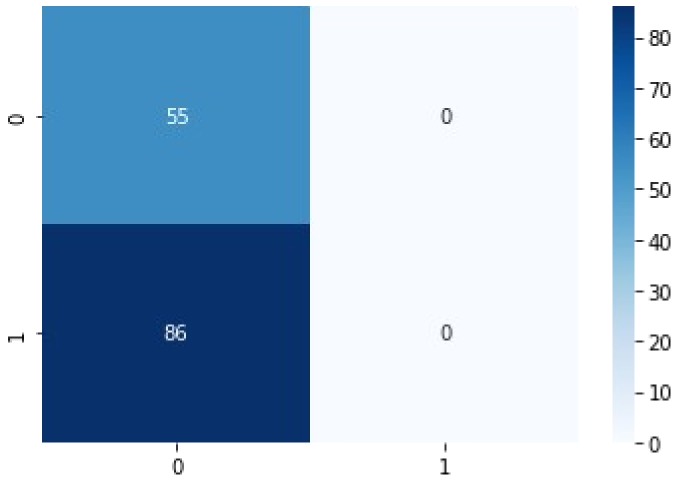
In [115…

```
#DATASET PREPARATION FOR MULTINOMIAL
                    col_name    =    ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
                                     ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.6,test_size=0.4




# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier  =  hmmlearn.hmm.MultinomialHMM()

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[0:141]


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, zero_division=1))


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: [[55
         0]
 [86  0]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.39      1.00      0.56        55
           g       1.00      0.00      0.00        86

    accuracy                           0.39       141
   macro avg       0.70      0.50      0.28       141
weighted avg       0.76      0.39      0.22       141

------------------------------------------------
------------------------------------------------
Accuracy:
0.3900709219858156
```



50-50 SPLIT WITHOUT PARAMETER TUNING

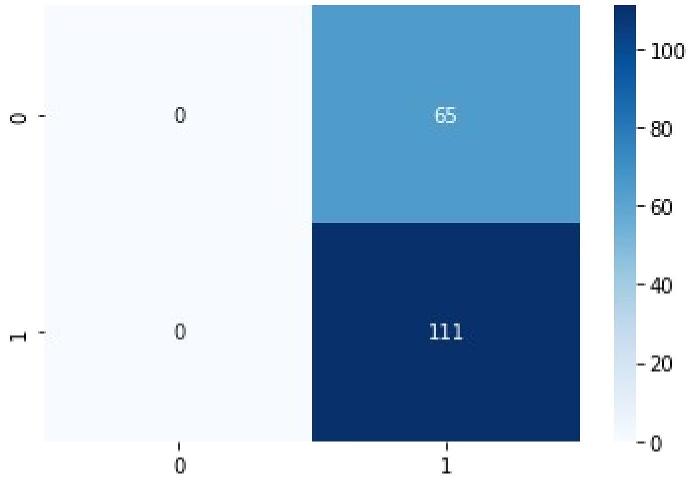
In [116…

```
#DATASET PREPARATION FOR MULTINOMIAL

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5



# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM()

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[0:176]


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, zero_division=1))


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
     65]
 [   0  111]]
-----------------------------------------------
-----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.00      0.00        65
           g       0.63      1.00      0.77       111

    accuracy                           0.63       176
   macro avg       0.82      0.50      0.39       176
weighted  avg       0.77      0.63      0.49       176

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.6306818181818182
```



40-60 SPLIT WITHOUT PARAMETER TUNING

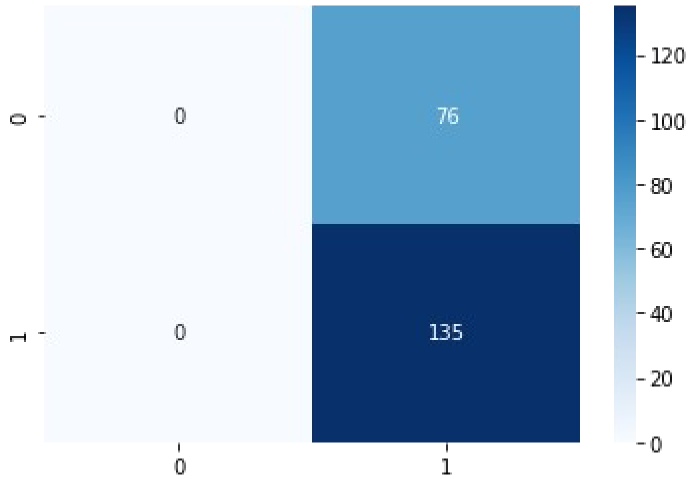
In [117…

```
#DATASET PREPARATION FOR MULTINOMIAL

col_name    =    ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

```
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM()

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[0:211]


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, zero_division=1))

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
   76]
 [  0 135]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.00      0.00        76
           g       0.64      1.00      0.78       135

    accuracy                           0.64       211
   macro avg       0.82      0.50      0.39       211
weighted avg       0.77      0.64      0.50       211

------------------------------------------------
------------------------------------------------
Accuracy:
0.6398104265402843
```



30-70 SPLIT WITHOUT PARAMETER TUNING

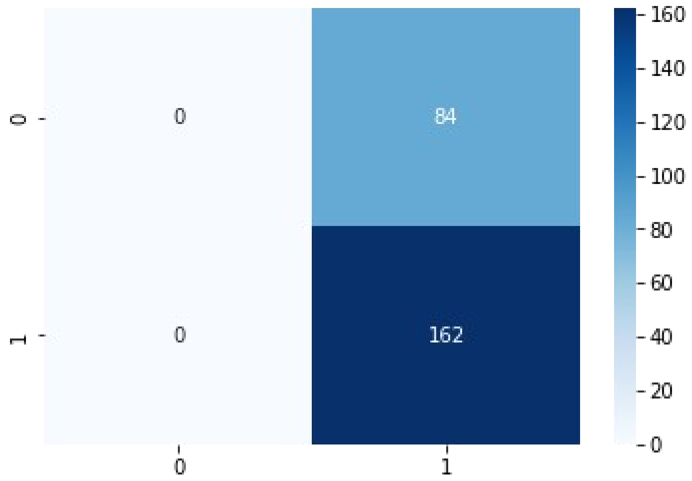
In [118…

```
#DATASET PREPARATION FOR MULTINOMIAL
```

```
col_name    =   ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
               ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.MultinomialHMM()

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)
```

```python
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, zero_division=1))


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
    84]
 [   0  162]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       1.00      0.00      0.00        84
           g       0.66      1.00      0.79       162

    accuracy                           0.66       246
   macro avg       0.83      0.50      0.40       246
weighted  avg      0.78      0.66      0.52       246

------------------------------------------------
------------------------------------------------
Accuracy:
0.6585365853658537
```



WITH PARAMETER TUNING MULTINOMIAL HMM

70-30 SPLIT WITH PARAMETER TUNING n_components, random_state, n_iter, algorithm, params

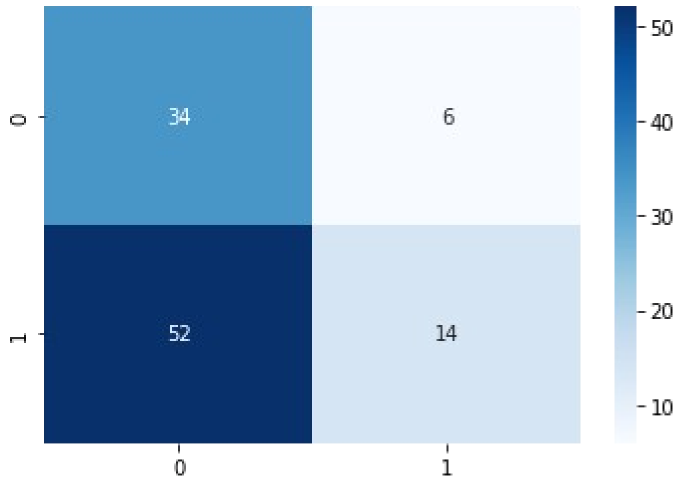
In [119…

```
#DATASET PREPARATION FOR MULTINOMIAL

 col_name    =   ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
              ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.MultinomialHMM(n_components=4,  random_state=15,n_iter=10,a

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, zero_division=1))


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
```



60-40 SPLIT WITH PARAMETER TUNING n_components, random_state, n_iter, algorithm, params

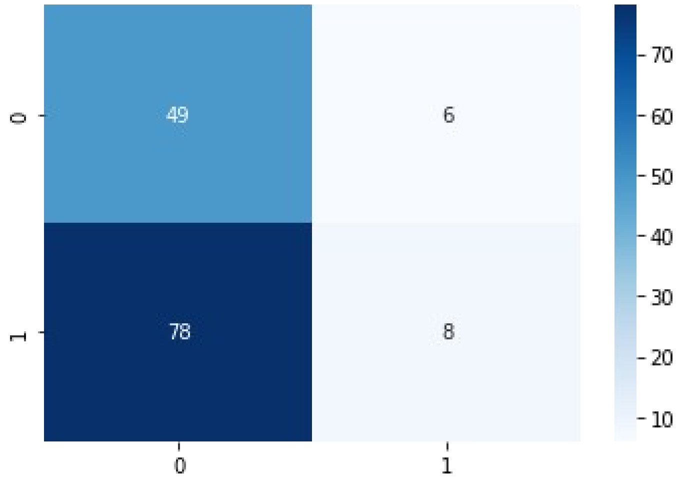
In [120…

```
#DATASET PREPARATION FOR MULTINOMIAL

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4, random_state=15,n_iter=10,a
```

```
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[0:141]


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, zero_division=1))


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: [[49
        6]
 [78  8]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.39      0.89      0.54        55
           g       0.57      0.09      0.16        86

    accuracy                           0.40       141
   macro avg       0.48      0.49      0.35       141
weighted  avg       0.50      0.40      0.31       141

------------------------------------------------
------------------------------------------------
Accuracy:
0.40425531914893614
```



50-50 SPLIT WITH PARAMETER TUNING n_components, random_state, n_iter, algorithm, params

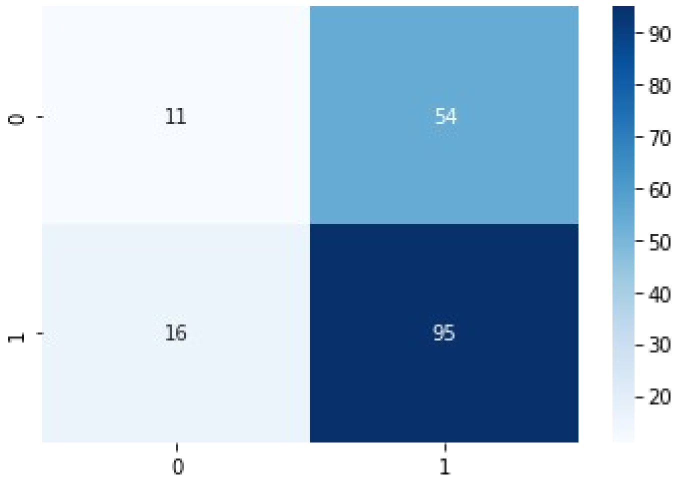
In [121…

```
#DATASET PREPARATION FOR MULTINOMIAL

col_name    =    ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.5,test_size=0.5
```

```python
# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4, random_state=15,n_iter=10,a

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[0:176]


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, zero_division=1))


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:
[[11 54]
 [16 95]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.41      0.17      0.24        65
           g       0.64      0.86      0.73       111

    accuracy                           0.60       176
   macro avg       0.52      0.51      0.48       176
weighted avg       0.55      0.60      0.55       176

------------------------------------------------
------------------------------------------------
Accuracy:
0.6022727272727273
```



40-60 SPLIT WITH PARAMETER TUNING n_components, random_state, n_iter, algorithm, params

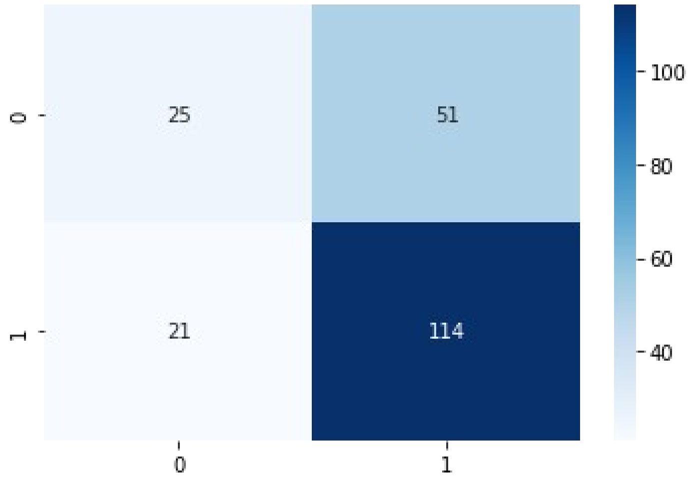
In [122…

```
#DATASET PREPARATION FOR MULTINOMIAL

col_name     =     ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3
```

```
df.columns = col_name

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

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

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4, random_state=15,n_iter=10,a

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[0:211]


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, zero_division=1))

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: [[ 25
        51]
 [ 21  114]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           b       0.54      0.33      0.41        76
           g       0.69      0.84      0.76       135

    accuracy                           0.66       211
   macro avg       0.62      0.59      0.58       211
weighted  avg       0.64      0.66      0.63       211

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.6587677725118484
```



30-70 SPLIT WITH PARAMETER TUNING n_components, random_state, n_iter, algorithm, params

In [123…

```
#DATASET PREPARATION FOR MULTINOMIAL

col_name    =    ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','
             ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','3

df.columns = col_name

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

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

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.MultinomialHMM(n_components=4,  random_state=15,n_iter=10,a

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
```

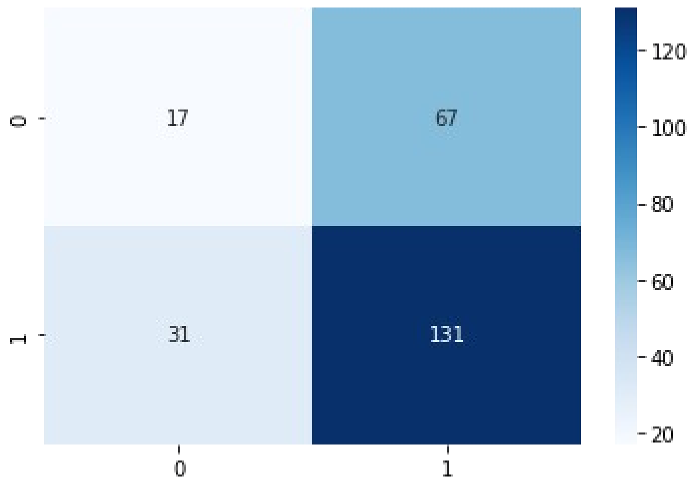
```
import math
row = len(X_test)
col = len(X_test[0])
new
for i in range(row):
    for j in range(col):
        X_test[i][j] = X_test[i][j]*10
        X_test[i][j] = math.floor(X_test[i][j])
    x = X_test[i].astype(np.int)
    new = np.vstack([new,x])

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, zero_division=1))


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



WORKING WITH WINE DATASET

Without and With Parameter
Tuning TABULATION

(CODE ALONGWITH OUTPUTS ATTACHED AT THE END
OF TABULATION)

CLASSIFIER	PARAMETER TUNING	TRAIN-TEST RATIO	PRECISION	RECALL	F1 SCORE	SUPPORT	ACCURACY
GAUSSIAN CLASSIFIER	No	70:30	0.76	0.33	0.14	54	0.27
	Yes		0.37	0.02	0.03	54	0.037
	No	60:40	0.76	0.33	0.14	72	0.27
	Yes		0.84	0.65	0.57	72	**0.70**
	No	50:50	0.78	0.33	0.16	89	0.35
	Yes		0.83	0.65	0.56	89	0.69
	No	40:60	0.77	0.33	0.16	107	0.31
	Yes		0.84	0.63	0.55	107	0.67
	No	30:70	0.78	0.33	0.16	125	0.32
	Yes		0.80	0.61	0.52	125	0.65

CLASSIFIER	PARAMETER TUNING	TRAIN-TEST RATIO	PRECISION	RECALL	F1 SCORE	SUPPORT	ACCURACY
GMM CLASSIFIER	No	70:30	0.76	0.33	0.14	54	0.27
	Yes		0.37	0.02	0.03	54	0.03
	No	60:40	0.76	0.33	0.14	72	0.27
	Yes		0.86	0.66	0.58	72	**0.72**
	No	50:50	0.78	0.33	0.16	89	0.32
	Yes		0.83	0.65	0.56	89	0.69
	No	40:60	0.77	0.33	0.16	107	0.31
	Yes		0.84	0.63	0.55	107	0.67
	No	30:70	0.78	0.33	0.16	125	0.32
	Yes		0.41	0.08	0.08	125	0.096

WINE DATASET

In [6]:

```
#DATASET PREPARATION AND IMPORTS

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','To
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/

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

WITHOUT PARAMETER TUNING GAUSSIAN

HMM 70-30 SPLIT WITHOUT PARAMETER

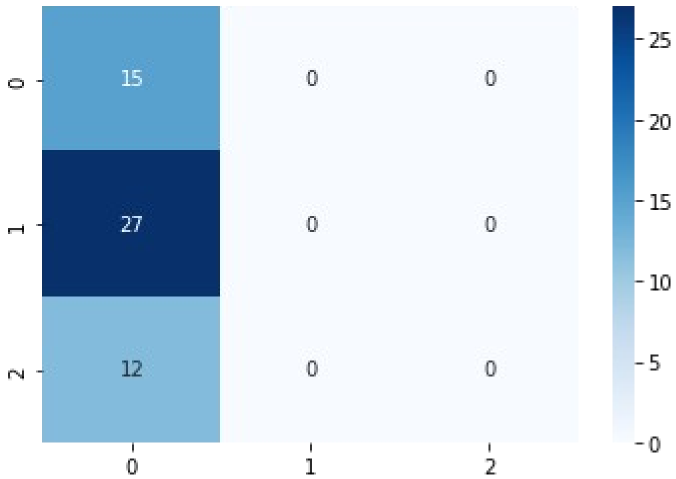
TUNING

In [7]:

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM() 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, zero_division=1))

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:
[[15  0  0]
 [27  0  0]
 [12  0  0]]
-------------------------
 ---------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.28      1.00      0.43        15
           2       1.00      0.00      0.00        27
           3       1.00      0.00      0.00        12

    accuracy                           0.28        54
   macro avg       0.76      0.33      0.14        54
weighted  avg      0.80      0.28      0.12        54

---------------------------------------------
---------------------------------------------
Accuracy:
0.2777777777777778
```



60-40 SPLIT WITHOUT PARAMETER TUNING

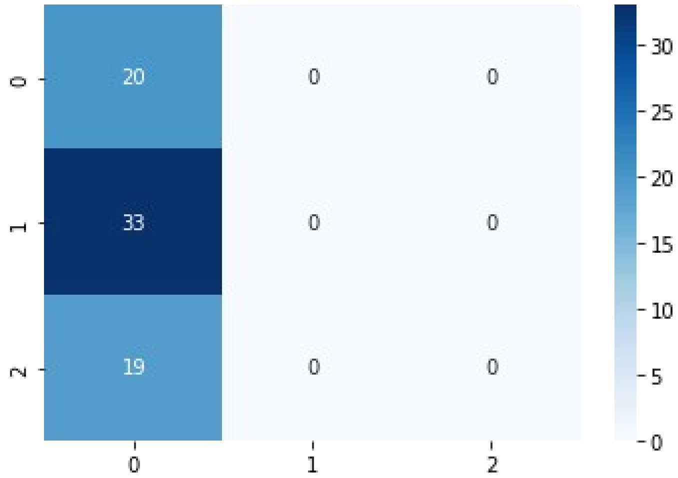
In [8]:

```
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.6,test_size=0.4)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm
```

```
classifier = hmm.GaussianHMM()
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, zero_division=1))

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]
 [33  0  0]
 [19  0  0]]
_________________________
 -----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.28      1.00      0.43        20
           2       1.00      0.00      0.00        33
           3       1.00      0.00      0.00        19

    accuracy                           0.28        72
   macro avg       0.76      0.33      0.14        72
weighted avg       0.80      0.28      0.12        72

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.2777777777777778
```



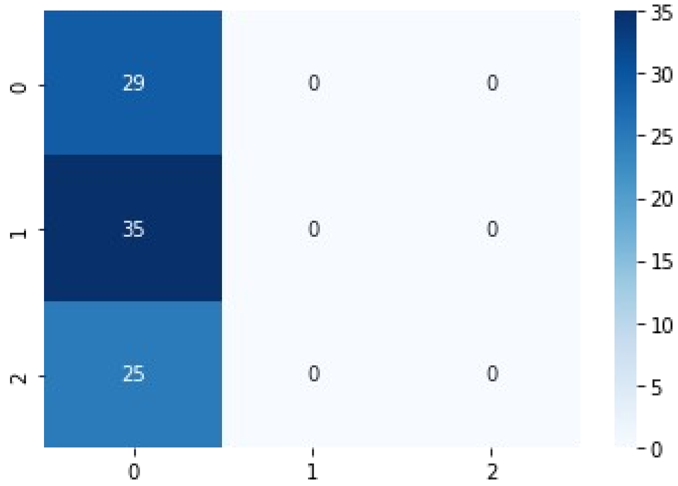
50-50 SPLIT WITHOUT PARAMETER TUNING

In [9]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM()
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, zero_division=1))

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:
[[29  0  0]
 [35  0  0]
 [25  0  0]]
____________________________
 ----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.33      1.00      0.49        29
           2       1.00      0.00      0.00        35
           3       1.00      0.00      0.00        25

    accuracy                           0.33        89
   macro avg       0.78      0.33      0.16        89
weighted avg       0.78      0.33      0.16        89

----------------------------------------------
----------------------------------------------
Accuracy:
0.3258426966292135
```



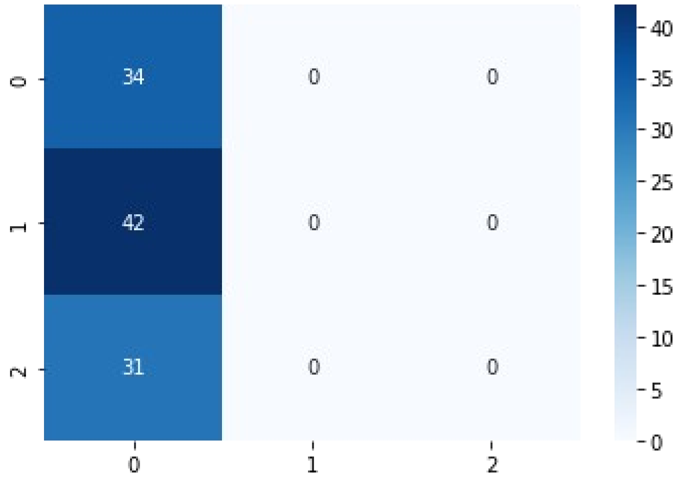
40-60 SPLIT WITHOUT PARAMETER TUNING

In [10]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM() classifier.fit(X_train)
```

```python
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, zero_division=1))

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   0   0]
 [42   0   0]
 [31   0   0]]
------------------------------------------------------
------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.32      1.00      0.48        34
           2       1.00      0.00      0.00        42
           3       1.00      0.00      0.00        31

    accuracy                           0.32       107
   macro avg       0.77      0.33      0.16       107
weighted avg       0.78      0.32      0.15       107

------------------------------------------------------
------------------------------------------------------
Accuracy:
0.3177570093457944
```



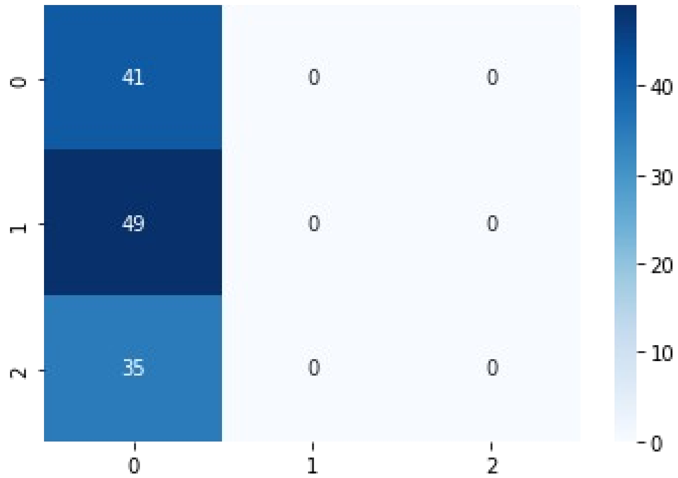
30-70 SPLIT WITHOUT PARAMETER TUNING

In [11]:

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM()
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, zero_division=1))

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:
[[41   0   0]
 [49   0   0]
 [35   0   0]]
------------------------------------------------------
  ---------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.33      1.00      0.49        41
           2       1.00      0.00      0.00        49
           3       1.00      0.00      0.00        35

    accuracy                           0.33       125
   macro avg       0.78      0.33      0.16       125
weighted  avg      0.78      0.33      0.16       125

---------------------------------------------
---------------------------------------------
Accuracy:
0.328
```



WITH PARAMETER TUNING GAUSSIAN HMM

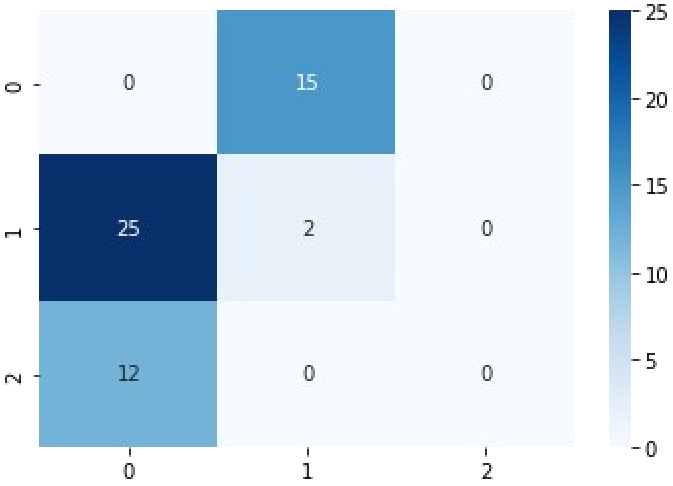
70-30 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

In [12]:

```
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)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm
```

```
classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori
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, zero_division=1))


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]
 [25  2  0]
 [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       1.00      0.00      0.00        12

    accuracy                           0.04        54
   macro avg       0.37      0.02      0.03        54
weighted avg       0.28      0.04      0.05        54

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

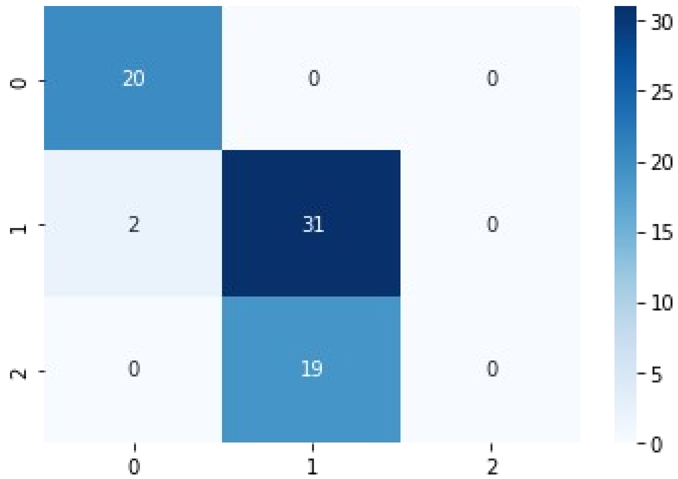


60-40 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori
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, zero_division=1))
```

```
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  19   0]]
__________________________________________
 -----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.91      1.00      0.95        20
           2       0.62      0.94      0.75        33
           3       1.00      0.00      0.00        19

    accuracy                           0.71        72
   macro avg       0.84      0.65      0.57        72
weighted  avg       0.80      0.71      0.61        72

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.7083333333333334
```



50-50 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

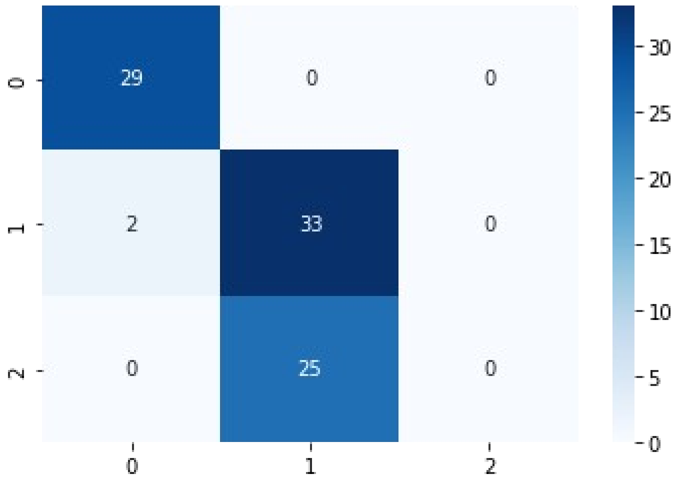
In [14]:

```
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.5,test_size=0.5

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
```

```python
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori
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, zero_division=1))


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:
[[29  0  0]
 [ 2 33  0]
 [ 0 25  0]]
------------------------------------------------------
------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.94      1.00      0.97        29
           2       0.57      0.94      0.71        35
           3       1.00      0.00      0.00        25

    accuracy                           0.70        89
   macro avg       0.83      0.65      0.56        89
weighted avg       0.81      0.70      0.59        89

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

Accuracy:
0.6966292134831461



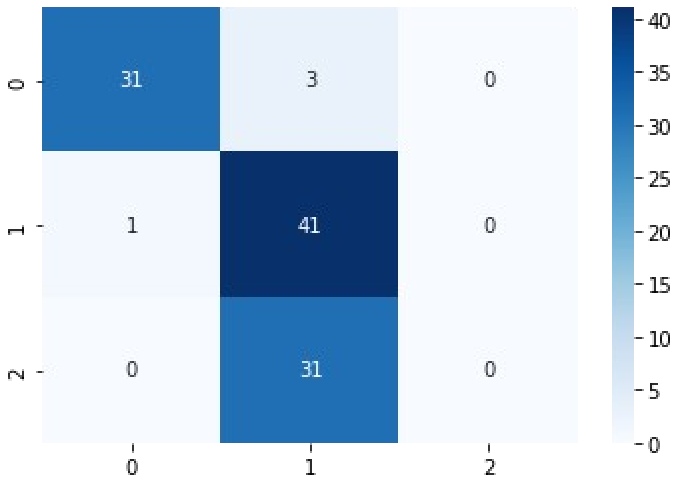
40-60 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

In [15]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori 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, zero_division=1))


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:
[[31   3   0]
 [ 1  41   0]
 [ 0  31   0]]
_________________________
 ------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.97      0.91      0.94        34
           2       0.55      0.98      0.70        42
           3       1.00      0.00      0.00        31

    accuracy                           0.67       107
   macro avg       0.84      0.63      0.55       107
weighted avg       0.81      0.67      0.57       107

------------------------------------------------
------------------------------------------------
Accuracy:
0.6728971962616822
```



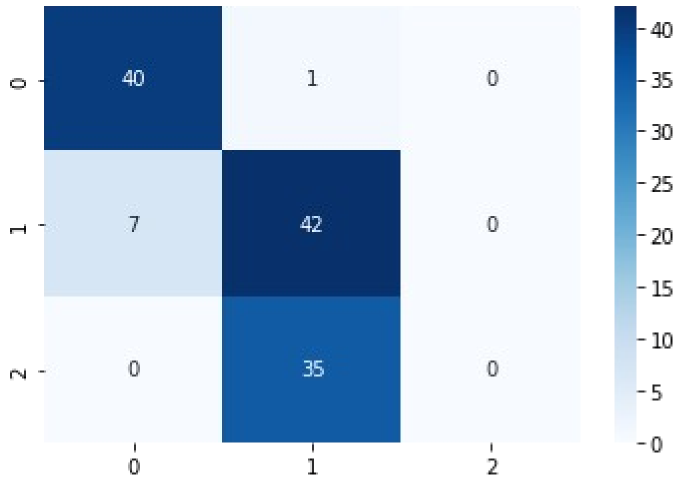
30-70 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

In [16]:
```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

```python
# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori
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, zero_division=1))


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:
[[40  1  0]
 [ 7 42  0]
 [ 0 35  0]]
------------------------------------------------------
------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.85      0.98      0.91        41
           2       0.54      0.86      0.66        49
           3       1.00      0.00      0.00        35

    accuracy                           0.66       125
   macro avg       0.80      0.61      0.52       125
weighted avg       0.77      0.66      0.56       125

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

Accuracy:
0.656



WITHOUT PARAMETER TUNING GMM HMM

70-30 SPLIT WITHOUT PARAMETER TUNING

In [17]:

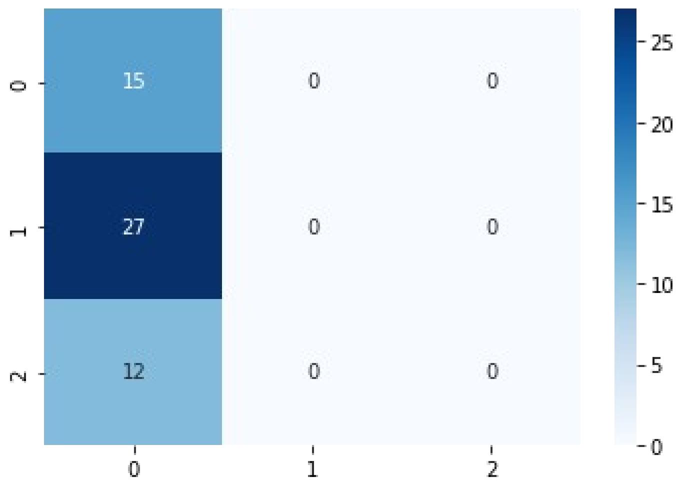
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GMMHMM() 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, zero_division=1))

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:
[[15   0   0]
 [27   0   0]
 [12   0   0]]
-------------------------
 ----------------------------------------------
Performance Evaluation
```

	precision	recall	f1-score	support
1	0.28	1.00	0.43	15
2	1.00	0.00	0.00	27
3	1.00	0.00	0.00	12
accuracy			0.28	54
macro avg	0.76	0.33	0.14	54
weighted avg	0.80	0.28	0.12	54

```
----------------------------------------------
----------------------------------------------
Accuracy:
0.2777777777777778
```



60-40 SPLIT WITHOUT PARAMETER TUNING

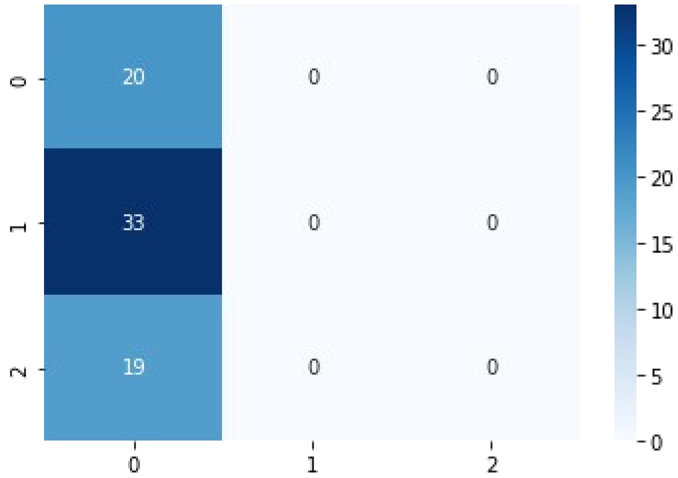
In [18]:

```
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.6,test_size=0.4

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
```

```python
from hmmlearn import hmm

classifier = hmm.GMMHMM() 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)
```

```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, zero_division=1))

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]
 [33  0  0]
 [19  0  0]]
__________________________________
 ------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.28      1.00      0.43        20
           2       1.00      0.00      0.00        33
           3       1.00      0.00      0.00        19

    accuracy                           0.28        72
   macro avg       0.76      0.33      0.14        72
weighted avg       0.80      0.28      0.12        72

------------------------------------------------
------------------------------------------------
Accuracy:
0.2777777777777778
```

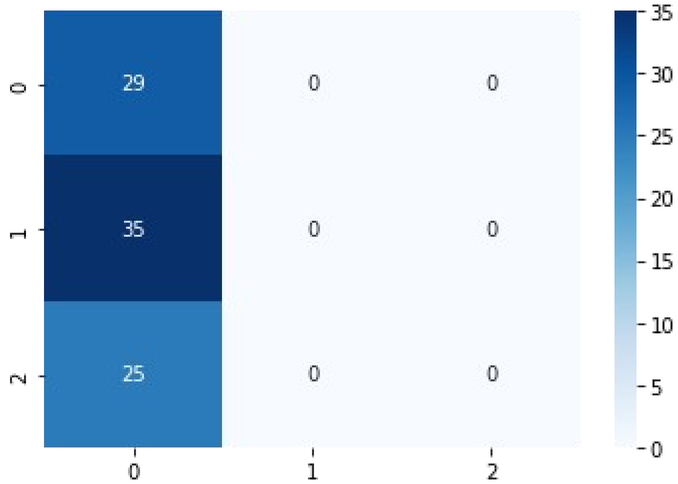


50-50 SPLIT WITHOUT PARAMETER TUNING

In [19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GMMHMM()
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, zero_division=1))

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:
[[29   0   0]
 [35   0   0]
 [25   0   0]]
---------------------------
 -----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.33      1.00      0.49        29
           2       1.00      0.00      0.00        35
           3       1.00      0.00      0.00        25

    accuracy                           0.33        89
   macro avg       0.78      0.33      0.16        89
weighted  avg       0.78      0.33      0.16        89

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.3258426966292135
```



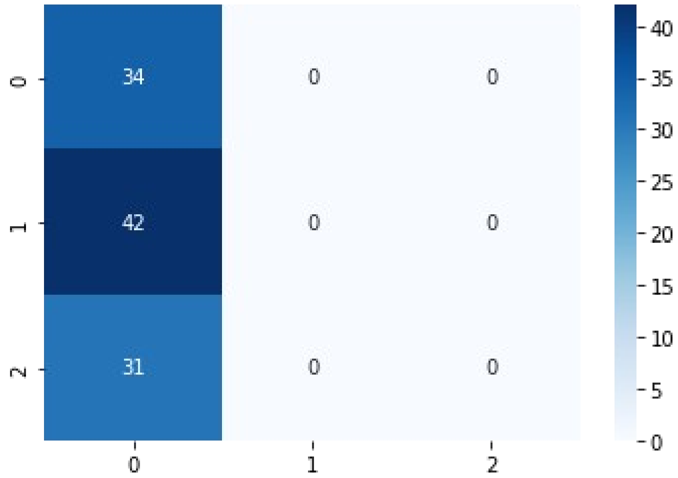
40-60 SPLIT WITHOUT PARAMETER TUNING

In [20]:

```
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.4,test_size=0.6

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier  =  hmm.GMMHMM() classifier.fit(X_train)
```

```python
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, zero_division=1))

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  0  0]
 [42  0  0]
 [31  0  0]]
-----------------------------------------------------
 -----------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.32      1.00      0.48        34
           2       1.00      0.00      0.00        42
           3       1.00      0.00      0.00        31

    accuracy                           0.32       107
   macro avg       0.77      0.33      0.16       107
weighted avg       0.78      0.32      0.15       107

-----------------------------------------------------
-----------------------------------------------------
Accuracy:
0.3177570093457944
```



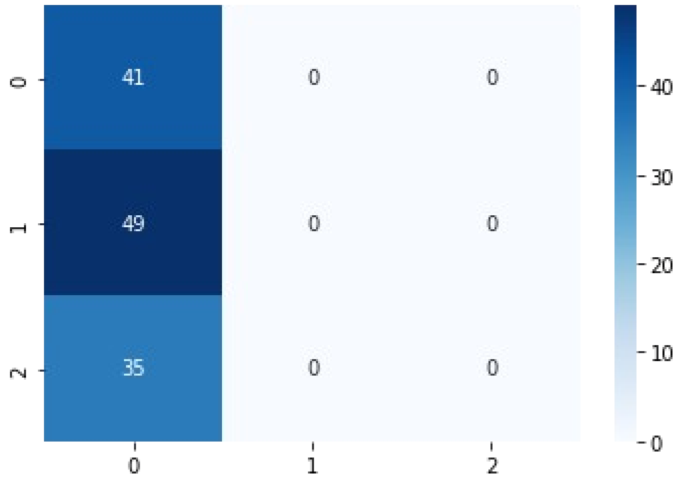
30-70 SPLIT WITHOUT PARAMETER TUNING

In [21]:

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GMMHMM()
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, zero_division=1))

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:
[[41   0   0]
 [49   0   0]
 [35   0   0]]
___________________________
 ----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.33      1.00      0.49        41
           2       1.00      0.00      0.00        49
           3       1.00      0.00      0.00        35

    accuracy                           0.33       125
   macro avg       0.78      0.33      0.16       125
weighted  avg      0.78      0.33      0.16       125

----------------------------------------------
----------------------------------------------
Accuracy:
0.328
```



WITH PARAMETER TUNING GMM HMM

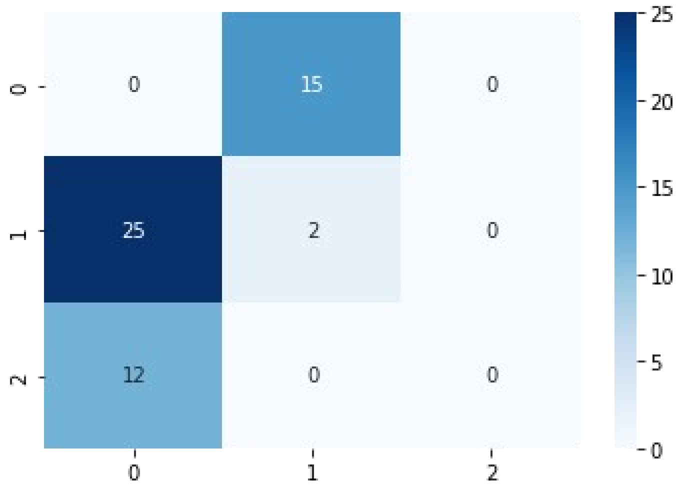
70-30 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

In [22]:

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm
```

```python
classifier = hmm.GMMHMM(n_components=2, covariance_type="full",n_iter=10,algorithm="
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, zero_division=1))


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]
 [25   2   0]
 [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       1.00      0.00      0.00        12

    accuracy                           0.04        54
   macro avg       0.37      0.02      0.03        54
weighted avg       0.28      0.04      0.05        54

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



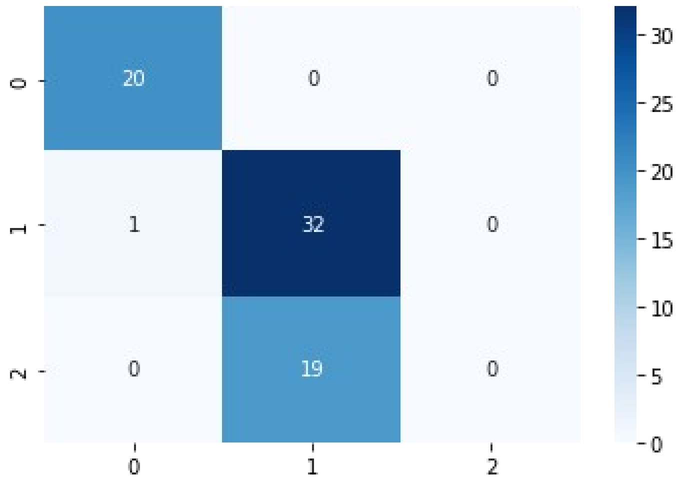
60-40 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

In [23]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GMMHMM(n_components=2, covariance_type="full",n_iter=10,algorithm='
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, zero_division=1))
```

```
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]
 [ 1 32  0]
 [ 0 19  0]]
--------------------------
----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.95      1.00      0.98        20
           2       0.63      0.97      0.76        33
           3       1.00      0.00      0.00        19

    accuracy                           0.72        72
   macro avg       0.86      0.66      0.58        72
weighted  avg      0.82      0.72      0.62        72

----------------------------------------------
----------------------------------------------
Accuracy:
0.7222222222222222
```



50-50 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

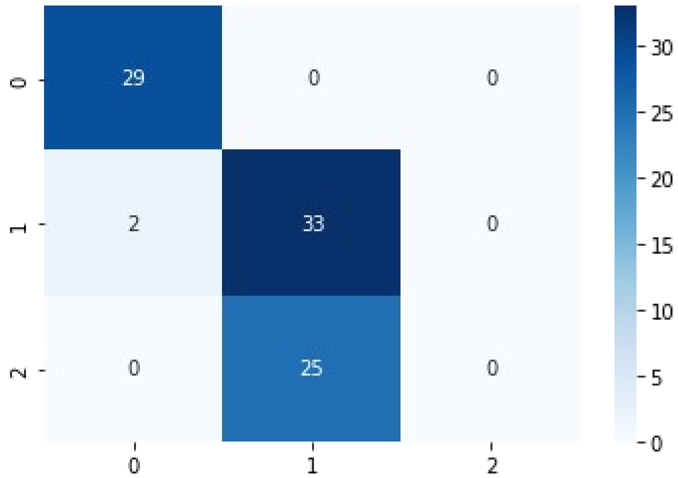
In [24]:

```
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.5,test_size=0.5

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
```

```
from hmmlearn import hmm

classifier = hmm.GMMHMM(n_components=2, covariance_type="full",n_iter=10,algorithm="
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, zero_division=1))


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:
[[29  0  0]
 [ 2 33  0]
 [ 0 25  0]]
-------------------------------------------------------
-------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.94      1.00      0.97        29
           2       0.57      0.94      0.71        35
           3       1.00      0.00      0.00        25

    accuracy                           0.70        89
   macro avg       0.83      0.65      0.56        89
weighted avg       0.81      0.70      0.59        89

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

Accuracy:
0.6966292134831461



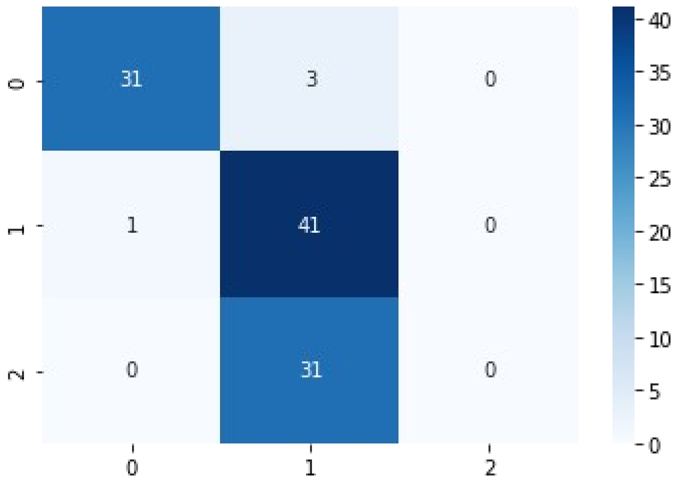
40-60 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

In [25]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GMMHMM(n_components=2, covariance_type="full",n_iter=10,algorithm=' 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, zero_division=1))


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:
[[31  3  0]
 [ 1 41  0]
 [ 0 31  0]]
___________________________
 -----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.97      0.91      0.94        34
           2       0.55      0.98      0.70        42
           3       1.00      0.00      0.00        31

    accuracy                           0.67       107
   macro avg       0.84      0.63      0.55       107
weighted avg       0.81      0.67      0.57       107

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.6728971962616822
```



30-70 SPLIT WITH PARAMETER TUNING n_components, covariance_type, n_iter, algorithm, verbose

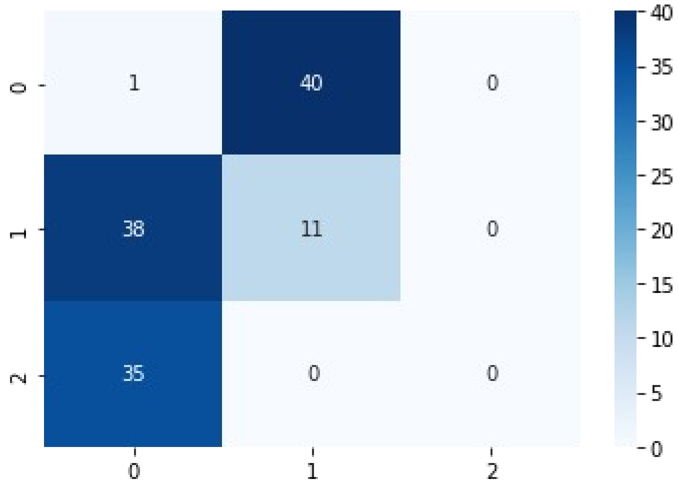
In [26]:

```
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

# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

```python
# Classification
from hmmlearn import hmm

classifier = hmm.GMMHMM(n_components=2, covariance_type="full",n_iter=10,algorithm="
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, zero_division=1))


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:
[[ 1 40  0]
 [38 11  0]
 [35  0  0]]
-----------------------------------------------------
-----------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           1       0.01      0.02      0.02        41
           2       0.22      0.22      0.22        49
           3       1.00      0.00      0.00        35

    accuracy                           0.10       125
   macro avg       0.41      0.08      0.08       125
weighted avg       0.37      0.10      0.09       125

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

Accuracy:
0.096



WORKING WITH BREAST CANCER DATASET

Without and With Parameter
Tuning TABULATION

(CODE ALONGWITH OUTPUTS ATTACHED AT THE END
OF TABULATION)

CLASSIFIER	PARAMETER TUNING	TRAIN-TEST RATIO	PRECISION	RECALL	F1 SCORE	SUPPORT	ACCURACY
GAUSSIAN CLASSIFIER	No	70:30	0.94	0.96	0.95	171	**0.95**
	Yes		0.94	0.95	0.94	171	0.94
	No	60:40	0.92	0.93	0.92	228	0.92
	Yes		0.94	0.95	0.94	228	0.94
	No	50:50	0.93	0.94	0.93	285	0.93
	Yes		0.07	0.06	0.06	285	0.06
	No	40:60	0.85	0.84	0.84	342	0.86
	Yes		0.85	0.84	0.84	342	0.86
	No	30:70	0.91	0.91	0.91	399	0.91
	Yes		0.91	0.91	0.92	399	0.91

CLASSIFIER	PARAMETER TUNING	TRAIN-TEST RATIO	PRECISION	RECALL	F1 SCORE	SUPPORT	ACCURACY
GMM CLASSIFIER	No	70:30	0.92	0.93	0.92	171	**0.92**
	Yes		0.92	0.93	0.92	171	**0.92**
	No	60:40	0.91	0.91	0.91	228	0.91
	Yes		0.91	0.91	0.91	228	0.91
	No	50:50	0.91	0.92	0.91	285	0.91
	Yes		0.91	0.92	0.91	285	0.91
	No	40:60	0.89	0.91	0.90	342	0.90
	Yes		0.89	0.91	0.9	342	0.90
	No	30:70	0.90	0.91	0.90	399	0.90
	Yes		0.9	0.78	0.8	399	0.083

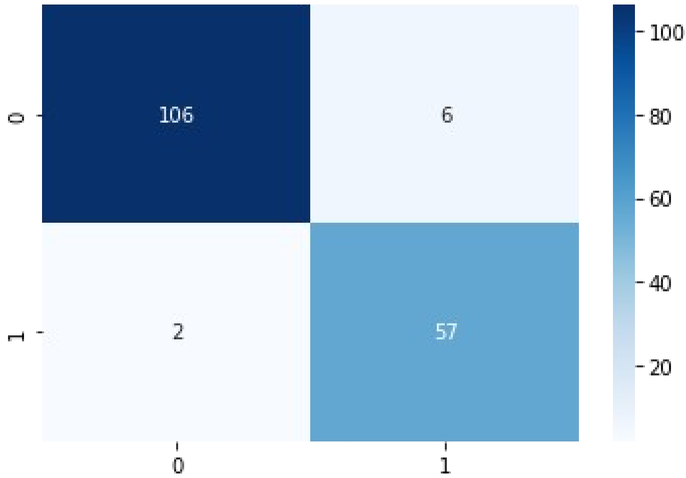
CLASSIFIER	PARAMETER TUNING	TRAIN-TEST RATIO	PRECISION	RECALL	F1 SCORE	SUPPORT	ACCURACY
MULTINOMIAL CLASSIFIER	No	70:30	0.51	0.51	0.51	171	0.57
	Yes		0.51	0.51	0.51	171	0.57
	No	60:40	0.54	0.54	0.54	228	**0.59**
	Yes		0.54	0.54	0.54	228	**0.59**
	No	50:50	0.54	0.54	0.54	285	0.57
	Yes		0.54	0.54	0.54	285	0.57
	No	40:60	0.53	0.53	0.53	342	0.58
	Yes		0.53	0.53	0.53	342	0.58
	No	30:70	0.54	0.54	0.54	399	0.57
	Yes		0.54	0.54	0.54	399	0.57

In [ ]:

```
# BREAST CANCER DATASET
# GaussianHMM(Without Tuning)[70-30 split]


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','1
            ,'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']

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
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
```



In [ ]:

```
# BREAST CANCER DATASET
# GaussianHMM(Without Tuning)[60-40  split]


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','1
,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
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: [[138
          11]
 [   5   74]]
-------------------------------------------------
```

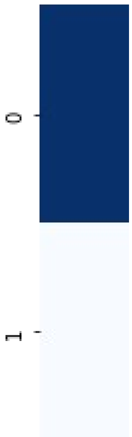
In [ ]:

```
# BREAST CANCER DATASET
# GaussianHMM(Without Tuning)[50-50 split]


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','1
'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm
```

Performance Evaluation

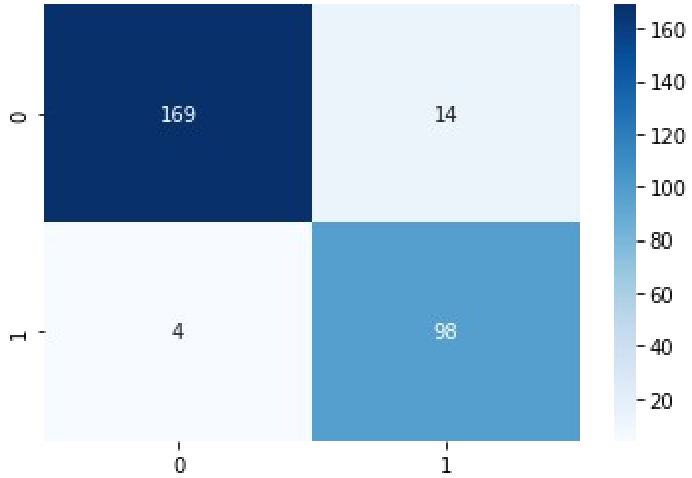
accuracy
macro avg
weighted

Accuracy : 0.92982456140350 88



```
classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",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: [[169
         14]
 [   4   98]]
------------------------------------------------------
------------------------------------------------------
Performance Evaluation
               precision    recall  f1-score   support

           B       0.98      0.92      0.95       183
           M       0.88      0.96      0.92       102

    accuracy                           0.94       285
   macro avg       0.93      0.94      0.93       285
weighted avg       0.94      0.94      0.94       285

------------------------------------------------------
------------------------------------------------------
Accuracy:
0.9368421052631579
```

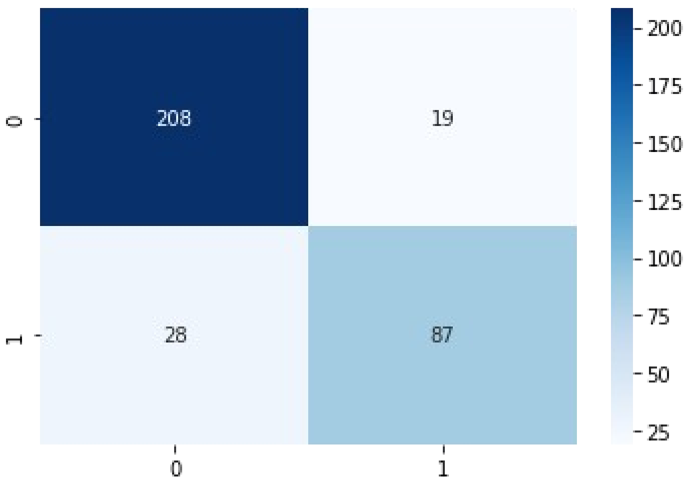


In [ ]:

```
# BREAST CANCER DATASET
# GaussianHMM(Without Tuning)[40-60 split]


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','1
           ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",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: [[208
        19]
 [ 28   87]]
-------------------------------------------------------
-------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.88      0.92      0.90       227
           M       0.82      0.76      0.79       115

    accuracy                           0.86       342
   macro avg       0.85      0.84      0.84       342
weighted  avg      0.86      0.86      0.86       342

-------------------------------------------------------
-------------------------------------------------------
Accuracy:
0.8625730994152047
```



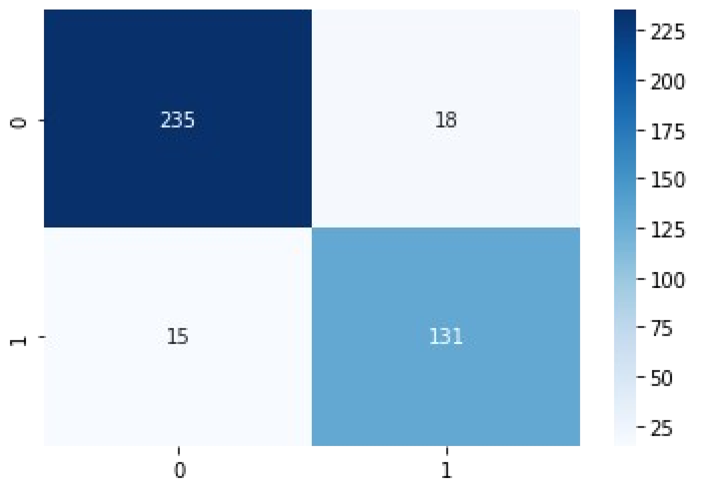
In [ ]:

```
# BREAST CANCER DATASET
```

```
# GaussianHMM(Without Tuning)[30-70 split]


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','1
            ,'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']

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",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: [[235
         18]
 [ 15  131]]
-----------------------------------------------
-----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.94      0.93      0.93       253
           M       0.88      0.90      0.89       146

    accuracy                           0.92       399
   macro avg       0.91      0.91      0.91       399
weighted  avg      0.92      0.92      0.92       399

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.9172932330827067
```



In [ ]:

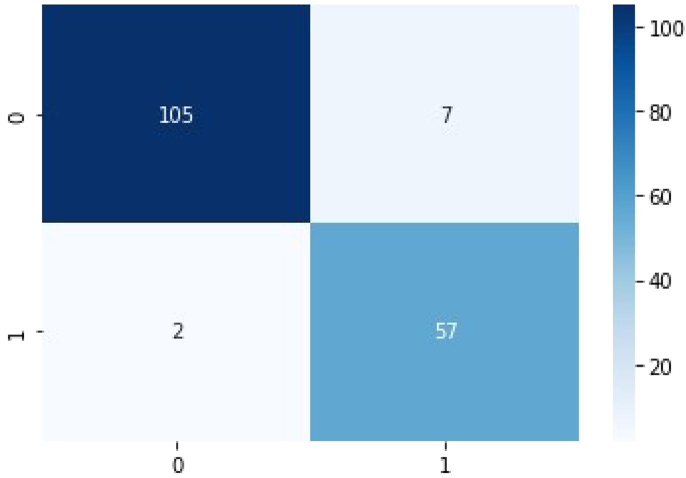
```
# BREAST CANCER DATASET
# GaussianHMM(With Tuning)[70-30 split]


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','1
,'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']
```

```
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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori
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]
 [   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

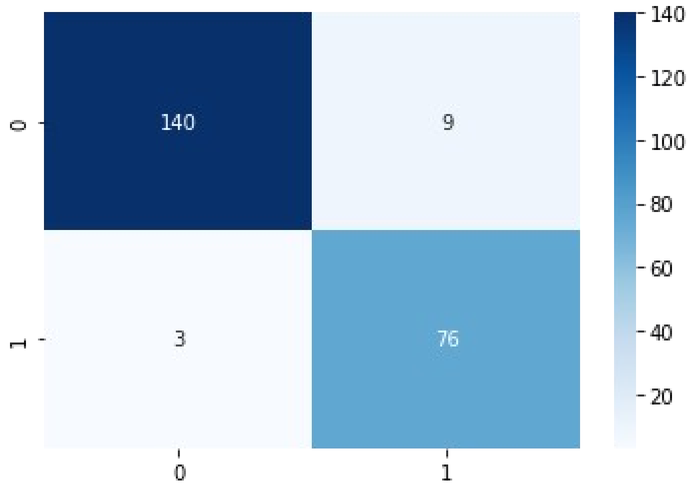


In [ ]:

```
# BREAST CANCER DATASET
# GaussianHMM(With Tuning)[60-40 split]


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','1
,'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']

from sklearn.model_selection import train_test_split

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,y,train_size=0.6,test_size=0.4


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier  =  hmm.GaussianHMM(n_components=2,  covariance_type="full",n_iter=10,algori 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: [[140
          9]
 [   3   76]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.98      0.94      0.96       149
           M       0.89      0.96      0.93        79

    accuracy                           0.95       228
   macro avg       0.94      0.95      0.94       228
weighted  avg      0.95      0.95      0.95       228

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

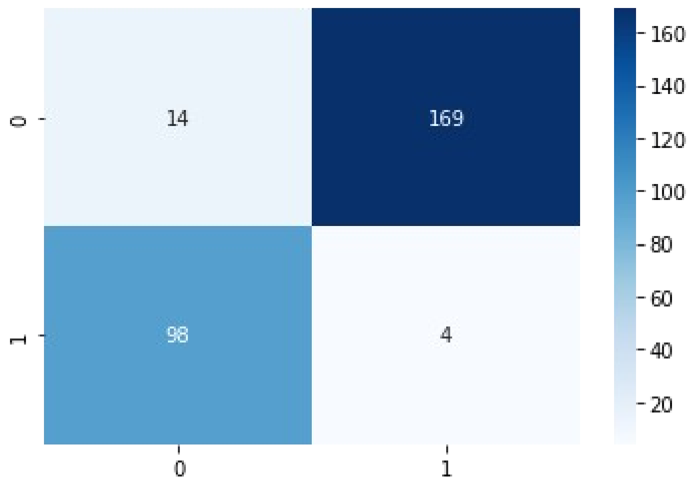


In [ ]:

```
# BREAST CANCER DATASET
# GaussianHMM(With Tuning)[50-50 split]


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','1
           ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori
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:
[[ 14 169]
 [ 98   4]]
------------------------------------------------------
------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.12      0.08      0.09       183
           M       0.02      0.04      0.03       102

    accuracy                           0.06       285
   macro avg       0.07      0.06      0.06       285
weighted avg       0.09      0.06      0.07       285

------------------------------------------------------
------------------------------------------------------
Accuracy:
0.06315789473684211
```

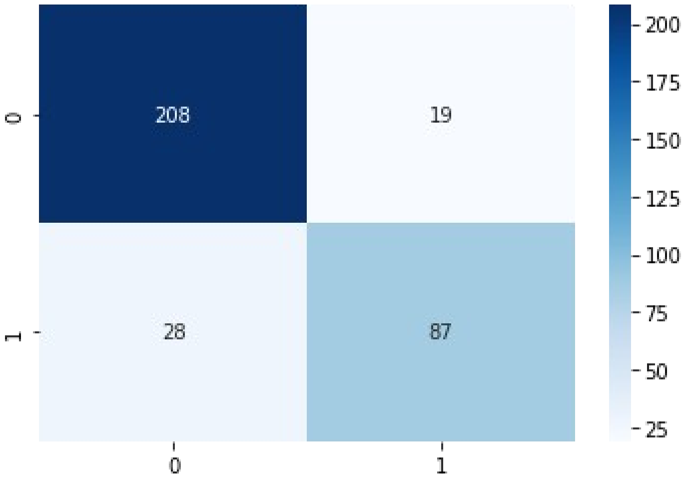


In [ ]:

```
# BREAST CANCER DATASET
# GaussianHMM(With Tuning)[40-60 split]


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','1
           ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori
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: [[208
          19]
 [ 28   87]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.88      0.92      0.90       227
           M       0.82      0.76      0.79       115

    accuracy                           0.86       342
   macro avg       0.85      0.84      0.84       342
weighted  avg       0.86      0.86      0.86       342

------------------------------------------------
------------------------------------------------
Accuracy:
0.8625730994152047
```



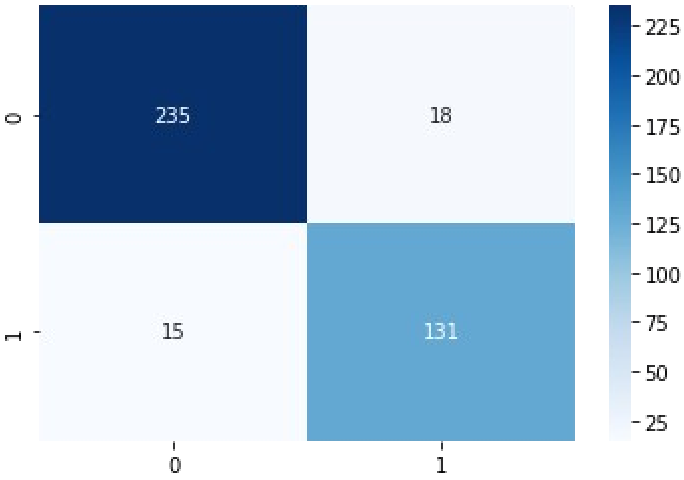
In [ ]:

```
# BREAST CANCER DATASET
# GaussianHMM(With Tuning)[30-70 split]


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','1
,'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']
```

```python
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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
from hmmlearn import hmm

classifier = hmm.GaussianHMM(n_components=2, covariance_type="full",n_iter=10,algori
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: [[235
         18]
 [ 15  131]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
```

```
              precision    recall  f1-score   support

           B       0.94      0.93      0.93       253
           M       0.88      0.90      0.89       146

    accuracy                           0.92       399
   macro avg       0.91      0.91      0.91       399
weighted avg       0.92      0.92      0.92       399

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.9172932330827067
```



In [ ]:

```
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
####################################################################################
```

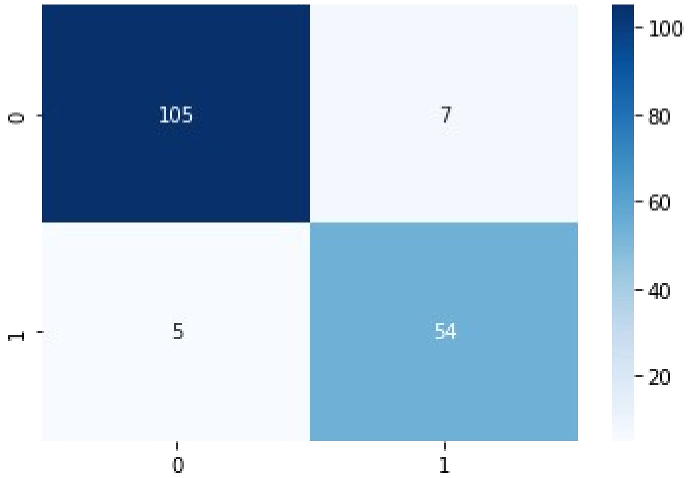
In [ ]:

```
# BREAST CANCER DATASET
# GMMHMM(Without Tuning)[70-30 split]


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','1
,'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']

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
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)
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
```

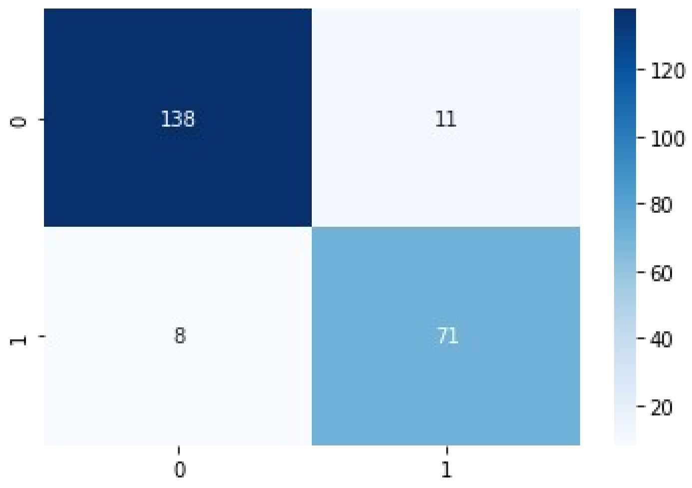


In [ ]:

```
# BREAST CANCER DATASET
# GMMHMM(Without Tuning)[60-40 split]


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','1
,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
```

```
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2, random_state=2) 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: [[138
        11]
 [   8   71]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.95      0.93      0.94       149
           M       0.87      0.90      0.88        79

    accuracy                           0.92       228
   macro avg       0.91      0.91      0.91       228
weighted avg       0.92      0.92      0.92       228

------------------------------------------------
------------------------------------------------
Accuracy:
0.9166666666666666
```

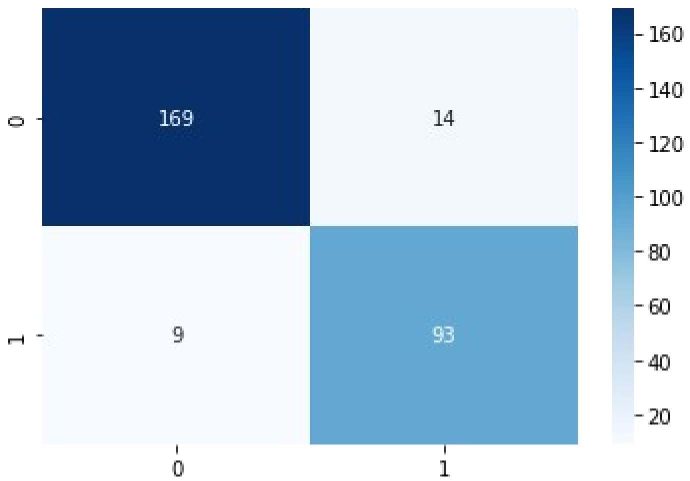


In [ ]:

```
# BREAST CANCER DATASET
# GMMHMM(Without Tuning)[50-50 split]


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','1
            ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
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)
sns.heatmap(cm, annot=True, fmt="d",cmap='Blues')
plt.show()
```

```
Confusion
Matrix: [[169
          14]
 [   9   93]]
--------------------------------------------------
--------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.95      0.92      0.94       183
           M       0.87      0.91      0.89       102

    accuracy                           0.92       285
   macro avg       0.91      0.92      0.91       285
weighted  avg       0.92      0.92      0.92       285

--------------------------------------------------
--------------------------------------------------
Accuracy:
0.9192982456140351
```



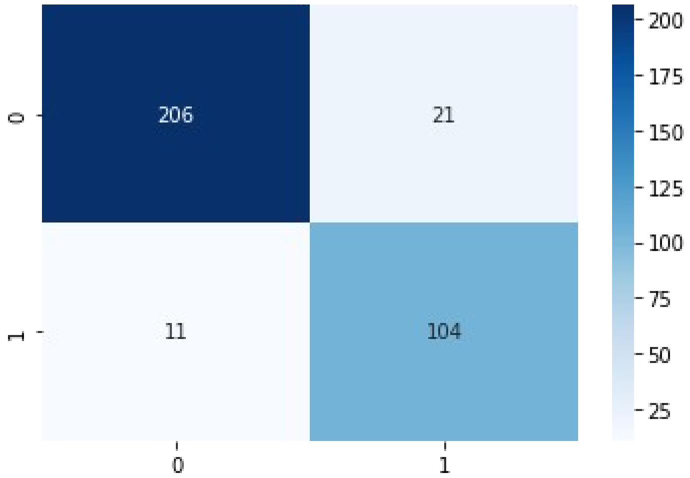
In [ ]:

```
# BREAST CANCER DATASET
```

```python
# GMMHMM(Without Tuning)[40-60 split]


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','1
            ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
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)
sns.heatmap(cm,  annot=True,  fmt="d",cmap='Blues') plt.show()
```

```
Confusion
Matrix: [[206
         21]
 [ 11  104]]
------------------------------------------------
------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.95      0.91      0.93       227
           M       0.83      0.90      0.87       115

    accuracy                           0.91       342
   macro avg       0.89      0.91      0.90       342
weighted  avg      0.91      0.91      0.91       342

------------------------------------------------
------------------------------------------------
Accuracy:
0.9064327485380117
```



In [ ]:

```
# BREAST CANCER DATASET
# GMMHMM(Without Tuning)[30-70 split]


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','1
,'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']
```

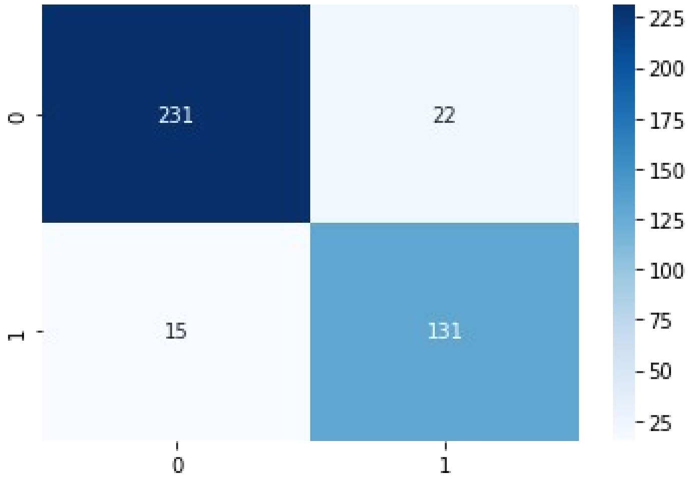
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.GMMHMM(n_components=2,  random_state=2)
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: [[231
          22]
 [ 15  131]]
---------------------------------------------------
---------------------------------------------------
Performance Evaluation
              precision    recall   f1-score    support
```

	precision	recall	f1-score	support
B	0.94	0.91	0.93	253
M	0.86	0.90	0.88	146
accuracy			0.91	399
macro avg	0.90	0.91	0.90	399
weighted avg	0.91	0.91	0.91	399

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

Accuracy:
0.9072681704260651

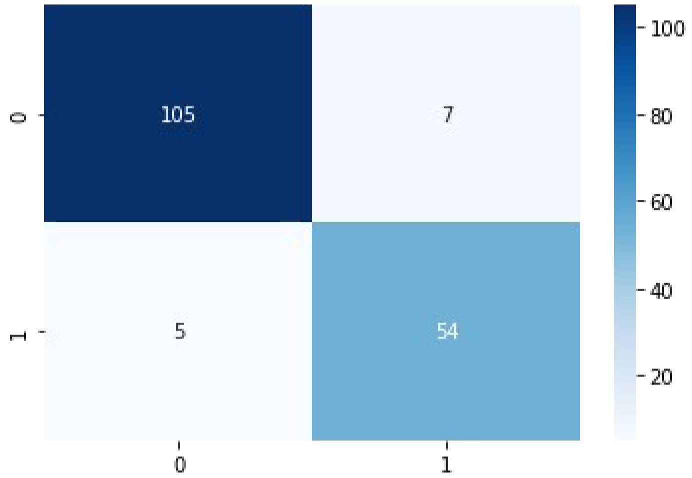


In [ ]:

```
# BREAST CANCER DATASET
# GMMHMM(With Tuning)[70-30 split]


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','1
,'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']

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2, random_state=10,covariance_type='di classifier.fit(X_trai
```

```
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
```

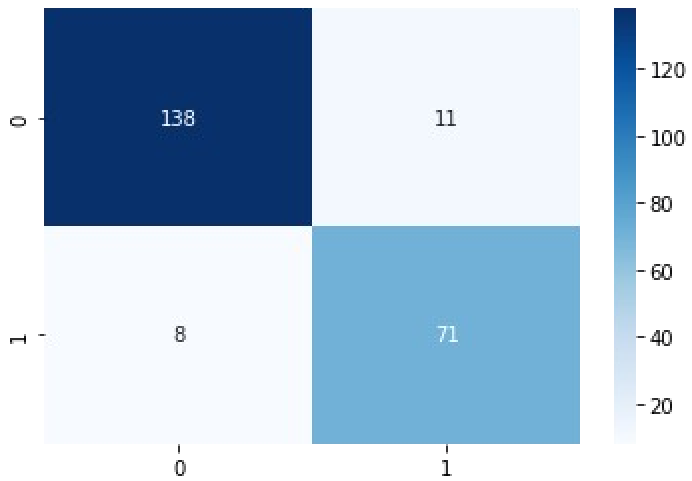


In [ ]:

```
# BREAST CANCER DATASET
# GMMHMM(With Tuning)[60-40 split]


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','1
           ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2, random_state=2,covariance_type='dia
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: [[138
          11]
 [  8  71]]
-----------------------------------------------------
-----------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.95      0.93      0.94       149
           M       0.87      0.90      0.88        79

    accuracy                           0.92       228
   macro avg       0.91      0.91      0.91       228
weighted  avg      0.92      0.92      0.92       228

-----------------------------------------------------
-----------------------------------------------------
Accuracy:
0.9166666666666666
```



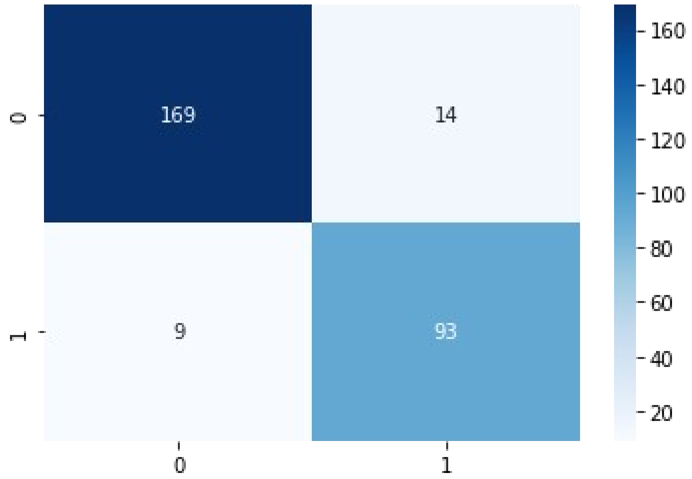
In [ ]:

BREAST CANCER DATASET

```python
# GMMHMM(With Tuning)[50-50 split]


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','1
           ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.GMMHMM(n_components=2,  random_state=10,covariance_type='di
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: [[169
          14]
 [   9   93]]
-----------------------------------------------
-----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.95      0.92      0.94       183
           M       0.87      0.91      0.89       102

    accuracy                           0.92       285
   macro avg       0.91      0.92      0.91       285
weighted avg       0.92      0.92      0.92       285

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.9192982456140351
```



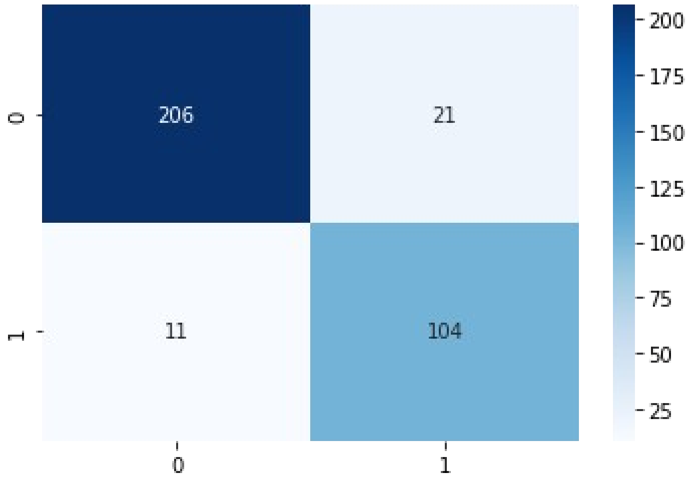
In [ ]:

```
# BREAST CANCER DATASET
# GMMHMM(With Tuning)[40-60 split]


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','1
,'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']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=2, random_state=10,covariance_type='di
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: [[206
         21]
 [ 11  104]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation
               precision    recall  f1-score   support
```

```
           B       0.95      0.91      0.93       227
           M       0.83      0.90      0.87       115

    accuracy                           0.91       342
   macro avg       0.89      0.91      0.90       342
weighted avg       0.91      0.91      0.91       342

-----------------------------------------------------
-----------------------------------------------------
Accuracy:
0.9064327485380117
```

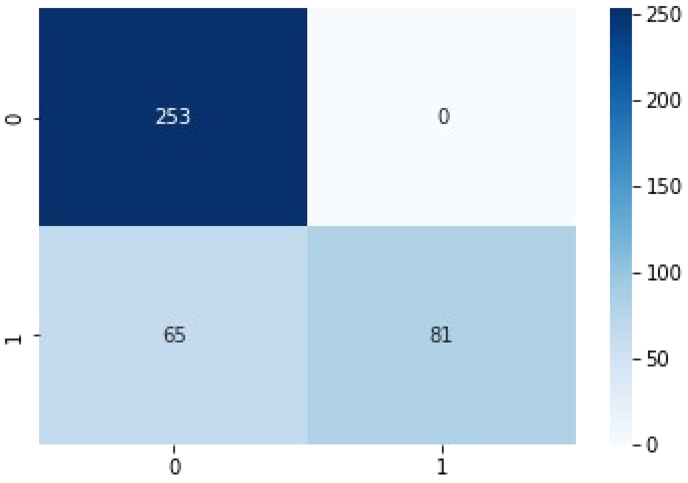


In [ ]:

```python
# BREAST CANCER DATASET
# GMMHMM(With Tuning)[30-70 split]


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','1
,'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']

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.GMMHMM(n_components=5, random_state=20,covariance_type='di classifier.fit(X_trai
```

```
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: [[253
           0]
 [ 65   81]]
------------------------------------------------------
------------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.80      1.00      0.89       253
           M       1.00      0.55      0.71       146

    accuracy                           0.84       399
   macro avg       0.90      0.78      0.80       399
weighted  avg      0.87      0.84      0.82       399

------------------------------------------------------
------------------------------------------------------
Accuracy:
0.8370927318295739
```



In [ ]:

```
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
##################################################################################################
```

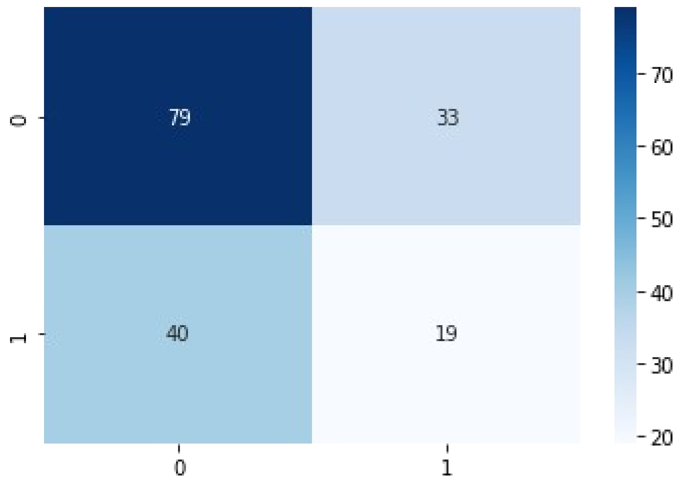
In [ ]:

```
# BREAST CANCER DATASET
# MultinomialHMM(With Tuning)[70-30 split]


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','1
,'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']

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

```
# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4, random_state=15,n_iter=10,a

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


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]


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
```



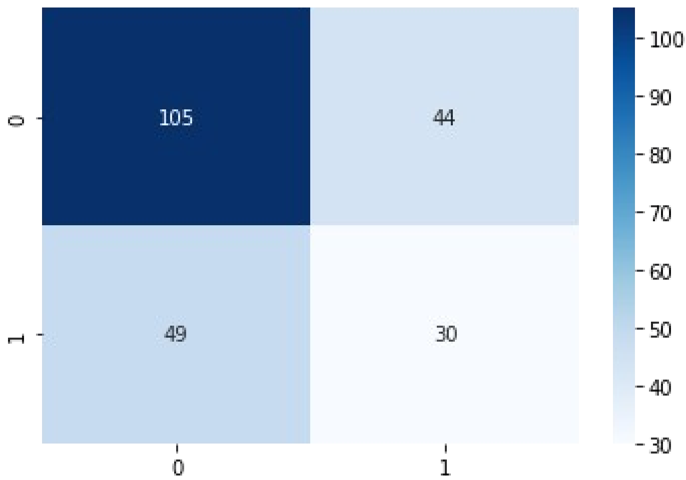
In [ ]:

```
# BREAST CANCER DATASET
# MultinomialHMM(With Tuning)[60-40 split]


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','1
,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4, random_state=15,n_iter=10,a

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


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[0:228]


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
         44]
 [ 49   30]]
-------------------------------------------------
-------------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.68      0.70      0.69       149
           M       0.41      0.38      0.39        79

    accuracy                           0.59       228
   macro avg       0.54      0.54      0.54       228
weighted  avg       0.59      0.59      0.59       228

-------------------------------------------------
-------------------------------------------------
Accuracy:
0.5921052631578947
```



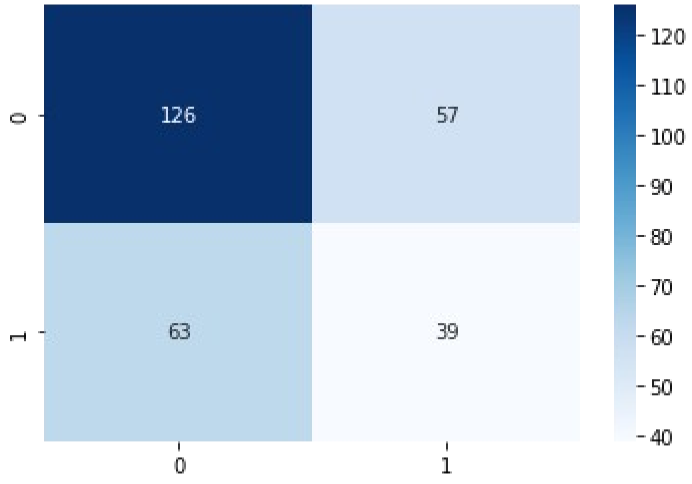
In [ ]:

BREAST CANCER DATASET

```
# MultinomialHMM(With Tuning)[50-50 split]


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','1
            ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.MultinomialHMM(n_components=4,  random_state=15,n_iter=10,a

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
```

```
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[0:285]




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: [[126
          57]
 [ 63   39]]
---------------------------------------------
---------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.67      0.69      0.68       183
           M       0.41      0.38      0.39       102

    accuracy                           0.58       285
   macro avg       0.54      0.54      0.54       285
weighted  avg       0.57      0.58      0.58       285

---------------------------------------------
---------------------------------------------
Accuracy:
0.5789473684210527
```



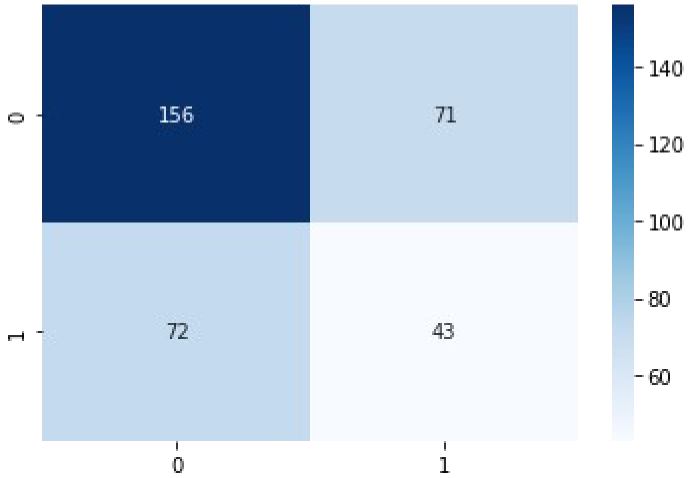
In [ ]:

```
# BREAST CANCER DATASET
# MultinomialHMM(With Tuning)[40-60 split]


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','1
            ,'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']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
import  hmmlearn
classifier  =  hmmlearn.hmm.MultinomialHMM(n_components=4,  random_state=15,n_iter=10,a

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
```

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

strings
strings = strings[0:342]


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: [[156
         71]
 [ 72   43]]
```

```
 ----------------------------------------------
----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.68      0.69      0.69       227
           M       0.38      0.37      0.38       115

    accuracy                           0.58       342
   macro avg       0.53      0.53      0.53       342
weighted avg       0.58      0.58      0.58       342

----------------------------------------------
----------------------------------------------
Accuracy:
0.5818713450292398
```



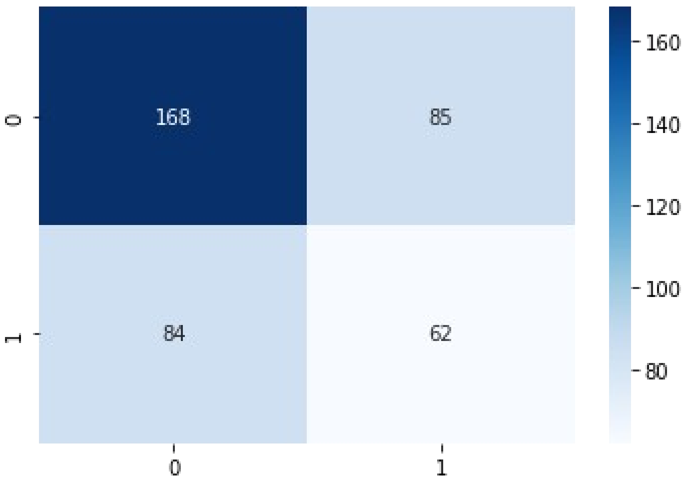
In [ ]:

```python
# BREAST CANCER DATASET
# MultinomialHMM(With Tuning)[30-70 split]


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','1
,'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']

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


# Feature Scaling
from sklearn.preprocessing import StandardScaler

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

# Classification
# from hmmlearn import hmm
```

```
import hmmlearn
classifier = hmmlearn.hmm.MultinomialHMM(n_components=4, random_state=15,n_iter=10,a

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


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[0:399]


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: [[168
        85]
 [ 84   62]]
-----------------------------------------------
-----------------------------------------------
Performance Evaluation
              precision    recall  f1-score   support

           B       0.67      0.66      0.67       253
           M       0.42      0.42      0.42       146

    accuracy                           0.58       399
   macro avg       0.54      0.54      0.54       399
weighted  avg      0.58      0.58      0.58       399

-----------------------------------------------
-----------------------------------------------
Accuracy:
0.5764411027568922
```



In [ ]:

```
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [13]:

In [ ]:

QUESTION 2

Construct a Deep Learning model using Convolutional Neural Network (CNN) for classification on the following four standard datasets:

1. CIFAR-10
2. MNIST
3. SAVEE
4. EmoDB

PERFORMANCE COMPARISION OF CONVOLUTIONAL NEURAL NETWORKS (CNN)

Dataset	Accuracy
CIFAR-10	0.70
MNIST	0.991
SAVEE	0.315
EmoDB	0.49

CODE AND OUTPUT ATTACHED BELOW

APPLYING **CNN** ON

CIFAR – 10 DATASET

IMPORT STATEMENTS AND DATASET

In [3]:

```
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

(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
```

In [4]:

```
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_1"
_________________________________________________________________
Layer  (type)                Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                65600
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

In [5]:

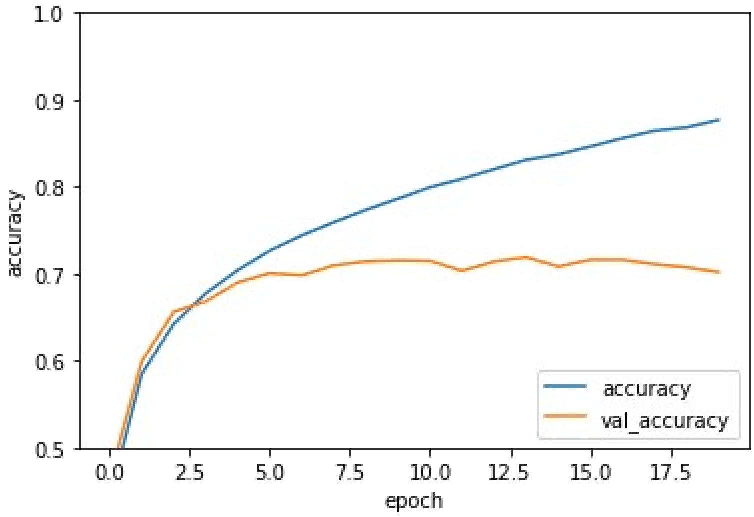
```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(fr

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

```
Epoch 1/20
1563/1563 [==============================] - 67s 42ms/step - loss: 1.5309 - accurac
```

```
y: 0.4417 - val_loss: 1.5367 - val_accuracy: 0.4646
Epoch 2/20
1563/1563 [==============================] - 65s 42ms/step - loss: 1.1710 - accurac
y: 0.5843 - val_loss: 1.1224 - val_accuracy: 0.5990
Epoch 3/20
1563/1563 [==============================] - 65s 42ms/step - loss: 1.0165 - accurac
y: 0.6421 - val_loss: 0.9825 - val_accuracy: 0.6560
Epoch 4/20
1563/1563 [==============================] - 65s 42ms/step - loss: 0.9195 - accurac
y: 0.6773 - val_loss: 0.9559 - val_accuracy: 0.6682
Epoch 5/20
1563/1563 [==============================] - 65s 41ms/step - loss: 0.8439 - accurac
y: 0.7041 - val_loss: 0.9026 - val_accuracy: 0.6897
Epoch 6/20
1563/1563 [==============================] - 65s 42ms/step - loss: 0.7828 - accurac
y: 0.7266 - val_loss: 0.8719 - val_accuracy: 0.7005
Epoch 7/20
1563/1563 [==============================] - 68s 44ms/step - loss: 0.7337 - accurac
y: 0.7441 - val_loss: 0.8932 - val_accuracy: 0.6980
Epoch 8/20
1563/1563 [==============================] - 68s 43ms/step - loss: 0.6880 - accurac
y: 0.7592 - val_loss: 0.8497 - val_accuracy: 0.7094
Epoch 9/20
1563/1563 [==============================] - 68s 44ms/step - loss: 0.6457 - accurac
y: 0.7734 - val_loss: 0.8456 - val_accuracy: 0.7140
Epoch 10/20
1563/1563 [==============================] - 67s 43ms/step - loss: 0.6068 - accurac
y: 0.7861 - val_loss: 0.8751 - val_accuracy: 0.7153
Epoch 11/20
1563/1563 [==============================] - 68s 44ms/step - loss: 0.5675 - accurac
y: 0.7993 - val_loss: 0.8580 - val_accuracy: 0.7147
Epoch 12/20
1563/1563 [==============================] - 68s 43ms/step - loss: 0.5410 - accurac
y: 0.8089 - val_loss: 0.9303 - val_accuracy: 0.7033
Epoch 13/20
1563/1563 [==============================] - 67s 43ms/step - loss: 0.5117 - accurac
y: 0.8200 - val_loss: 0.9169 - val_accuracy: 0.7139
Epoch 14/20
1563/1563 [==============================] - 67s 43ms/step - loss: 0.4802 - accurac
y: 0.8308 - val_loss: 0.9111 - val_accuracy: 0.7189
Epoch 15/20
1563/1563 [==============================] - 67s 43ms/step - loss: 0.4572 - accurac
y: 0.8371 - val_loss: 0.9975 - val_accuracy: 0.7082
Epoch 16/20
1563/1563 [==============================] - 69s 44ms/step - loss: 0.4299 - accurac
y: 0.8461 - val_loss: 0.9821 - val_accuracy: 0.7159
Epoch 17/20
1563/1563 [==============================] - 67s 43ms/step - loss: 0.4087 - accurac
y: 0.8557 - val_loss: 1.0078 - val_accuracy: 0.7159
Epoch 18/20
1563/1563 [==============================] - 68s 43ms/step - loss: 0.3813 - accurac
y: 0.8643 - val_loss: 1.0568 - val_accuracy: 0.7109
Epoch 19/20
1563/1563 [==============================] - 67s 43ms/step - loss: 0.3690 - accurac
y: 0.8682 - val_loss: 1.1018 - val_accuracy: 0.7073
Epoch 20/20
1563/1563 [==============================] - 67s 43ms/step - loss: 0.3479 - accurac
y: 0.8765 - val_loss: 1.1768 - val_accuracy: 0.7017
```

In [6]:

```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()
```



In [7]:

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

```
313/313 - 3s - loss: 1.1768 - accuracy: 0.7017
```

APPLYING **CNN**

ON

MNIST DATASET

IMPORT STATEMENTS AND DATASET

In [ ]:

```
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

(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
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
ist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

In [ ]:

```
train_images = np.reshape(train_images, train_images.shape + (1,))
test_images = np.reshape(test_images, test_images.shape + (1,))

train_images[0].shape
```

Out[ ]:

```
(28, 28, 1)
```

In [ ]:

```
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"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 10)	650

===

Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

In [ ]:

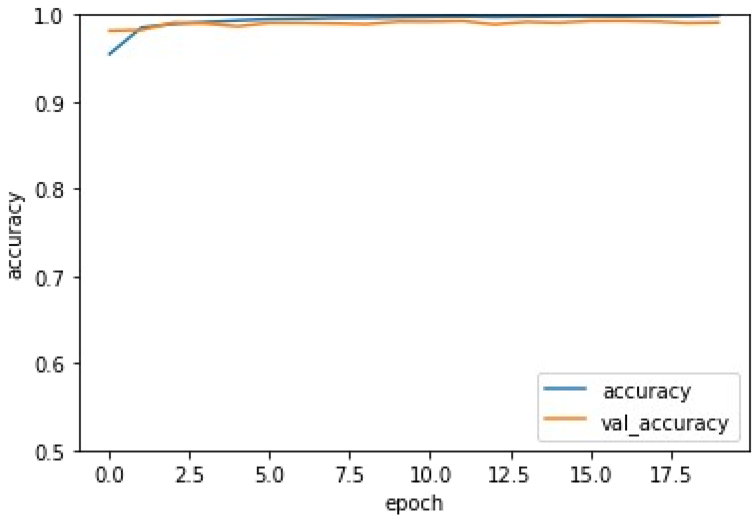
```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(fr

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

```
Epoch 1/20
1875/1875 [==============================] - 61s 32ms/step - loss: 0.1446 - accurac
y: 0.9548 - val_loss: 0.0558 - val_accuracy: 0.9816
Epoch 2/20
1875/1875 [==============================] - 59s 32ms/step - loss: 0.0474 - accurac
y: 0.9853 - val_loss: 0.0568 - val_accuracy: 0.9824
Epoch 3/20
1875/1875 [==============================] - 59s 32ms/step - loss: 0.0336 - accurac
y: 0.9894 - val_loss: 0.0249 - val_accuracy: 0.9910
Epoch 4/20
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0270 - accurac
y: 0.9915 - val_loss: 0.0326 - val_accuracy: 0.9899
Epoch 5/20
1875/1875 [==============================] - 59s 32ms/step - loss: 0.0207 - accurac
y: 0.9934 - val_loss: 0.0429 - val_accuracy: 0.9867
Epoch 6/20
1875/1875 [==============================] - 59s 32ms/step - loss: 0.0166 - accurac
y: 0.9949 - val_loss: 0.0337 - val_accuracy: 0.9907
Epoch 7/20
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0141 - accurac
y: 0.9955 - val_loss: 0.0337 - val_accuracy: 0.9907
Epoch 8/20
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0117 - accurac
y: 0.9963 - val_loss: 0.0369 - val_accuracy: 0.9901
Epoch 9/20
1875/1875 [==============================] - 60s 32ms/step - loss: 0.0106 - accurac
y: 0.9965 - val_loss: 0.0481 - val_accuracy: 0.9892
Epoch 10/20
1875/1875 [==============================] - 60s 32ms/step - loss: 0.0091 - accurac
y: 0.9971 - val_loss: 0.0344 - val_accuracy: 0.9922
Epoch 11/20
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0077 - accurac
y: 0.9974 - val_loss: 0.0354 - val_accuracy: 0.9921
Epoch 12/20
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0067 - accurac
y: 0.9980 - val_loss: 0.0306 - val_accuracy: 0.9929
Epoch 13/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0075 - accurac
y: 0.9974 - val_loss: 0.0503 - val_accuracy: 0.9889
Epoch 14/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0065 - accurac
y: 0.9978 - val_loss: 0.0420 - val_accuracy: 0.9918
Epoch 15/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0055 - accurac
y: 0.9981 - val_loss: 0.0451 - val_accuracy: 0.9908
Epoch 16/20
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0060 - accurac
y: 0.9981 - val_loss: 0.0389 - val_accuracy: 0.9928
Epoch 17/20
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0062 - accurac
y: 0.9980 - val_loss: 0.0411 - val_accuracy: 0.9930
Epoch 18/20
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0051 - accurac
y: 0.9984 - val_loss: 0.0383 - val_accuracy: 0.9923
Epoch 19/20
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0058 - accurac
y: 0.9982 - val_loss: 0.0441 - val_accuracy: 0.9906
Epoch 20/20
```

1875/1875 [==============================] - 60s 32ms/step - loss: 0.0033 - accuracy: 0.9989 - val_loss: 0.0504 - val_accuracy: 0.9911

In [ ]:

```
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()
```



In [ ]:

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

313/313 - 3s - loss: 0.0504 - accuracy: 0.9911

APPLYING **CNN**

ON

SAVEE DATASET

In [ ]:

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

In [ ]:

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

In [2]:

```
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_mel
    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
```

In [8]:

```
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.s
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]))
```

In [9]:

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

In [10]:

```
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=
```

In [11]:

```
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"

_________________________________________________________________
Layer  (type)                 Output Shape              Param  #
=================================================================
conv2d (Conv2D)              (None, 155,  318,  32)     320
_________________________________________________________________
max_pooling2d  (MaxPooling2D) (None,  77,  159,  32)     0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75,  157,  64)      18496
_________________________________________________________________
max_pooling2d_1  (MaxPooling2  (None,  37,  78,  64)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 35,  76,  64)       36928
_________________________________________________________________
flatten   (Flatten)          (None, 170240)             0
_________________________________________________________________
dense  (Dense)               (None,  64)                10895424
_________________________________________________________________
dense_1  (Dense)             (None,  10)                650
=================================================================
Total params: 10,951,818
Trainable params: 10,951,818
```

Non-trainable params: 0

In [12]:

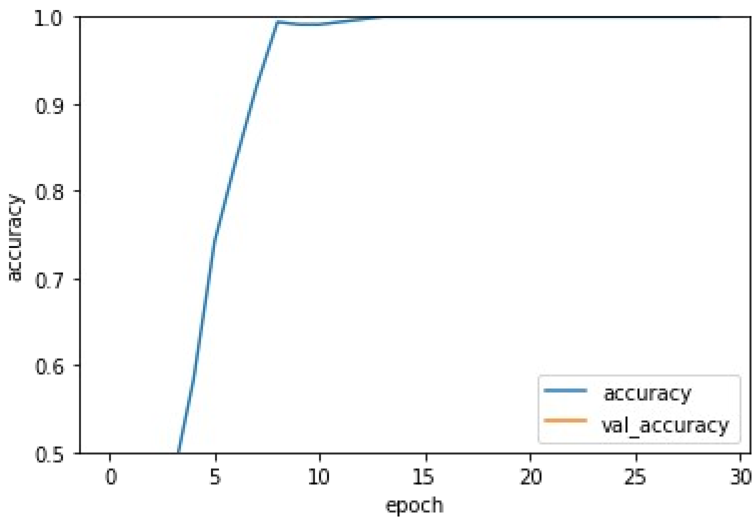
```python
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(fr

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

```
Epoch 1/30
11/11 [==============================] - 28s        2s/step - loss: 3.1605 - accuracy: 0.18
45 - val_loss: 2.2921 - val_accuracy: 0.1528 Epoch 2/30
11/11 [==============================] - 26s
08 - val_loss: 2.1275 - val_accuracy: 0.1528     2s/step - loss: 2.0202 - accuracy: 0.27
Epoch 3/30
11/11 [==============================] - 26s
36 - val_loss: 1.9424 - val_accuracy: 0.2431     2s/step - loss: 1.8117 - accuracy: 0.30
Epoch 4/30
11/11 [==============================] - 26s
73 - val_loss: 1.9249 - val_accuracy: 0.2500     2s/step - loss: 1.4582 - accuracy: 0.46
Epoch 5/30
11/11 [==============================] - 26s
33 - val_loss: 1.8715 - val_accuracy: 0.3194     2s/step - loss: 1.1255 - accuracy: 0.58
Epoch 6/30
11/11 [==============================] - 26s
11 - val_loss: 1.9895 - val_accuracy: 0.3542     2s/step - loss: 0.7514 - accuracy: 0.74
Epoch 7/30
11/11 [==============================] - 26s
33 - val_loss: 2.2782 - val_accuracy: 0.2986     2s/step - loss: 0.4764 - accuracy: 0.83
Epoch 8/30
11/11 [==============================] - 26s
96 - val_loss: 2.7056 - val_accuracy: 0.3750     2s/step - loss: 0.2938 - accuracy: 0.91
Epoch 9/30
11/11 [==============================] - 26s
40 - val_loss: 2.9734 - val_accuracy: 0.3542     2s/step - loss: 0.1100 - accuracy: 0.99
Epoch 10/30
11/11 [==============================] - 26s
11 - val_loss: 3.7372 - val_accuracy: 0.3125     2s/step - loss: 0.0435 - accuracy: 0.99
Epoch 11/30
11/11 [==============================] - 26s
11 - val_loss: 3.8469 - val_accuracy: 0.4028     2s/step - loss: 0.0382 - accuracy: 0.99
Epoch 12/30
11/11 [==============================] - 26s
40 - val_loss: 3.9630 - val_accuracy: 0.3611     2s/step - loss: 0.0193 - accuracy: 0.99
Epoch 13/30
11/11 [==============================] - 26s
70 - val_loss: 4.4897 - val_accuracy: 0.3194     2s/step - loss: 0.0088 - accuracy: 0.99
Epoch 14/30
11/11 [==============================] - 26s
00 - val_loss: 4.5158 - val_accuracy: 0.3403     2s/step - loss: 0.0028 - accuracy: 1.00
Epoch 15/30
11/11 [==============================] - 26s
00 - val_loss: 4.6630 - val_accuracy: 0.3403     2s/step - loss: 0.0018 - accuracy: 1.00
Epoch 16/30
11/11 [==============================] - 26s 2s/step - loss: 8.1752e-04 - accuracy:
1.0000 - val_loss: 4.7943 - val_accuracy: 0.3403
Epoch 17/30
11/11 [==============================] - 26s 2s/step - loss: 6.2663e-04 - accuracy:
1.0000 - val_loss: 4.9100 - val_accuracy: 0.3472
Epoch 18/30
11/11 [==============================] - 26s 2s/step - loss: 5.0990e-04 - accuracy:
1.0000 - val_loss: 4.9722 - val_accuracy: 0.3472
Epoch 19/30
11/11 [==============================] - 26s 2s/step - loss: 4.2045e-04 - accuracy:
1.0000 - val_loss: 5.0247 - val_accuracy: 0.3472
Epoch 20/30
11/11 [==============================] - 26s 2s/step - loss: 3.6234e-04 - accuracy:
1.0000 - val_loss: 5.0764 - val_accuracy: 0.3403
Epoch 21/30
```

```
11/11 [==============================] - 26s 2s/step - loss: 3.1751e-04 - accuracy: 1.0000 - val_loss: 5.1274 - val_accuracy: 0.3403
Epoch 22/30
11/11 [==============================] - 26s 2s/step - loss: 2.8307e-04 - accuracy: 1.0000 - val_loss: 5.1800 - val_accuracy: 0.3403
Epoch 23/30
11/11 [==============================] - 26s 2s/step - loss: 2.5461e-04 - accuracy: 1.0000 - val_loss: 5.2211 - val_accuracy: 0.3333
Epoch 24/30
11/11 [==============================] - 26s 2s/step - loss: 2.3146e-04 - accuracy: 1.0000 - val_loss: 5.2606 - val_accuracy: 0.3264
Epoch 25/30
11/11 [==============================] - 26s 2s/step - loss: 2.1190e-04 - accuracy: 1.0000 - val_loss: 5.2975 - val_accuracy: 0.3264
Epoch 26/30
11/11 [==============================] - 26s 2s/step - loss: 1.9411e-04 - accuracy: 1.0000 - val_loss: 5.3341 - val_accuracy: 0.3264
Epoch 27/30
11/11 [==============================] - 26s 2s/step - loss: 1.7741e-04 - accuracy: 1.0000 - val_loss: 5.3717 - val_accuracy: 0.3194
Epoch 28/30
11/11 [==============================] - 26s 2s/step - loss: 1.6377e-04 - accuracy: 1.0000 - val_loss: 5.4069 - val_accuracy: 0.3125
Epoch 29/30
11/11 [==============================] - 26s 2s/step - loss: 1.5132e-04 - accuracy: 1.0000 - val_loss: 5.4432 - val_accuracy: 0.3125
Epoch 30/30
11/11 [==============================] - 26s 2s/step - loss: 1.3950e-04 - accuracy: 1.0000 - val_loss: 5.4739 - val_accuracy: 0.3125
```

In [13]:

```
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()
```



In [14]:

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

```
5/5 - 3s - loss: 5.4739 - accuracy: 0.3125
```

APPLYING **CNN**

ON

EmoDB DATASET

In [4]:

```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_mel
    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
```

In [5]:

```python
# 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]))
```

In [6]:
```
X = np.asarray(X, dtype=np.float32) y = np.asarray(y, dtype=np.float32)
```

In [7]:
```
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=
```

In [8]:
```
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"

Layer  (type)                Output Shape             Param #
=================================================================
conv2d (Conv2D)              (None, 155, 318, 32)     320

max_pooling2d (MaxPooling2D) (None, 77, 159, 32)      0

conv2d_1 (Conv2D)            (None, 75, 157, 64)      18496

max_pooling2d_1 (MaxPooling2 (None, 37, 78, 64)       0

conv2d_2 (Conv2D)            (None, 35, 76, 64)       36928

flatten  (Flatten)           (None, 170240)           0

dense (Dense)                (None, 64)               10895424

dense_1 (Dense)              (None, 10)               650
=================================================================
Total params: 10,951,818
Trainable params: 10,951,818
Non-trainable params: 0
```

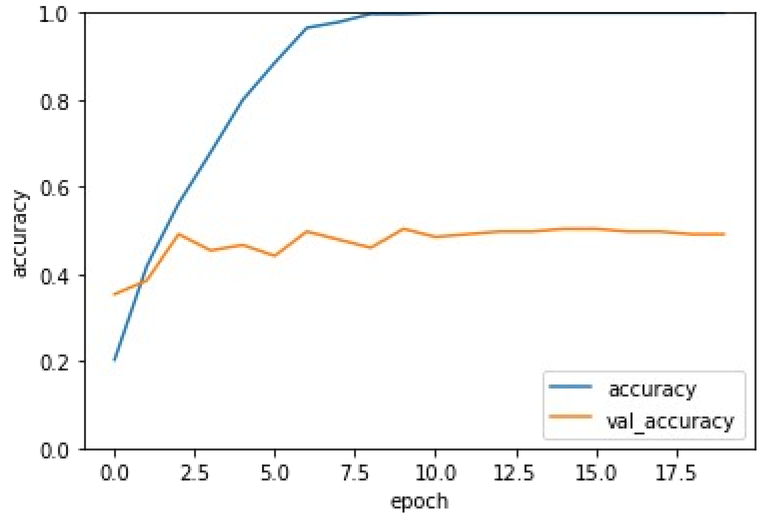
In [9]:
```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(fr

history = model.fit(X_train,y_train,epochs=20,validation_data=(X_test,y_test))
```

```
Epoch 1/20
12/12 [==============================] - 31s 3s/step - loss: 3.3983 - accuracy:
0.20
32 - val_loss: 1.7978 - val_accuracy: 0.3540
Epoch 2/20
12/12 [==============================] - 30s 3s/step - loss: 1.5264 - accuracy:
0.41
71 - val_loss: 1.4124 - val_accuracy: 0.3851
```

```
Epoch 3/20
12/12 [==============================] - 29s 2s/step - loss: 1.1416 - accuracy: 0.56
15 - val_loss: 1.3522 - val_accuracy: 0.4907 Epoch
4/20
12/12 [==============================] - 29s 2s/step - loss: 0.8115 - accuracy: 0.67
91 - val_loss: 1.3487 - val_accuracy: 0.4534
Epoch 5/20
12/12 [==============================] - 29s 2s/step - loss: 0.5488 - accuracy: 0.79
95 - val_loss: 1.7667 - val_accuracy: 0.4658
Epoch 6/20
12/12 [==============================] - 29s 2s/step - loss: 0.3237 - accuracy: 0.88
50 - val_loss: 1.9519 - val_accuracy: 0.4410
Epoch 7/20
12/12 [==============================] - 29s 2s/step - loss: 0.1483 - accuracy: 0.96
52 - val_loss: 2.2482 - val_accuracy: 0.4969
Epoch 8/20
12/12 [==============================] - 29s 2s/step - loss: 0.0647 - accuracy: 0.97
86 - val_loss: 2.6644 - val_accuracy: 0.4783
Epoch 9/20
12/12 [==============================] - 29s 2s/step - loss: 0.0223 - accuracy: 0.99
73 - val_loss: 2.9553 - val_accuracy: 0.4596
Epoch 10/20
12/12 [==============================] - 29s 2s/step - loss: 0.0090 - accuracy: 0.99
73 - val_loss: 3.3734 - val_accuracy: 0.5031
Epoch 11/20
12/12 [==============================] - 29s 2s/step - loss: 0.0031 - accuracy: 1.00
00 - val_loss: 3.7336 - val_accuracy: 0.4845
Epoch 12/20
12/12 [==============================] - 29s 2s/step - loss: 0.0016 - accuracy: 1.00
00 - val_loss: 3.7510 - val_accuracy: 0.4907
Epoch 13/20
12/12 [==============================] - 29s 2s/step - loss: 9.8465e-04 - accuracy:
1.0000 - val_loss: 3.7821 - val_accuracy: 0.4969
Epoch 14/20
12/12 [==============================] - 29s 2s/step - loss: 6.7053e-04 - accuracy:
1.0000 - val_loss: 3.8717 - val_accuracy: 0.4969
Epoch 15/20
12/12 [==============================] - 29s 2s/step - loss: 5.1129e-04 - accuracy:
1.0000 - val_loss: 3.9343 - val_accuracy: 0.5031
Epoch 16/20
12/12 [==============================] - 29s 2s/step - loss: 4.0118e-04 - accuracy:
1.0000 - val_loss: 4.0046 - val_accuracy: 0.5031
Epoch 17/20
12/12 [==============================] - 29s 2s/step - loss: 3.6354e-04 - accuracy:
1.0000 - val_loss: 4.0746 - val_accuracy: 0.4969
Epoch 18/20
12/12 [==============================] - 29s 2s/step - loss: 2.9631e-04 - accuracy:
1.0000 - val_loss: 4.1104 - val_accuracy: 0.4969
Epoch 19/20
12/12 [==============================] - 29s 2s/step - loss: 2.5358e-04 - accuracy:
1.0000 - val_loss: 4.1465 - val_accuracy: 0.4907
Epoch 20/20
12/12 [==============================] - 29s 2s/step - loss: 2.2702e-04 - accuracy:
1.0000 - val_loss: 4.1814 - val_accuracy: 0.4907
```

In [10]:

```
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()
```



In [11]:

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

6/6 - 3s - loss: 4.1814 - accuracy: 0.4907

QUESTION 3

Experiment with the following Deep Learning models on the above the four datasets and show the performance comparison among the models along with that of CNN:

1. **VGG-16**
2. **ResNet-50**
3. **Recurrent Neural Networks (RNN)**
4. **AlexNet**
5. **GoogLeNet**

Apply different values of train-test set splits and report the corresponding results for the Deep Learning models.

Generate the image (heat map) of the confusion matrix for the best case of every Deep Learning

model. Also, generate the images of training & loss generation curves. For each dataset, generate

an image illustrating **Receiver Operating Characteristic (ROC) curve and Area Under Curve (AUC)** for the best case of every Deep Learning model only.

Try to achieve accuracy >=80%.

Show the performance comparison among Deep Learning models in a table along with a detailed discussion.

PERFORMANCE COMPARISION OF DEEP LEARNING MODELS

Models	Dataset	Accuracy
VGG-16	CIFAR-10	**9.8**
	MNIST	10.95
	SAVEE	12.92
	EmoDB	25
ResNet-50	CIFAR-10	27
	MNIST	**99**
	SAVEE	**99**
	EmoDB	**92**
Recurrent Neural Networks (RNN)	CIFAR-10	29
	MNIST	97
	SAVEE	43
	EmoDB	55
AlexNet	CIFAR-10	7.5
	MNIST	11.69
	SAVEE	23.74
	EmoDB	23.36
GoogLeNet	CIFAR-10	26.6
	MNIST	**99**
	SAVEE	38
	EmoDB	36

CODE AND OUTPUT ATTACHED BELOW

CONCLUSIONS

1. CIFAR-10 has maximum accuracy in VGG-16 DL model.
2. MNIST has maximum accuracy in ResNet-50 and GoogleNet DL model.
3. SAVEE has maximum accuracy in ResNet-50 DL model.
4. EmoDB has maximum accuracy in ResNet-50 DL model.
5. We can conclude from this that out of all the models here, the RESNET- 50 model consistently provides good accuracy.

VGG-16

In [11]:

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

Mounted at /content/drive

In [1]:

```
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
```

```
Requirement already satisfied: keras_applications in /usr/local/lib/python3.7/dist-p
ackages (1.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from
keras_applications) (3.1.0)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-package
s (from keras_applications) (1.19.5)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-pack
ages (from h5py->keras_applications) (1.5.2)
```

In [2]:

```
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
```

In [3]:

```
def preprocess_data(X_train):

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

return X_train
```

In [4]:

```
WEIGHTS_PATH = 'https://github.com/fchollet/deep-learning-models/releases/download/v
WEIGHTS_PATH_NO_TOP = 'https://github.com/fchollet/deep-learning-models/releases/dow


def VGG16(include_top=True, weights='imagenet',
          input_tensor=None, input_shape=None,
          pooling=None,
          classes=1000):
    """Instantiates the VGG16 architecture.

    Optionally loads weights pre-trained
    on ImageNet. Note that when using TensorFlow,
    for best performance you should set
    `image_data_format="channels_last"` in your Keras
    config at ~/.keras/keras.json.

    The model and the weights are compatible with both
    TensorFlow and Theano. The data format
    convention used by the model is the
    one specified in your Keras config
    file.

    # Arguments
        include_top: whether to include the 3 fully-connected
            layers at the top of the network.
        weights: one of `None` (random initialization)
            or "imagenet" (pre-training on ImageNet).
        input_tensor: optional Keras tensor (i.e. output of `layers.Input()`)
            to use as image input for the model.
        input_shape: optional shape tuple, only to be specified
            if `include_top` is False (otherwise the input shape
            has to be `(224, 224, 3)` (with `channels_last` data format)
            or `(3, 224, 244)` (with `channels_first` data format).
            It should have exactly 3 inputs channels,
            and width and height should be no smaller than 48.
            E.g. `(200, 200, 3)` would be one valid value.
        pooling: Optional pooling mode for feature
            extraction when `include_top` is `False`.
            - `None` means that the output of the model will be
                the 4D tensor output of the
                last convolutional layer.
            - `avg` means that global average
                pooling will be applied to the output
                of the last convolutional layer, and
                thus
                the output of the model will be a 2D tensor.
            - `max` means that global max pooling will
                be applied.
        classes: optional number of classes to classify images
            into, only to be specified if `include_top` is True, and
            if no `weights` argument is specified.

    # Returns
        A Keras model instance.

    # Raises
        ValueError: in case of invalid argument for `weights`,
            or invalid input shape.
    """
    if weights not in {'imagenet', None}:
        raise ValueError('The `weights` argument should be either '
                         '`None` (random initialization) or `imagenet` '
                         '(pre-training on ImageNet).')
```

```
if weights == 'imagenet' and include_top and classes != 1000:
```

```
        raise ValueError('If using `weights` as imagenet with
                         `include_top`' ' as true, `classes` should be
                         1000')
# Determine proper input shape
input_shape   =   _obtain_input_shape(input_shape,
                                      default_size=224,
                                      min_size=48,
                                      data_format=K.image_data_format(),
                                      require_flatten=include_top)

if input_tensor is None:
    img_input = Input(shape=input_shape)
else:
    if not K.is_keras_tensor(input_tensor):
        img_input = Input(tensor=input_tensor, shape=input_shape)
    else:
        img_input = input_tensor
# Block 1
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv1')(i
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv1')(
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv2')(
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

# Block 3
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv1')(
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv2')(
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv3')(
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)

# Block 4
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv1')(
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv2')(
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv3')(
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

# Block 5
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv1')(
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv2')(
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(
x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)

if include_top:
    # Classification block
    x = Flatten(name='flatten')(x)
    x = Dense(4096, activation='relu', name='fc1')(x)
    x = Dense(4096, activation='relu', name='fc2')(x)
    x = Dense(classes, activation='softmax', name='predictions')(x)
else:
    if pooling == 'avg':
        x = GlobalAveragePooling2D()(x)
    elif pooling == 'max':
        x = GlobalMaxPooling2D()(x)

# Ensure that the model takes into account
# any potential predecessors of `input_tensor`.
if input_tensor is not None:
    inputs = get_source_inputs(input_tensor)
else:
    inputs = img_input
# Create model.
model = Model(inputs, x, name='vgg16')
```

```
    # load weights
    if weights == 'imagenet':
        if include_top:
            weights_path = get_file('vgg16_weights_tf_dim_ordering_tf_kernels.h5',
                                    WEIGHTS_PATH,
                                    cache_subdir='models')
        else:
            weights_path = get_file('vgg16_weights_tf_dim_ordering_tf_kernels_notop.
                                    WEIGHTS_PATH_NO_TOP,
                                    cache_subdir='models')
        model.load_weights(weights_path)
        if K.backend() == 'theano':
            layer_utils.convert_all_kernels_in_model(model)

        if K.image_data_format() == 'channels_first':
            if include_top:
                maxpool = model.get_layer(name='block5_pool')
                shape = maxpool.output_shape[1:]
                dense = model.get_layer(name='fc1')
                layer_utils.convert_dense_weights_data_format(dense, shape, 'channe

            if K.backend() == 'tensorflow':
                warnings.warn('You are using the TensorFlow backend, yet you '
                              'are using the Theano '
                              'image data format convention '
                              '(`image_data_format="channels_first"`). '
                              'For best performance, set '
                              '`image_data_format="channels_last"` in '
                              'your Keras config '
                              'at ~/.keras/keras.json.')
    return model
```

In [5]:

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

CIFAR-10 Dataset

In [ ]:

```
(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]
```

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:
```
X_train_resized = preprocess_data(X_train_resized)
X_test_resized = preprocess_data(X_test_resized)
```

In [ ]:
```
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)

# img_path = 'a.jpg'
# img = image.load_img(img_path, target_size=(224, 224))
# x = image.img_to_array(img)
# x = np.expand_dims(x, axis=0)
# x = preprocess_input(x)
# print('Input image shape:', x.shape)

# preds = model.predict(x)
# print('Predicted:', decode_predictions(preds))
```

```
Epoch 1/5
63/63 [==============================] - 97s 855ms/step - loss: nan - accuracy: 0.09
85
Epoch 2/5
63/63 [==============================] - 47s 743ms/step - loss: nan - accuracy: 0.10
10
Epoch 3/5
63/63 [==============================] - 47s 745ms/step - loss: nan - accuracy: 0.10
10
Epoch 4/5
63/63 [==============================] - 47s 744ms/step - loss: nan - accuracy: 0.10
10
Epoch 5/5
63/63 [==============================] - 47s 744ms/step - loss: nan - accuracy: 0.10
10
```

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

```
63/63 [==============================] - 14s 225ms/step - loss: nan - accuracy: 0.09
80
```

Out[ ]:
```
[nan, 0.09799999743700027]
```

MNIST Dataset

In [6]:
```
(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]
```

In [7]:
```
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)

In [8]:

```
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]))
```

In [9]:

```
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
```

In [10]:

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

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

Out[10]: [2.6351511478424072, 0.10949999839067459]

SAVEE Dataset

In [ ]:

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

In [13]:

```
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_mel mel_db = (librosa.power_t

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
```

In [15]:

```
# 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]))
```

In [17]:

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

In [21]:

```
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=
```

In [25]:

```
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)
```

In [26]:

```
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)
```

```
Epoch 1/50
8/8 [==============================] - 7s 725ms/step - loss: nan - accuracy: 0.0917
Epoch 2/50
8/8 [==============================] - 6s 705ms/step - loss: nan - accuracy: 0.1208
Epoch 3/50
8/8 [==============================] - 6s 707ms/step - loss: nan - accuracy: 0.1208
Epoch 4/50
8/8 [==============================] - 6s 705ms/step - loss: nan - accuracy: 0.1208
Epoch 5/50
8/8 [==============================] - 6s 708ms/step - loss: nan - accuracy: 0.1208
Epoch 6/50
8/8 [==============================] - 6s 708ms/step - loss: nan - accuracy: 0.1208
Epoch 7/50
8/8 [==============================] - 6s 706ms/step - loss: nan - accuracy: 0.1208
Epoch 8/50
8/8 [==============================] - 6s 709ms/step - loss: nan - accuracy: 0.1208
Epoch 9/50
8/8 [==============================] - 6s 709ms/step - loss: nan - accuracy: 0.1208
Epoch 10/50
8/8 [==============================] - 6s 707ms/step - loss: nan - accuracy: 0.1208
Epoch 11/50
8/8 [==============================] - 6s 706ms/step - loss: nan - accuracy: 0.1208
Epoch 12/50
8/8 [==============================] - 6s 708ms/step - loss: nan - accuracy: 0.1208
Epoch 13/50
```

```
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 14/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 15/50
8/8 [==============================]   - 6s  709ms/step - loss: nan - accuracy: 0.1208
Epoch 16/50
8/8 [==============================]   - 6s  706ms/step - loss: nan - accuracy: 0.1208
Epoch 17/50
8/8 [==============================]   - 6s  710ms/step - loss: nan - accuracy: 0.1208
Epoch 18/50
8/8 [==============================]   - 6s  711ms/step - loss: nan - accuracy: 0.1208
Epoch 19/50
8/8 [==============================]   - 6s  704ms/step - loss: nan - accuracy: 0.1208
Epoch 20/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 21/50
8/8 [==============================]   - 6s  708ms/step - loss: nan - accuracy: 0.1208
Epoch 22/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 23/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 24/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 25/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 26/50
8/8 [==============================]   - 6s  705ms/step - loss: nan - accuracy: 0.1208
Epoch 27/50
8/8 [==============================]   - 6s  705ms/step - loss: nan - accuracy: 0.1208
Epoch 28/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 29/50
8/8 [==============================]   - 6s  709ms/step - loss: nan - accuracy: 0.1208
Epoch 30/50
8/8 [==============================]   - 6s  706ms/step - loss: nan - accuracy: 0.1208
Epoch 31/50
8/8 [==============================]   - 6s  705ms/step - loss: nan - accuracy: 0.1208
Epoch 32/50
8/8 [==============================]   - 6s  705ms/step - loss: nan - accuracy: 0.1208
Epoch 33/50
8/8 [==============================]   - 6s  709ms/step - loss: nan - accuracy: 0.1208
Epoch 34/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 35/50
8/8 [==============================]   - 6s  706ms/step - loss: nan - accuracy: 0.1208
Epoch 36/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 37/50
8/8 [==============================]   - 6s  703ms/step - loss: nan - accuracy: 0.1208
Epoch 38/50
8/8 [==============================]   - 6s  705ms/step - loss: nan - accuracy: 0.1208
Epoch 39/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 40/50
8/8 [==============================]   - 6s  706ms/step - loss: nan - accuracy: 0.1208
Epoch 41/50
8/8 [==============================]   - 6s  705ms/step - loss: nan - accuracy: 0.1208
Epoch 42/50
8/8 [==============================]   - 6s  707ms/step - loss: nan - accuracy: 0.1208
Epoch 43/50
8/8 [==============================]   - 6s  706ms/step - loss: nan - accuracy: 0.1208
Epoch 44/50
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
```

In [28]:

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

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

Out[28]:

```
[nan, 0.12916666269302368]
```

EmoDb Dataset

In [ ]:

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

In [30]:

```
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_mel
    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
```

In [31]:

```
# 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]))
```

In [32]:

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

In [33]:

```
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=
```

In [34]:

```
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)
```

In [35]:

```
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)
```

```
Epoch 1/20
9/9 [==============================] - 13s 1s/step - loss: nan - accuracy: 0.1610
Epoch 2/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
Epoch 3/20
9/9 [==============================] - 6s 713ms/step - loss: nan - accuracy: 0.2247
Epoch 4/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
Epoch 5/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
```

```
Epoch 6/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
Epoch 7/20
9/9 [==============================] - 6s 713ms/step - loss: nan - accuracy: 0.2247
Epoch 8/20
9/9 [==============================] - 6s 713ms/step - loss: nan - accuracy: 0.2247
Epoch 9/20
9/9 [==============================] - 6s 712ms/step - loss: nan - accuracy: 0.2247
Epoch 10/20
9/9 [==============================] - 6s 711ms/step - loss: nan - accuracy: 0.2247
Epoch 11/20
9/9 [==============================] - 6s 715ms/step - loss: nan - accuracy: 0.2247
Epoch 12/20
9/9 [==============================] - 6s 710ms/step - loss: nan - accuracy: 0.2247
Epoch 13/20
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
```

In [36]:

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

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

Out[36]:

```
[nan, 0.25]
```

RESNET-50

In [14]:

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

Mounted at /content/drive

In [1]:

```
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
```

```
Requirement already satisfied: keras_applications in /usr/local/lib/python3.7/dist-p
ackages (1.0.8)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-package
s (from keras_applications) (1.19.5)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from
keras_applications) (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-pack
ages (from h5py->keras_applications) (1.5.2)
```

In [2]:

```
WEIGHTS_PATH = 'https://github.com/fchollet/deep-learning-models/releases/download/v
WEIGHTS_PATH_NO_TOP = 'https://github.com/fchollet/deep-learning-models/releases/dow


def identity_block(input_tensor, kernel_size, filters, stage, block):
"""The identity block is the block that has no conv layer at shortcut. # Arguments
input_tensor: input tensor
kernel_size: defualt 3, the kernel size of middle conv layer at main path filters: list of integers, the filterss of 3 conv
stage: integer, current stage label, used for generating layer names
block: 'a','b'..., current block label, used for generating layer names
```

```
    # Returns
        Output tensor for the block.
    """
    filters1, filters2, filters3 = filters
    if K.image_data_format() == 'channels_last':
        bn_axis = 3
    else:
        bn_axis = 1
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    x = Conv2D(filters1, (1, 1), name=conv_name_base + '2a')(input_tensor)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2a')(x)
    x = Activation('relu')(x)

    x = Conv2D(filters2, kernel_size,
               padding='same', name=conv_name_base + '2b')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2b')(x)
    x = Activation('relu')(x)

    x = Conv2D(filters3, (1, 1), name=conv_name_base + '2c')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2c')(x)

    x = layers.add([x, input_tensor])
    x = Activation('relu')(x)
    return x


def conv_block(input_tensor, kernel_size, filters, stage, block, strides=(2, 2)):
    """conv_block is the block that has a conv layer at shortcut
    # Arguments
        input_tensor: input tensor
        kernel_size: defualt 3, the kernel size of middle conv layer at main path
        filters: list of integers, the filterss of 3 conv layer at main path
        stage: integer, current stage label, used for generating layer names
        block: 'a','b'..., current block label, used for generating layer names
    # Returns
        Output tensor for the block.
    Note that from stage 3, the first conv layer at main path is with strides=(2,2)
    And the shortcut should have strides=(2,2) as well
    """
    filters1, filters2, filters3 = filters
    if K.image_data_format() == 'channels_last':
        bn_axis = 3
    else:
        bn_axis = 1
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    x = Conv2D(filters1, (1, 1), strides=strides,
               name=conv_name_base + '2a')(input_tensor)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2a')(x)
    x = Activation('relu')(x)

    x = Conv2D(filters2, kernel_size, padding='same',
               name=conv_name_base + '2b')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2b')(x)
    x = Activation('relu')(x)

    x = Conv2D(filters3, (1, 1), name=conv_name_base + '2c')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2c')(x)

    shortcut = Conv2D(filters3, (1, 1), strides=strides,
                      name=conv_name_base + '1')(input_tensor)
    shortcut = BatchNormalization(axis=bn_axis, name=bn_name_base + '1')(shortcut)
```

```
        x = layers.add([x, shortcut])
        x = Activation('relu')(x)
        return x


def ResNet50(include_top=True, weights='imagenet',
             input_tensor=None, input_shape=None,
             pooling=None,
             classes=1000):
    """Instantiates the ResNet50 architecture.
    Optionally loads weights pre-trained
    on ImageNet. Note that when using TensorFlow,
    for best performance you should set
    `image_data_format="channels_last"` in your Keras
    config at ~/.keras/keras.json.
    The model and the weights are compatible with both
    TensorFlow and Theano. The data format
    convention used by the model is the
    one specified in your Keras config
    file.
    # Arguments
        include_top: whether to include the fully-connected
            layer at the top of the network.
        weights: one of `None` (random initialization)
            or "imagenet" (pre-training on ImageNet).
        input_tensor: optional Keras tensor (i.e. output of `layers.Input()`)
            to use as image input for the model.
        input_shape: optional shape tuple, only to be specified
            if `include_top` is False (otherwise the input shape
            has to be `(224, 224, 3)` (with `channels_last` data format)
            or `(3, 224, 244)` (with `channels_first` data format).
            It should have exactly 3 inputs channels,
            and width and height should be no smaller than 197.
            E.g. `(200, 200, 3)` would be one valid value.
        pooling: Optional pooling mode for feature
            extraction when `include_top` is `False`.
            - `None` means that the output of the model will be
                the 4D tensor output of the
                last convolutional layer.
            - `avg` means that global average
                pooling will be applied to the output
                of the last convolutional layer, and
                thus
                the output of the model will be a 2D tensor.
            - `max` means that global max pooling will
                be applied.
        classes: optional number of classes to classify images
            into, only to be specified if `include_top` is True, and
            if no `weights` argument is specified.
    # Returns
        A Keras model instance.
    # Raises
        ValueError: in case of invalid argument for `weights`,
            or invalid input shape.
    """
    if weights not in {'imagenet', None}:
        raise ValueError('The `weights` argument should be either '
                         '`None` (random initialization) or `imagenet` '
                         '(pre-training on ImageNet).')

    if weights == 'imagenet' and include_top and classes != 1000:
        raise ValueError('If using `weights` as imagenet with
                         `include_top`' ' as true, `classes` should be
                         1000')
```

```
# Determine proper input shape
input_shape = _obtain_input_shape(input_shape,
```

```
                                      default_size=224,
                                      min_size=197,
                                      data_format=K.image_data_format(),
                                      require_flatten=include_top)

    if input_tensor is None:
        img_input = Input(shape=input_shape)
    else:
        if not K.is_keras_tensor(input_tensor):
            img_input = Input(tensor=input_tensor, shape=input_shape)
        else:
            img_input = input_tensor
    if K.image_data_format() == 'channels_last':
        bn_axis = 3
    else:
        bn_axis = 1

    x = ZeroPadding2D((3, 3))(img_input)
    x = Conv2D(64, (7, 7), strides=(2, 2), name='conv1')(x)
    x = BatchNormalization(axis=bn_axis, name='bn_conv1')(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2))(x)

    x = conv_block(x, 3, [64, 64, 256], stage=2, block='a', strides=(1, 1))
    x = identity_block(x, 3, [64, 64, 256], stage=2, block='b')
    x = identity_block(x, 3, [64, 64, 256], stage=2, block='c')

    x = conv_block(x, 3, [128, 128, 512], stage=3, block='a')
    x = identity_block(x, 3, [128, 128, 512], stage=3, block='b')
    x = identity_block(x, 3, [128, 128, 512], stage=3, block='c')
    x = identity_block(x, 3, [128, 128, 512], stage=3, block='d')

    x = conv_block(x, 3, [256, 256, 1024], stage=4, block='a')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='b')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='c')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='d')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='e')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='f')

    x = conv_block(x, 3, [512, 512, 2048], stage=5, block='a')
    x = identity_block(x, 3, [512, 512, 2048], stage=5, block='b')
    x = identity_block(x, 3, [512, 512, 2048], stage=5, block='c')

    x = AveragePooling2D((7, 7), name='avg_pool')(x)

    if include_top:
        x = Flatten()(x)
        x = Dense(classes, activation='softmax', name='fc1000')(x)
    else:
        if pooling == 'avg':
            x = GlobalAveragePooling2D()(x)
        elif pooling == 'max':
            x = GlobalMaxPooling2D()(x)

    # Ensure that the model takes into account
    # any potential predecessors of `input_tensor`.
    if input_tensor is not None:
        inputs = get_source_inputs(input_tensor)
    else:
        inputs = img_input
    # Create model.
    model = Model(inputs, x, name='resnet50')

    # load weights
    if weights == 'imagenet':
```

```python
        if include_top:
            weights_path = get_file('resnet50_weights_tf_dim_ordering_tf_kernels.h5'
                                    WEIGHTS_PATH,
                                    cache_subdir='models',
                                    md5_hash='a7b3fe01876f51b976af0dea6bc144eb')
        else:
            weights_path = get_file('resnet50_weights_tf_dim_ordering_tf_kernels_not
                                    WEIGHTS_PATH_NO_TOP,
                                    cache_subdir='models',
                                    md5_hash='a268eb855778b3df3c7506639542a6af')
        model.load_weights(weights_path)
        if K.backend() == 'theano':
            layer_utils.convert_all_kernels_in_model(model)

        if K.image_data_format() == 'channels_first':
            if include_top:
                maxpool = model.get_layer(name='avg_pool')
                shape = maxpool.output_shape[1:]
                dense = model.get_layer(name='fc1000')
                layer_utils.convert_dense_weights_data_format(dense, shape, 'channel

            if K.backend() == 'tensorflow':
                warnings.warn('You are using the TensorFlow backend, yet you '
                              'are using the Theano '
                              'image data format convention '
                              '(`image_data_format="channels_first"`). '
                              'For best performance, set '
                              '`image_data_format="channels_last"` in '
                              'your Keras config '
                              'at ~/.keras/keras.json.')
    return model
```

In [3]:

```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
```

In [4]:

```python
def preprocess_data(X_train):

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

    return X_train
```

CIFAR-10 DATASET

In [ ]:

```python
(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]
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
170508288/170498071 [==============================] - 3s 0us/step
```

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:

```
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)
```

In [ ]:

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

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50_weights_tf_dim_ordering_tf_kernels.h5
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
```

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

Out[ ]:

```
[16.839269638061523, 0.0]
```

MNIST Dataset

In [5]:

```
(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]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
ist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

In [6]:

```
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)
```

In [7]:

```
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]))
```

In [9]:

```
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)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/down
load/v0.2/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102858752/102853048 [==============================] - 5s 0us/step
102866944/102853048 [==============================] - 5s 0us/step
Epoch 1/10
63/63 [==============================] - 81s 715ms/step - loss: 2.8611 - accuracy:
0.1030
Epoch 2/10
63/63 [==============================] - 44s 696ms/step - loss: 1.0983 - accuracy:
0.6265
Epoch 3/10
63/63 [==============================] - 44s 695ms/step - loss: 0.2511 - accuracy:
0.9315
Epoch 4/10
63/63 [==============================] - 44s 698ms/step - loss: 0.1633 - accuracy:
0.9570
Epoch 5/10
```

```
63/63 [==============================] - 44s 702ms/step - loss: 0.1118 - accuracy:
0.9710
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
```

In [12]:

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

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

Out[12]: [13.156914710998535, 0.0]

SAVEE Dataset

In [ ]:

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

In [5]:

```
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_mel mel_db = (librosa.power_

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
```

In [6]:

```
# 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]))
```

In [7]:

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

In [8]:

```
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=
```

In [11]:

```
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)
```

In [12]:
```
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 1/10
8/8 [==============================] - 10s 671ms/step - loss: 5.4114 - accuracy:
0.1
042
Epoch 2/10
```

In [13]:
```
model.evaluate(X_test_resized, y_test)
```

```
8/8 [==============================]  - 5s  668ms/step - loss:  2.2010  - accuracy: 0.28
75
Epoch 3/10
8/8 [==============================]  - 5s  670ms/step - loss:  1.4778  - accuracy: 0.51
25
Epoch 4/10
8/8 [==============================]  - 5s  669ms/step - loss:  1.0197  - accuracy: 0.68
33
Epoch 5/10
8/8 [==============================]  - 5s  667ms/step - loss:  0.5054  - accuracy: 0.91
67
Epoch 6/10
8/8 [==============================]  - 5s  669ms/step - loss:  0.2390  - accuracy: 0.98
33
Epoch 7/10
8/8 [==============================]  - 5s  671ms/step - loss:  0.0966  - accuracy: 1.00
00
Epoch 8/10
8/8 [==============================]  - 5s  668ms/step - loss:  0.0691  - accuracy: 1.00
00
Epoch 9/10
8/8 [==============================]  - 5s  669ms/step - loss:  0.0550  - accuracy: 1.00
00
Epoch 10/10
8/8 [==============================]  - 5s  666ms/step - loss:  0.0281  - accuracy: 1.00
00
```

```
8/8 [==============================] - 3s 215ms/step - loss: 8.7594 - accuracy: 0.00
00e+00
```

Out[13]: [8.759380340576172, 0.0]

EmoDB Dataset

In [ ]:

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

In [15]:

```
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_mel
```

```
        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
```

In [16]:

```
# 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]))
```

In [17]:

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

In [18]:

```
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=
```

In [19]:

```
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)
```

In [20]:

```
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 1/10
9/9 [==============================] - 12s 857ms/step - loss: 5.1684 - accuracy:
0.1
685
Epoch 2/10
```

In [21]:

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

```
9/9 [==============================] - 6s 665ms/step - loss: 1.7511 - accuracy: 0.41
57
Epoch 3/10
9/9 [==============================] - 6s 663ms/step - loss: 1.1062 - accuracy: 0.63
67
Epoch 4/10
9/9 [==============================] - 6s 661ms/step - loss: 0.6534 - accuracy: 0.76
78
Epoch 5/10
9/9 [==============================] - 6s 662ms/step - loss: 0.3835 - accuracy: 0.89
14
Epoch 6/10
9/9 [==============================] - 6s 662ms/step - loss: 0.3716 - accuracy: 0.86
89
Epoch 7/10
```

```
9/9 [==============================] - 6s 662ms/step - loss: 0.2297 - accuracy: 0.92
13
Epoch 8/10
9/9 [==============================] - 6s 661ms/step - loss: 0.1096 - accuracy: 0.97
75
Epoch 9/10
9/9 [==============================] - 6s 664ms/step - loss: 0.1170 - accuracy: 0.98
50
Epoch 10/10
9/9 [==============================] - 6s 659ms/step - loss: 0.2414 - accuracy: 0.92
51
```

```
9/9 [==============================] - 4s 304ms/step - loss: 7.2902 - accuracy: 0.00
00e+00
```

Out[21]: [7.290168285369873, 0.0]

RECURRENT NEURAN NETWORKS

In [1]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

CIFAR 10 DATASET

In [ ]:
```python
import os
import tensorflow as tf
import keras
from tensorflow.keras import layers
from tensorflow.keras import Model
from os import getcwd
```

In [ ]:
```python
cifar10 = tf.keras.datasets.cifar10
(training_images, training_labels), (test_images, test_labels) = cifar10.load_data()
```

In [ ]:
```python
print(len(training_images))
print(len(test_images))
```

```
50000
10000
```

In [ ]:
```python
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
```

In [ ]:
```python
model = tf.keras.models.Sequential([
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input_shape=(1024,3),
        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')
        ])
```

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

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

```
Epoch 1/10
200/200 [==============================] - 123s 557ms/step - loss: 2.1137 - accurac
y: 0.1966
Epoch 2/10
200/200 [==============================] - 112s 558ms/step - loss: 2.0086 - accurac
y: 0.2529
Epoch 3/10
200/200 [==============================] - 111s 557ms/step - loss: 2.0085 - accurac
y: 0.2645
```

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

In [ ]:

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

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

Out[ ]:

```
[1.9600898027420044, 0.29120001196861267]
```

MNIST DATASET

In [ ]:

```
import torch
```

In [ ]:

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

Out[ ]:

```
device(type='cuda')
```

In [ ]:

```
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()
)
```

In [ ]:

```
print(train_data)
```

```
Dataset MNIST
```

Number of datapoints: 60000
Root location: data
Split: Train

```
        StandardTransform
    Transform: ToTensor()
```

In [ ]:

```python
print(test_data)
```

```
Dataset MNIST
    Number of datapoints: 10000
    Root location: data
    Split: Test
    StandardTransform
Transform: ToTensor()
```

In [ ]:

```python
print(train_data.data.size())
```

```
torch.Size([60000, 28, 28])
```

In [ ]:

```python
print(train_data.targets.size())
```

```
torch.Size([60000])
```

In [ ]:

```python
print(train_data.data[0])
```

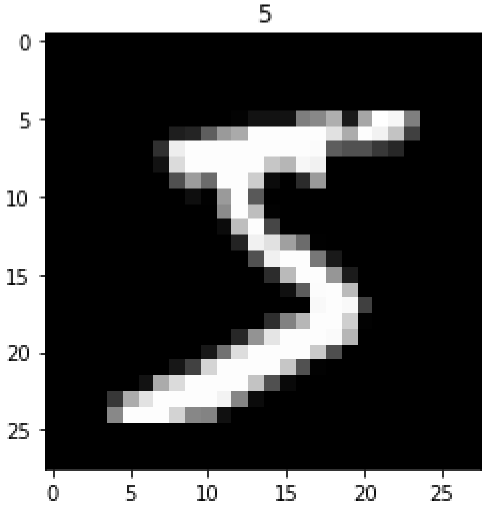
```
tensor([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,  18,
          18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
         253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253, 253,
         253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253, 253,
         198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253, 205,
          11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,  90,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253, 190,
           2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190, 253,
          70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35, 241,
         225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  81,
         240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39, 148,
         229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221, 253,
```

```
 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253, 253,
 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
```

```
[  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253, 195,
  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
[  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,  11,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
[  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0]],

dtype=torch.uint8)
```

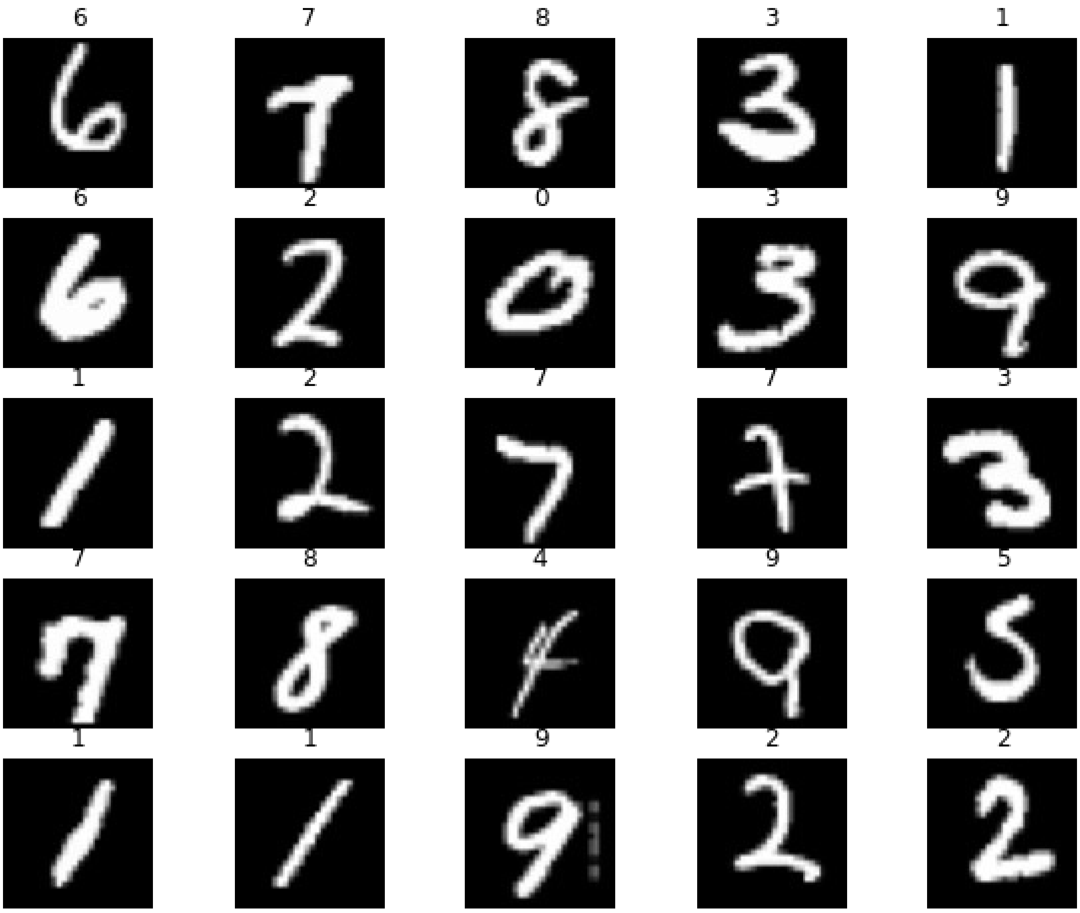
In [ ]:

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



In [ ]:

```
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()
```



In [ ]:

```
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
```

Out[ ]:

```
{'test': <torch.utils.data.dataloader.DataLoader at 0x7fae59428850>,
 'train': <torch.utils.data.dataloader.DataLoader at 0x7fae59449e10>}
```

In [ ]:

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

In [ ]:

```
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
```

In [ ]:

```
class  RNN(nn.Module):
```

```
pass
model = RNN().to(device) print(model)
```

RNN()

In [ ]:

```
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 output
        out, hidden = self.lstm(x, (h0, c0))  # out: tensor of shape (batch_size, se

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

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

```
RNN(
  (lstm): LSTM(28, 128, num_layers=2, batch_first=True)
  (fc): Linear(in_features=128, out_features=10, bias=True)
)
```

In [ ]:

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

Out[ ]:

```
CrossEntropyLoss()
```

In [ ]:

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

Out[ ]:

```
Adam (
Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    eps: 1e-08
    lr: 0.01
    weight_decay: 0
)
```

In [ ]:

```
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)
```

```
Epoch [1/2], Step [100/600], Loss: 0.6104
Epoch [1/2], Step [200/600], Loss: 0.2625
Epoch [1/2], Step [300/600], Loss: 0.1447
Epoch [1/2], Step [400/600], Loss: 0.2647
Epoch [1/2], Step [500/600], Loss: 0.1042
Epoch [1/2], Step [600/600], Loss: 0.0769
Epoch [2/2], Step [100/600], Loss: 0.0376
Epoch [2/2], Step [200/600], Loss: 0.0225
Epoch [2/2], Step [300/600], Loss: 0.0473
Epoch [2/2], Step [400/600], Loss: 0.0719
Epoch [2/2], Step [500/600], Loss: 0.1155
Epoch [2/2], Step [600/600], Loss: 0.1507
```

In [ ]:

```
# 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 * corre
```

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

SAVEE Dataset

In [ ]:

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

In [3]:

```
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_mel
    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
```

In [8]:

```
# 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]))
```

In [9]:

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

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

In [11]:

```
X.shape , y.shape
```

Out[11]:

```
((480, 157, 320), (480,))
```

In [14]:

```
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=
```

In [15]:

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

In [18]:

```
model = tf.keras.models.Sequential([
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input_shape=(157,320)
        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')
        ])
```

In [19]:

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

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

```
Epoch 1/50
6/6 [==============================] - 9s 127ms/step - loss: 2.1944 - accuracy: 0.19
10
Epoch 2/50
6/6 [==============================] - 1s 126ms/step - loss: 2.0548 - accuracy: 0.24
31
Epoch 3/50
6/6 [==============================] - 1s 126ms/step - loss: 1.9929 - accuracy: 0.24
31
Epoch 4/50
6/6 [==============================] - 1s 127ms/step - loss: 1.9573 - accuracy: 0.24
31
Epoch 5/50
6/6 [==============================] - 1s 129ms/step - loss: 1.9331 - accuracy: 0.24
31
Epoch 6/50
6/6 [==============================] - 1s 127ms/step - loss: 1.9149 - accuracy: 0.25
00
Epoch 7/50
6/6 [==============================] - 1s 130ms/step - loss: 1.9023 - accuracy: 0.24
65
```

```
Epoch 8/50
6/6 [==============================] - 1s 125ms/step - loss: 1.9052 - accuracy: 0.25
69
Epoch 9/50
6/6 [==============================] - 1s 127ms/step - loss: 1.8863 - accuracy: 0.25
35
Epoch 10/50
6/6 [==============================] - 1s 125ms/step - loss: 1.8672 - accuracy: 0.28
47
Epoch 11/50
6/6 [==============================] - 1s 126ms/step - loss: 1.8465 - accuracy: 0.28
82
Epoch 12/50
6/6 [==============================] - 1s 127ms/step - loss: 1.8392 - accuracy: 0.27
78
Epoch 13/50
6/6 [==============================] - 1s 132ms/step - loss: 1.8321 - accuracy: 0.28
47
Epoch 14/50
6/6 [==============================] - 1s 131ms/step - loss: 1.8343 - accuracy: 0.26
74
Epoch 15/50
6/6 [==============================] - 1s 127ms/step - loss: 1.8393 - accuracy: 0.25
69
Epoch 16/50
6/6 [==============================] - 1s 124ms/step - loss: 1.8227 - accuracy: 0.27
78
Epoch 17/50
6/6 [==============================] - 1s 130ms/step - loss: 1.7737 - accuracy: 0.30
56
Epoch 18/50
6/6 [==============================] - 1s 126ms/step - loss: 1.7376 - accuracy: 0.35
42
Epoch 19/50
6/6 [==============================] - 1s 129ms/step - loss: 1.8251 - accuracy: 0.33
33
Epoch 20/50
6/6 [==============================] - 1s 127ms/step - loss: 1.8050 - accuracy: 0.28
82
Epoch 21/50
6/6 [==============================] - 1s 131ms/step - loss: 1.7694 - accuracy: 0.31
94
Epoch 22/50
6/6 [==============================] - 1s 127ms/step - loss: 1.7708 - accuracy: 0.33
33
Epoch 23/50
6/6 [==============================] - 1s 131ms/step - loss: 1.7016 - accuracy: 0.35
76
Epoch 24/50
6/6 [==============================] - 1s 124ms/step - loss: 1.7182 - accuracy: 0.35
76
Epoch 25/50
6/6 [==============================] - 1s 129ms/step - loss: 1.6337 - accuracy: 0.36
11
Epoch 26/50
6/6 [==============================] - 1s 131ms/step - loss: 1.5874 - accuracy: 0.41
67
Epoch 27/50
6/6 [==============================] - 1s 125ms/step - loss: 1.5535 - accuracy: 0.39
93
Epoch 28/50
6/6 [==============================] - 1s 125ms/step - loss: 1.5196 - accuracy: 0.36
11
Epoch 29/50
6/6 [==============================] - 1s 128ms/step - loss: 1.5726 - accuracy: 0.38
89
Epoch 30/50
6/6 [==============================] - 1s 128ms/step - loss: 1.4901 - accuracy: 0.45
49
```

```
Epoch 31/50
6/6 [==============================] - 1s 126ms/step - loss: 1.3938 - accuracy: 0.45
14
Epoch 32/50
6/6 [==============================] - 1s 129ms/step - loss: 1.4611 - accuracy: 0.40
28
Epoch 33/50
6/6 [==============================] - 1s 130ms/step - loss: 1.3808 - accuracy: 0.43
06
Epoch 34/50
6/6 [==============================] - 1s 128ms/step - loss: 1.3642 - accuracy: 0.45
14
Epoch 35/50
6/6 [==============================] - 1s 128ms/step - loss: 1.3113 - accuracy: 0.44
10
Epoch 36/50
6/6 [==============================] - 1s 129ms/step - loss: 1.2760 - accuracy: 0.48
26
Epoch 37/50
6/6 [==============================] - 1s 125ms/step - loss: 1.2692 - accuracy: 0.47
57
Epoch 38/50
6/6 [==============================] - 1s 128ms/step - loss: 1.3421 - accuracy: 0.44
10
Epoch 39/50
6/6 [==============================] - 1s 129ms/step - loss: 1.2649 - accuracy: 0.46
53
Epoch 40/50
6/6 [==============================] - 1s 129ms/step - loss: 1.2730 - accuracy: 0.49
31
Epoch 41/50
6/6 [==============================] - 1s 128ms/step - loss: 1.2156 - accuracy: 0.50
69
Epoch 42/50
6/6 [==============================] - 1s 131ms/step - loss: 1.1851 - accuracy: 0.53
47
Epoch 43/50
6/6 [==============================] - 1s 123ms/step - loss: 1.1742 - accuracy: 0.54
86
Epoch 44/50
6/6 [==============================] - 1s 128ms/step - loss: 1.3160 - accuracy: 0.49
31
Epoch 45/50
6/6 [==============================] - 1s 127ms/step - loss: 1.2075 - accuracy: 0.52
08
Epoch 46/50
6/6 [==============================] - 1s 128ms/step - loss: 1.1717 - accuracy: 0.53
47
Epoch 47/50
6/6 [==============================] - 1s 125ms/step - loss: 1.1327 - accuracy: 0.56
25
Epoch 48/50
6/6 [==============================] - 1s 128ms/step - loss: 1.1397 - accuracy: 0.57
29
Epoch 49/50
6/6 [==============================] - 1s 129ms/step - loss: 1.0978 - accuracy: 0.57
64
Epoch 50/50
6/6 [==============================] - 1s 126ms/step - loss: 1.0690 - accuracy: 0.56
94
```

In [20]:

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

```
6/6 [==============================] - 2s 57ms/step - loss: 1.4087 - accuracy: 0.432
3
```

Out[20]: [1.408677577972412, 0.4322916567325592]

EmoDB Dataset

In [ ]:

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

In [22]:

```
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_mel
    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
```

In [23]:

```
# 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]))
```

In [24]:

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

In [25]:

```
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=
```

In [26]:

```
model = tf.keras.models.Sequential([
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input_shape=(157,320)
        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')
        ])
```

In [27]:

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

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

```
Epoch 1/50
8/8 [==============================] - 10s 130ms/step - loss: 2.1718 - accuracy: 0.1
631
Epoch 2/50
8/8 [==============================] - 1s 130ms/step - loss: 2.0208 - accuracy: 0.18
98
Epoch 3/50
8/8 [==============================] - 1s 130ms/step - loss: 1.9197 - accuracy: 0.25
40
Epoch 4/50
8/8 [==============================] - 1s 127ms/step - loss: 1.8459 - accuracy: 0.27
01
Epoch 5/50
8/8 [==============================] - 1s 130ms/step - loss: 1.7682 - accuracy: 0.32
89
Epoch 6/50
8/8 [==============================] - 1s 127ms/step - loss: 1.6464 - accuracy: 0.44
39
Epoch 7/50
8/8 [==============================] - 1s 129ms/step - loss: 1.5401 - accuracy: 0.40
64
Epoch 8/50
8/8 [==============================] - 1s 127ms/step - loss: 1.4142 - accuracy: 0.41
71
Epoch 9/50
8/8 [==============================] - 1s 132ms/step - loss: 1.3430 - accuracy: 0.48
66
Epoch 10/50
```

```
8/8 [==============================] - 1s 131ms/step - loss: 1.2661 - accuracy: 0.51
07
Epoch 11/50
8/8 [==============================] - 1s 129ms/step - loss: 1.2101 - accuracy: 0.50
53
Epoch 12/50
8/8 [==============================] - 1s 131ms/step - loss: 1.1619 - accuracy: 0.53
74
Epoch 13/50
8/8 [==============================] - 1s 131ms/step - loss: 1.0971 - accuracy: 0.54
81
Epoch 14/50
8/8 [==============================] - 1s 130ms/step - loss: 1.1003 - accuracy: 0.55
08
Epoch 15/50
8/8 [==============================] - 1s 127ms/step - loss: 1.0850 - accuracy: 0.56
15
Epoch 16/50
8/8 [==============================] - 1s 130ms/step - loss: 1.0485 - accuracy: 0.58
02
Epoch 17/50
8/8 [==============================] - 1s 132ms/step - loss: 1.0593 - accuracy: 0.56
42
Epoch 18/50
8/8 [==============================] - 1s 130ms/step - loss: 0.9345 - accuracy: 0.63
10
Epoch 19/50
8/8 [==============================] - 1s 131ms/step - loss: 1.0418 - accuracy: 0.59
89
Epoch 20/50
8/8 [==============================] - 1s 129ms/step - loss: 0.9683 - accuracy: 0.63
10
Epoch 21/50
8/8 [==============================] - 1s 128ms/step - loss: 0.9145 - accuracy: 0.61
23
Epoch 22/50
8/8 [==============================] - 1s 132ms/step - loss: 0.8651 - accuracy: 0.64
71
Epoch 23/50
8/8 [==============================] - 1s 128ms/step - loss: 0.7903 - accuracy: 0.66
58
Epoch 24/50
8/8 [==============================] - 1s 128ms/step - loss: 0.7931 - accuracy: 0.68
98
Epoch 25/50
8/8 [==============================] - 1s 128ms/step - loss: 0.6393 - accuracy: 0.74
87
Epoch 26/50
8/8 [==============================] - 1s 127ms/step - loss: 0.6302 - accuracy: 0.75
67
Epoch 27/50
8/8 [==============================] - 1s 130ms/step - loss: 0.7492 - accuracy: 0.70
86
Epoch 28/50
8/8 [==============================] - 1s 130ms/step - loss: 0.6786 - accuracy: 0.72
99
Epoch 29/50
8/8 [==============================] - 1s 130ms/step - loss: 0.5873 - accuracy: 0.75
94
Epoch 30/50
8/8 [==============================] - 1s 133ms/step - loss: 0.8354 - accuracy: 0.69
79
Epoch 31/50
8/8 [==============================] - 1s 129ms/step - loss: 0.5967 - accuracy: 0.78
07
Epoch 32/50
8/8 [==============================] - 1s 126ms/step - loss: 0.5241 - accuracy: 0.79
41
Epoch 33/50
```

```
8/8 [==============================] - 1s 131ms/step - loss: 0.4724 - accuracy: 0.82
09
Epoch 34/50
8/8 [==============================] - 1s 129ms/step - loss: 0.4658 - accuracy: 0.83
96
Epoch 35/50
8/8 [==============================] - 1s 131ms/step - loss: 0.4471 - accuracy: 0.83
96
Epoch 36/50
8/8 [==============================] - 1s 133ms/step - loss: 0.4040 - accuracy: 0.84
76
Epoch 37/50
8/8 [==============================] - 1s 128ms/step - loss: 0.3737 - accuracy: 0.86
36
Epoch 38/50
8/8 [==============================] - 1s 129ms/step - loss: 0.3504 - accuracy: 0.88
50
Epoch 39/50
8/8 [==============================] - 1s 127ms/step - loss: 0.3232 - accuracy: 0.88
50
Epoch 40/50
8/8 [==============================] - 1s 125ms/step - loss: 0.3720 - accuracy: 0.85
56
Epoch 41/50
8/8 [==============================] - 1s 129ms/step - loss: 0.4313 - accuracy: 0.82
89
Epoch 42/50
8/8 [==============================] - 1s 129ms/step - loss: 0.4042 - accuracy: 0.82
62
Epoch 43/50
8/8 [==============================] - 1s 127ms/step - loss: 0.4327 - accuracy: 0.83
16
Epoch 44/50
8/8 [==============================] - 1s 127ms/step - loss: 0.3212 - accuracy: 0.88
50
Epoch 45/50
8/8 [==============================] - 1s 130ms/step - loss: 0.3093 - accuracy: 0.88
24
Epoch 46/50
8/8 [==============================] - 1s 129ms/step - loss: 0.3036 - accuracy: 0.89
30
Epoch 47/50
8/8 [==============================] - 1s 128ms/step - loss: 0.2900 - accuracy: 0.89
30
Epoch 48/50
8/8 [==============================] - 1s 131ms/step - loss: 0.2550 - accuracy: 0.89
84
Epoch 49/50
8/8 [==============================] - 1s 130ms/step - loss: 0.2239 - accuracy: 0.91
44
Epoch 50/50
8/8 [==============================] - 1s 127ms/step - loss: 0.2858 - accuracy: 0.89
84
```

In [28]:

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

```
6/6 [==============================] - 2s 54ms/step - loss: 1.7593 - accuracy: 0.559
0
```

Out[28]:

```
[1.7592660188674927, 0.5590062141418457]
```

ALEXNET

In [1]:

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

Mounted at /content/drive

In [ ]:

```
# 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
```

In [ ]:

```
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))

# 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()
```

In [ ]:

```
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
```

CIFAR 10 DATASET

In [ ]:

```
# Command line parameters # args = parse_args()

# Create AlexNet model
model = alexnet_model()

# Print
```

```
# if  args.print_model: #model.summary()
```

In [ ]:

```
(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]
```

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:

```
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: 0.12
40
Epoch 2/5
16/16 [==============================] - 59s 4s/step - loss: 2.6332 - accuracy: 0.09
60
Epoch 3/5
16/16 [==============================] - 60s 4s/step - loss: 2.5360 - accuracy: 0.12
20
Epoch 4/5
16/16 [==============================] - 59s 4s/step - loss: 2.4222 - accuracy: 0.14
00
Epoch 5/5
16/16 [==============================] - 60s 4s/step - loss: 2.3684 - accuracy: 0.14
00
```

In [ ]:

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

```
7/7 [==============================] - 6s 753ms/step - loss: 2.3611 - accuracy: 0.07
50
```

Out[ ]:

```
[2.3610661029815674, 0.07500000298023224]
```

NMIST Dataset

In [ ]:

```
(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]
```

In [ ]:

```
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
```

In [ ]:

```
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]))
```

In [ ]:

```
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: 0.2
610
Epoch 2/5
63/63 [==============================] - 476s 8s/step - loss: 1.7821 - accuracy: 0.3
680
Epoch 3/5
63/63 [==============================] - 468s 7s/step - loss: 1.6773 - accuracy: 0.4
395
Epoch 4/5
63/63 [==============================] - 469s 7s/step - loss: 1.5820 - accuracy: 0.4
810
Epoch 5/5
63/63 [==============================] - 472s 7s/step - loss: 1.5318 - accuracy: 0.5
040
```

In [ ]:

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

```
63/63 [==============================] - 107s 2s/step - loss: 2.3417 - accuracy:
0.1
170
```

Out[ ]:

```
[2.3417277336120605, 0.11699999868869781]
```

SAVEE Dataset

In [ ]:

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

In [ ]:

```
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_mel
    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
```

In [ ]:

```
# 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]))
```

In [ ]:

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

In [ ]:

```
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=
```

In [ ]:

```
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)
```

In [ ]:

```
model = alexnet_model()

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

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

```
Epoch 1/10
8/8 [==============================] - 99s 7s/step - loss: 2.6041 - accuracy: 0.1167
Epoch 2/10
8/8 [==============================] - 56s 7s/step - loss: 2.5481 - accuracy: 0.1167
Epoch 3/10
8/8 [==============================] - 57s 7s/step - loss: 2.4258 - accuracy: 0.1958
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
```

In [ ]:

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

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

Out[ ]:

[2.275780200958252, 0.23749999701976776]

EmoDB Database

In [ ]:

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

In [ ]:

```
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_mel
    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
```

In [ ]:

```
# 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]))
```

In [ ]:

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

In [ ]:

```
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=
```

In [ ]:

```
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)
```

In [ ]:

```
model = alexnet_model()

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

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

```
Epoch 1/10
11/11 [==============================] - 78s 7s/step - loss: 2.6594 - accuracy: 0.0997
Epoch 2/10
11/11 [==============================] - 75s 7s/step - loss: 2.4658 - accuracy: 0.1121
Epoch 3/10
11/11 [==============================] - 75s 7s/step - loss: 2.4722 - accuracy: 0.1402
Epoch 4/10
11/11 [==============================] - 75s 7s/step - loss: 2.3171 - accuracy: 0.1776
Epoch 5/10
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: 0.23
```

36
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:
0.26
79
Epoch 10/10
11/11 [==============================] - 76s 7s/step - loss: 1.9726 - accuracy:
0.31
15
```

In [ ]:

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

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

Out[ ]:

```
[2.1748006343841553, 0.23364485800266266]
```

GOOGLENET

In [1]:

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

Mounted at /content/drive

CIFAR 10 DATASET

In [ ]:

```
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

    conv_3x3 = Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu',
    conv_3x3 = Conv2D(filters_3x3, (3, 3), padding='same', activation='relu', kernel

    conv_5x5 = Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu',
    conv_5x5 = Conv2D(filters_5x5, (5, 5), padding='same', activation='relu', kernel

    pool_proj = MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
    pool_proj = Conv2D(filters_pool_proj, (1, 1), padding='same', activation='relu',

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

    return output
```

In [ ]:

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

In [ ]:

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

x = Conv2D(64, (7, 7), padding='same', strides=(2, 2), activation='relu', name='conv x = MaxPool2D((3, 3), pad
x = Conv2D(64, (1, 1), padding='same', strides=(1, 1), activation='relu', name='conv x = Conv2D(192, (3, 3), p

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, 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)
```

In [ ]:

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

In [ ]:

```
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
```

In [ ]:

```
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(i

# 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
```

In [ ]:

```
X_train, y_train, X_test, y_test = load_cifar10_data(224, 224)
```

In [ ]:

```
model = Model(input_layer, [x, x1, x2], name='inception_v1')
```

In [ ]:

```
model.summary()
```

```
Model: "inception_v1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv_1_7x7/2 (Conv2D)           (None, 112, 112, 64) 9472        input_1[0][0]
__________________________________________________________________________________________________
max_pool_1_3x3/2 (MaxPooling2D) (None, 56, 56, 64)   0           conv_1_7x7/2[0][0]
__________________________________________________________________________________________________
conv_2a_3x3/1 (Conv2D)          (None, 56, 56, 64)   4160        max_pool_1_3x3/2[0]
[0]
__________________________________________________________________________________________________
conv_2b_3x3/1 (Conv2D)          (None, 56, 56, 192)  110784      conv_2a_3x3/1[0][0]
__________________________________________________________________________________________________
max_pool_2_3x3/2 (MaxPooling2D) (None, 28, 28, 192)  0           conv_2b_3x3/1[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 28, 28, 96)   18528       max_pool_2_3x3/2[0]
[0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 28, 28, 16)   3088        max_pool_2_3x3/2[0]
[0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 28, 28, 192)  0           max_pool_2_3x3/2[0]
[0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 28, 28, 64)   12352       max_pool_2_3x3/2[0]
[0]
__________________________________________________________________________________________________
```

conv2d_2 (Conv2D)	(None, 28, 28, 128)	110720	conv2d_1[0][0]
conv2d_4 (Conv2D)	(None, 28, 28, 32)	12832	conv2d_3[0][0]
conv2d_5 (Conv2D)	(None, 28, 28, 32)	6176	max_pooling2d[0][0]
inception_3a (Concatenate)	(None, 28, 28, 256)	0	conv2d[0][0] conv2d_2[0][0] conv2d_4[0][0] conv2d_5[0][0]
conv2d_7 (Conv2D)	(None, 28, 28, 128)	32896	inception_3a[0][0]
conv2d_9 (Conv2D)	(None, 28, 28, 32)	8224	inception_3a[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 28, 28, 256)	0	inception_3a[0][0]
conv2d_6 (Conv2D)	(None, 28, 28, 128)	32896	inception_3a[0][0]
conv2d_8 (Conv2D)	(None, 28, 28, 192)	221376	conv2d_7[0][0]
conv2d_10 (Conv2D)	(None, 28, 28, 96)	76896	conv2d_9[0][0]
conv2d_11 (Conv2D) [0]	(None, 28, 28, 64)	16448	max_pooling2d_1[0]
inception_3b (Concatenate)	(None, 28, 28, 480)	0	conv2d_6[0][0] conv2d_8[0][0] conv2d_10[0][0] conv2d_11[0][0]
max_pool_3_3x3/2 (MaxPooling2D)	(None, 14, 14, 480)	0	inception_3b[0][0]
conv2d_13 (Conv2D) [0]	(None, 14, 14, 96)	46176	max_pool_3_3x3/2[0]
conv2d_15 (Conv2D) [0]	(None, 14, 14, 16)	7696	max_pool_3_3x3/2[0]
max_pooling2d_2 (MaxPooling2D) [0]	(None, 14, 14, 480)	0	max_pool_3_3x3/2[0]
conv2d_12 (Conv2D) [0]	(None, 14, 14, 192)	92352	max_pool_3_3x3/2[0]
conv2d_14 (Conv2D)	(None, 14, 14, 208)	179920	conv2d_13[0][0]
conv2d_16 (Conv2D)	(None, 14, 14, 48)	19248	conv2d_15[0][0]

conv2d_17 (Conv2D) [0]	(None, 14, 14, 64)	30784	max_pooling2d_2[0]
inception_4a (Concatenate)	(None, 14, 14, 512)	0	conv2d_12[0][0] conv2d_14[0][0] conv2d_16[0][0] conv2d_17[0][0]
conv2d_20 (Conv2D)	(None, 14, 14, 112)	57456	inception_4a[0][0]
conv2d_22 (Conv2D)	(None, 14, 14, 24)	12312	inception_4a[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0	inception_4a[0][0]
conv2d_19 (Conv2D)	(None, 14, 14, 160)	82080	inception_4a[0][0]
conv2d_21 (Conv2D)	(None, 14, 14, 224)	226016	conv2d_20[0][0]
conv2d_23 (Conv2D)	(None, 14, 14, 64)	38464	conv2d_22[0][0]
conv2d_24 (Conv2D) [0]	(None, 14, 14, 64)	32832	max_pooling2d_3[0]
inception_4b (Concatenate)	(None, 14, 14, 512)	0	conv2d_19[0][0] conv2d_21[0][0] conv2d_23[0][0] conv2d_24[0][0]
conv2d_26 (Conv2D)	(None, 14, 14, 128)	65664	inception_4b[0][0]
conv2d_28 (Conv2D)	(None, 14, 14, 24)	12312	inception_4b[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 512)	0	inception_4b[0][0]
conv2d_25 (Conv2D)	(None, 14, 14, 128)	65664	inception_4b[0][0]
conv2d_27 (Conv2D)	(None, 14, 14, 256)	295168	conv2d_26[0][0]
conv2d_29 (Conv2D)	(None, 14, 14, 64)	38464	conv2d_28[0][0]
conv2d_30 (Conv2D) [0]	(None, 14, 14, 64)	32832	max_pooling2d_4[0]
inception_4c (Concatenate)	(None, 14, 14, 512)	0	conv2d_25[0][0] conv2d_27[0][0] conv2d_29[0][0] conv2d_30[0][0]
conv2d_32 (Conv2D)	(None, 14, 14, 144)	73872	inception_4c[0][0]

conv2d_34 (Conv2D)	(None, 14, 14, 32)	16416	inception_4c[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 512)	0	inception_4c[0][0]
conv2d_31 (Conv2D)	(None, 14, 14, 112)	57456	inception_4c[0][0]
conv2d_33 (Conv2D)	(None, 14, 14, 288)	373536	conv2d_32[0][0]
conv2d_35 (Conv2D)	(None, 14, 14, 64)	51264	conv2d_34[0][0]
conv2d_36 (Conv2D)	(None, 14, 14, 64)	32832	max_pooling2d_5[0][0]
inception_4d (Concatenate)	(None, 14, 14, 528)	0	conv2d_31[0][0] conv2d_33[0][0] conv2d_35[0][0] conv2d_36[0][0]
conv2d_39 (Conv2D)	(None, 14, 14, 160)	84640	inception_4d[0][0]
conv2d_41 (Conv2D)	(None, 14, 14, 32)	16928	inception_4d[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 14, 14, 528)	0	inception_4d[0][0]
conv2d_38 (Conv2D)	(None, 14, 14, 256)	135424	inception_4d[0][0]
conv2d_40 (Conv2D)	(None, 14, 14, 320)	461120	conv2d_39[0][0]
conv2d_42 (Conv2D)	(None, 14, 14, 128)	102528	conv2d_41[0][0]
conv2d_43 (Conv2D)	(None, 14, 14, 128)	67712	max_pooling2d_6[0][0]
inception_4e (Concatenate)	(None, 14, 14, 832)	0	conv2d_38[0][0] conv2d_40[0][0] conv2d_42[0][0] conv2d_43[0][0]
max_pool_4_3x3/2 (MaxPooling2D)	(None, 7, 7, 832)	0	inception_4e[0][0]
conv2d_45 (Conv2D)	(None, 7, 7, 160)	133280	max_pool_4_3x3/2[0][0]
conv2d_47 (Conv2D)	(None, 7, 7, 32)	26656	max_pool_4_3x3/2[0][0]
max_pooling2d_7 (MaxPooling2D)	(None, 7, 7, 832)	0	max_pool_4_3x3/2[0][0]
conv2d_44 (Conv2D)	(None, 7, 7, 256)	213248	max_pool_4_3x3/2[0]

[0]

conv2d_46 (Conv2D)	(None, 7, 7, 320)	461120	conv2d_45[0][0]
conv2d_48 (Conv2D)	(None, 7, 7, 128)	102528	conv2d_47[0][0]
conv2d_49 (Conv2D) [0]	(None, 7, 7, 128)	106624	max_pooling2d_7[0]
inception_5a (Concatenate)	(None, 7, 7, 832)	0	conv2d_44[0][0] conv2d_46[0][0] conv2d_48[0][0] conv2d_49[0][0]
conv2d_51 (Conv2D)	(None, 7, 7, 192)	159936	inception_5a[0][0]
conv2d_53 (Conv2D)	(None, 7, 7, 48)	39984	inception_5a[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 7, 7, 832)	0	inception_5a[0][0]
average_pooling2d (AveragePooli	(None, 4, 4, 512)	0	inception_4a[0][0]
average_pooling2d_1 (AveragePoo	(None, 4, 4, 528)	0	inception_4d[0][0]
conv2d_50 (Conv2D)	(None, 7, 7, 384)	319872	inception_5a[0][0]
conv2d_52 (Conv2D)	(None, 7, 7, 384)	663936	conv2d_51[0][0]
conv2d_54 (Conv2D)	(None, 7, 7, 128)	153728	conv2d_53[0][0]
conv2d_55 (Conv2D) [0]	(None, 7, 7, 128)	106624	max_pooling2d_8[0]
conv2d_18 (Conv2D) [0] [0]	(None, 4, 4, 128)	65664	average_pooling2d
conv2d_37 (Conv2D) [0] [0]	(None, 4, 4, 128)	67712	average_pooling2d_1
inception_5b (Concatenate)	(None, 7, 7, 1024)	0	conv2d_50[0][0] conv2d_52[0][0] conv2d_54[0][0] conv2d_55[0][0]
flatten (Flatten)	(None, 2048)	0	conv2d_18[0][0]
flatten_1 (Flatten)	(None, 2048)	0	conv2d_37[0][0]
avg_pool_5_3x3/1 (GlobalAverage	(None, 1024)	0	inception_5b[0][0]

dense (Dense)	(None, 1024)	2098176	flatten[0][0]
dense_1 (Dense)	(None, 1024)	2098176	flatten_1[0][0]
dropout_2 (Dropout) [0]	(None, 1024)	0	avg_pool_5_3x3/1[0]
dropout (Dropout)	(None, 1024)	0	dense[0][0]
dropout_1 (Dropout)	(None, 1024)	0	dense_1[0][0]
output (Dense)	(None, 10)	10250	dropout_2[0][0]
auxilliary_output_1 (Dense)	(None, 10)	10250	dropout[0][0]
auxilliary_output_2 (Dense)	(None, 10)	10250	dropout_1[0][0]

```
==================================================================================
==============
Total params: 10,334,030
Trainable params: 10,334,030
Non-trainable params: 0
```

In [ ]:

```python
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', 'categor
```

In [ ]:

```python
history = model.fit(X_train, [y_train, y_train, y_train], validation_data=(X_test, [
```

```
Epoch 1/10

Epoch 00001: LearningRateScheduler setting learning rate to 0.01.
20/20 [==============================] - 111s 3s/step - loss: 3.8503 -
output_loss: 2.4237 - auxilliary_output_1_loss: 2.3730 - auxilliary_output_2_loss: 2.3825
- outpu t_accuracy: 0.1000 - auxilliary_output_1_accuracy: 0.1092 -
auxilliary_output_2_accu racy: 0.0862 - val_loss: 3.6912 - val_output_loss: 2.3097 -
val_auxilliary_output_1_ loss: 2.3023 - val_auxilliary_output_2_loss: 2.3027 -
val_output_accuracy: 0.0990  -
val_auxilliary_output_1_accuracy: 0.0990 - val_auxilliary_output_2_accuracy: 0.1085
Epoch 2/10

Epoch 00002: LearningRateScheduler setting learning rate to 0.01.
20/20 [==============================] - 38s 2s/step - loss: 3.7276 - output_loss:
2.3437 - auxilliary_output_1_loss: 2.3055 - auxilliary_output_2_loss: 2.3076 - outpu
t_accuracy: 0.0990 - auxilliary_output_1_accuracy: 0.0982 - auxilliary_output_2_accu
```

racy: 0.1060 - val_loss: 3.7074 - val_output_loss: 2.3267 - val_auxilliary_output_1_ loss: 2.2999 - val_auxilliary_output_2_loss: 2.3022 - val_output_accuracy: 0.0925 - val_auxilliary_output_1_accuracy: 0.0995 - val_auxilliary_output_2_accuracy: 0.1080
Epoch 3/10

Epoch 00003: LearningRateScheduler setting learning rate to 0.01.
20/20 [==============================] - 38s 2s/step - loss: 3.7174 - output_loss: 2.3355 - auxilliary_output_1_loss: 2.3006 - auxilliary_output_2_loss: 2.3057 - outpu t_accuracy: 0.0986 - auxilliary_output_1_accuracy: 0.1138 - auxilliary_output_2_accu racy: 0.1068 - val_loss: 3.6837 - val_output_loss: 2.3036 - val_auxilliary_output_1_ loss: 2.2982 - val_auxilliary_output_2_loss: 2.3022 - val_output_accuracy: 0.1080 -
val_auxilliary_output_1_accuracy: 0.1770 - val_auxilliary_output_2_accuracy: 0.0980
Epoch 4/10

Epoch 00004: LearningRateScheduler setting learning rate to 0.01.
20/20 [==============================] - 38s 2s/step - loss: 3.7021 - output_loss: 2.3220 - auxilliary_output_1_loss: 2.2979 - auxilliary_output_2_loss: 2.3026 - outpu t_accuracy: 0.0998 - auxilliary_output_1_accuracy: 0.1270 - auxilliary_output_2_accu racy: 0.1080 - val_loss: 3.6890 - val_output_loss: 2.3109 - val_auxilliary_output_1_ loss: 2.2924 - val_auxilliary_output_2_loss: 2.3010 - val_output_accuracy: 0.1085 -
val_auxilliary_output_1_accuracy: 0.1725 - val_auxilliary_output_2_accuracy: 0.1310
Epoch 5/10

Epoch 00005: LearningRateScheduler setting learning rate to 0.01.
20/20 [==============================] - 38s 2s/step - loss: 3.6894 - output_loss: 2.3129 - auxilliary_output_1_loss: 2.2875 - auxilliary_output_2_loss: 2.3009 - outpu t_accuracy: 0.1108 - auxilliary_output_1_accuracy: 0.1494 - auxilliary_output_2_accu racy: 0.1074 - val_loss: 3.6642 - val_output_loss: 2.2924 - val_auxilliary_output_1_ loss: 2.2741 - val_auxilliary_output_2_loss: 2.2986 - val_output_accuracy: 0.1085 -
val_auxilliary_output_1_accuracy: 0.2110 - val_auxilliary_output_2_accuracy: 0.1085
Epoch 6/10

Epoch 00006: LearningRateScheduler setting learning rate to 0.01.
20/20 [==============================] - 38s 2s/step - loss: 3.6344 - output_loss: 2.2691 - auxilliary_output_1_loss: 2.2596 - auxilliary_output_2_loss: 2.2915 - outpu t_accuracy: 0.1482 - auxilliary_output_1_accuracy: 0.1660 - auxilliary_output_2_accu racy: 0.1326 - val_loss: 3.5432 - val_output_loss: 2.1983 - val_auxilliary_output_1_ loss: 2.2150 - val_auxilliary_output_2_loss: 2.2677 - val_output_accuracy: 0.1730 -
val_auxilliary_output_1_accuracy: 0.1900 - val_auxilliary_output_2_accuracy: 0.1645
Epoch 7/10

Epoch 00007: LearningRateScheduler setting learning rate to 0.01.
20/20 [==============================] - 38s 2s/step - loss: 3.5920 - output_loss: 2.2447 - auxilliary_output_1_loss: 2.2258 - auxilliary_output_2_loss: 2.2653 - outpu t_accuracy: 0.1558 - auxilliary_output_1_accuracy: 0.1642 - auxilliary_output_2_accu racy: 0.1410 - val_loss: 3.4822 - val_output_loss: 2.1881 - val_auxilliary_output_1_ loss: 2.1318 - val_auxilliary_output_2_loss: 2.1821 - val_output_accuracy: 0.1675 -
val_auxilliary_output_1_accuracy: 0.2250 - val_auxilliary_output_2_accuracy: 0.1905
Epoch 8/10

Epoch 00008: LearningRateScheduler setting learning rate to 0.0096.
20/20 [==============================] - 38s 2s/step - loss: 3.4126 - output_loss: 2.1287 - auxilliary_output_1_loss: 2.1271 - auxilliary_output_2_loss: 2.1526 - outpu t_accuracy: 0.1820 - auxilliary_output_1_accuracy: 0.2020 - auxilliary_output_2_accu racy: 0.1834 - val_loss: 3.3223 - val_output_loss: 2.0856 - val_auxilliary_output_1_ loss: 2.0574 - val_auxilliary_output_2_loss: 2.0650 - val_output_accuracy: 0.2305 -
val_auxilliary_output_1_accuracy: 0.2400 - val_auxilliary_output_2_accuracy: 0.2240
Epoch 9/10

Epoch 00009: LearningRateScheduler setting learning rate to 0.0096.
20/20 [==============================] - 38s 2s/step - loss: 3.3359 - output_loss: 2.0873 - auxilliary_output_1_loss: 2.0735 - auxilliary_output_2_loss: 2.0886 - outpu t_accuracy: 0.2194 - auxilliary_output_1_accuracy: 0.2160 - auxilliary_output_2_accu racy: 0.2060 - val_loss: 3.2636 - val_output_loss: 2.0530 - val_auxilliary_output_1_ loss: 2.0111 - val_auxilliary_output_2_loss: 2.0244 -

val_output_accuracy: 0.2470 -
val_auxilliary_output_1_accuracy: 0.2630 - val_auxilliary_output_2_accuracy: 0.2585
Epoch 10/10

Epoch 00010: LearningRateScheduler setting learning rate to 0.0096.

20/20 [==============================] - 38s 2s/step - loss: 3.2849 - output_loss: 2.0561 - auxilliary_output_1_loss: 2.0420 - auxilliary_output_2_loss: 2.0541 - outpu t_accuracy: 0.2276 - auxilliary_output_1_accuracy: 0.2414 - auxilliary_output_2_accu racy: 0.2194 - val_loss: 3.2485 - val_output_loss: 2.0485 - val_auxilliary_output_1_ loss: 1.9923 - val_auxilliary_output_2_loss: 2.0076 - val_output_accuracy: 0.2355 - val_auxilliary_output_1_accuracy: 0.2735 - val_auxilliary_output_2_accuracy: 0.2660

MNIST DATASET

In [ ]:

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

In [ ]:

```
(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]
```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn ist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step

In [ ]:

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

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

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

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

In [ ]:

```
inp = layers.Input(shape=(32, 32, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="b

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

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

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)
```

In [ ]:

```
model = Model(inputs = inp, outputs = [out, aux1, aux2])
```

In [ ]:

```
model.compile(optimizer='adam', loss=[losses.sparse_categorical_crossentropy, losses
```

In [ ]:

```
history = model.fit(x_train, [y_train, y_train, y_train], validation_data=(x_val, [y
```

```
Epoch 1/10
907/907 [==============================] - 333s 367ms/step - loss: 0.1901 - dense_6_
loss: 0.1282 - dense_3_loss: 0.0972 - dense_5_loss: 0.1092 - dense_6_accuracy: 0.962
0 - dense_3_accuracy: 0.9702 - dense_5_accuracy: 0.9671 - val_loss: 0.1184 -
val_den se_6_loss: 0.0799 - val_dense_3_loss: 0.0618 - val_dense_5_loss: 0.0665 -
val_dense_ 6_accuracy: 0.9750 - val_dense_3_accuracy: 0.9835 -
val_dense_5_accuracy: 0.9825
Epoch 2/10
907/907 [==============================] - 333s 367ms/step - loss: 0.1405 - dense_6_
loss: 0.0927 - dense_3_loss: 0.0753 - dense_5_loss: 0.0842 - dense_6_accuracy: 0.972
3 - dense_3_accuracy: 0.9773 - dense_5_accuracy: 0.9744 - val_loss: 0.0731 -
val_den se_6_loss: 0.0478 - val_dense_3_loss: 0.0377 - val_dense_5_loss: 0.0467 -
val_dense_ 6_accuracy: 0.9865 - val_dense_3_accuracy: 0.9915 -
val_dense_5_accuracy: 0.9870
Epoch 3/10
907/907 [==============================] - 333s 367ms/step - loss: 0.1204 - dense_6_
loss: 0.0804 - dense_3_loss: 0.0632 - dense_5_loss: 0.0702 - dense_6_accuracy: 0.975
7 - dense_3_accuracy: 0.9810 - dense_5_accuracy: 0.9792 - val_loss: 0.0618 -
val_den se_6_loss: 0.0435 - val_dense_3_loss: 0.0331 - val_dense_5_loss: 0.0279 -
val_dense_ 6_accuracy: 0.9885 - val_dense_3_accuracy: 0.9920 -
val_dense_5_accuracy: 0.9945
Epoch 4/10
907/907 [==============================] - 332s 367ms/step - loss: 0.0971 - dense_6_
loss: 0.0632 - dense_3_loss: 0.0545 - dense_5_loss: 0.0586 - dense_6_accuracy: 0.980
8 - dense_3_accuracy: 0.9832 - dense_5_accuracy: 0.9826 - val_loss: 0.0610 -
val_den se_6_loss: 0.0390 - val_dense_3_loss: 0.0372 - val_dense_5_loss: 0.0360 -
val_dense_ 6_accuracy: 0.9915 - val_dense_3_accuracy: 0.9915 -
```

val_dense_5_accuracy: 0.9920
Epoch 5/10

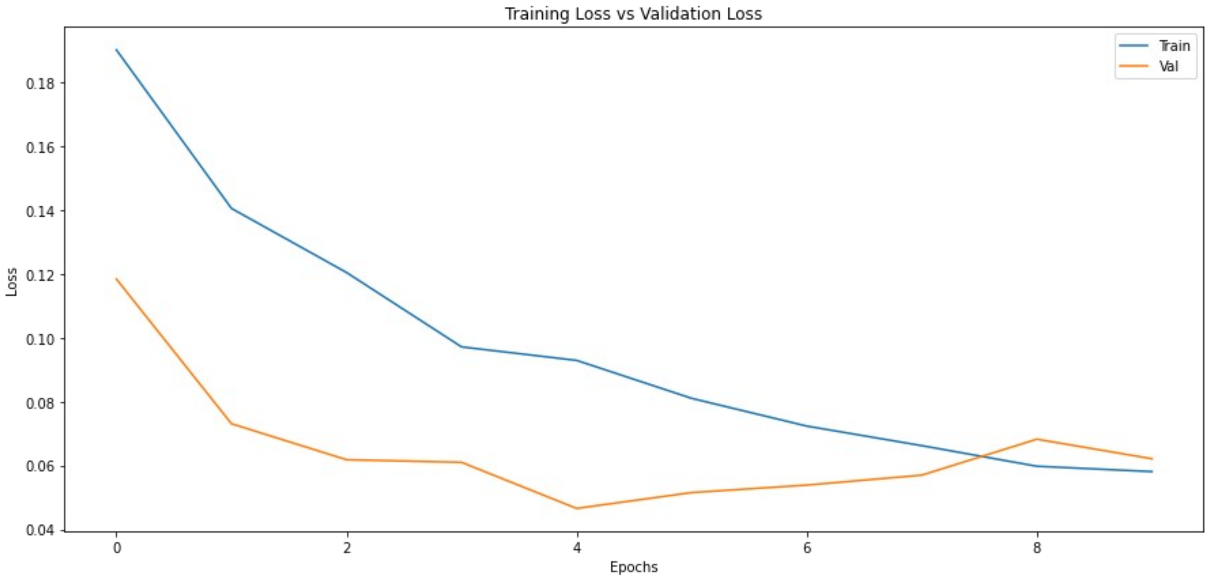
```
907/907 [==============================] - 332s 366ms/step - loss: 0.0929 - dense_6_loss: 0.0610 - dense_3_loss: 0.0507 - dense_5_loss: 0.0558 - dense_6_accuracy: 0.9815 - dense_3_accuracy: 0.9849 - dense_5_accuracy: 0.9831 - val_loss: 0.0466 - val_dense_6_loss: 0.0296 - val_dense_3_loss: 0.0311 - val_dense_5_loss: 0.0256 - val_dense_6_accuracy: 0.9925 - val_dense_3_accuracy: 0.9935 - val_dense_5_accuracy: 0.9920
Epoch 6/10
907/907 [==============================] - 331s 365ms/step - loss: 0.0810 - dense_6_loss: 0.0528 - dense_3_loss: 0.0456 - dense_5_loss: 0.0484 - dense_6_accuracy: 0.9840 - dense_3_accuracy: 0.9859 - dense_5_accuracy: 0.9846 - val_loss: 0.0515 - val_dense_6_loss: 0.0300 - val_dense_3_loss: 0.0386 - val_dense_5_loss: 0.0332 - val_dense_6_accuracy: 0.9915 - val_dense_3_accuracy: 0.9920 - val_dense_5_accuracy: 0.9925
Epoch 7/10
907/907 [==============================] - 330s 364ms/step - loss: 0.0724 - dense_6_loss: 0.0467 - dense_3_loss: 0.0411 - dense_5_loss: 0.0444 - dense_6_accuracy: 0.9856 - dense_3_accuracy: 0.9875 - dense_5_accuracy: 0.9867 - val_loss: 0.0539 - val_dense_6_loss: 0.0373 - val_dense_3_loss: 0.0303 - val_dense_5_loss: 0.0250 - val_dense_6_accuracy: 0.9910 - val_dense_3_accuracy: 0.9960 - val_dense_5_accuracy: 0.9945
Epoch 8/10
907/907 [==============================] - 332s 366ms/step - loss: 0.0662 - dense_6_loss: 0.0428 - dense_3_loss: 0.0373 - dense_5_loss: 0.0407 - dense_6_accuracy: 0.9866 - dense_3_accuracy: 0.9882 - dense_5_accuracy: 0.9879 - val_loss: 0.0570 - val_dense_6_loss: 0.0375 - val_dense_3_loss: 0.0326 - val_dense_5_loss: 0.0325 - val_dense_6_accuracy: 0.9905 - val_dense_3_accuracy: 0.9945 - val_dense_5_accuracy: 0.9940
Epoch 9/10
907/907 [==============================] - 330s 364ms/step - loss: 0.0598 - dense_6_loss: 0.0374 - dense_3_loss: 0.0366 - dense_5_loss: 0.0379 - dense_6_accuracy: 0.9881 - dense_3_accuracy: 0.9890 - dense_5_accuracy: 0.9885 - val_loss: 0.0682 - val_dense_6_loss: 0.0421 - val_dense_3_loss: 0.0338 - val_dense_5_loss: 0.0533 - val_dense_6_accuracy: 0.9925 - val_dense_3_accuracy: 0.9930 - val_dense_5_accuracy: 0.9895
Epoch 10/10
907/907 [==============================] - 331s 365ms/step - loss: 0.0581 - dense_6_loss: 0.0371 - dense_3_loss: 0.0322 - dense_5_loss: 0.0378 - dense_6_accuracy: 0.9887 - dense_3_accuracy: 0.9905 - dense_5_accuracy: 0.9887 - val_loss: 0.0621 - val_dense_6_loss: 0.0433 - val_dense_3_loss: 0.0319 - val_dense_5_loss: 0.0309 - val_dense_6_accuracy: 0.9890 - val_dense_3_accuracy: 0.9930 - val_dense_5_accuracy: 0.9925
```

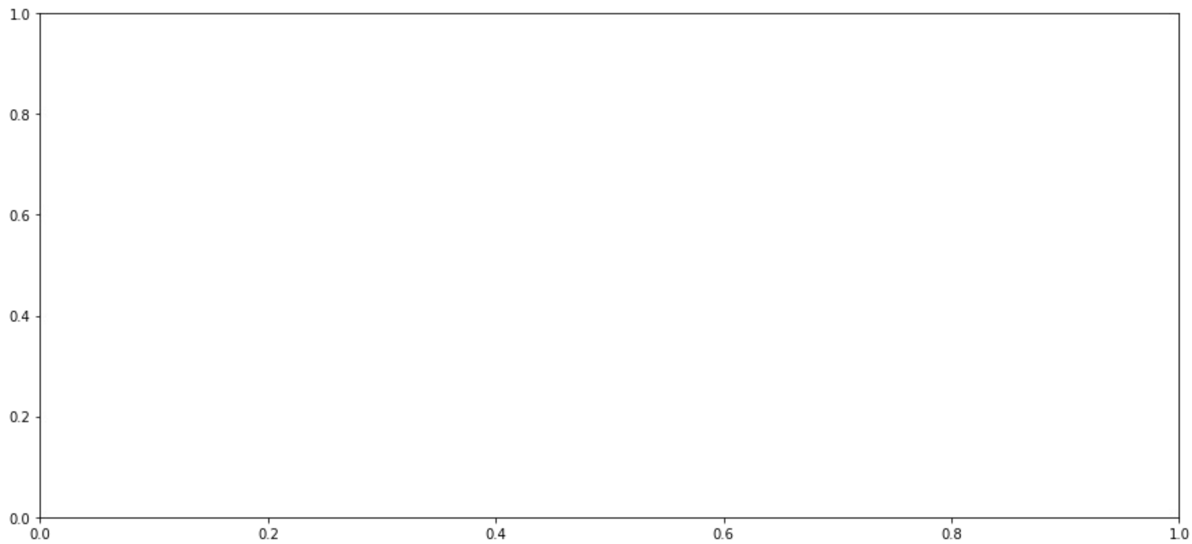
In [ ]:

```
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'])
```

Out[ ]:

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





In [ ]:

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

313/313 [==============================] - 24s 69ms/step - loss: 0.0543 - dense_6_loss: 0.0382 - dense_3_loss: 0.0257 - dense_5_loss: 0.0279 - dense_6_accuracy: 0.9874 - dense_3_accuracy: 0.9926 - dense_5_accuracy: 0.9903

Out[ ]:

```
[0.05425573140382767,
 0.03818700090050697,
 0.02569129876792431,
 0.027871087193489075,
 0.9873999953269958,
 0.9926000237464905,
 0.9902999997138977]
```

SAVEE Dataset

In [ ]:

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

In [3]:

```
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_mel
    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
```

In [4]:

```
# 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]))
```

In [5]:

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

In [6]:

```
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=
```

In [7]:

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

In [8]:

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

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

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

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

In [9]:

```
inp = layers.Input(shape=(157, 320, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="b

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

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

In [10]:

```
model = Model(inputs = inp, outputs = [out, aux1, aux2])

model.compile(optimizer='adam', loss=[losses.sparse_categorical_crossentropy, losses
```

In [11]:

```
history = model.fit(X_train, [y_train, y_train, y_train], validation_data=(X_test, [
```

```
Epoch 1/30
5/5 [==============================] - 43s 1s/step - loss: 3.7075 - dense_4_loss:
2. 3287 - dense_1_loss: 2.2312 - dense_3_loss: 2.3650 - dense_4_accuracy: 0.1979 -
dens e_1_accuracy: 0.1771 - dense_3_accuracy: 0.1840 - val_loss: 3.3708 -
val_dense_4_los s: 2.1198 - val_dense_1_loss: 2.0493 - val_dense_3_loss: 2.1207 -
val_dense_4_accura cy: 0.1042 - val_dense_1_accuracy: 0.1354 -
val_dense_3_accuracy: 0.1042
Epoch 2/30
5/5 [==============================] - 2s 512ms/step - loss: 3.4439 - dense_4_loss:
2.1523 - dense_1_loss: 2.1182 - dense_3_loss: 2.1872 - dense_4_accuracy: 0.0938 - de
nse_1_accuracy: 0.1354 - dense_3_accuracy: 0.1111 - val_loss: 3.3694 - val_dense_4_l
oss: 2.0954 - val_dense_1_loss: 2.0870 - val_dense_3_loss: 2.1597 - val_dense_4_accu
racy: 0.1302 - val_dense_1_accuracy: 0.1302 - val_dense_3_accuracy: 0.1042
Epoch 3/30
5/5 [==============================] - 2s 510ms/step - loss: 3.2806 - dense_4_loss:
2.0352 - dense_1_loss: 2.0429 - dense_3_loss: 2.1084 - dense_4_accuracy: 0.1528 - de
nse_1_accuracy: 0.1285 - dense_3_accuracy: 0.1354 - val_loss: 3.0642 - val_dense_4_l
oss: 1.9016 - val_dense_1_loss: 1.9144 - val_dense_3_loss: 1.9607 - val_dense_4_accu
racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.1042
Epoch 4/30
5/5 [==============================] - 2s 449ms/step - loss: 3.1873 - dense_4_loss:
1.9948 - dense_1_loss: 1.9546 - dense_3_loss: 2.0205 - dense_4_accuracy: 0.2153 - de
nse_1_accuracy: 0.2361 - dense_3_accuracy: 0.1944 - val_loss: 3.1250 - val_dense_4_l
oss: 1.9474 - val_dense_1_loss: 1.9391 - val_dense_3_loss: 1.9860 - val_dense_4_accu
racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 5/30
5/5 [==============================] - 2s 450ms/step - loss: 3.1442 - dense_4_loss:
1.9528 - dense_1_loss: 1.9735 - dense_3_loss: 1.9977 - dense_4_accuracy: 0.2535 - de
nse_1_accuracy: 0.2361 - dense_3_accuracy: 0.2361 - val_loss: 3.0796 - val_dense_4_l
oss: 1.9226 - val_dense_1_loss: 1.9217 - val_dense_3_loss: 1.9350 - val_dense_4_accu
racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 6/30
5/5 [==============================] - 2s 443ms/step - loss: 3.1400 - dense_4_loss:
1.9551 - dense_1_loss: 1.9676 - dense_3_loss: 1.9821 - dense_4_accuracy: 0.2396 - de
nse_1_accuracy: 0.2396 - dense_3_accuracy: 0.2361 - val_loss: 3.0815 - val_dense_4_l
oss: 1.9252 - val_dense_1_loss: 1.9249 - val_dense_3_loss: 1.9293 - val_dense_4_accu
racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 7/30
5/5 [==============================] - 2s 449ms/step - loss: 3.1226 -
dense_4_loss:
```

1.9453 - dense_1_loss: 1.9527 - dense_3_loss: 1.9717 - dense_4_accuracy: 0.2431 - de nse_1_accuracy: 0.2188 - dense_3_accuracy: 0.2361 - val_loss: 3.0612 - val_dense_4_l oss: 1.9125 - val_dense_1_loss: 1.9107 - val_dense_3_loss: 1.9184 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 8/30
5/5 [==============================] - 2s 507ms/step - loss: 3.1009 - dense_4_loss: 1.9255 - dense_1_loss: 1.9675 - dense_3_loss: 1.9506 - dense_4_accuracy: 0.2465 - de nse_1_accuracy: 0.2326 - dense_3_accuracy: 0.2292 - val_loss: 3.0480 - val_dense_4_l oss: 1.9020 - val_dense_1_loss: 1.9074 - val_dense_3_loss: 1.9124 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 9/30
5/5 [==============================] - 2s 448ms/step - loss: 3.0927 - dense_4_loss: 1.9269 - dense_1_loss: 1.9428 - dense_3_loss: 1.9432 - dense_4_accuracy: 0.2396 - de nse_1_accuracy: 0.2257 - dense_3_accuracy: 0.2361 - val_loss: 3.0549 - val_dense_4_l oss: 1.9050 - val_dense_1_loss: 1.9166 - val_dense_3_loss: 1.9165 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 10/30
5/5 [==============================] - 2s 512ms/step - loss: 3.0785 - dense_4_loss: 1.9220 - dense_1_loss: 1.9162 - dense_3_loss: 1.9389 - dense_4_accuracy: 0.2500 - de nse_1_accuracy: 0.2292 - dense_3_accuracy: 0.2431 - val_loss: 3.0163 - val_dense_4_l oss: 1.8823 - val_dense_1_loss: 1.8911 - val_dense_3_loss: 1.8887 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 11/30
5/5 [==============================] - 2s 511ms/step - loss: 3.0487 - dense_4_loss: 1.8934 - dense_1_loss: 1.9367 - dense_3_loss: 1.9144 - dense_4_accuracy: 0.2431 - de nse_1_accuracy: 0.2361 - dense_3_accuracy: 0.2326 - val_loss: 2.9904 - val_dense_4_l oss: 1.8704 - val_dense_1_loss: 1.8717 - val_dense_3_loss: 1.8616 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 12/30
5/5 [==============================] - 2s 510ms/step - loss: 3.0880 - dense_4_loss: 1.9187 - dense_1_loss: 1.9478 - dense_3_loss: 1.9501 - dense_4_accuracy: 0.2292 - de nse_1_accuracy: 0.2431 - dense_3_accuracy: 0.2292 - val_loss: 2.9760 - val_dense_4_l oss: 1.8550 - val_dense_1_loss: 1.8789 - val_dense_3_loss: 1.8577 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 13/30
5/5 [==============================] - 2s 516ms/step - loss: 3.0064 - dense_4_loss: 1.8649 - dense_1_loss: 1.9037 - dense_3_loss: 1.9012 - dense_4_accuracy: 0.2500 - de nse_1_accuracy: 0.2569 - dense_3_accuracy: 0.2361 - val_loss: 2.9414 - val_dense_4_l oss: 1.8328 - val_dense_1_loss: 1.8465 - val_dense_3_loss: 1.8491 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 14/30
5/5 [==============================] - 2s 453ms/step - loss: 2.9520 - dense_4_loss: 1.8353 - dense_1_loss: 1.8597 - dense_3_loss: 1.8627 - dense_4_accuracy: 0.2326 - de nse_1_accuracy: 0.2500 - dense_3_accuracy: 0.2396 - val_loss: 2.7950 - val_dense_4_l oss: 1.7310 - val_dense_1_loss: 1.7772 - val_dense_3_loss: 1.7697 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.2604 - val_dense_3_accuracy: 0.2604
Epoch 15/30
5/5 [==============================] - 2s 447ms/step - loss: 2.9183 - dense_4_loss: 1.8202 - dense_1_loss: 1.8169 - dense_3_loss: 1.8435 - dense_4_accuracy: 0.2431 - de nse_1_accuracy: 0.2986 - dense_3_accuracy: 0.2500 - val_loss: 2.7127 - val_dense_4_l oss: 1.6981 - val_dense_1_loss: 1.6853 - val_dense_3_loss: 1.6967 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.3646 - val_dense_3_accuracy: 0.2604
Epoch 16/30
5/5 [==============================] - 2s 450ms/step - loss: 2.9170 - dense_4_loss: 1.8502 - dense_1_loss: 1.7662 - dense_3_loss: 1.7898 - dense_4_accuracy: 0.2049 - de nse_1_accuracy: 0.3160 - dense_3_accuracy: 0.2674 - val_loss: 2.6895 - val_dense_4_l oss: 1.6972 - val_dense_1_loss: 1.6399 - val_dense_3_loss: 1.6678 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.3490 - val_dense_3_accuracy: 0.2604
Epoch 17/30
5/5 [==============================] - 2s 450ms/step - loss: 2.8261 - dense_4_loss: 1.7665 - dense_1_loss: 1.7318 - dense_3_loss: 1.8001 - dense_4_accuracy: 0.2361 - de nse_1_accuracy: 0.3299 - dense_3_accuracy: 0.2917 - val_loss: 2.7954 - val_dense_4_l oss: 1.7517 - val_dense_1_loss: 1.7163 - val_dense_3_loss: 1.7627 - val_dense_4_accu racy: 0.2604 - val_dense_1_accuracy: 0.3229 - val_dense_3_accuracy: 0.3438
Epoch 18/30
5/5 [==============================] - 2s 452ms/step - loss: 2.7408 - dense_4_loss: 1.7274 - dense_1_loss: 1.6739 - dense_3_loss: 1.7039 - dense_4_accuracy: 0.2917 - de nse_1_accuracy: 0.3333 - dense_3_accuracy: 0.3229 - val_loss: 2.5295 - val_dense_4_l oss: 1.6045 - val_dense_1_loss: 1.5372 - val_dense_3_loss: 1.5459 - val_dense_4_accu

racy: 0.3698 - val_dense_1_accuracy: 0.3958 - val_dense_3_accuracy: 0.3698
Epoch 19/30
5/5 [==============================] - 2s 515ms/step - loss: 2.6459 - dense_4_loss: 1.6658 - dense_1_loss: 1.6213 - dense_3_loss: 1.6459 - dense_4_accuracy: 0.3472 - dense_1_accuracy: 0.3507 - dense_3_accuracy: 0.3542 - val_loss: 2.3957 - val_dense_4_loss: 1.4989 - val_dense_1_loss: 1.4865 - val_dense_3_loss: 1.5027 - val_dense_4_accuracy: 0.3802 - val_dense_1_accuracy: 0.3906 - val_dense_3_accuracy: 0.3750
Epoch 20/30
5/5 [==============================] - 2s 453ms/step - loss: 2.5887 - dense_4_loss: 1.6192 - dense_1_loss: 1.5936 - dense_3_loss: 1.6383 - dense_4_accuracy: 0.3333 - dense_1_accuracy: 0.3368 - dense_3_accuracy: 0.3229 - val_loss: 2.3250 - val_dense_4_loss: 1.4582 - val_dense_1_loss: 1.4467 - val_dense_3_loss: 1.4426 - val_dense_4_accuracy: 0.3906 - val_dense_1_accuracy: 0.3802 - val_dense_3_accuracy: 0.3854
Epoch 21/30
5/5 [==============================] - 2s 455ms/step - loss: 2.4929 - dense_4_loss: 1.5598 - dense_1_loss: 1.5861 - dense_3_loss: 1.5242 - dense_4_accuracy: 0.3403 - dense_1_accuracy: 0.3472 - dense_3_accuracy: 0.4062 - val_loss: 2.8551 - val_dense_4_loss: 1.8494 - val_dense_1_loss: 1.6330 - val_dense_3_loss: 1.7195 - val_dense_4_accuracy: 0.2396 - val_dense_1_accuracy: 0.3281 - val_dense_3_accuracy: 0.3021
Epoch 22/30
5/5 [==============================] - 2s 450ms/step - loss: 2.6071 - dense_4_loss: 1.6355 - dense_1_loss: 1.5847 - dense_3_loss: 1.6540 - dense_4_accuracy: 0.3333 - dense_1_accuracy: 0.3368 - dense_3_accuracy: 0.3160 - val_loss: 2.3769 - val_dense_4_loss: 1.4988 - val_dense_1_loss: 1.4541 - val_dense_3_loss: 1.4730 - val_dense_4_accuracy: 0.3802 - val_dense_1_accuracy: 0.4010 - val_dense_3_accuracy: 0.3854
Epoch 23/30
5/5 [==============================] - 2s 512ms/step - loss: 2.4752 - dense_4_loss: 1.5466 - dense_1_loss: 1.5351 - dense_3_loss: 1.5600 - dense_4_accuracy: 0.3993 - dense_1_accuracy: 0.4167 - dense_3_accuracy: 0.3681 - val_loss: 2.2871 - val_dense_4_loss: 1.4387 - val_dense_1_loss: 1.4061 - val_dense_3_loss: 1.4219 - val_dense_4_accuracy: 0.3958 - val_dense_1_accuracy: 0.3906 - val_dense_3_accuracy: 0.3958
Epoch 24/30
5/5 [==============================] - 2s 512ms/step - loss: 2.4079 - dense_4_loss: 1.5108 - dense_1_loss: 1.4736 - dense_3_loss: 1.5168 - dense_4_accuracy: 0.3681 - dense_1_accuracy: 0.3819 - dense_3_accuracy: 0.3819 - val_loss: 2.2459 - val_dense_4_loss: 1.4099 - val_dense_1_loss: 1.3882 - val_dense_3_loss: 1.3987 - val_dense_4_accuracy: 0.3906 - val_dense_1_accuracy: 0.3854 - val_dense_3_accuracy: 0.3854
Epoch 25/30
5/5 [==============================] - 2s 518ms/step - loss: 2.4873 - dense_4_loss: 1.5496 - dense_1_loss: 1.5495 - dense_3_loss: 1.5763 - dense_4_accuracy: 0.3542 - dense_1_accuracy: 0.4062 - dense_3_accuracy: 0.3299 - val_loss: 2.2399 - val_dense_4_loss: 1.4034 - val_dense_1_loss: 1.3862 - val_dense_3_loss: 1.4023 - val_dense_4_accuracy: 0.3802 - val_dense_1_accuracy: 0.3958 - val_dense_3_accuracy: 0.3854
Epoch 26/30
5/5 [==============================] - 2s 454ms/step - loss: 2.3686 - dense_4_loss: 1.4740 - dense_1_loss: 1.4742 - dense_3_loss: 1.5078 - dense_4_accuracy: 0.3576 - dense_1_accuracy: 0.3993 - dense_3_accuracy: 0.3646 - val_loss: 2.2973 - val_dense_4_loss: 1.4385 - val_dense_1_loss: 1.4257 - val_dense_3_loss: 1.4371 - val_dense_4_accuracy: 0.3802 - val_dense_1_accuracy: 0.3750 - val_dense_3_accuracy: 0.3854
Epoch 27/30
5/5 [==============================] - 2s 456ms/step - loss: 2.2876 - dense_4_loss: 1.4199 - dense_1_loss: 1.4513 - dense_3_loss: 1.4410 - dense_4_accuracy: 0.3889 - dense_1_accuracy: 0.3993 - dense_3_accuracy: 0.3924 - val_loss: 2.1548 - val_dense_4_loss: 1.3449 - val_dense_1_loss: 1.3436 - val_dense_3_loss: 1.3559 - val_dense_4_accuracy: 0.3802 - val_dense_1_accuracy: 0.3906 - val_dense_3_accuracy: 0.4010
Epoch 28/30
5/5 [==============================] - 2s 456ms/step - loss: 2.2950 - dense_4_loss: 1.4286 - dense_1_loss: 1.4445 - dense_3_loss: 1.4435 - dense_4_accuracy: 0.3889 - dense_1_accuracy: 0.4097 - dense_3_accuracy: 0.3715 - val_loss: 2.1400 - val_dense_4_loss: 1.3402 - val_dense_1_loss: 1.3266 - val_dense_3_loss: 1.3395 - val_dense_4_accuracy: 0.3958 - val_dense_1_accuracy: 0.3958 - val_dense_3_accuracy: 0.4219
Epoch 29/30
5/5 [==============================] - 2s 456ms/step - loss: 2.1741 - dense_4_loss: 1.3459 - dense_1_loss: 1.4084 - dense_3_loss: 1.3524 - dense_4_accuracy: 0.4132 - dense_1_accuracy: 0.4062 - dense_3_accuracy: 0.4375 - val_loss: 2.1231 - val_dense_4_loss: 1.3301 - val_dense_1_loss: 1.3179 - val_dense_3_loss: 1.3254 - val_dense_4_accuracy: 0.4010 - val_dense_1_accuracy: 0.4323 - val_dense_3_accuracy: 0.4219
Epoch 30/30
5/5 [==============================] - 2s 454ms/step - loss: 2.1850 -
dense_4_loss:

1.3632 - dense_1_loss: 1.3858 - dense_3_loss: 1.3533 - dense_4_accuracy: 0.3681 - de nse_1_accuracy: 0.4167 - dense_3_accuracy: 0.3924 - val_loss: 2.3244 - val_dense_4_l oss: 1.4581 - val_dense_1_loss: 1.4512 - val_dense_3_loss: 1.4365 - val_dense_4_accu racy: 0.3542 - val_dense_1_accuracy: 0.3333 - val_dense_3_accuracy: 0.3802

In [12]:
```
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()
```

In [13]:
```
model.evaluate(X_test, y_test)
```

6/6 [==============================] - 1s 92ms/step - loss: 2.3244 - dense_4_loss: 1.4581 - dense_1_loss: 1.4512 - dense_3_loss: 1.4365 - dense_4_accuracy: 0.3542 - de nse_1_accuracy: 0.3333 - dense_3_accuracy: 0.3802

Out[13]:
```
[2.324364423751831,
 1.458065390586853,
 1.4511628150939941,
 1.4365006685256958,
 0.3541666567325592,
 0.3333333432674408,
 0.3802083432674408]
```

EmoDB Dataset

In [14]:
```
!unzip "/content/drive/MyDrive/EmoDB.zip"
```

Archive: /content/drive/MyDrive/EmoDB.zip

creating: lablaut/
inflating: lablaut/14a04Lbxx.lablaut
inflating: lablaut/03a07Fbxx.lablaut
inflating: lablaut/16b03Faxx.lablaut
inflating: lablaut/15a05Lbxx.lablaut
inflating: lablaut/16a02Lbxx.lablaut
inflating: lablaut/14a04Aaxx.lablaut
inflating: lablaut/12b03Taxx.lablaut
inflating: lablaut/16a05Laxx.lablaut
inflating: lablaut/16b03Taxx.lablaut
inflating: lablaut/11a05Fcxx.lablaut
inflating: lablaut/03a02Taxx.lablaut
inflating: lablaut/09b03Edxx.lablaut
inflating: lablaut/12a01Fbxx.lablaut
inflating: lablaut/10a02Faxx.lablaut
inflating: lablaut/08a04Tbxx.lablaut
inflating: lablaut/09a05Tbxx.lablaut
inflating:
lablaut/09b03Wbxx.lablaut
inflating: lablaut/03b03Nbxx.lablaut
inflating:
lablaut/08b09Waxx.lablaut
inflating: lablaut/09b10Ndxx.lablaut
inflating: lablaut/08a07Taxx.lablaut
inflating:
lablaut/10b09Wbxx.lablaut
inflating:
lablaut/08b01Waxx.lablaut
inflating: lablaut/13a07Tcxx.lablaut
inflating: lablaut/15a07Faxx.lablaut
inflating: lablaut/03b02Naxx.lablaut
inflating: lablaut/16a07Tdxx.lablaut
inflating:
lablaut/11b02Wbxx.lablaut
inflating: lablaut/14a04Tcxx.lablaut
inflating: lablaut/13b10Ecxx.lablaut
inflating: lablaut/16a04Faxx.lablaut
inflating:
lablaut/11b09Waxx.lablaut
inflating: lablaut/11a07Taxx.lablaut
inflating: lablaut/16a02Tcxx.lablaut
inflating:
lablaut/13b10Waxx.lablaut
inflating:
lablaut/12b02Waxx.lablaut
inflating:
lablaut/16b10Wbxx.lablaut
inflating: lablaut/14b10Ebxx.lablaut
inflating: lablaut/15b01Ecxx.lablaut
inflating: lablaut/11b01Ncxx.lablaut
inflating: lablaut/16a05Tbxx.lablaut
inflating: lablaut/11b10Ldxx.lablaut
inflating: lablaut/13b10Ncxx.lablaut
inflating:
lablaut/03a07Wcxx.lablaut
inflating:
lablaut/16b03Wbxx.lablaut
inflating: lablaut/16a01Tbxx.lablaut
inflating: lablaut/15b02Ndxx.lablaut
inflating: lablaut/16b09Ebxx.lablaut
inflating: lablaut/08b01Lbxx.lablaut
inflating: lablaut/16b01Ebxx.lablaut
inflating: lablaut/10b02Naxx.lablaut
inflating:
lablaut/03a02Wbxx.lablaut
inflating: lablaut/14b10Adxx.lablaut
inflating: lablaut/14b10Nbxx.lablaut
inflating: lablaut/13b01Ldxx.lablaut
inflating: lablaut/13b02Nbxx.lablaut
inflating: lablaut/14b03Adxx.lablaut
inflating: lablaut/08b01Aaxx.lablaut

inflating: lablaut/13b09Naxx.lablaut
inflating:
lablaut/03a05Waxx.lablaut
inflating:
lablaut/03a01Waxx.lablaut
inflating:
lablaut/08a04Wcxx.lablaut
inflating: lablaut/11b01Lbxx.lablaut
inflating: lablaut/11b02Abxx.lablaut
inflating: lablaut/15b10Acxx.lablaut
inflating:
lablaut/09a05Wcxx.lablaut
inflating: lablaut/14b02Naxx.lablaut
inflating: lablaut/10b03Laxx.lablaut

inflating: lablaut/03a01Ncxx.lablaut
inflating:
lablaut/11a05Wdxx.lablaut
inflating: lablaut/08b03Fexx.lablaut
inflating: lablaut/13b03Lbxx.lablaut
inflating: lablaut/03a04Adxx.lablaut
inflating:
lablaut/11a04Wcxx.lablaut
inflating: lablaut/15b01Lbxx.lablaut
inflating: lablaut/09b03Fdxx.lablaut
inflating: lablaut/16b02Lbxx.lablaut
inflating:
lablaut/12a01Wcxx.lablaut
inflating: lablaut/09a02Ebxx.lablaut
inflating: lablaut/03b01Tdxx.lablaut
inflating: lablaut/11a02Ecxx.lablaut
inflating: lablaut/16b09Laxx.lablaut
inflating:
lablaut/14a07Wcxx.lablaut
inflating: lablaut/16b01Laxx.lablaut
inflating: lablaut/16b02Aaxx.lablaut
inflating:
lablaut/10a01Waxx.lablaut
inflating: lablaut/09a01Eaxx.lablaut
inflating: lablaut/08a05Nbxx.lablaut
inflating: lablaut/11b02Fdxx.lablaut
inflating:
lablaut/13a01Wbxx.lablaut
inflating:
lablaut/16a05Wcxx.lablaut
inflating:
lablaut/12a07Waxx.lablaut
inflating: lablaut/03a05Aaxx.lablaut
inflating: lablaut/11a02Ncxx.lablaut
inflating: lablaut/11b01Fcxx.lablaut
inflating:
lablaut/16a04Wbxx.lablaut
inflating: lablaut/10a04Nbxx.lablaut
inflating:
lablaut/14a05Waxx.lablaut
inflating:
lablaut/14a01Waxx.lablaut
inflating: lablaut/11a05Adxx.lablaut
inflating: lablaut/15a05Ebxx.lablaut
inflating:
lablaut/16a07Waxx.lablaut
inflating: lablaut/16b03Fdxx.lablaut
inflating: lablaut/12a02Nbxx.lablaut
inflating: lablaut/11a04Acxx.lablaut
inflating: lablaut/14a07Ldxx.lablaut
inflating: lablaut/11b02Tdxx.lablaut
inflating: lablaut/13b02Fbxx.lablaut
inflating: lablaut/16a05Eaxx.lablaut
inflating: lablaut/14b09Ea.lablaut
inflating: lablaut/10a02Abxx.lablaut
inflating: lablaut/09a02Laxx.lablaut
inflating: lablaut/13a05Lcxx.lablaut
inflating: lablaut/14b09Tdxx.lablaut
inflating: lablaut/15a01Nbxx.lablaut
inflating: lablaut/16b09Fbxx.lablaut
inflating: lablaut/14a07Naxx.lablaut
inflating: lablaut/16a02Nbxx.lablaut
inflating: lablaut/03a05Fcxx.lablaut
inflating: lablaut/10a05Aaxx.lablaut
inflating: lablaut/10b03Tbxx.lablaut
inflating: lablaut/16b10Tdxx.lablaut
inflating: lablaut/15a04Acxx.lablaut
inflating: lablaut/13a04Lbxx.lablaut
inflating: lablaut/16a04Lcxx.lablaut
inflating: lablaut/14a05Lbxx.lablaut
inflating: lablaut/14a02Abxx.lablaut

inflating: lablaut/08a07Fdxx.lablaut
inflating: lablaut/16a07Lbxx.lablaut
inflating: lablaut/09a04Fdxx.lablaut
inflating: lablaut/16a04Abxx.lablaut
inflating: lablaut/14a05Aaxx.lablaut
inflating: lablaut/03a07Faxx.lablaut
inflating: lablaut/14a01Aaxx.lablaut
inflating: lablaut/15a01Laxx.lablaut
inflating:
lablaut/03b10Wcxx.lablaut
inflating: lablaut/03a05Tcxx.lablaut

inflating: lablaut/16b01Tbxx.lablaut
inflating:
lablaut/03b03Wcxx.lablaut
inflating:
lablaut/03b02Wbxx.lablaut
inflating: lablaut/13a01Fdxx.lablaut
inflating: lablaut/14a02Fdxx.lablaut
inflating:
lablaut/03b09Waxx.lablaut
inflating:
lablaut/09b02Wdxx.lablaut
inflating: lablaut/09a01Faxx.lablaut
inflating: lablaut/13a04Fcxx.lablaut
inflating: lablaut/11a05Fbxx.lablaut
inflating:
lablaut/03b01Waxx.lablaut
inflating: lablaut/03b09Ncxx.lablaut
inflating:
lablaut/11b01Wdxx.lablaut
inflating:
lablaut/12b02Wdxx.lablaut
inflating: lablaut/15a05Fbxx.lablaut
inflating: lablaut/11a02Tcxx.lablaut
inflating: lablaut/15a01Fbxx.lablaut
inflating:
lablaut/12b09Wcxx.lablaut
inflating: lablaut/13a02Faxx.lablaut
inflating:
lablaut/09b10Waxx.lablaut
inflating: lablaut/03b10Naxx.lablaut
inflating:
lablaut/10b02Wbxx.lablaut
inflating:
lablaut/14b10Wcxx.lablaut
inflating:
lablaut/11b03Wbxx.lablaut
inflating: lablaut/14a05Tcxx.lablaut
inflating: lablaut/09b09Eaxx.lablaut
inflating: lablaut/10a07Taxx.lablaut
inflating: lablaut/08b09Nbxx.lablaut
inflating: lablaut/09b01Eaxx.lablaut
inflating: lablaut/03b01Lbxx.lablaut
inflating:
lablaut/12b10Waxx.lablaut
inflating: lablaut/11b01Ebxx.lablaut
inflating: lablaut/14a04Tbxx.lablaut
inflating: lablaut/12a05Taxx.lablaut
inflating:
lablaut/14b02Wbxx.lablaut
inflating:
lablaut/15b03Wbxx.lablaut
inflating: lablaut/08b03Lcxx.lablaut
inflating: lablaut/13a02Taxx.lablaut
inflating:
lablaut/14b09Waxx.lablaut
inflating: lablaut/11b09Adxx.lablaut
inflating:
lablaut/03a04Wcxx.lablaut
inflating: lablaut/09b01Naxx.lablaut
inflating:
lablaut/16b10Waxx.lablaut
inflating:
lablaut/15b02Waxx.lablaut
inflating: lablaut/12b02Adxx.lablaut
inflating: lablaut/16b02Ebxx.lablaut
inflating: lablaut/09b03Lbxx.lablaut
inflating: lablaut/11b03Lcxx.lablaut
inflating: lablaut/12b09Acxx.lablaut
inflating: lablaut/10b09Lbxx.lablaut
inflating:
lablaut/09a07Wdxx.lablaut

inflating: lablaut/09b10Aaxx.lablaut
inflating: lablaut/15b09Nbxx.lablaut
inflating: lablaut/10b01Lbxx.lablaut
inflating:
lablaut/08a01Wcxx.lablaut
inflating: lablaut/09a05Edxx.lablaut
inflating: lablaut/15b10Lcxx.lablaut
inflating: lablaut/03a02Ncxx.lablaut
inflating: lablaut/13b09Abxx.lablaut
inflating: lablaut/10b01Aaxx.lablaut
inflating: lablaut/15b03Lcxx.lablaut
inflating: lablaut/14b09Lbxx.lablaut
inflating: lablaut/13b01Abxx.lablaut
inflating: lablaut/08b10Fdxx.lablaut
inflating:
lablaut/09a05Wbxx.lablaut
inflating: lablaut/13b10Laxx.lablaut
inflating:
lablaut/09a01Wbxx.lablaut
inflating: lablaut/16b10Lbxx.lablaut
inflating: lablaut/15b02Lbxx.lablaut

inflating:
lablaut/11a01Wcxx.lablaut
inflating:
lablaut/10a07Wbxx.lablaut
inflating: lablaut/09a07Ebxx.lablaut
inflating:
lablaut/12a02Wcxx.lablaut
inflating: lablaut/15b09Laxx.lablaut
inflating:
lablaut/09a04Waxx.lablaut
inflating:
lablaut/15a02Wdxx.lablaut
inflating: lablaut/16b10Aaxx.lablaut
inflating: lablaut/15b02Aaxx.lablaut
inflating:
lablaut/14a04Wcxx.lablaut
inflating: lablaut/03b09Tcxx.lablaut
inflating:
lablaut/12a05Wbxx.lablaut
inflating:
lablaut/10a02Waxx.lablaut
inflating: lablaut/11a04Ndxx.lablaut
inflating: lablaut/09a02Eaxx.lablaut
inflating: lablaut/13a01Ecxx.lablaut
inflating: lablaut/12a05Ndxx.lablaut
inflating:
lablaut/15a04Wbxx.lablaut
inflating:
lablaut/13a05Waxx.lablaut
inflating: lablaut/08a01Naxx.lablaut
inflating:
lablaut/16a05Wbxx.lablaut
inflating: lablaut/09a05Lcxx.lablaut
inflating:
lablaut/16a01Wbxx.lablaut
inflating: lablaut/10a01Nbxx.lablaut
inflating:
lablaut/14a02Waxx.lablaut
inflating: lablaut/11a01Ldxx.lablaut
inflating: lablaut/14b09Fcxx.lablaut
inflating: lablaut/08b02Tcxx.lablaut
inflating: lablaut/14a02Ncxx.lablaut
inflating: lablaut/11b10Tdxx.lablaut
inflating: lablaut/15a07Ncxx.lablaut
inflating: lablaut/14b01Fcxx.lablaut
inflating: lablaut/12b02Fbxx.lablaut
inflating: lablaut/08b09Tbxx.lablaut
inflating: lablaut/11b03Tdxx.lablaut
inflating: lablaut/11a05Naxx.lablaut
inflating: lablaut/15a01Eaxx.lablaut
inflating: lablaut/16a04Ncxx.lablaut
inflating: lablaut/08a02Laxx.lablaut
inflating: lablaut/12a02Acxx.lablaut
inflating: lablaut/16a02Eaxx.lablaut
inflating: lablaut/10a02Lbxx.lablaut
inflating: lablaut/09b02Tbxx.lablaut
inflating: lablaut/13a07Naxx.lablaut
inflating: lablaut/13a02Lcxx.lablaut
inflating: lablaut/14a07Lcxx.lablaut
inflating: lablaut/16a07Nbxx.lablaut
inflating: lablaut/12a05Abxx.lablaut
inflating: lablaut/03a02Fcxx.lablaut
inflating: lablaut/08a05Fexx.lablaut
inflating: lablaut/15a05Naxx.lablaut
inflating: lablaut/13a01Lbxx.lablaut
inflating: lablaut/12a07Laxx.lablaut
inflating: lablaut/16b01Faxx.lablaut
inflating: lablaut/15a04Abxx.lablaut
inflating: lablaut/13a05Aaxx.lablaut
inflating: lablaut/15b02Tcxx.lablaut
inflating: lablaut/16a05Abxx.lablaut

inflating: lablaut/12b01Taxx.lablaut
inflating: lablaut/15a02Laxx.lablaut
inflating: lablaut/16a07Laxx.lablaut
inflating: lablaut/14b03Taxx.lablaut
inflating: lablaut/11a04Fdxx.lablaut
inflating:
lablaut/03b10Wbxx.lablaut
inflating: lablaut/14a07Fdxx.lablaut
inflating:
lablaut/08b02Wdxx.lablaut
inflating: lablaut/15a04Fdxx.lablaut
inflating:
lablaut/08b09Wcxx.lablaut
inflating: lablaut/03a04Taxx.lablaut

inflating: lablaut/11a02Fbxx.lablaut
inflating:
lablaut/09b02Wcxx.lablaut
inflating: lablaut/08a02Tbxx.lablaut
inflating: lablaut/14a05Fbxx.lablaut
inflating:
lablaut/08b10Waxx.lablaut
inflating:
lablaut/09b01Wbxx.lablaut
inflating: lablaut/03b01Nbxx.lablaut
inflating: lablaut/09b09Ndxx.lablaut
inflating: lablaut/08a05Taxx.lablaut
inflating: lablaut/16a07Fbxx.lablaut
inflating:
lablaut/13b10Wcxx.lablaut
inflating: lablaut/10a05Tbxx.lablaut
inflating: lablaut/08b10Ncxx.lablaut
inflating:
lablaut/10b03Wbxx.lablaut
inflating: lablaut/13a05Tcxx.lablaut
inflating:
lablaut/13b03Wcxx.lablaut
inflating: lablaut/03b10Abxx.lablaut
inflating: lablaut/14b03Edxx.lablaut
inflating: lablaut/13b09Ecxx.lablaut
inflating:
lablaut/11b10Waxx.lablaut
inflating:
lablaut/15b01Wcxx.lablaut
inflating: lablaut/13b01Ecxx.lablaut
inflating: lablaut/08b02Nbxx.lablaut
inflating:
lablaut/11b03Waxx.lablaut
inflating: lablaut/16a04Tcxx.lablaut
inflating: lablaut/11b10Aexx.lablaut
inflating: lablaut/11b10Ncxx.lablaut
inflating: lablaut/03b09Laxx.lablaut
inflating:
lablaut/14b03Wbxx.lablaut
inflating: lablaut/09b03Nbxx.lablaut
inflating: lablaut/12a02Taxx.lablaut
inflating:
lablaut/13b09Waxx.lablaut
inflating: lablaut/03b02Aaxx.lablaut
inflating: lablaut/10b09Adxx.lablaut
inflating:
lablaut/16b09Wbxx.lablaut
inflating:
lablaut/13b01Waxx.lablaut
inflating: lablaut/08b01Naxx.lablaut
inflating: lablaut/11b09Ldxx.lablaut
inflating:
lablaut/15b10Waxx.lablaut
inflating:
lablaut/16b01Wbxx.lablaut
inflating: lablaut/14b01Ebxx.lablaut
inflating: lablaut/12b02Eaxx.lablaut
inflating: lablaut/09b02Naxx.lablaut
inflating: lablaut/16b10Ebxx.lablaut
inflating: lablaut/10b10Lcxx.lablaut
inflating: lablaut/13b01Ncxx.lablaut
inflating:
lablaut/15b03Waxx.lablaut
inflating: lablaut/15b10Ncxx.lablaut
inflating: lablaut/11b09Naxx.lablaut
inflating: lablaut/08b10Aaxx.lablaut
inflating: lablaut/08b02Laxx.lablaut
inflating: lablaut/12b02Naxx.lablaut
inflating: lablaut/08b02Ffxx.lablaut
inflating: lablaut/13b02Lcxx.lablaut
inflating: lablaut/13b03Acxx.lablaut

inflating: lablaut/13b03Naxx.lablaut
inflating:
lablaut/08a02Wcxx.lablaut
inflating: lablaut/16b03Adxx.lablaut
inflating: lablaut/03a07Ncxx.lablaut
inflating: lablaut/16b03Nbxx.lablaut
inflating: lablaut/15b09Acxx.lablaut
inflating: lablaut/10b02Aaxx.lablaut
inflating: lablaut/08b01Fexx.lablaut
inflating: lablaut/15b01Naxx.lablaut
inflating: lablaut/16b01Lcxx.lablaut
inflating: lablaut/12b03Laxx.lablaut
inflating:
lablaut/09a02Wbxx.lablaut
inflating:
lablaut/11a02Wcxx.lablaut
inflating: lablaut/16b09Abxx.lablaut

inflating:
lablaut/10a04Wbxx.lablaut
inflating: lablaut/03a04Lcxx.lablaut
inflating: lablaut/14b02Aaxx.lablaut
inflating:
lablaut/13a04Wcxx.lablaut
inflating: lablaut/08a04Ncxx.lablaut
inflating: lablaut/15b03Aaxx.lablaut
inflating: lablaut/14a04Edxx.lablaut
inflating: lablaut/16b03Laxx.lablaut
inflating:
lablaut/14a01Wcxx.lablaut
inflating:
lablaut/13a07Wbxx.lablaut
inflating: lablaut/11a01Ndxx.lablaut
inflating: lablaut/13a02Ecxx.lablaut
inflating:
lablaut/14a04Wbxx.lablaut
inflating: lablaut/09a04Nbxx.lablaut
inflating: lablaut/08a05Lcxx.lablaut
inflating: lablaut/11b03Fcxx.lablaut
inflating: lablaut/08a01Lcxx.lablaut
inflating: lablaut/08a02Acxx.lablaut
inflating:
lablaut/13a02Waxx.lablaut
inflating: lablaut/10a05Ldxx.lablaut
inflating: lablaut/08a02Naxx.lablaut
inflating: lablaut/09a07Naxx.lablaut
inflating: lablaut/16a01Ecxx.lablaut
inflating:
lablaut/16a02Wbxx.lablaut
inflating: lablaut/13b09Fcxx.lablaut
inflating: lablaut/08b10Tcxx.lablaut
inflating: lablaut/15a07Ebxx.lablaut
inflating: lablaut/11a02Ldxx.lablaut
inflating: lablaut/13a02Ncxx.lablaut
inflating:
lablaut/15a04Waxx.lablaut
inflating: lablaut/13b01Fcxx.lablaut
inflating: lablaut/08b03Tcxx.lablaut
inflating: lablaut/08a01Abxx.lablaut
inflating: lablaut/10a01Acxx.lablaut
inflating: lablaut/14a01Eaxx.lablaut
inflating: lablaut/15a04Ncxx.lablaut
inflating: lablaut/11a05Lcxx.lablaut
inflating: lablaut/13a05Nbxx.lablaut
inflating: lablaut/08a07Laxx.lablaut
inflating: lablaut/12b09Tdxx.lablaut
inflating: lablaut/13a01Nbxx.lablaut
inflating: lablaut/15a02Eaxx.lablaut
inflating: lablaut/12a07Acxx.lablaut
inflating: lablaut/16a07Eaxx.lablaut
inflating: lablaut/16a01Ncxx.lablaut
inflating: lablaut/03a04Fdxx.lablaut
inflating: lablaut/09a04Laxx.lablaut
inflating: lablaut/13b10Faxx.lablaut
inflating: lablaut/13a04Acxx.lablaut
inflating: lablaut/16b10Fbxx.lablaut
inflating: lablaut/11a01Abxx.lablaut
inflating: lablaut/14a05Acxx.lablaut
inflating: lablaut/12a05Lbxx.lablaut
inflating: lablaut/14a05Naxx.lablaut
inflating: lablaut/10a07Aaxx.lablaut
inflating: lablaut/14a01Acxx.lablaut
inflating: lablaut/12a01Lbxx.lablaut
inflating: lablaut/14a01Naxx.lablaut
inflating: lablaut/15b09Faxx.lablaut
inflating: lablaut/08a02Fexx.lablaut
inflating: lablaut/15a02Acxx.lablaut
inflating: lablaut/15a02Naxx.lablaut
inflating: lablaut/14b02Tcxx.lablaut

inflating: lablaut/15b03Tcxx.lablaut
inflating: lablaut/08a01Fdxx.lablaut
inflating: lablaut/14a07Aaxx.lablaut
inflating: lablaut/16a01Lbxx.lablaut
inflating: lablaut/14a02Laxx.lablaut
inflating: lablaut/03a01Faxx.lablaut

inflating: lablaut/16b10Tbxx.lablaut
inflating: lablaut/10a04Fdxx.lablaut
inflating: lablaut/16a04Laxx.lablaut
inflating: lablaut/15b09Taxx.lablaut
inflating:
lablaut/03b01Wcxx.lablaut
inflating: lablaut/03b10Ecxx.lablaut
inflating: lablaut/13a07Fdxx.lablaut
inflating:
lablaut/08b03Wdxx.lablaut
inflating: lablaut/03b10Ncxx.lablaut
inflating: lablaut/11a05Tdxx.lablaut
inflating: lablaut/08a07Tbxx.lablaut
inflating: lablaut/16a05Fcxx.lablaut
inflating: lablaut/16a01Fcxx.lablaut
inflating: lablaut/15a07Fbxx.lablaut
inflating:
lablaut/09b09Waxx.lablaut
inflating:
lablaut/14b02Wdxx.lablaut
inflating: lablaut/14a05Faxx.lablaut
inflating: lablaut/09a07Taxx.lablaut
inflating: lablaut/13b03Edxx.lablaut
inflating:
lablaut/14b09Wcxx.lablaut
inflating: lablaut/14a07Tcxx.lablaut
inflating:
lablaut/10b10Waxx.lablaut
inflating: lablaut/16a07Faxx.lablaut
inflating:
lablaut/14b01Wcxx.lablaut
inflating:
lablaut/12b02Wbxx.lablaut
inflating:
lablaut/15b02Wcxx.lablaut
inflating: lablaut/08b03Nbxx.lablaut
inflating: lablaut/14a02Tbxx.lablaut
inflating:
lablaut/15b09Wbxx.lablaut
inflating: lablaut/08b09Lcxx.lablaut
inflating:
lablaut/12b01Waxx.lablaut
inflating: lablaut/10b01Eaxx.lablaut
inflating: lablaut/03b02Laxx.lablaut
inflating: lablaut/13a04Taxx.lablaut
inflating:
lablaut/13b02Waxx.lablaut
inflating: lablaut/11b10Adxx.lablaut
inflating: lablaut/14a05Taxx.lablaut
inflating:
lablaut/16b02Wbxx.lablaut
inflating:
lablaut/03a02Wcxx.lablaut
inflating: lablaut/08b09Abxx.lablaut
inflating: lablaut/12b10Ldxx.lablaut
inflating: lablaut/11b03Nbxx.lablaut
inflating: lablaut/15a02Taxx.lablaut
inflating: lablaut/14b09Eaxx.lablaut
inflating:
lablaut/03a05Wbxx.lablaut
inflating: lablaut/08b10Laxx.lablaut
inflating:
lablaut/16b01Waxx.lablaut
inflating: lablaut/12b10Acxx.lablaut
inflating: lablaut/11b02Naxx.lablaut
inflating: lablaut/03a05Ndxx.lablaut
inflating: lablaut/15b10Nbxx.lablaut
inflating: lablaut/16b03Eaxx.lablaut
inflating:
lablaut/03a04Waxx.lablaut
inflating:

lablaut/08a07Wcxx.lablaut
inflating: lablaut/15b03Nbxx.lablaut
inflating: lablaut/14b09Acxx.lablaut
inflating: lablaut/11b01Abxx.lablaut
inflating: lablaut/03a04Ncxx.lablaut
inflating: lablaut/14b01Acxx.lablaut
inflating: lablaut/14b01Naxx.lablaut
inflating: lablaut/10b02Laxx.lablaut
inflating: lablaut/14b10Lbxx.lablaut
inflating:
lablaut/09a07Wbxx.lablaut
inflating: lablaut/14b03Lbxx.lablaut
inflating: lablaut/08b09Fdxx.lablaut
inflating: lablaut/13b09Laxx.lablaut
inflating:
lablaut/11a07Wcxx.lablaut
inflating: lablaut/16b09Lbxx.lablaut
inflating:
lablaut/08a05Waxx.lablaut

inflating: lablaut/08b01Fdxx.lablaut
inflating:
lablaut/08a01Waxx.lablaut
inflating:
lablaut/12a04Wcxx.lablaut
inflating:
lablaut/10a05Wbxx.lablaut
inflating:
lablaut/13a05Wcxx.lablaut
inflating: lablaut/03b01Faxx.lablaut
inflating:
lablaut/14a02Wcxx.lablaut
inflating: lablaut/12a02Ecxx.lablaut
inflating: lablaut/11b09Fdxx.lablaut
inflating:
lablaut/10a04Waxx.lablaut
inflating: lablaut/16b01Aaxx.lablaut
inflating: lablaut/03b03Tcxx.lablaut
inflating: lablaut/10b10Fcxx.lablaut
inflating:
lablaut/16a04Wcxx.lablaut
inflating:
lablaut/14a05Wbxx.lablaut
inflating: lablaut/09a05Nbxx.lablaut
inflating: lablaut/03a07Laxx.lablaut
inflating: lablaut/09a01Nbxx.lablaut
inflating:
lablaut/12a02Waxx.lablaut
inflating: lablaut/13b03Fdxx.lablaut
inflating: lablaut/03b02Tbxx.lablaut
inflating: lablaut/08a07Naxx.lablaut
inflating:
lablaut/15a02Wbxx.lablaut
inflating: lablaut/10a07Adxx.lablaut
inflating: lablaut/14a07Ebxx.lablaut
inflating: lablaut/16a02Ecxx.lablaut
inflating: lablaut/11a07Ldxx.lablaut
inflating: lablaut/09b03Faxx.lablaut
inflating:
lablaut/15a05Waxx.lablaut
inflating: lablaut/13a05Eaxx.lablaut
inflating:
lablaut/15a01Waxx.lablaut
inflating: lablaut/13a01Eaxx.lablaut
inflating: lablaut/08a02Abxx.lablaut
inflating: lablaut/16b02Fdxx.lablaut
inflating: lablaut/11b09Tdxx.lablaut
inflating: lablaut/12a01Nbxx.lablaut
inflating: lablaut/14a02Eaxx.lablaut
inflating: lablaut/13b09Fbxx.lablaut
inflating: lablaut/10a02Naxx.lablaut
inflating: lablaut/11a07Acxx.lablaut
inflating: lablaut/10b01Faxx.lablaut
inflating: lablaut/13a02Adxx.lablaut
inflating: lablaut/08a04Laxx.lablaut
inflating: lablaut/16a04Eaxx.lablaut
inflating: lablaut/15a07Ldxx.lablaut
inflating: lablaut/14b02Fbxx.lablaut
inflating: lablaut/08a04Ffxx.lablaut
inflating: lablaut/13b03Tdxx.lablaut
inflating: lablaut/13a01Acxx.lablaut
inflating: lablaut/10a07Laxx.lablaut
inflating: lablaut/15a07Acxx.lablaut
inflating: lablaut/13a07Lbxx.lablaut
inflating: lablaut/14b01Faxx.lablaut
inflating: lablaut/09b03Taxx.lablaut
inflating: lablaut/14b10Tcxx.lablaut
inflating: lablaut/11a01Aaxx.lablaut
creating: labsilb/
inflating:
labsilb/13a07Na.labsilb

inflating: labsilb/08a02Tb.labsilb
inflating: labsilb/09b01Wb.labsilb
inflating: labsilb/15b10Wa.labsilb
inflating: labsilb/16b10Fb.labsilb
inflating: labsilb/08a07Wc.labsilb
inflating:
labsilb/15a05Na.labsilb
inflating: labsilb/14b02Tc.labsilb
inflating: labsilb/14b10Lb.labsilb
inflating: labsilb/13b03Fd.labsilb
inflating: labsilb/11a04Fd.labsilb
inflating: labsilb/14b01Wc.labsilb

inflating: labsilb/16b09Lb.labsilb
inflating: labsilb/16a04Fa.labsilb
inflating: labsilb/12a02Wc.labsilb
inflating: labsilb/16b03La.labsilb
inflating: labsilb/13b01Ld.labsilb
inflating: labsilb/14a05Aa.labsilb
inflating: labsilb/11b10Wa.labsilb
inflating: labsilb/11a02Ld.labsilb
inflating: labsilb/03a02Wb.labsilb
inflating:
labsilb/03b10Na.labsilb
inflating:
labsilb/11a05Na.labsilb inflating:
labsilb/13b09Wa.labsilb inflating:
labsilb/08a07La.labsilb inflating:
labsilb/09b03Ed.labsilb inflating:
labsilb/13b09Ab.labsilb inflating:
labsilb/16b01Fa.labsilb inflating:
labsilb/12b03La.labsilb inflating:
labsilb/15a07Fb.labsilb inflating:
labsilb/11a05Wd.labsilb inflating:
labsilb/08b10Wa.labsilb inflating:
labsilb/03b09La.labsilb inflating:
labsilb/14b02Aa.labsilb inflating:
labsilb/03a05Nd.labsilb
inflating: labsilb/14a07Eb.labsilb
inflating: labsilb/13a07Lb.labsilb
inflating:
labsilb/11b02Na.labsilb
inflating: labsilb/15a05Lb.labsilb
inflating: labsilb/14a01Aa.labsilb
inflating: labsilb/03a04Wc.labsilb
inflating: labsilb/16a05Ab.labsilb
inflating: labsilb/08b01Fd.labsilb
inflating:
labsilb/10a02Na.labsilb
inflating: labsilb/11a07Ta.labsilb
inflating: labsilb/11b03Td.labsilb
inflating:
labsilb/16b03Nb.labsilb
inflating:
labsilb/13a05Nb.labsilb inflating:
labsilb/15b09Wb.labsilb inflating:
labsilb/16b03Ta.labsilb inflating:
labsilb/13b10Ec.labsilb inflating:
labsilb/03a04Ad.labsilb inflating:
labsilb/15b03Wa.labsilb inflating:
labsilb/10b02Aa.labsilb inflating:
labsilb/09a02La.labsilb inflating:
labsilb/16a02Nb.labsilb
inflating: labsilb/15b09Ac.labsilb
inflating: labsilb/14b03Lb.labsilb
inflating:
labsilb/09b01Na.labsilb
inflating: labsilb/16b02Eb.labsilb
inflating: labsilb/15b02Lb.labsilb
inflating: labsilb/03b01Wc.labsilb
inflating: labsilb/10b10Fc.labsilb
inflating: labsilb/08a07Ta.labsilb
inflating:
labsilb/08a01Na.labsilb
inflating:
labsilb/13b02Nb.labsilb
inflating: labsilb/08b03Fe.labsilb
inflating:
labsilb/10a04Nb.labsilb inflating:
labsilb/08b02Wd.labsilb inflating:
labsilb/11b03Wa.labsilb inflating:
labsilb/12b02Wa.labsilb inflating:
labsilb/08b01Aa.labsilb inflating:
labsilb/12a02Nb.labsilb

inflating: labsilb/13b02Fb.labsilb
inflating: labsilb/13b01Wa.labsilb
inflating:
labsilb/13a01Nb.labsilb
inflating: labsilb/08a02Fe.labsilb
inflating: labsilb/16a04Nc.labsilb
inflating: labsilb/14a05Wb.labsilb
inflating: labsilb/13b01Ab.labsilb
inflating: labsilb/15a04Wb.labsilb

inflating: labsilb/10a02Lb.labsilb
inflating:
labsilb/09b03Nb.labsilb
inflating: labsilb/12a07Ac.labsilb
inflating: labsilb/09b03Ta.labsilb
inflating: labsilb/13a05Lc.labsilb
inflating: labsilb/03b02Aa.labsilb
inflating: labsilb/14a05Ac.labsilb
inflating: labsilb/03b10Nc.labsilb
inflating: labsilb/09a07Wb.labsilb
inflating: labsilb/11b10Ad.labsilb
inflating: labsilb/10a05Wb.labsilb
inflating: labsilb/08a04Ff.labsilb
inflating: labsilb/11a05Fc.labsilb
inflating: labsilb/16b01Tb.labsilb
inflating: labsilb/14b02Wb.labsilb
inflating: labsilb/16b10Aa.labsilb
inflating: labsilb/14a02Nc.labsilb
inflating: labsilb/14a02Tb.labsilb
inflating: labsilb/08a02Ab.labsilb
inflating:
labsilb/09b10Nd.labsilb inflating:
labsilb/14a07Wc.labsilb inflating:
labsilb/03a04Ta.labsilb inflating:
labsilb/13b02Lc.labsilb inflating:
labsilb/13b03Ac.labsilb inflating:
labsilb/15b01Ec.labsilb inflating:
labsilb/16a05Wc.labsilb inflating:
labsilb/11a04Ac.labsilb inflating:
labsilb/13a02Ec.labsilb inflating:
labsilb/13b10Fa.labsilb inflating:
labsilb/10b03Tb.labsilb inflating:
labsilb/14a01Ac.labsilb inflating:
labsilb/15b09Fa.labsilb inflating:
labsilb/10b02Wb.labsilb inflating:
labsilb/14b09Ea.labsilb inflating:
labsilb/13b09La.labsilb inflating:
labsilb/03b01Nb.labsilb
inflating: labsilb/13a05Tc.labsilb
inflating: labsilb/14a04Tc.labsilb
inflating: labsilb/16b03Fd.labsilb
inflating:
labsilb/11b09Na.labsilb inflating:
labsilb/13a04Wc.labsilb inflating:
labsilb/16a02Tc.labsilb inflating:
labsilb/03a05Wb.labsilb inflating:
labsilb/08b10La.labsilb inflating:
labsilb/15b10Nb.labsilb
inflating: labsilb/16a05La.labsilb
inflating: labsilb/09b10Aa.labsilb
inflating: labsilb/08a07Fd.labsilb
inflating: labsilb/03a04Lc.labsilb
inflating:
labsilb/14a05Na.labsilb
inflating: labsilb/14b10Eb.labsilb
inflating: labsilb/09a05Wc.labsilb
inflating: labsilb/03b09Tc.labsilb
inflating: labsilb/14a05Fa.labsilb
inflating: labsilb/10a07La.labsilb
inflating: labsilb/16b09Eb.labsilb
inflating: labsilb/09a05Ed.labsilb
inflating: labsilb/11a02Wc.labsilb
inflating: labsilb/16b03Ea.labsilb
inflating: labsilb/13a05Ea.labsilb
inflating: labsilb/12a01Wc.labsilb
inflating: labsilb/03b02Wb.labsilb
inflating: labsilb/13a01Fd.labsilb
inflating: labsilb/12b02Ad.labsilb
inflating: labsilb/03a02Nc.labsilb
inflating: labsilb/13a05Aa.labsilb
inflating:

labsilb/09a07Na.labsilb inflating:
labsilb/10b10Wa.labsilb inflating:
labsilb/14a04Aa.labsilb

inflating: labsilb/03a07Wc.labsilb
inflating: labsilb/16a07Lb.labsilb
inflating: labsilb/11a01Ld.labsilb
inflating: labsilb/16a02Ea.labsilb
inflating: labsilb/03a02Fc.labsilb
inflating: labsilb/15a02La.labsilb
inflating:
labsilb/13b03Na.labsilb
inflating: labsilb/13b09Fb.labsilb
inflating:
labsilb/14b02Na.labsilb inflating:
labsilb/09b02Wc.labsilb inflating:
labsilb/15b01Na.labsilb inflating:
labsilb/16b10Wb.labsilb inflating:
labsilb/09a07Wd.labsilb inflating:
labsilb/08a02Wc.labsilb inflating:
labsilb/14a01Na.labsilb
inflating: labsilb/11a05Td.labsilb
inflating: labsilb/13a02Fa.labsilb
inflating: labsilb/15b10Lc.labsilb
inflating: labsilb/14b02Wd.labsilb
inflating: labsilb/15a05Wa.labsilb
inflating: labsilb/09b09Wa.labsilb
inflating: labsilb/13a01Ea.labsilb
inflating: labsilb/13b10Nc.labsilb
inflating:
labsilb/10b02Na.labsilb
inflating: labsilb/15a05Eb.labsilb
inflating: labsilb/14a05Lb.labsilb
inflating: labsilb/03a04Fd.labsilb
inflating: labsilb/03a07La.labsilb
inflating: labsilb/10a07Ta.labsilb
inflating:
labsilb/15b03Nb.labsilb
inflating: labsilb/11b02Td.labsilb
inflating: labsilb/08a05Fe.labsilb
inflating: labsilb/10b01Ea.labsilb
inflating: labsilb/13a04Ta.labsilb
inflating: labsilb/11b01Wd.labsilb
inflating: labsilb/15b02Wa.labsilb
inflating: labsilb/10b01Aa.labsilb
inflating: labsilb/09a02Ea.labsilb
inflating: labsilb/08a02La.labsilb
inflating: labsilb/15a02Ta.labsilb
inflating: labsilb/16b01Wa.labsilb
inflating: labsilb/14b09Ac.labsilb
inflating: labsilb/13b03Lb.labsilb
inflating: labsilb/14a02Wa.labsilb
inflating: labsilb/03b10Ab.labsilb
inflating:
labsilb/08b01Na.labsilb inflating:
labsilb/15a01Wa.labsilb inflating:
labsilb/16b01Eb.labsilb inflating:
labsilb/15b01Lb.labsilb inflating:
labsilb/14a07Lc.labsilb inflating:
labsilb/11b03Nb.labsilb
inflating:
labsilb/09a05Nb.labsilb
inflating: labsilb/15a07Ac.labsilb
inflating: labsilb/10b09Wb.labsilb
inflating: labsilb/14a02Ab.labsilb
inflating: labsilb/08a05Wa.labsilb
inflating: labsilb/14b10Wc.labsilb
inflating: labsilb/09a04Wa.labsilb
inflating: labsilb/12b02Fb.labsilb
inflating: labsilb/12b01Wa.labsilb
inflating:
labsilb/12a01Nb.labsilb
inflating:
labsilb/03b02Na.labsilb inflating:
labsilb/10a02Wa.labsilb inflating:

labsilb/15a04Nc.labsilb inflating:
labsilb/12a01Fb.labsilb inflating:
labsilb/16b03Wb.labsilb inflating:
labsilb/14b10Ad.labsilb inflating:
labsilb/11b02Ab.labsilb inflating:
labsilb/08b09Tb.labsilb

inflating: labsilb/14a04Wb.labsilb
inflating:
labsilb/08b03Nb.labsilb
inflating: labsilb/11a07Ac.labsilb
inflating: labsilb/03a07Fb.labsilb
inflating: labsilb/10a02Ab.labsilb
inflating: labsilb/15b03Lc.labsilb
inflating: labsilb/16a02Wb.labsilb
inflating: labsilb/11a01Ab.labsilb
inflating: labsilb/03a01Fa.labsilb
inflating:
labsilb/09a01Nb.labsilb
inflating: labsilb/16a02Ec.labsilb
inflating: labsilb/12b09Wc.labsilb
inflating: labsilb/08a01Wa.labsilb
inflating: labsilb/08b01Lb.labsilb
inflating: labsilb/11b09Ld.labsilb
inflating: labsilb/10a04Wb.labsilb
inflating: labsilb/13a02Nc.labsilb
inflating: labsilb/14a07Tc.labsilb
inflating: labsilb/16b09La.labsilb
inflating: labsilb/11b03Lc.labsilb
inflating: labsilb/09a05Lc.labsilb
inflating: labsilb/14a07Fd.labsilb
inflating: labsilb/08b10Tc.labsilb
inflating: labsilb/13a01Wb.labsilb
inflating: labsilb/08b10Fd.labsilb
inflating: labsilb/14b01Ac.labsilb
inflating: labsilb/12a02Ec.labsilb
inflating: labsilb/16a04Wc.labsilb
inflating: labsilb/13a01Ec.labsilb
inflating: labsilb/12a02Ac.labsilb
inflating: labsilb/08a04Nc.labsilb
inflating: labsilb/13a01Ac.labsilb
inflating: labsilb/08a04Tb.labsilb
inflating: labsilb/09b03Wb.labsilb
inflating: labsilb/11b01Nc.labsilb
inflating: labsilb/11b01Fc.labsilb
inflating: labsilb/08b03Lc.labsilb
inflating:
labsilb/12a05Nd.labsilb inflating:
labsilb/09a02Wb.labsilb inflating:
labsilb/15b03Tc.labsilb inflating:
labsilb/15a07Fa.labsilb inflating:
labsilb/03b10Wc.labsilb inflating:
labsilb/16b10Lb.labsilb inflating:
labsilb/15b02Wc.labsilb inflating:
labsilb/12a04Wc.labsilb inflating:
labsilb/16b02Fd.labsilb inflating:
labsilb/16a07Td.labsilb inflating:
labsilb/14b03Ed.labsilb inflating:
labsilb/10a05Ld.labsilb inflating:
labsilb/14a07Aa.labsilb inflating:
labsilb/14a02Wc.labsilb inflating:
labsilb/14b10Nb.labsilb
inflating: labsilb/14b03Ad.labsilb
inflating: labsilb/11a05Ad.labsilb
inflating: labsilb/16a05Ea.labsilb
inflating: labsilb/03a05Fc.labsilb
inflating: labsilb/08b10Aa.labsilb
inflating: labsilb/16a04La.labsilb
inflating: labsilb/11b09Td.labsilb
inflating: labsilb/13b10Wa.labsilb
inflating: labsilb/16b09Fb.labsilb
inflating: labsilb/16b03Fa.labsilb
inflating: labsilb/11a02Tc.labsilb
inflating: labsilb/14b09Lb.labsilb
inflating: labsilb/10a07Aa.labsilb
inflating: labsilb/16a07Wa.labsilb
inflating: labsilb/11a01Wc.labsilb
inflating: labsilb/15b03Aa.labsilb

inflating: labsilb/08a07Na.labsilb

inflating: labsilb/03a01Nc.labsilb
inflating: labsilb/16b01La.labsilb
inflating: labsilb/16b02Aa.labsilb
inflating: labsilb/08b03Tc.labsilb
inflating: labsilb/15a02Ea.labsilb
inflating: labsilb/11b09Wa.labsilb
inflating: labsilb/15a01La.labsilb
inflating: labsilb/16b10Tb.labsilb
inflating:
labsilb/14b01Na.labsilb
inflating: labsilb/10b09Lb.labsilb
inflating: labsilb/13b03Td.labsilb
inflating: labsilb/14b01Fa.labsilb
inflating: labsilb/10b03La.labsilb
inflating: labsilb/12a07Wa.labsilb
inflating: labsilb/08a01Wc.labsilb
inflating: labsilb/09a04La.labsilb
inflating: labsilb/15b10Ac.labsilb
inflating: labsilb/14a05Wa.labsilb
inflating: labsilb/15a05Fb.labsilb
inflating: labsilb/15a04Wa.labsilb
inflating: labsilb/08b09Wa.labsilb
inflating: labsilb/03b03Wc.labsilb
inflating: labsilb/09b03Fa.labsilb
inflating: labsilb/14a04Lb.labsilb
inflating: labsilb/15a04Ab.labsilb
inflating: labsilb/03a02Wc.labsilb
inflating: labsilb/08b09Ab.labsilb
inflating: labsilb/10b01Fa.labsilb
inflating: labsilb/16a02Lb.labsilb
inflating: labsilb/14b09Fc.labsilb
inflating: labsilb/14b03Ta.labsilb
inflating: labsilb/10b10Lc.labsilb
inflating: labsilb/08b02La.labsilb
inflating: labsilb/11a05Fb.labsilb
inflating: labsilb/09a07Eb.labsilb
inflating: labsilb/15a07Nc.labsilb
inflating: labsilb/09a01Ea.labsilb
inflating:
labsilb/15a01Nb.labsilb inflating:
labsilb/13a02Wa.labsilb inflating:
labsilb/14a01Wa.labsilb inflating:
labsilb/15a01Fb.labsilb inflating:
labsilb/16a05Wb.labsilb inflating:
labsilb/08a05Nb.labsilb
inflating: labsilb/13a01Lb.labsilb
inflating: labsilb/08a05Ta.labsilb
inflating:
labsilb/09a04Nb.labsilb
inflating: labsilb/14b10Tc.labsilb
inflating: labsilb/16a04Lc.labsilb
inflating: labsilb/12b01Ta.labsilb
inflating: labsilb/13b10Wc.labsilb
inflating: labsilb/08b01Fe.labsilb
inflating: labsilb/09b03Lb.labsilb
inflating: labsilb/10a07Wb.labsilb
inflating: labsilb/12b10Ld.labsilb
inflating: labsilb/10a01Wa.labsilb
inflating: labsilb/14a04Tb.labsilb
inflating: labsilb/15b03Wb.labsilb
inflating: labsilb/10b01Lb.labsilb
inflating: labsilb/12a05Wb.labsilb
inflating: labsilb/16b02Wb.labsilb
inflating: labsilb/11b01Ab.labsilb
inflating: labsilb/03b01Fa.labsilb
inflating:
labsilb/08b02Nb.labsilb
inflating: labsilb/11a05Lc.labsilb
inflating: labsilb/15a02Wb.labsilb
inflating: labsilb/03a05Wa.labsilb
inflating: labsilb/16a01Wb.labsilb

inflating: labsilb/16b01Lc.labsilb
inflating: labsilb/13a04Ac.labsilb

inflating: labsilb/08a07Tb.labsilb
inflating: labsilb/15a07Ld.labsilb
inflating: labsilb/08b01Wa.labsilb
inflating: labsilb/16a01Ec.labsilb
inflating: labsilb/15a02Ac.labsilb
inflating: labsilb/16b10Td.labsilb
inflating: labsilb/03b09Nc.labsilb
inflating: labsilb/11b03Wb.labsilb
inflating: labsilb/09a05Wb.labsilb
inflating: labsilb/13b10La.labsilb
inflating: labsilb/12b02Wb.labsilb
inflating:
labsilb/03b03Nb.labsilb
inflating: labsilb/13a07Tc.labsilb
inflating: labsilb/11b09Ad.labsilb
inflating: labsilb/15b09La.labsilb
inflating: labsilb/14b01Fc.labsilb
inflating: labsilb/08a05Lc.labsilb
inflating: labsilb/13a07Fd.labsilb
inflating:
labsilb/09b09Nd.labsilb
inflating: labsilb/13b01Ec.labsilb
inflating: labsilb/16a04Tc.labsilb
inflating: labsilb/11a07Ld.labsilb
inflating: labsilb/03a02Ta.labsilb
inflating:
labsilb/13b09Na.labsilb
inflating: labsilb/11a02Ec.labsilb
inflating: labsilb/16a07La.labsilb
inflating: labsilb/08b09Wc.labsilb
inflating: labsilb/03a01Wa.labsilb
inflating: labsilb/03b01Lb.labsilb
inflating: labsilb/14b09Td.labsilb
inflating:
labsilb/14a07Na.labsilb inflating:
labsilb/16b10Wa.labsilb inflating:
labsilb/15b02Nd.labsilb inflating:
labsilb/09a01Wb.labsilb inflating:
labsilb/13b03Wc.labsilb inflating:
labsilb/11b10Ae.labsilb inflating:
labsilb/11a04Wc.labsilb inflating:
labsilb/12a07La.labsilb inflating:
labsilb/15b01Wc.labsilb inflating:
labsilb/08a01Lc.labsilb inflating:
labsilb/13b03Ed.labsilb inflating:
labsilb/03a04Nc.labsilb inflating:
labsilb/14a02Fd.labsilb inflating:
labsilb/14a01Wc.labsilb inflating:
labsilb/16a04Ea.labsilb inflating:
labsilb/09b09Ea.labsilb inflating:
labsilb/12b10Wa.labsilb inflating:
labsilb/15b09Nb.labsilb
inflating: labsilb/15b09Ta.labsilb
inflating: labsilb/13a02Ad.labsilb
inflating: labsilb/14b09Wa.labsilb
inflating: labsilb/08a04Wc.labsilb
inflating: labsilb/11b02Fd.labsilb
inflating: labsilb/09a04Fd.labsilb
inflating:
labsilb/16a07Nb.labsilb
inflating:
labsilb/11a01Nd.labsilb
inflating:
labsilb/15a02Na.labsilb
inflating: labsilb/16a07Fb.labsilb
inflating: labsilb/09b10Wa.labsilb
inflating: labsilb/15a07Eb.labsilb
inflating: labsilb/16b01Aa.labsilb
inflating: labsilb/14a02Ea.labsilb
inflating: labsilb/15a02Wd.labsilb
inflating: labsilb/08b02Tc.labsilb

inflating: labsilb/15a01Ea.labsilb
inflating: labsilb/15b10Nc.labsilb
inflating:
labsilb/12b02Na.labsilb inflating:
labsilb/16b09Wb.labsilb inflating:
labsilb/14a05Ta.labsilb

inflating: labsilb/10b02La.labsilb
inflating: labsilb/08a04La.labsilb
inflating: labsilb/12b02Wd.labsilb
inflating:
labsilb/08b09Nb.labsilb inflating:
labsilb/13a05Wa.labsilb inflating:
labsilb/14a05Fb.labsilb inflating:
labsilb/03b03Tc.labsilb inflating:
labsilb/03a07Fa.labsilb inflating:
labsilb/11b10Nc.labsilb inflating:
labsilb/09b02Na.labsilb
inflating: labsilb/12a05Lb.labsilb
inflating: labsilb/16b02Lb.labsilb
inflating: labsilb/13a04Lb.labsilb
inflating:
labsilb/08a02Na.labsilb
inflating: labsilb/09a07Ta.labsilb
inflating: labsilb/13b09Fc.labsilb
inflating: labsilb/16a01Lb.labsilb
inflating: labsilb/08b03Wd.labsilb
inflating: labsilb/09b02Wd.labsilb
inflating: labsilb/09a01Fa.labsilb
inflating: labsilb/09b01Ea.labsilb
inflating: labsilb/13b02Wa.labsilb
inflating: labsilb/10a04Wa.labsilb
inflating: labsilb/14b02Fb.labsilb
inflating: labsilb/03b09Wa.labsilb
inflating: labsilb/13a02Ta.labsilb
inflating: labsilb/08b10Nc.labsilb
inflating: labsilb/13a07Wb.labsilb
inflating: labsilb/12a02Wa.labsilb
inflating: labsilb/16a05Tb.labsilb
inflating: labsilb/14b01Eb.labsilb
inflating: labsilb/16a04Wb.labsilb
inflating: labsilb/12a01Lb.labsilb
inflating: labsilb/03b02La.labsilb
inflating: labsilb/08b09Lc.labsilb
inflating:
labsilb/10a01Nb.labsilb inflating:
labsilb/14b09Wc.labsilb inflating:
labsilb/11b10Ld.labsilb inflating:
labsilb/03b10Wb.labsilb inflating:
labsilb/14b03Wb.labsilb inflating:
labsilb/13a04Fc.labsilb inflating:
labsilb/16b01Wb.labsilb inflating:
labsilb/16a01Nc.labsilb inflating:
labsilb/16a01Tb.labsilb inflating:
labsilb/16a01Fc.labsilb inflating:
labsilb/14a07Ld.labsilb inflating:
labsilb/13a02Lc.labsilb inflating:
labsilb/11b09Fd.labsilb inflating:
labsilb/10b03Wb.labsilb inflating:
labsilb/13b01Nc.labsilb inflating:
labsilb/08b02Ff.labsilb inflating:
labsilb/11b03Fc.labsilb inflating:
labsilb/11b02Wb.labsilb inflating:
labsilb/11a02Nc.labsilb inflating:
labsilb/10b09Ad.labsilb inflating:
labsilb/13b01Fc.labsilb inflating:
labsilb/11a07Wc.labsilb inflating:
labsilb/14a05Tc.labsilb inflating:
labsilb/03a07Nc.labsilb inflating:
labsilb/13a05Wc.labsilb inflating:
labsilb/03b01Wa.labsilb inflating:
labsilb/14a04Wc.labsilb inflating:
labsilb/15a04Fd.labsilb inflating:
labsilb/16a07Ea.labsilb inflating:
labsilb/08b09Fd.labsilb inflating:
labsilb/11b10Td.labsilb inflating:
labsilb/16b03Ad.labsilb
creating: silb/

inflating: silb/16a05Fc.silb

inflating: silb/11b09Td.silb
inflating: silb/13a02Ta.silb
inflating:
silb/03b01Wc.silb
inflating: silb/09b03Lb.silb
inflating: silb/08b09Fd.silb
inflating: silb/11b01Eb.silb
inflating: silb/10a07Aa.silb
inflating: silb/14b09Lb.silb
inflating: silb/09b03Ed.silb
inflating: silb/15a01Nb.silb
inflating: silb/16a07Ea.silb
inflating: silb/13a07Tc.silb
inflating: silb/14b02Tc.silb
inflating: silb/08b09Tb.silb
inflating: silb/11b10Ld.silb
inflating: silb/16b03Fa.silb
inflating: silb/16a05Tb.silb
inflating: silb/03a07Nc.silb
inflating: silb/11b01Lb.silb
inflating: silb/11b02Ab.silb
inflating:
silb/08a07Wc.silb
inflating:
silb/09b02Wc.silb
inflating: silb/13a05Aa.silb
inflating: silb/16a02Ea.silb
inflating: silb/08b10La.silb
inflating: silb/10a05Wb.silb
inflating: silb/16a07La.silb
inflating: silb/14b03Wb.silb
inflating: silb/03a02Nc.silb
inflating: silb/10a02Ab.silb
inflating: silb/09b02Wd.silb
inflating:
silb/08a02Wc.silb
inflating: silb/15b09Nb.silb
inflating: silb/13a04Lb.silb
inflating: silb/16a07Lb.silb
inflating: silb/10a07Ad.silb
inflating: silb/16a02Ec.silb
inflating: silb/12a05Ta.silb
inflating: silb/16a02Lb.silb
inflating: silb/16b03Fd.silb
inflating: silb/13b10Wa.silb
inflating: silb/10b09Lb.silb
inflating: silb/16b03Ta.silb
inflating: silb/16a01Wb.silb
inflating: silb/16b10Aa.silb
inflating: silb/08b01Lb.silb
inflating: silb/11b09Na.silb
inflating: silb/12a07La.silb
inflating: silb/03a04Fd.silb
inflating: silb/03a04Ta.silb
inflating: silb/10b03Wb.silb
inflating: silb/09b01Na.silb
inflating:
silb/13b10Wc.silb
inflating: silb/11a01Nd.silb
inflating: silb/13b01Wa.silb
inflating: silb/16b09Wb.silb
inflating: silb/15a05Eb.silb
inflating: silb/09b10Nd.silb
inflating: silb/16b01Aa.silb
inflating: silb/08a01Na.silb
inflating: silb/12a02Ec.silb
inflating: silb/13b02Lc.silb
inflating: silb/13b03Ac.silb
inflating: silb/15a04Wa.silb
inflating: silb/13a07Na.silb
inflating: silb/14b02Na.silb

inflating: silb/12b03Ta.silb
inflating: silb/15a05Lb.silb
inflating: silb/03a05Wa.silb

inflating: silb/10a04Nb.silb
inflating: silb/09b03Fa.silb
inflating: silb/09a05Tb.silb
inflating: silb/15a04Wb.silb
inflating: silb/09a07Eb.silb
inflating: silb/09a02Ea.silb
inflating: silb/03a05Wb.silb
inflating:
silb/12a01Wc.silb
inflating: silb/08b09Nb.silb
inflating: silb/09a02Eb.silb
inflating: silb/09a02La.silb
inflating: silb/12b10Ac.silb
inflating: silb/03b09La.silb
inflating: silb/13a02Nc.silb
inflating: silb/03b02Tb.silb
inflating: silb/14b09Fc.silb
inflating:
silb/12b09Wc.silb
inflating: silb/09b03Fd.silb
inflating: silb/10b02Na.silb
inflating: silb/16a07Fa.silb
inflating: silb/09b03Ta.silb
inflating: silb/15b02Wa.silb
inflating: silb/11a01Aa.silb
inflating: silb/11b01Fc.silb
inflating: silb/15b09Ac.silb
inflating: silb/09a01Wb.silb
inflating: silb/14a04Aa.silb
inflating: silb/09b10Aa.silb
inflating: silb/16a07Fb.silb
inflating: silb/11a05Lc.silb
inflating: silb/14a02Wa.silb
inflating: silb/11a01Ab.silb
inflating: silb/09b09Wa.silb
inflating: silb/15b03Lc.silb
inflating: silb/13a04Fc.silb
inflating:
silb/11a04Wc.silb
inflating:
silb/14a07Wc.silb
inflating:
silb/15b02Wc.silb
inflating: silb/14b10Eb.silb
inflating: silb/11b10Td.silb
inflating: silb/16b03Nb.silb
inflating: silb/13a04Ta.silb
inflating:
silb/03b03Wc.silb
inflating: silb/08b10Fd.silb
inflating:
silb/14a02Wc.silb
inflating: silb/14b10Lb.silb
inflating: silb/14b09Td.silb
inflating: silb/12a05Nd.silb
inflating: silb/14b01Eb.silb
inflating: silb/13a01Ea.silb
inflating: silb/14b02Aa.silb
inflating: silb/16a04Ea.silb
inflating: silb/13b02Fb.silb
inflating: silb/10a07Wb.silb
inflating: silb/08a05Lc.silb
inflating: silb/11b02Wb.silb
inflating: silb/10a02Wa.silb
inflating: silb/08a01Ab.silb
inflating: silb/08b10Tc.silb
inflating: silb/08b01Fd.silb
inflating: silb/11b09Ad.silb
inflating: silb/11b03Lc.silb
inflating: silb/03a04Nc.silb
inflating: silb/13a05Wa.silb

inflating:
silb/08a04Wc.silb
inflating: silb/15b10Nb.silb
inflating: silb/09a05Nb.silb
inflating: silb/16a04La.silb
inflating: silb/08b01Fe.silb

inflating: silb/16a07Td.silb
inflating: silb/15a05Fb.silb
inflating: silb/16a02Tc.silb
inflating: silb/13a01Ec.silb
inflating: silb/15b10Nc.silb
inflating: silb/13a01Lb.silb
inflating: silb/15b01Na.silb
inflating: silb/03a01Fa.silb
inflating: silb/08b09Ab.silb
inflating: silb/16a05Ab.silb
inflating: silb/10b01Ea.silb
inflating:
silb/13a05Wc.silb
inflating: silb/03b02Na.silb
inflating: silb/13b09La.silb
inflating: silb/14a01Na.silb
inflating: silb/16a04Lc.silb
inflating: silb/10b10Lc.silb
inflating: silb/13b09Ec.silb
inflating: silb/10b02Aa.silb
inflating: silb/16b10Wa.silb
inflating: silb/13a02Ad.silb
inflating: silb/08b03Lc.silb
inflating: silb/10b01Lb.silb
inflating: silb/16b10Wb.silb
inflating: silb/15a07Eb.silb
inflating: silb/11a05Fb.silb
inflating: silb/16b02Eb.silb
inflating: silb/15a02Ea.silb
inflating: silb/09b03Nb.silb
inflating: silb/11b10Nc.silb
inflating: silb/09a07Ta.silb
inflating: silb/08b02Wd.silb
inflating: silb/12a05Ab.silb
inflating: silb/16b01Wa.silb
inflating: silb/11a05Fc.silb
inflating: silb/16b02Lb.silb
inflating: silb/15a02La.silb
inflating:
silb/13b03Wc.silb inflating:
silb/16b01Wb.silb inflating:
silb/15a01Wa.silb inflating:
silb/12b10Wa.silb inflating:
silb/11b01Nc.silb inflating:
silb/10a01Nb.silb inflating:
silb/12b02Ea.silb inflating:
silb/15a07Ld.silb inflating:
silb/16b03Ad.silb inflating:
silb/16a07Nb.silb inflating:
silb/03a07Wc.silb inflating:
silb/03a02Wb.silb inflating:
silb/09a04La.silb inflating:
silb/03b09Tc.silb inflating:
silb/08b10Nc.silb inflating:
silb/16a02Nb.silb inflating:
silb/08b01Na.silb inflating:
silb/15b03Tc.silb inflating:
silb/14b01Fa.silb inflating:
silb/03a02Wc.silb
inflating: silb/03b10Ec.silb
inflating: silb/12b01Wa.silb
inflating: silb/15b09Wb.silb
inflating: silb/03a04Ad.silb
inflating: silb/11b03Fc.silb
inflating: silb/11a05Td.silb
inflating: silb/15b10Ac.silb
inflating: silb/11a02Ec.silb
inflating: silb/16a04Fa.silb
inflating: silb/08a05Ta.silb
inflating: silb/03b02Aa.silb
inflating: silb/09b10Wa.silb

inflating: silb/14a05Lb.silb
inflating: silb/14a01Aa.silb
inflating: silb/14b01Fc.silb
inflating: silb/08a05Fe.silb
inflating: silb/11a07Ld.silb
inflating: silb/14a04Wb.silb
inflating: silb/03b01Lb.silb
inflating: silb/13b02Nb.silb
inflating: silb/15a05Na.silb
inflating: silb/14b10Tc.silb
inflating: silb/11b09Wa.silb
inflating: silb/08a07La.silb
inflating:
silb/11a01Wc.silb
inflating:
silb/14a04Wc.silb
inflating: silb/11a02Ld.silb
inflating: silb/10b10Fc.silb
inflating: silb/12a02Nb.silb
inflating: silb/10b01Fa.silb
inflating: silb/14a01Ac.silb
inflating: silb/13b09Fb.silb
inflating: silb/13a01Fd.silb
inflating: silb/08a02La.silb
inflating: silb/11b03Td.silb
inflating: silb/09b01Wb.silb
inflating: silb/08a01Wa.silb
inflating: silb/13b09Fc.silb
inflating: silb/10a04Wa.silb
inflating: silb/09a07Na.silb
inflating: silb/15a07Fa.silb
inflating: silb/11b10Ad.silb
inflating: silb/11b01Ab.silb
inflating: silb/14b09Ac.silb
inflating: silb/14b03Lb.silb
inflating: silb/08b10Aa.silb
inflating: silb/10a04Wb.silb
inflating: silb/08b03Fe.silb
inflating: silb/15a07Fb.silb
inflating: silb/11b10Ae.silb
inflating: silb/16a04Tc.silb
inflating: silb/13a07Wb.silb
inflating: silb/14b02Wb.silb
inflating: silb/03a01Nc.silb
inflating: silb/13a02Wa.silb
inflating: silb/11a05Na.silb
inflating: silb/14b03Ed.silb
inflating: silb/08b09Wa.silb
inflating:
silb/08a01Wc.silb
inflating: silb/08b03Tc.silb
inflating: silb/10a05Ld.silb
inflating: silb/10a01Ac.silb
inflating: silb/13b10La.silb
inflating: silb/16a05Wb.silb
inflating: silb/13a04Ac.silb
inflating: silb/15b03Nb.silb
inflating: silb/16a01Ec.silb
inflating: silb/16a01Lb.silb
inflating: silb/08b01Aa.silb
inflating: silb/16b02Fd.silb
inflating: silb/03b09Nc.silb
inflating: silb/14b02Wd.silb
inflating: silb/13b10Ec.silb
inflating: silb/10b03La.silb
inflating:
silb/08b09Wc.silb
inflating:
silb/16a05Wc.silb
inflating: silb/16b09Eb.silb
inflating: silb/16b09La.silb

```
inflating:  silb/15a02Ta.silb
inflating:  silb/12b02Fb.silb
inflating: silb/16b09Lb.silb
```

inflating: silb/10b02Wb.silb
inflating: silb/14a05Fa.silb
inflating: silb/10b09Ad.silb
inflating: silb/13b01Ec.silb
inflating: silb/14b10Nb.silb
inflating: silb/11a02Fb.silb
inflating: silb/03b01Fa.silb
inflating: silb/08a05Nb.silb
inflating: silb/14a05Fb.silb
inflating: silb/12a05Wb.silb
inflating: silb/11b03Nb.silb
inflating: silb/09a04Fd.silb
inflating: silb/12a07Ac.silb
inflating: silb/16b03Wb.silb
inflating: silb/12a01Lb.silb
inflating: silb/14b01Na.silb
inflating: silb/11a07Ta.silb
inflating: silb/13b01Ld.silb
inflating: silb/12a02Ac.silb
inflating: silb/09a01Ea.silb
inflating: silb/14a05Ta.silb
inflating: silb/15b10Wa.silb
inflating: silb/13a01Nb.silb
inflating:
silb/03a04Wc.silb
inflating: silb/13b09Na.silb
inflating: silb/14a07Eb.silb
inflating: silb/14a02Ea.silb
inflating: silb/11a02Tc.silb
inflating: silb/09a05Wb.silb
inflating: silb/16a04Nc.silb
inflating: silb/08b03Nb.silb
inflating: silb/15b03Aa.silb
inflating: silb/09b09Ea.silb
inflating: silb/14a05Tc.silb
inflating: silb/08a07Fd.silb
inflating: silb/08a07Ta.silb
inflating:
silb/09a05Wc.silb
inflating: silb/15b02Lb.silb
inflating: silb/14a02La.silb
inflating: silb/08a07Tb.silb
inflating: silb/09b02Tb.silb
inflating: silb/14a01Wa.silb
inflating: silb/03b01Td.silb
inflating: silb/14a07Lc.silb
inflating: silb/13b10Fa.silb
inflating: silb/11b10Wa.silb
inflating: silb/14b03Ta.silb
inflating: silb/03b02Wb.silb
inflating: silb/08a02Fe.silb
inflating: silb/08a02Tb.silb
inflating: silb/11a05Ad.silb
inflating:
silb/15b01Wc.silb
inflating: silb/10a05Tb.silb
inflating: silb/14a07Ld.silb
inflating: silb/15a02Na.silb
inflating: silb/08a04La.silb
inflating: silb/16a01Fc.silb
inflating: silb/14b09Wa.silb
inflating: silb/10a07La.silb
inflating:
silb/14a01Wc.silb
inflating: silb/13a05Ea.silb
inflating: silb/09b03Wb.silb
inflating: silb/15a07Nc.silb
inflating: silb/16b09Fb.silb
inflating: silb/10a01Wa.silb
inflating:
silb/14b09Wc.silb

inflating: silb/16a01Tb.silb
inflating: silb/03b10Na.silb
inflating: silb/10a02Lb.silb

inflating: silb/12b02Na.silb
inflating: silb/14b10Ad.silb
inflating: silb/08b10Wa.silb
inflating: silb/16a07Wa.silb
inflating: silb/13b01Fc.silb
inflating: silb/12a01Fb.silb
inflating: silb/09a04Nb.silb
inflating: silb/10b03Tb.silb
inflating: silb/13a05Lc.silb
inflating: silb/14b01Ac.silb
inflating: silb/14a05Na.silb
inflating: silb/11b01Wd.silb
inflating: silb/16a04Ab.silb
inflating: silb/08b02La.silb
inflating: silb/03b10Nc.silb
inflating:
silb/13a04Wc.silb inflating:
silb/16a02Wb.silb inflating:
silb/13a01Ac.silb inflating:
silb/08b01Wa.silb inflating:
silb/16b10Eb.silb inflating:
silb/03a05Fc.silb inflating:
silb/10b09Wb.silb inflating:
silb/15a04Fd.silb inflating:
silb/11a02Nc.silb inflating:
silb/03b01Nb.silb inflating:
silb/10b01Aa.silb inflating:
silb/09a01Fa.silb inflating:
silb/13b09Ab.silb inflating:
silb/16b10Lb.silb inflating:
silb/08a07Na.silb inflating:
silb/09b02Na.silb inflating:
silb/12a07Wa.silb inflating:
silb/13b02Wa.silb inflating:
silb/13b03Lb.silb inflating:
silb/16b01Eb.silb inflating:
silb/15a01Ea.silb inflating:
silb/16b01La.silb inflating:
silb/16b02Aa.silb inflating:
silb/08a02Na.silb inflating:
silb/12a02Wa.silb inflating:
silb/13b03Ed.silb inflating:
silb/03a07La.silb inflating:
silb/15a05Wa.silb inflating:
silb/03a05Tc.silb inflating:
silb/15a01La.silb inflating:
silb/11a04Fd.silb inflating:
silb/15a07Ac.silb inflating:
silb/16b01Lc.silb inflating:
silb/14a07Fd.silb inflating:
silb/12a02Wc.silb inflating:
silb/03a01Wa.silb inflating:
silb/12b09Td.silb inflating:
silb/15b09La.silb inflating:
silb/15a02Ac.silb inflating:
silb/14a02Fd.silb inflating:
silb/10a02Fa.silb inflating:
silb/09a07Wb.silb inflating:
silb/14a07Tc.silb inflating:
silb/15b02Tc.silb inflating:
silb/03b09Wa.silb inflating:
silb/14a02Tb.silb inflating:
silb/12b10Ld.silb inflating:
silb/03b10Ab.silb inflating:
silb/15b03Wa.silb inflating:
silb/03b03Tc.silb inflating:
silb/09a02Wb.silb inflating:
silb/14a05Aa.silb inflating:
silb/16a01Nc.silb inflating:
silb/13b10Nc.silb

inflating: silb/11a07Ac.silb
inflating: silb/15b03Wb.silb
inflating: silb/11b02Fd.silb
inflating: silb/09a07Wd.silb
inflating: silb/10a07Ta.silb
inflating: silb/14a04Lb.silb
inflating: silb/09b01Ea.silb
inflating: silb/08a04Tb.silb
inflating: silb/12b02Ad.silb
inflating: silb/14a04Ed.silb
inflating: silb/14a05Ac.silb
inflating: silb/11a05Wd.silb
inflating: silb/08a04Ff.silb
inflating: silb/11a01Ld.silb
inflating: silb/13b01Nc.silb
inflating: silb/12a01Nb.silb
inflating: silb/08a05Wa.silb
inflating: silb/11b03Wa.silb
inflating: silb/16b10Fb.silb
inflating: silb/10a05Aa.silb
inflating: silb/11b02Td.silb
inflating: silb/15a04Nc.silb
inflating: silb/16a05Ea.silb
inflating: silb/13a05Tc.silb
inflating: silb/11b03Wb.silb
inflating: silb/08a02Ab.silb
inflating:
silb/14b10Wc.silb
inflating: silb/11b09Ld.silb
inflating: silb/16b01Fa.silb
inflating: silb/13a07Lb.silb
inflating: silb/03a07Fa.silb
inflating: silb/08a01Lc.silb
inflating: silb/08a02Ac.silb
inflating: silb/16a05La.silb
inflating: silb/03a05Nd.silb
inflating: silb/13a02Ec.silb
inflating: silb/14a07Na.silb
inflating: silb/03a07Fb.silb
inflating: silb/13b03Fd.silb
inflating: silb/10b10Wa.silb
inflating: silb/08b02Ff.silb
inflating: silb/09a01Nb.silb
inflating: silb/16b10Tb.silb
inflating: silb/15a01Fb.silb
inflating: silb/08b02Tc.silb
inflating:
silb/14b01Wc.silb inflating:
silb/13a01Wb.silb inflating:
silb/14b03Ad.silb inflating:
silb/16a04Wb.silb inflating:
silb/13a02Lc.silb inflating:
silb/08b09Lc.silb inflating:
silb/15b09Fa.silb inflating:
silb/13b09Wa.silb inflating:
silb/03b03Nb.silb inflating:
silb/10b02La.silb inflating:
silb/16a04Wc.silb
inflating: silb/03a02Fc.silb
inflating: silb/16b03Ea.silb
inflating: silb/16b10Td.silb
inflating: silb/16b01Tb.silb
inflating: silb/11a04Nd.silb
inflating: silb/15b02Nd.silb
inflating: silb/14a02Nc.silb
inflating: silb/13b03Td.silb
inflating: silb/03a02Ta.silb
inflating: silb/16b09Ab.silb
inflating: silb/16b03La.silb
inflating: silb/11b02Na.silb
inflating: silb/08b03Wd.silb

inflating: silb/12a05Lb.silb
inflating: silb/13b01Ab.silb
inflating: silb/09b09Nd.silb
inflating: silb/15b09Ta.silb
inflating: silb/10a02Na.silb
inflating: silb/03a05Aa.silb
inflating: silb/16b02Wb.silb
inflating: silb/12b01Ta.silb
inflating: silb/08a04Nc.silb
inflating: silb/15a04Ab.silb
inflating:
silb/12a04Wc.silb inflating:
silb/15a02Wb.silb inflating:
silb/13a05Nb.silb inflating:
silb/15a04Ac.silb inflating:
silb/03a04Lc.silb inflating:
silb/12b03La.silb inflating:
silb/09a04Wa.silb inflating:
silb/14a04Tb.silb inflating:
silb/12b02Wa.silb inflating:
silb/12b09Ac.silb inflating:
silb/15a02Wd.silb inflating:
silb/11b09Fd.silb inflating:
silb/14a01Ea.silb inflating:
silb/15b10Lc.silb inflating:
silb/14a07Aa.silb inflating:
silb/09a05Ed.silb inflating:
silb/08b02Nb.silb inflating:
silb/15b02Aa.silb inflating:
silb/14b02Fb.silb inflating:
silb/03b10Wb.silb inflating:
silb/14a04Tc.silb inflating:
silb/13a02Fa.silb inflating:
silb/09a05Lc.silb inflating:
silb/12b02Wb.silb inflating:
silb/14a05Wa.silb inflating:
silb/03b02La.silb inflating:
silb/13b03Na.silb inflating:
silb/15b01Ec.silb inflating:
silb/15b01Lb.silb inflating:
silb/03b10Wc.silb
inflating:
silb/11a07Wc.silb inflating:
silb/03b01Wa.silb inflating:
silb/08a01Fd.silb inflating:
silb/14a05Wb.silb inflating:
silb/11a04Ac.silb inflating:
silb/14b09Ea.silb inflating:
silb/10a04Fd.silb inflating:
silb/14a02Ab.silb inflating:
silb/13a07Fd.silb inflating:
silb/12b02Wd.silb inflating:
silb/11a02Wc.silb
creating: wav/
inflating: wav/16a05Tb.wav
inflating:
wav/13a05Aa.wav inflating:
wav/03a02Nc.wav inflating:
wav/13a04Lb.wav inflating:
wav/12a05Ta.wav inflating:
wav/09a05Tb.wav inflating:
wav/08b10Fd.wav inflating:
wav/16b01Wa.wav
inflating: wav/16b02Lb.wav
inflating: wav/16b01Wb.wav
inflating:
wav/16b03Ad.wav inflating:
wav/15b03Tc.wav inflating:
wav/12b01Wa.wav
inflating: wav/11b03Td.wav
inflating: wav/09b01Wb.wav

inflating: wav/08b03Tc.wav
inflating: wav/16a05Wb.wav

inflating: wav/16a05Wc.wav
inflating: wav/16b09La.wav
inflating: wav/16b09Lb.wav
inflating: wav/13b01Ec.wav
inflating: wav/12a05Wb.wav
inflating: wav/12a07Ac.wav
inflating: wav/11a07Ta.wav
inflating: wav/09a05Wb.wav
inflating: wav/08a07Ta.wav
inflating: wav/09a05Wc.wav
inflating: wav/08a07Tb.wav
inflating:
wav/13a05Ea.wav inflating:
wav/12b02Na.wav
inflating: wav/13b01Fc.wav
inflating: wav/09b02Na.wav
inflating: wav/15a01La.wav
inflating: wav/15a02Ac.wav
inflating: wav/14a02Tb.wav
inflating: wav/15b03Wa.wav
inflating: wav/15b03Wb.wav
inflating: wav/11a01Ld.wav
inflating: wav/11b03Wa.wav
inflating: wav/11b03Wb.wav
inflating:
wav/08a02Ab.wav inflating:
wav/08a01Lc.wav inflating:
wav/08a02Ac.wav inflating:
wav/16b03Ea.wav inflating:
wav/03a02Ta.wav inflating:
wav/08b03Wd.wav
inflating: wav/09b09Nd.wav
inflating: wav/15b10Lc.wav
inflating: wav/11a07Wc.wav
inflating:
wav/09b03Ed.wav inflating:
wav/15a01Nb.wav
inflating:
wav/16a07Ea.wav inflating:
wav/11b10Ld.wav inflating:
wav/16b03Fa.wav inflating:
wav/08a07Wc.wav
inflating: wav/08b10La.wav
inflating: wav/16b03Fd.wav
inflating: wav/11a01Nd.wav
inflating: wav/08a01Na.wav
inflating: wav/09b03Fa.wav
inflating:
wav/09a07Eb.wav inflating:
wav/09b03Fd.wav inflating:
wav/16a07Fa.wav inflating:
wav/14a04Aa.wav inflating:
wav/16a07Fb.wav inflating:
wav/14a02Wa.wav
inflating: wav/13a04Ta.wav
inflating: wav/14a02Wc.wav
inflating: wav/10a02Wa.wav
inflating: wav/15b10Nb.wav
inflating:
wav/15b10Nc.wav inflating:
wav/15a02Ea.wav inflating:
wav/11b10Nc.wav inflating:
wav/03a02Wb.wav
inflating:
wav/08b10Nc.wav inflating:
wav/03a02Wc.wav
inflating:
wav/03a04Ad.wav inflating:
wav/11a02Ec.wav inflating:
wav/11a02Fb.wav inflating:
wav/13b01Ld.wav inflating:

wav/09b02Tb.wav inflating:
wav/08a02Fe.wav inflating:
wav/13a05Lc.wav inflating:
wav/13a04Wc.wav
inflating:
wav/13b09Ab.wav inflating:
wav/12b09Td.wav

inflating:
wav/14a04Ed.wav inflating:
wav/13b01Nc.wav inflating:
wav/16b03La.wav inflating:
wav/16b02Wb.wav
inflating: wav/13a05Nb.wav
inflating: wav/12b03La.wav
inflating: wav/12b02Wa.wav
inflating: wav/12b02Wb.wav
inflating: wav/10a04Fd.wav
inflating: wav/12b02Wd.wav
inflating: wav/09b03Lb.wav
inflating: wav/09b02Wc.wav
inflating: wav/16a07La.wav
inflating: wav/09b02Wd.wav
inflating: wav/16a07Lb.wav
inflating:
wav/16b10Aa.wav inflating:
wav/12a07La.wav inflating:
wav/03a04Fd.wav inflating:
wav/16b09Wb.wav
inflating: wav/12b10Ac.wav
inflating: wav/12b09Wc.wav
inflating:
wav/09b10Aa.wav inflating:
wav/09b09Wa.wav
inflating: wav/11b10Td.wav
inflating: wav/16b03Nb.wav
inflating: wav/13b02Fb.wav
inflating: wav/08b10Tc.wav
inflating: wav/13b09Ec.wav
inflating: wav/09b03Nb.wav
inflating: wav/15a02La.wav
inflating: wav/15a01Wa.wav
inflating: wav/16a07Nb.wav
inflating: wav/11a01Wc.wav
inflating: wav/11a02Ld.wav
inflating: wav/13b09Fb.wav
inflating: wav/08a02La.wav
inflating: wav/08a01Wa.wav
inflating: wav/13b09Fc.wav
inflating: wav/09a07Na.wav
inflating: wav/08a01Wc.wav
inflating: wav/16a01Ec.wav
inflating:
wav/09a01Ea.wav inflating:
wav/15b10Wa.wav inflating:
wav/11b10Wa.wav
inflating: wav/15a02Na.wav
inflating: wav/16a01Fc.wav
inflating: wav/08b10Wa.wav
inflating: wav/12a01Fb.wav
inflating: wav/14b01Ac.wav
inflating:
wav/16b10Eb.wav inflating:
wav/11a02Nc.wav inflating:
wav/10b01Aa.wav inflating:
wav/09a01Fa.wav inflating:
wav/08a02Na.wav
inflating:
wav/14a05Aa.wav inflating:
wav/14a04Lb.wav inflating:
wav/14a05Ac.wav inflating:
wav/16b10Fb.wav inflating:
wav/10a05Aa.wav inflating:
wav/13a05Tc.wav inflating:
wav/03a05Aa.wav inflating:
wav/03a04Lc.wav inflating:
wav/16b03Ta.wav inflating:
wav/13b01Wa.wav
inflating: wav/13b02Lc.wav

```
inflating: wav/13b03Ac.wav
inflating: wav/12b03Ta.wav
inflating: wav/10a04Nb.wav
inflating: wav/09b03Ta.wav
```

inflating:
wav/14b01Eb.wav inflating:
wav/03a04Nc.wav inflating:
wav/13a05Wa.wav
inflating: wav/16a07Td.wav
inflating:
wav/10b01Ea.wav inflating:
wav/13a05Wc.wav
inflating: wav/13b09La.wav
inflating: wav/09a07Ta.wav
inflating: wav/14b01Fa.wav
inflating: wav/14b01Fc.wav
inflating: wav/13b02Nb.wav
inflating: wav/10b01Fa.wav
inflating: wav/16a01Lb.wav
inflating: wav/15a02Ta.wav
inflating: wav/14a05Fa.wav
inflating: wav/03b01Fa.wav
inflating: wav/14a05Fb.wav
inflating: wav/16b03Wb.wav
inflating: wav/12a01Lb.wav
inflating: wav/12a02Ac.wav
inflating: wav/13b09Na.wav
inflating: wav/11a02Tc.wav
inflating: wav/08a02Tb.wav
inflating: wav/09b03Wb.wav
inflating: wav/16a07Wa.wav
inflating: wav/03a05Fc.wav
inflating: wav/16b10Lb.wav
inflating: wav/12a07Wa.wav
inflating:
wav/13b03Ed.wav inflating:
wav/09a07Wb.wav
inflating: wav/12b10Ld.wav
inflating:
wav/16a01Nc.wav inflating:
wav/09a07Wd.wav
inflating: wav/12a01Nb.wav
inflating: wav/13b03Fd.wav
inflating: wav/09a01Nb.wav
inflating:
wav/15a04Ab.wav inflating:
wav/15a02Wb.wav
inflating: wav/15a04Ac.wav
inflating: wav/14a04Tb.wav
inflating: wav/15a02Wd.wav
inflating: wav/14a04Tc.wav
inflating: wav/11a04Ac.wav
inflating: wav/13a07Fd.wav
inflating: wav/11a02Wc.wav
inflating:
wav/16a02Ea.wav inflating:
wav/08a02Wc.wav
inflating: wav/16a02Ec.wav
inflating: wav/03a04Ta.wav
inflating: wav/09b10Nd.wav
inflating: wav/12a02Ec.wav
inflating:
wav/09a02Ea.wav inflating:
wav/09a02Eb.wav inflating:
wav/14b02Aa.wav inflating:
wav/10b02Aa.wav inflating:
wav/10b01Lb.wav inflating:
wav/03b02Aa.wav inflating:
wav/14a05Lb.wav inflating:
wav/14a04Wb.wav
inflating: wav/03b01Lb.wav
inflating: wav/14a04Wc.wav
inflating: wav/10a04Wa.wav
inflating: wav/14b09Ac.wav
inflating: wav/10a04Wb.wav

inflating: wav/10a05Ld.wav
inflating:
wav/10b09Ad.wav inflating:
wav/14b01Na.wav inflating:
wav/03a04Wc.wav
inflating: wav/16a01Tb.wav

```
inflating: wav/14a05Na.wav
inflating: wav/13a01Ac.wav
inflating: wav/15a04Fd.wav
inflating: wav/03b01Nb.wav
inflating: wav/13b02Wa.wav
inflating: wav/13b03Lb.wav
inflating: wav/11a04Fd.wav
inflating: wav/08a04Ff.wav
inflating: wav/13a07Lb.wav
inflating: wav/03a05Nd.wav
inflating: wav/16b10Tb.wav
inflating: wav/13b09Wa.wav
inflating: wav/16b10Td.wav
inflating: wav/14b02Fb.wav
inflating: wav/13b03Na.wav
inflating: wav/14b09Ea.wav
inflating: wav/16a02Lb.wav
inflating: wav/16a01Wb.wav
inflating: wav/13a07Na.wav
inflating: wav/12a01Wc.wav
inflating: wav/09a02La.wav
inflating: wav/14b09Fc.wav
inflating: wav/09a01Wb.wav
inflating: wav/13a01Ea.wav
inflating: wav/13a01Ec.wav
inflating: wav/16b10Wa.wav
inflating: wav/16b10Wb.wav
inflating: wav/12b10Wa.wav
inflating: wav/16a02Nb.wav
inflating: wav/09b10Wa.wav
inflating: wav/12a02Nb.wav
inflating: wav/13a01Fd.wav
inflating: wav/11b01Ab.wav
inflating: wav/08b01Aa.wav
inflating: wav/13b10Ec.wav
inflating: wav/14a05Ta.wav
inflating: wav/14a05Tc.wav
inflating: wav/03b01Td.wav
inflating: wav/13b10Fa.wav
inflating: wav/11a05Ad.wav
inflating: wav/10a05Tb.wav
inflating: wav/08a04La.wav
inflating: wav/03a05Tc.wav
inflating: wav/15a04Nc.wav
inflating: wav/14b01Wc.wav
inflating: wav/14b03Ad.wav
inflating: wav/10b02La.wav
inflating: wav/11a04Nd.wav
inflating: wav/13b03Td.wav
inflating: wav/08a04Nc.wav
inflating: wav/14a07Aa.wav
inflating: wav/14a05Wa.wav
inflating: wav/03b02La.wav
inflating: wav/15b01Ec.wav
inflating: wav/03b01Wa.wav
inflating: wav/14a05Wb.wav
inflating: wav/03b01Wc.wav
inflating: wav/11b01Eb.wav
inflating: wav/10a07Aa.wav
inflating: wav/14b09Lb.wav
inflating: wav/13a07Tc.wav
inflating: wav/10a05Wb.wav
inflating: wav/10a07Ad.wav
inflating: wav/10b09Lb.wav
inflating: wav/15a05Eb.wav
inflating: wav/14b02Na.wav
inflating: wav/03a05Wa.wav
inflating: wav/03a05Wb.wav
inflating: wav/03b09La.wav
```

inflating: wav/10b02Na.wav
inflating: wav/11b01Fc.wav
inflating: wav/08b01Fd.wav
inflating: wav/08b01Fe.wav
inflating: wav/15a05Fb.wav
inflating: wav/16a02Tc.wav
inflating: wav/13a01Lb.wav
inflating: wav/03b02Na.wav
inflating:
wav/13a02Ad.wav inflating:
wav/11a05Fb.wav inflating:
wav/11a05Fc.wav inflating:
wav/13b03Wc.wav
inflating: wav/08a05Fe.wav
inflating: wav/13a07Wb.wav
inflating:
wav/14b03Ed.wav inflating:
wav/13b10La.wav inflating:
wav/03b09Nc.wav inflating:
wav/13a01Nb.wav
inflating:
wav/14a07Eb.wav inflating:
wav/16a04Ab.wav inflating:
wav/16a02Wb.wav inflating:
wav/12a02Wa.wav
inflating: wav/14a07Fd.wav
inflating: wav/12a02Wc.wav
inflating: wav/09a02Wb.wav
inflating:
wav/13b10Nc.wav inflating:
wav/08a04Tb.wav inflating:
wav/03a07Fa.wav inflating:
wav/13a02Ec.wav inflating:
wav/03a07Fb.wav inflating:
wav/15b02Aa.wav inflating:
wav/13a02Fa.wav inflating:
wav/15b01Lb.wav inflating:
wav/14b02Tc.wav inflating:
wav/11b01Lb.wav inflating:
wav/11b02Ab.wav inflating:
wav/08b01Lb.wav inflating:
wav/15a04Wa.wav
inflating: wav/15a05Lb.wav
inflating: wav/15a04Wb.wav
inflating: wav/03b02Tb.wav
inflating: wav/15b09Ac.wav
inflating: wav/11a05Lc.wav
inflating: wav/11a04Wc.wav
inflating: wav/14b09Td.wav
inflating:
wav/16a04Ea.wav inflating:
wav/08a05Lc.wav inflating:
wav/11b09Ad.wav inflating:
wav/08a04Wc.wav
inflating: wav/15b01Na.wav
inflating:
wav/08b09Ab.wav inflating:
wav/11b01Nc.wav inflating:
wav/03b09Tc.wav inflating:
wav/08b01Na.wav
inflating: wav/16a04Fa.wav
inflating:
wav/14a01Aa.wav inflating:
wav/15a05Na.wav
inflating: wav/14a01Ac.wav
inflating: wav/14b03Lb.wav
inflating: wav/14b02Wb.wav
inflating: wav/11a05Na.wav
inflating: wav/10a01Ac.wav
inflating: wav/14b02Wd.wav
inflating: wav/10b03La.wav

inflating: wav/10b02Wb.wav
inflating: wav/08a05Nb.wav
inflating: wav/09a04Fd.wav
inflating: wav/14a07Lc.wav
inflating: wav/03b02Wb.wav

inflating: wav/14a07Ld.wav
inflating: wav/14b09Wa.wav
inflating: wav/10a07La.wav
inflating: wav/14b09Wc.wav
inflating: wav/14b10Ad.wav
inflating: wav/10b09Wb.wav
inflating: wav/03a07La.wav
inflating: wav/03b09Wa.wav
inflating: wav/03b10Ab.wav
inflating: wav/11b02Fd.wav
inflating: wav/14a07Na.wav
inflating: wav/08b02Ff.wav
inflating: wav/13a01Wb.wav
inflating: wav/13a02Lc.wav
inflating: wav/15b09Fa.wav
inflating: wav/03b03Nb.wav
inflating: wav/11b09Fd.wav
inflating: wav/14a01Ea.wav
inflating: wav/08b09Fd.wav
inflating: wav/03a07Nc.wav
inflating: wav/13b10Wa.wav
inflating: wav/13b10Wc.wav
inflating: wav/16b01Aa.wav
inflating: wav/13a02Nc.wav
inflating: wav/14b10Eb.wav
inflating: wav/16a04La.wav
inflating: wav/03a01Fa.wav
inflating: wav/16a05Ab.wav
inflating: wav/16a04Lc.wav
inflating: wav/12a05Ab.wav
inflating: wav/09a04La.wav
inflating: wav/03b10Ec.wav
inflating: wav/11a05Td.wav
inflating: wav/08a05Ta.wav
inflating: wav/10b10Fc.wav
inflating: wav/16a04Nc.wav
inflating: wav/15b03Aa.wav
inflating: wav/15b02Lb.wav
inflating: wav/14b03Ta.wav
inflating: wav/15b01Wc.wav
inflating: wav/09a04Nb.wav
inflating: wav/10b03Tb.wav
inflating: wav/11b01Wd.wav
inflating: wav/08b02La.wav
inflating: wav/08b01Wa.wav
inflating: wav/16b01Eb.wav
inflating: wav/15a05Wa.wav
inflating: wav/15a07Ac.wav
inflating: wav/15b09La.wav
inflating: wav/14a07Tc.wav
inflating: wav/03b03Tc.wav
inflating: wav/11a07Ac.wav
inflating: wav/10a07Ta.wav
inflating: wav/09b01Ea.wav
inflating: wav/11a05Wd.wav
inflating: wav/08a05Wa.wav
inflating: wav/16a05Ea.wav
inflating: wav/11b09Ld.wav
inflating: wav/16b01Fa.wav
inflating: wav/08b09Lc.wav
inflating: wav/15b02Nd.wav
inflating: wav/11b02Na.wav
inflating: wav/09a05Ed.wav
inflating: wav/08b02Nb.wav
inflating: wav/14a02Ab.wav
inflating: wav/16a05Fc.wav
inflating: wav/13a02Ta.wav
inflating: wav/14b03Wb.wav
inflating: wav/10a02Ab.wav

inflating: wav/15b09Nb.wav
inflating: wav/11b09Na.wav
inflating: wav/10b03Wb.wav
inflating: wav/08b09Nb.wav
inflating: wav/14a07Wc.wav
inflating: wav/03b03Wc.wav
inflating: wav/14b10Lb.wav
inflating: wav/10a07Wb.wav
inflating: wav/14a01Na.wav
inflating: wav/10b10Lc.wav
inflating:
wav/15a07Eb.wav inflating:
wav/10a01Nb.wav inflating:
wav/03a07Wc.wav
inflating: wav/11b03Fc.wav
inflating: wav/15a07Fa.wav
inflating: wav/08b03Fe.wav
inflating: wav/15a07Fb.wav
inflating: wav/16a04Tc.wav
inflating:
wav/03a01Nc.wav inflating:
wav/13a02Wa.wav
inflating: wav/13a04Ac.wav
inflating: wav/14b10Nb.wav
inflating:
wav/14a02Ea.wav inflating:
wav/08a07Fd.wav inflating:
wav/03b10Na.wav
inflating:
wav/03b10Nc.wav inflating:
wav/16b01La.wav inflating:
wav/16b02Aa.wav inflating:
wav/16b01Lc.wav inflating:
wav/14a02Fd.wav inflating:
wav/10a02Fa.wav inflating:
wav/15b02Tc.wav inflating:
wav/12b02Ad.wav inflating:
wav/11b02Td.wav inflating:
wav/16a05La.wav inflating:
wav/08b02Tc.wav inflating:
wav/16a04Wb.wav inflating:
wav/16a04Wc.wav
inflating: wav/03a02Fc.wav
inflating:
wav/16b09Ab.wav inflating:
wav/12a05Lb.wav inflating:
wav/15b09Ta.wav inflating:
wav/12a04Wc.wav inflating:
wav/09a04Wa.wav
inflating: wav/12b09Ac.wav
inflating: wav/09a05Lc.wav
inflating: wav/11b09Td.wav
inflating: wav/08b09Tb.wav
inflating: wav/09b01Na.wav
inflating: wav/15b02Wa.wav
inflating:
wav/11a01Aa.wav inflating:
wav/11a01Ab.wav inflating:
wav/15b03Lc.wav inflating:
wav/13a04Fc.wav inflating:
wav/15b02Wc.wav
inflating: wav/12a05Nd.wav
inflating: wav/11b02Wb.wav
inflating:
wav/08a01Ab.wav inflating:
wav/11b03Lc.wav inflating:
wav/09a05Nb.wav
inflating: wav/08b03Lc.wav
inflating:
wav/16b02Eb.wav inflating:
wav/08b02Wd.wav

inflating:
wav/12b02Ea.wav inflating:
wav/15a07Ld.wav inflating:
wav/15b09Wb.wav
inflating: wav/15b10Ac.wav
inflating: wav/11a07Ld.wav
inflating: wav/14b10Tc.wav

```
inflating: wav/11b09Wa.wav
inflating: wav/08a07La.wav
inflating:
wav/11b10Ad.wav inflating:
wav/08b10Aa.wav inflating:
wav/11b10Ae.wav inflating:
wav/08b09Wa.wav
inflating: wav/15b03Nb.wav
inflating: wav/16b02Fd.wav
inflating: wav/08b09Wc.wav
inflating:
wav/16b09Eb.wav inflating:
wav/12b02Fb.wav inflating:
wav/11b03Nb.wav inflating:
wav/08b03Nb.wav
inflating:
wav/09b09Ea.wav inflating:
wav/14a02La.wav inflating:
wav/14a01Wa.wav inflating:
wav/14a01Wc.wav
inflating:
wav/15a07Nc.wav inflating:
wav/16b09Fb.wav inflating:
wav/10a01Wa.wav
inflating: wav/10a02Lb.wav
inflating: wav/08a07Na.wav
inflating:
wav/15a01Ea.wav inflating:
wav/03a01Wa.wav inflating:
wav/14b10Wc.wav inflating:
wav/10b10Wa.wav
inflating: wav/15a01Fb.wav
inflating: wav/16b01Tb.wav
inflating:
wav/14a02Nc.wav inflating:
wav/13b01Ab.wav inflating:
wav/10a02Na.wav
inflating: wav/12b01Ta.wav
inflating: wav/03b10Wb.wav
inflating: wav/03b10Wc.wav
inflating: wav/08a01Fd.wav
inflating: erkennung.txt
inflating: erklaerung.txt
```

In [15]:

```
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_mel mel_db = (librosa.power_

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
```

In [16]:

```
# 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]))
```

In [17]:

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

In [18]:

```
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=
```

In [19]:

```
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 path2  =  layers.Conv2D(filters_
```

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

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

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

In [20]:

```
inp = layers.Input(shape=(157, 320, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="b

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)
```

In [21]:

```
model = Model(inputs = inp, outputs = [out, aux1, aux2])

model.compile(optimizer='adam', loss=[losses.sparse_categorical_crossentropy, losses
```

In [22]:

```
history = model.fit(X_train, [y_train, y_train, y_train], validation_data=(X_test, [
```

Epoch 1/30
6/6 [==============================] - 11s 1s/step - loss: 3.4348 - dense_9_loss: 2. 1473 - dense_6_loss: 2.1199 - dense_8_loss: 2.1716 - dense_9_accuracy: 0.1620 - dens e_6_accuracy: 0.1963 - dense_8_accuracy: 0.2025 - val_loss: 3.3562 - val_dense_9_los s: 2.0970 - val_dense_6_loss: 2.0468 - val_dense_8_loss: 2.1506 - val_dense_9_accura cy: 0.2570 - val_dense_6_accuracy: 0.1776 - val_dense_8_accuracy: 0.1075

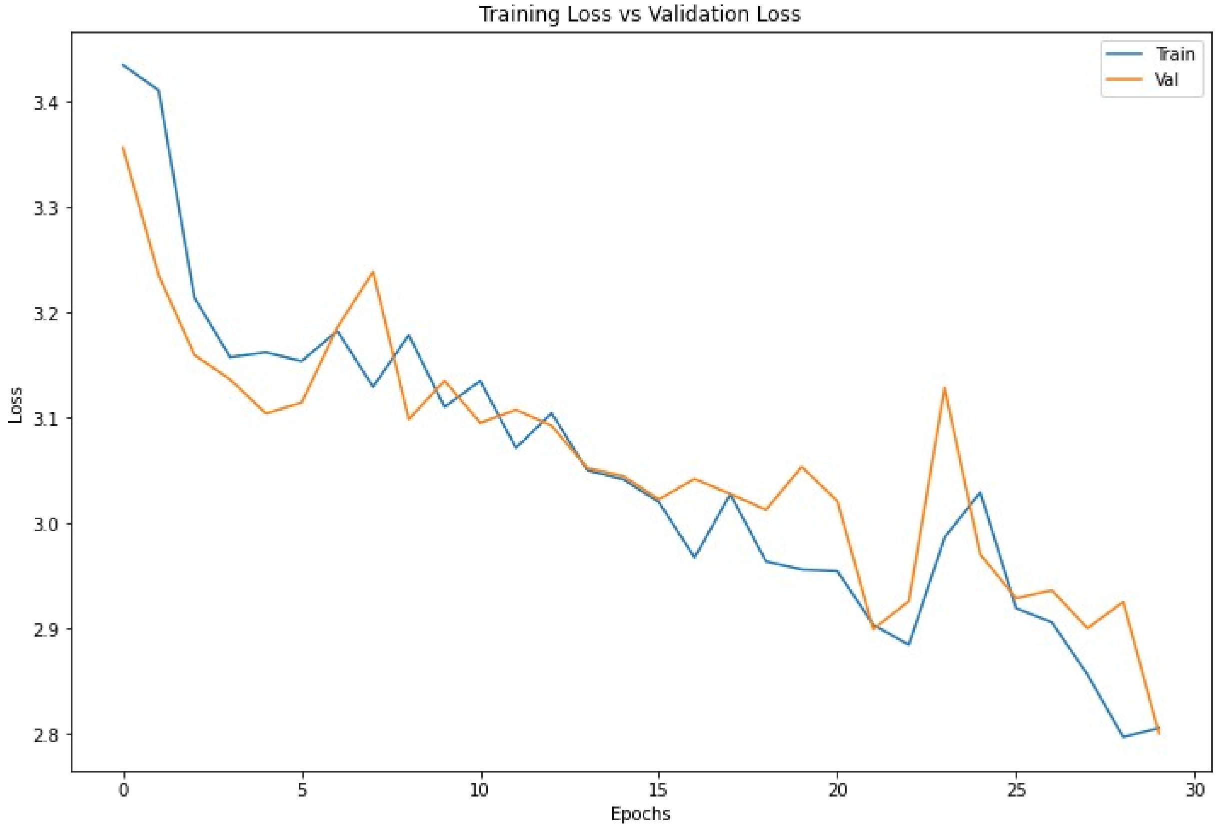
```
Epoch 2/30
6/6 [==============================] - 2s 423ms/step - loss: 3.4108 - dense_9_loss:
2.1265 - dense_6_loss: 2.0958 - dense_8_loss: 2.1852 - dense_9_accuracy: 0.1215 - de
nse_6_accuracy: 0.2025 - dense_8_accuracy: 0.1558 - val_loss: 3.2353 - val_dense_9_l
oss: 2.0143 - val_dense_6_loss: 1.9862 - val_dense_8_loss: 2.0839 - val_dense_9_accu
racy: 0.1028 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.1075
Epoch 3/30
6/6 [==============================] - 2s 420ms/step - loss: 3.2141 - dense_9_loss:
2.0217 - dense_6_loss: 1.9672 - dense_8_loss: 2.0074 - dense_9_accuracy: 0.1589 - de
nse_6_accuracy: 0.2181 - dense_8_accuracy: 0.1589 - val_loss: 3.1598 - val_dense_9_l
oss: 1.9761 - val_dense_6_loss: 1.9544 - val_dense_8_loss: 1.9912 - val_dense_9_accu
racy: 0.1215 - val_dense_6_accuracy: 0.2944 - val_dense_8_accuracy: 0.2570
Epoch 4/30
6/6 [==============================] - 3s 423ms/step - loss: 3.1577 - dense_9_loss:
1.9725 - dense_6_loss: 1.9663 - dense_8_loss: 1.9841 - dense_9_accuracy: 0.1526 - de
nse_6_accuracy: 0.1745 - dense_8_accuracy: 0.1963 - val_loss: 3.1364 - val_dense_9_l
oss: 1.9780 - val_dense_6_loss: 1.9127 - val_dense_8_loss: 1.9486 - val_dense_9_accu
racy: 0.1449 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.2570
Epoch 5/30
6/6 [==============================] - 2s 421ms/step - loss: 3.1622 - dense_9_loss:
1.9809 - dense_6_loss: 1.9399 - dense_8_loss: 1.9981 - dense_9_accuracy: 0.1963 - de
nse_6_accuracy: 0.2056 - dense_8_accuracy: 0.2025 - val_loss: 3.1043 - val_dense_9_l
oss: 1.9530 - val_dense_6_loss: 1.8987 - val_dense_8_loss: 1.9387 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.2570
Epoch 6/30
6/6 [==============================] - 2s 421ms/step - loss: 3.1539 - dense_9_loss:
1.9659 - dense_6_loss: 1.9675 - dense_8_loss: 1.9927 - dense_9_accuracy: 0.2243 - de
nse_6_accuracy: 0.2274 - dense_8_accuracy: 0.2056 - val_loss: 3.1145 - val_dense_9_l
oss: 1.9558 - val_dense_6_loss: 1.9173 - val_dense_8_loss: 1.9452 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.2570
Epoch 7/30
6/6 [==============================] - 2s 420ms/step - loss: 3.1826 - dense_9_loss:
1.9950 - dense_6_loss: 1.9679 - dense_8_loss: 1.9907 - dense_9_accuracy: 0.2212 - de
nse_6_accuracy: 0.2305 - dense_8_accuracy: 0.1963 - val_loss: 3.1860 - val_dense_9_l
oss: 1.9891 - val_dense_6_loss: 1.9936 - val_dense_8_loss: 1.9958 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.2570
Epoch 8/30
6/6 [==============================] - 2s 420ms/step - loss: 3.1296 - dense_9_loss:
1.9507 - dense_6_loss: 1.9535 - dense_8_loss: 1.9762 - dense_9_accuracy: 0.2181 - de
nse_6_accuracy: 0.2118 - dense_8_accuracy: 0.1931 - val_loss: 3.2386 - val_dense_9_l
oss: 2.0339 - val_dense_6_loss: 2.0069 - val_dense_8_loss: 2.0088 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.2570
Epoch 9/30
6/6 [==============================] - 2s 419ms/step - loss: 3.1788 - dense_9_loss:
1.9753 - dense_6_loss: 1.9628 - dense_8_loss: 2.0488 - dense_9_accuracy: 0.2212 - de
nse_6_accuracy: 0.2523 - dense_8_accuracy: 0.1869 - val_loss: 3.0988 - val_dense_9_l
oss: 1.9370 - val_dense_6_loss: 1.9253 - val_dense_8_loss: 1.9471 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.2570
Epoch 10/30
6/6 [==============================] - 2s 421ms/step - loss: 3.1103 - dense_9_loss:
1.9383 - dense_6_loss: 1.9409 - dense_8_loss: 1.9659 - dense_9_accuracy: 0.2274 - de
nse_6_accuracy: 0.2336 - dense_8_accuracy: 0.1931 - val_loss: 3.1353 - val_dense_9_l
oss: 1.9619 - val_dense_6_loss: 1.9348 - val_dense_8_loss: 1.9765 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.2570
Epoch 11/30
6/6 [==============================] - 2s 418ms/step - loss: 3.1352 - dense_9_loss:
1.9541 - dense_6_loss: 1.9574 - dense_8_loss: 1.9793 - dense_9_accuracy: 0.2150 - de
nse_6_accuracy: 0.2118 - dense_8_accuracy: 0.1869 - val_loss: 3.0954 - val_dense_9_l
oss: 1.9431 - val_dense_6_loss: 1.8902 - val_dense_8_loss: 1.9508 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.2570
Epoch 12/30
6/6 [==============================] - 3s 422ms/step - loss: 3.0718 - dense_9_loss:
1.9190 - dense_6_loss: 1.8908 - dense_8_loss: 1.9518 - dense_9_accuracy: 0.2305 - de
nse_6_accuracy: 0.2399 - dense_8_accuracy: 0.2056 - val_loss: 3.1077 - val_dense_9_l
oss: 1.9607 - val_dense_6_loss: 1.8640 - val_dense_8_loss: 1.9594 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.3084 - val_dense_8_accuracy: 0.2617
Epoch 13/30
6/6 [==============================] - 2s 421ms/step - loss: 3.1045 - dense_9_loss:
1.9518 - dense_6_loss: 1.8865 - dense_8_loss: 1.9560 - dense_9_accuracy: 0.1807 - de
```

nse_6_accuracy: 0.2710 - dense_8_accuracy: 0.1994 - val_loss: 3.0927 - val_dense_9_l
oss: 1.9642 - val_dense_6_loss: 1.8241 - val_dense_8_loss: 1.9378 - val_dense_9_accu
racy: 0.1215 - val_dense_6_accuracy: 0.3131 - val_dense_8_accuracy: 0.2570
Epoch 14/30
6/6 [==============================] - 2s 420ms/step - loss: 3.0503 - dense_9_loss:
1.9241 - dense_6_loss: 1.8323 - dense_8_loss: 1.9218 - dense_9_accuracy: 0.2212 - de
nse_6_accuracy: 0.2897 - dense_8_accuracy: 0.2461 - val_loss: 3.0524 - val_dense_9_l
oss: 1.9292 - val_dense_6_loss: 1.8200 - val_dense_8_loss: 1.9242 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.3224 - val_dense_8_accuracy: 0.2570
Epoch 15/30
6/6 [==============================] - 2s 417ms/step - loss: 3.0419 - dense_9_loss:
1.9176 - dense_6_loss: 1.8138 - dense_8_loss: 1.9341 - dense_9_accuracy: 0.2243 - de
nse_6_accuracy: 0.3209 - dense_8_accuracy: 0.2181 - val_loss: 3.0450 - val_dense_9_l
oss: 1.9304 - val_dense_6_loss: 1.7860 - val_dense_8_loss: 1.9295 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.3271 - val_dense_8_accuracy: 0.2570
Epoch 16/30
6/6 [==============================] - 2s 422ms/step - loss: 3.0203 - dense_9_loss:
1.9193 - dense_6_loss: 1.7365 - dense_8_loss: 1.9334 - dense_9_accuracy: 0.2305 - de
nse_6_accuracy: 0.3520 - dense_8_accuracy: 0.2399 - val_loss: 3.0228 - val_dense_9_l
oss: 1.9291 - val_dense_6_loss: 1.7423 - val_dense_8_loss: 1.9035 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.3224 - val_dense_8_accuracy: 0.2570
Epoch 17/30
6/6 [==============================] - 2s 418ms/step - loss: 2.9676 - dense_9_loss:
1.9036 - dense_6_loss: 1.6555 - dense_8_loss: 1.8910 - dense_9_accuracy: 0.2181 - de
nse_6_accuracy: 0.3364 - dense_8_accuracy: 0.2461 - val_loss: 3.0423 - val_dense_9_l
oss: 1.9260 - val_dense_6_loss: 1.8040 - val_dense_8_loss: 1.9168 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2430 - val_dense_8_accuracy: 0.3178
Epoch 18/30
6/6 [==============================] - 2s 415ms/step - loss: 3.0278 - dense_9_loss:
1.9179 - dense_6_loss: 1.7855 - dense_8_loss: 1.9141 - dense_9_accuracy: 0.2150 - de
nse_6_accuracy: 0.2773 - dense_8_accuracy: 0.2897 - val_loss: 3.0280 - val_dense_9_l
oss: 1.9323 - val_dense_6_loss: 1.7583 - val_dense_8_loss: 1.8939 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2944 - val_dense_8_accuracy: 0.3318
Epoch 19/30
6/6 [==============================] - 3s 423ms/step - loss: 2.9640 - dense_9_loss:
1.9146 - dense_6_loss: 1.6611 - dense_8_loss: 1.8366 - dense_9_accuracy: 0.2555 - de
nse_6_accuracy: 0.3863 - dense_8_accuracy: 0.2960 - val_loss: 3.0129 - val_dense_9_l
oss: 1.9254 - val_dense_6_loss: 1.7189 - val_dense_8_loss: 1.9060 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.3037 - val_dense_8_accuracy: 0.2617
Epoch 20/30
6/6 [==============================] - 2s 419ms/step - loss: 2.9562 - dense_9_loss:
1.9040 - dense_6_loss: 1.6664 - dense_8_loss: 1.8410 - dense_9_accuracy: 0.2305 - de
nse_6_accuracy: 0.3427 - dense_8_accuracy: 0.2897 - val_loss: 3.0538 - val_dense_9_l
oss: 1.9108 - val_dense_6_loss: 1.7944 - val_dense_8_loss: 2.0154 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.3364 - val_dense_8_accuracy: 0.2897
Epoch 21/30
6/6 [==============================] - 2s 419ms/step - loss: 2.9548 - dense_9_loss:
1.8982 - dense_6_loss: 1.6623 - dense_8_loss: 1.8596 - dense_9_accuracy: 0.2461 - de
nse_6_accuracy: 0.3458 - dense_8_accuracy: 0.2804 - val_loss: 3.0211 - val_dense_9_l
oss: 1.8970 - val_dense_6_loss: 1.8481 - val_dense_8_loss: 1.8988 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.2103 - val_dense_8_accuracy: 0.1869
Epoch 22/30
6/6 [==============================] - 2s 417ms/step - loss: 2.9038 - dense_9_loss:
1.8782 - dense_6_loss: 1.6369 - dense_8_loss: 1.7817 - dense_9_accuracy: 0.2274 - de
nse_6_accuracy: 0.2960 - dense_8_accuracy: 0.2897 - val_loss: 2.8996 - val_dense_9_l
oss: 1.8772 - val_dense_6_loss: 1.6389 - val_dense_8_loss: 1.7689 - val_dense_9_accu
racy: 0.2570 - val_dense_6_accuracy: 0.3505 - val_dense_8_accuracy: 0.3084
Epoch 23/30
6/6 [==============================] - 2s 421ms/step - loss: 2.8848 - dense_9_loss:
1.8863 - dense_6_loss: 1.5750 - dense_8_loss: 1.7534 - dense_9_accuracy: 0.2617 - de
nse_6_accuracy: 0.3583 - dense_8_accuracy: 0.2866 - val_loss: 2.9262 - val_dense_9_l
oss: 1.9005 - val_dense_6_loss: 1.6990 - val_dense_8_loss: 1.7199 - val_dense_9_accu
racy: 0.2944 - val_dense_6_accuracy: 0.3645 - val_dense_8_accuracy: 0.3598
Epoch 24/30
6/6 [==============================] - 2s 419ms/step - loss: 2.9867 - dense_9_loss:
1.9122 - dense_6_loss: 1.7453 - dense_8_loss: 1.8364 - dense_9_accuracy: 0.2679 - de
nse_6_accuracy: 0.3676 - dense_8_accuracy: 0.3645 - val_loss: 3.1287 - val_dense_9_l
oss: 1.9493 - val_dense_6_loss: 1.8893 - val_dense_8_loss: 2.0418 - val_dense_9_accu
racy: 0.1121 - val_dense_6_accuracy: 0.2383 - val_dense_8_accuracy: 0.1729

```
Epoch 25/30
6/6 [==============================] - 2s 416ms/step - loss: 3.0295 - dense_9_loss: 1.9105 - dense_6_loss: 1.8124 - dense_8_loss: 1.9174 - dense_9_accuracy: 0.1807 - dense_6_accuracy: 0.2430 - dense_8_accuracy: 0.2150 - val_loss: 2.9706 - val_dense_9_loss: 1.9398 - val_dense_6_loss: 1.6632 - val_dense_8_loss: 1.7729 - val_dense_9_accuracy: 0.2336 - val_dense_6_accuracy: 0.3364 - val_dense_8_accuracy: 0.3224
Epoch 26/30
6/6 [==============================] - 2s 421ms/step - loss: 2.9195 - dense_9_loss: 1.8782 - dense_6_loss: 1.7151 - dense_8_loss: 1.7559 - dense_9_accuracy: 0.2555 - dense_6_accuracy: 0.3053 - dense_8_accuracy: 0.2928 - val_loss: 2.9289 - val_dense_9_loss: 1.8914 - val_dense_6_loss: 1.6787 - val_dense_8_loss: 1.7799 - val_dense_9_accuracy: 0.2570 - val_dense_6_accuracy: 0.3364 - val_dense_8_accuracy: 0.2944
Epoch 27/30
6/6 [==============================] - 2s 418ms/step - loss: 2.9062 - dense_9_loss: 1.8775 - dense_6_loss: 1.6509 - dense_8_loss: 1.7779 - dense_9_accuracy: 0.2274 - dense_6_accuracy: 0.3240 - dense_8_accuracy: 0.2710 - val_loss: 2.9366 - val_dense_9_loss: 1.8877 - val_dense_6_loss: 1.7431 - val_dense_8_loss: 1.7533 - val_dense_9_accuracy: 0.2570 - val_dense_6_accuracy: 0.2570 - val_dense_8_accuracy: 0.3364
Epoch 28/30
6/6 [==============================] - 2s 419ms/step - loss: 2.8565 - dense_9_loss: 1.8677 - dense_6_loss: 1.6412 - dense_8_loss: 1.6549 - dense_9_accuracy: 0.2243 - dense_6_accuracy: 0.3458 - dense_8_accuracy: 0.3146 - val_loss: 2.9005 - val_dense_9_loss: 1.8728 - val_dense_6_loss: 1.7038 - val_dense_8_loss: 1.7217 - val_dense_9_accuracy: 0.2570 - val_dense_6_accuracy: 0.3037 - val_dense_8_accuracy: 0.3271
Epoch 29/30
6/6 [==============================] - 2s 420ms/step - loss: 2.7974 - dense_9_loss: 1.8339 - dense_6_loss: 1.5888 - dense_8_loss: 1.6230 - dense_9_accuracy: 0.2492 - dense_6_accuracy: 0.3925 - dense_8_accuracy: 0.3427 - val_loss: 2.9255 - val_dense_9_loss: 1.9051 - val_dense_6_loss: 1.6680 - val_dense_8_loss: 1.7333 - val_dense_9_accuracy: 0.3178 - val_dense_6_accuracy: 0.3271 - val_dense_8_accuracy: 0.3178
Epoch 30/30
6/6 [==============================] - 2s 416ms/step - loss: 2.8058 - dense_9_loss: 1.8568 - dense_6_loss: 1.5645 - dense_8_loss: 1.5989 - dense_9_accuracy: 0.2461 - dense_6_accuracy: 0.3738 - dense_8_accuracy: 0.3489 - val_loss: 2.8009 - val_dense_9_loss: 1.7844 - val_dense_6_loss: 1.6509 - val_dense_8_loss: 1.7374 - val_dense_9_accuracy: 0.3084 - val_dense_6_accuracy: 0.3879 - val_dense_8_accuracy: 0.3645
```

In [23]:

```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()
```



In [24]:
```
model.evaluate(X_test, y_test)
```

```
7/7 [==============================] - 2s 89ms/step - loss: 2.8009 - dense_9_loss:
1.7844 - dense_6_loss: 1.6509 - dense_8_loss: 1.7374 - dense_9_accuracy: 0.3084 - de
nse_6_accuracy: 0.3879 - dense_8_accuracy: 0.3645
```

Out[24]:
```
[2.8008506298065186,
 1.7843737602233887,
 1.6508854627609253,
 1.7373706102371216,
 0.30841121077537537,
 0.38785046339035034,
 0.3644859790802002]
```