**Machine Learning Lab**

**Assignment 3**

***Name – Anmol Raj***

***Roll - 001811001069***

***Semester - 7***

***Year - 4***

***Department - Information Technology(IT)***

***Wine Dataset***

***Importing the Dataset :-***

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

df = pd.read\_csv("wine.data",header=**None**)

col\_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total phenols','Flavanoids',

'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315 of diluted wines','Proline']

df.columns = col\_name

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

y = df['Class']

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**from** **hmmlearn** **import** hmm

classifier = hmm.GaussianHMM(n\_components=3, covariance\_type="full")

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 0:

strings[i] = 1

**elif** y\_pred[i] == 1:

strings[i] = 2

**else**:

strings[i] = 3

strings = strings.astype(np.int)

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

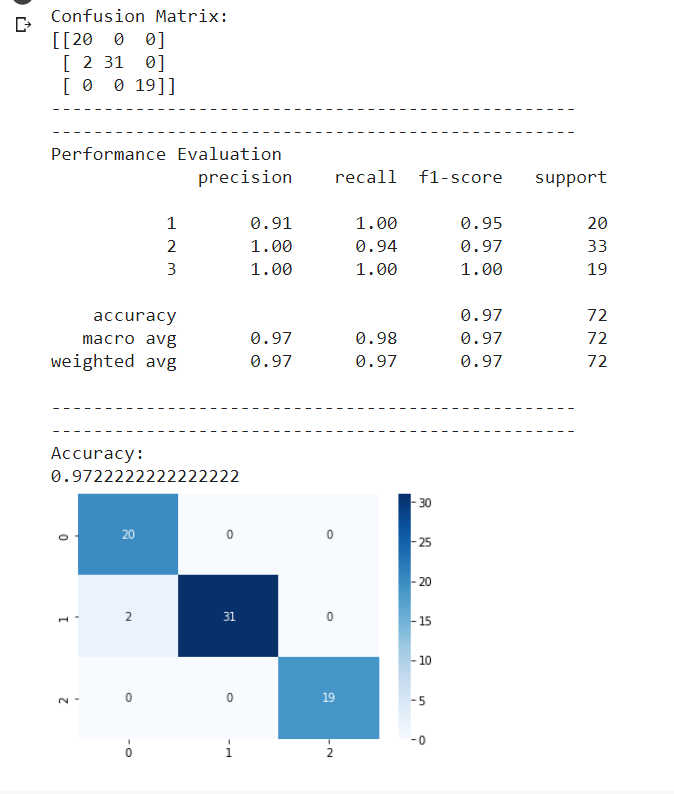
**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**from** **hmmlearn** **import** hmm

classifier = hmm.GaussianHMM(n\_components=3, covariance\_type="full", n\_iter=5,algorithm='viterbi',verbose=**False**

)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 0:

strings[i] = 1

**elif** y\_pred[i] == 1:

strings[i] = 2

**else**:

strings[i] = 3

strings = strings.astype(np.int)

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

Confusion Matrix:

[[ 0 15 0]

[ 2 2 23]

[12 0 0]]

--------------------------------------------------

--------------------------------------------------

Performance Evaluation

precision recall f1-score support

1 0.00 0.00 0.00 15

2 0.12 0.07 0.09 27

3 0.00 0.00 0.00 12

accuracy 0.04 54

macro avg 0.04 0.02 0.03 54

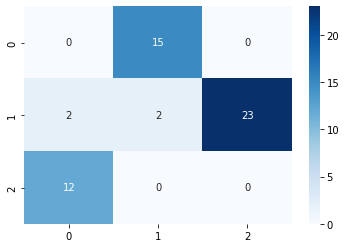
weighted avg 0.06 0.04 0.05 54

--------------------------------------------------

--------------------------------------------------

Accuracy:

0.037037037037037035

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.GMMHMM(n\_components=3, random\_state=10,covariance\_type='full',algorithm='viterbi',n\_iter=10)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 0:

strings[i] = 3

**elif** y\_pred[i] == 1:

strings[i] = 2

**else**:

strings[i] = 1

strings = strings.astype(np.int)

**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))

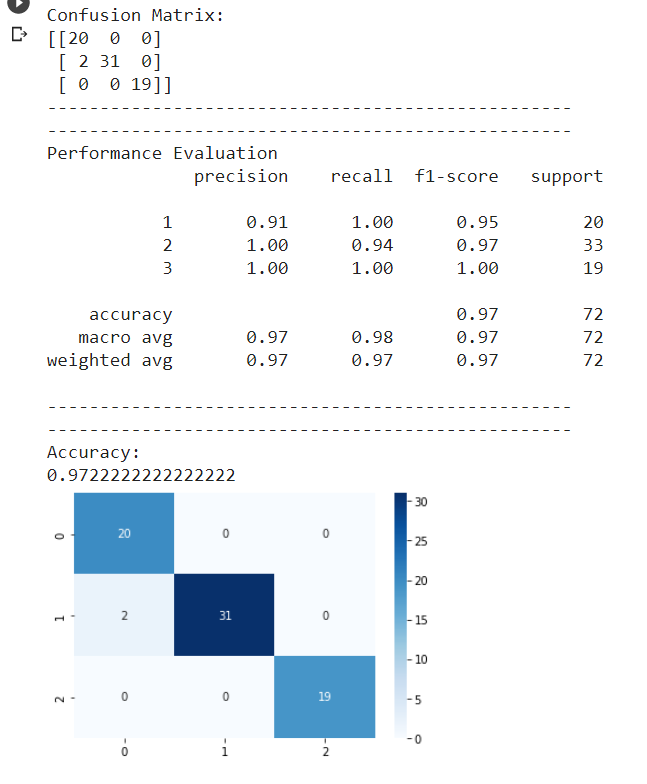
**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.3,test\_size=0.7,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**from** **hmmlearn** **import** hmm

classifier = hmm.GaussianHMM(n\_components=3, covariance\_type="full", n\_iter=5,algorithm='viterbi',verbose=**False**

)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 0:

strings[i] = 1

**elif** y\_pred[i] == 1:

strings[i] = 2

**else**:

strings[i] = 3

strings = strings.astype(np.int)

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

Confusion Matrix:

[[39 0 2]

[ 5 0 44]

[ 3 24 8]]

--------------------------------------------------

--------------------------------------------------

Performance Evaluation

precision recall f1-score support

1 0.83 0.95 0.89 41

2 0.00 0.00 0.00 49

3 0.15 0.23 0.18 35

accuracy 0.38 125

macro avg 0.33 0.39 0.36 125

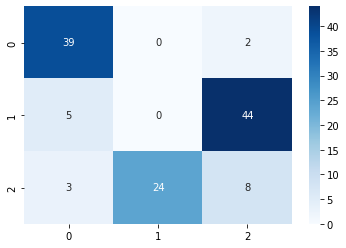
weighted avg 0.31 0.38 0.34 125

--------------------------------------------------

--------------------------------------------------

Accuracy:

0.376

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.MultinomialHMM(n\_components=4, random\_state=15,n\_iter=10,algorithm='viterbi',params='ste')

**import** **math**

row = len(X\_train)

col = len(X\_train[0])

new = [1] \* 33

**for** i **in** range(row):

**for** j **in** range(col):

X\_train[i][j] = X\_train[i][j]\*10

X\_train[i][j] = math.floor(X\_train[i][j])

x = X\_train[i].astype(np.int)

new = np.vstack([new,x])

y = new

y = np.absolute(y)

X\_train = y

**import** **math**

row = len(X\_test)

col = len(X\_test[0])

new

**for** i **in** range(row):

**for** j **in** range(col):

X\_test[i][j] = X\_test[i][j]\*10

X\_test[i][j] = math.floor(X\_test[i][j])

x = X\_test[i].astype(np.int)

new = np.vstack([new,x])

y = new

y = np.absolute(y)

X\_test = y

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("b")

**else**:

strings[i] = ("g")

strings

strings = strings[0:246]

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**2) Ionosphere Dataset**

***Importing the dataset***

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*# Dataset Preparation*

df = pd.read\_csv("ionosphere.data",header=**None**)

col\_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19'

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

df.columns = col\_name

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

y = df['Class']

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**from** **hmmlearn** **import** hmm

classifier = hmm.GaussianHMM(n\_components=2, covariance\_type="full")

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

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()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**from** **hmmlearn** **import** hmm

classifier = hmm.GaussianHMM(n\_components=2, covariance\_type="full",n\_iter=5,algorithm='viterbi',verbose=**False**)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("g")

**else**:

strings[i] = ("b")

strings

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.GMMHMM(n\_components=2, random\_state=10,covariance\_type='full',algorithm='viterbi',n\_iter=10)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("g")

**else**:

strings[i] = ("b")

strings

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.GMMHMM(n\_components=2, random\_state=10,covariance\_type='full',algorithm='viterbi',n\_iter=10)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("g")

**else**:

strings[i] = ("b")

strings

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

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()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.MultinomialHMM(n\_components=4, random\_state=15,n\_iter=10,algorithm='viterbi',params='ste')

**import** **math**

row = len(X\_train)

col = len(X\_train[0])

new = [1] \* 33

**for** i **in** range(row):

**for** j **in** range(col):

X\_train[i][j] = X\_train[i][j]\*10

X\_train[i][j] = math.floor(X\_train[i][j])

x = X\_train[i].astype(np.int)

new = np.vstack([new,x])

y = new

y = np.absolute(y)

X\_train = y

**import** **math**

row = len(X\_test)

col = len(X\_test[0])

new

**for** i **in** range(row):

**for** j **in** range(col):

X\_test[i][j] = X\_test[i][j]\*10

X\_test[i][j] = math.floor(X\_test[i][j])

x = X\_test[i].astype(np.int)

new = np.vstack([new,x])

y = new

y = np.absolute(y)

X\_test = y

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("g")

**else**:

strings[i] = ("b")

strings

strings = strings[0:106]

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

Confusion Matrix:

[[34 6]

[52 14]]

--------------------------------------------------

--------------------------------------------------

Performance Evaluation

precision recall f1-score support

b 0.40 0.85 0.54 40

g 0.70 0.21 0.33 66

accuracy 0.45 106

macro avg 0.55 0.53 0.43 106

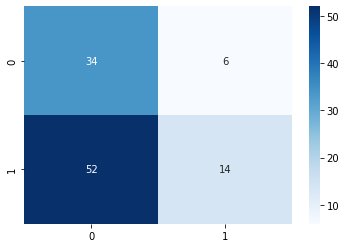
weighted avg 0.59 0.45 0.41 106

--------------------------------------------------

--------------------------------------------------

Accuracy:

0.4528301886792453

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.3,test\_size=0.7,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.MultinomialHMM(n\_components=4, random\_state=15,n\_iter=10,algorithm='viterbi',params='ste')

**import** **math**

row = len(X\_train)

col = len(X\_train[0])

new = [1] \* 33

**for** i **in** range(row):

**for** j **in** range(col):

X\_train[i][j] = X\_train[i][j]\*10

X\_train[i][j] = math.floor(X\_train[i][j])

x = X\_train[i].astype(np.int)

new = np.vstack([new,x])

y = new

y = np.absolute(y)

X\_train = y

**import** **math**

row = len(X\_test)

col = len(X\_test[0])

new

**for** i **in** range(row):

**for** j **in** range(col):

X\_test[i][j] = X\_test[i][j]\*10

X\_test[i][j] = math.floor(X\_test[i][j])

x = X\_test[i].astype(np.int)

new = np.vstack([new,x])

y = new

y = np.absolute(y)

X\_test = y

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("g")

**else**:

strings[i] = ("b")

strings

strings = strings[0:106]

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

Confusion Matrix:

[[ 17 67]

[ 31 131]]

--------------------------------------------------

--------------------------------------------------

Performance Evaluation

precision recall f1-score support

b 0.35 0.20 0.26 84

g 0.66 0.81 0.73 162

accuracy 0.60 246

macro avg 0.51 0.51 0.49 246

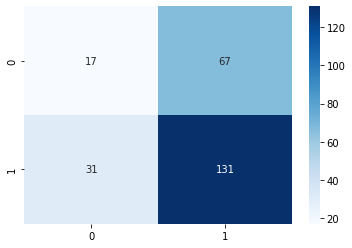
weighted avg 0.56 0.60 0.57 246

--------------------------------------------------

--------------------------------------------------

Accuracy:

0.6016260162601627

****

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

**3) Breast Cancer Dataset**

***Importing the Dataset***

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*# Dataset Preparation*

df = pd.read\_csv("wdbc.data",header=**None**)

col\_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19'

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

df.columns = col\_name

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

y = df['Class']

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**from** **hmmlearn** **import** hmm

classifier = hmm.GaussianHMM(n\_components=2, covariance\_type="full")

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("M")

**else**:

strings[i] = ("B")

strings

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

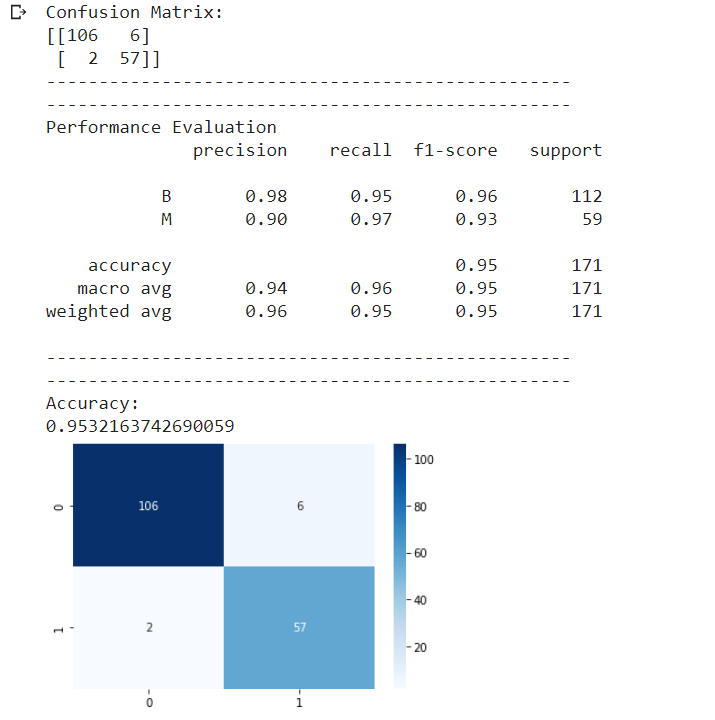
**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**from** **hmmlearn** **import** hmm

classifier = hmm.GaussianHMM(n\_components=2, covariance\_type="full",n\_iter=10,algorithm='viterbi',verbose=**False**)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("M")

**else**:

strings[i] = ("B")

strings

**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))

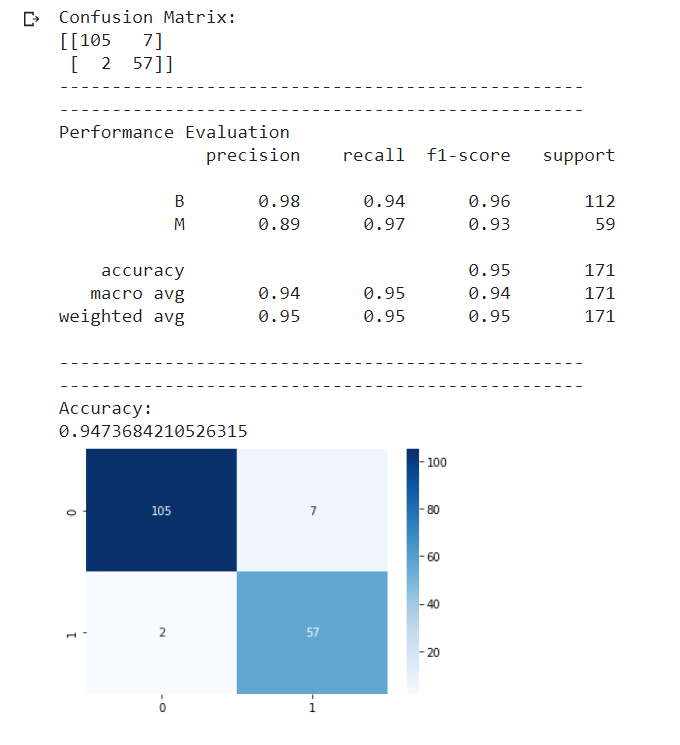
**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.GMMHMM(n\_components=2, random\_state=10)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("M")

**else**:

strings[i] = ("B")

strings

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

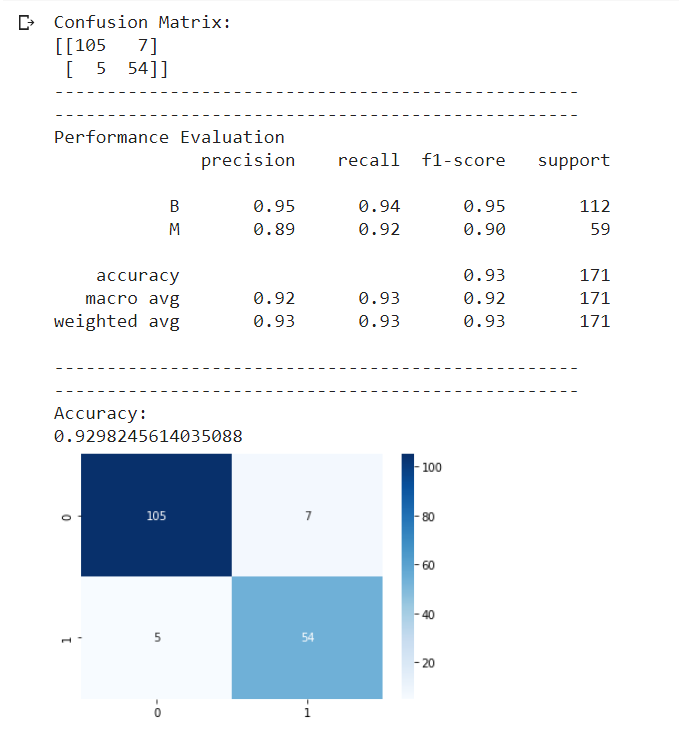
**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.GMMHMM(n\_components=2, random\_state=10,covariance\_type='diag',algorithm='viterbi',n\_iter=10)

classifier.fit(X\_train)

y\_pred = classifier.predict(X\_test)

size = len(y\_pred)

strings = np.empty(size, np.unicode\_)

**for** i **in** range (size):

**if** y\_pred[i] == 1:

strings[i] = ("M")

**else**:

strings[i] = ("B")

strings

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, accuracy\_score

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, strings))

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

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

print("Performance Evaluation")

print(classification\_report(y\_test, strings))

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

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

print("Accuracy:")

print(accuracy\_score(y\_test, strings))

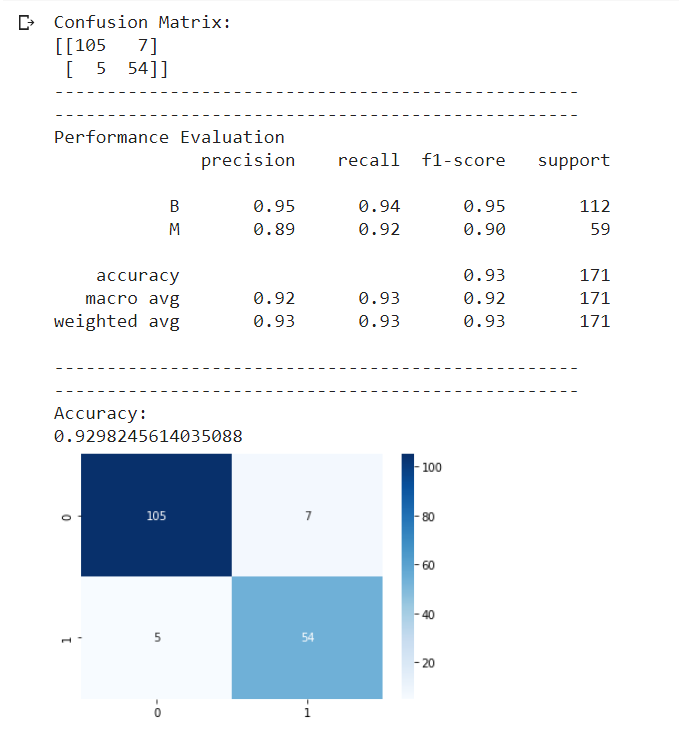
**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

cm = confusion\_matrix(y\_test, strings)

sns.heatmap(cm, annot=**True**, fmt="d",cmap='Blues')

plt.show()

****

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.7,test\_size=0.3,random\_state=10)

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**import** **hmmlearn**

classifier = hmmlearn.hmm.MultinomialHMM(n\_components=4, random\_state=15,n\_iter=10,algorithm='viterbi',params='ste')

**import** **math**

row = len(X\_train)

col = len(X\_train[0])

new

**for** i **in** range(row):

**for** j **in** range(col):

X\_train[i][j] = X\_train[i][j]\*10

X\_train[i][j] = math.floor(X\_train[i][j])

x = X\_train[i].astype(np.int)

new = np.vstack([new,x])

y = new

y = np.absolute(y)

X\_train = y

**import** **math**

row = len(X\_test)

col = len(X\_test[0])

new

**for** i **in** range(row):

**for** j **in** range(col):

X\_test[i][j] = X\_test[i][j]\*10

X\_test[i][j] = math.floor(X\_test[i][j])

x = X\_test[i].astype(np.int)

new = np.vstack([new,x])

y = new

y = np.absolute(y)

X\_test = y

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

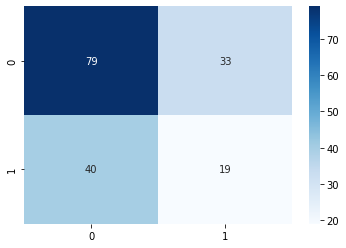
weighted avg 0.56 0.57 0.57 171

--------------------------------------------------

--------------------------------------------------

Accuracy:

0.5730994152046783

****

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

1. **CIFAR-10**

***Importing dataset* :-**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **tensorflow** **as** **tf**

*# dataset preparation*

**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

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))

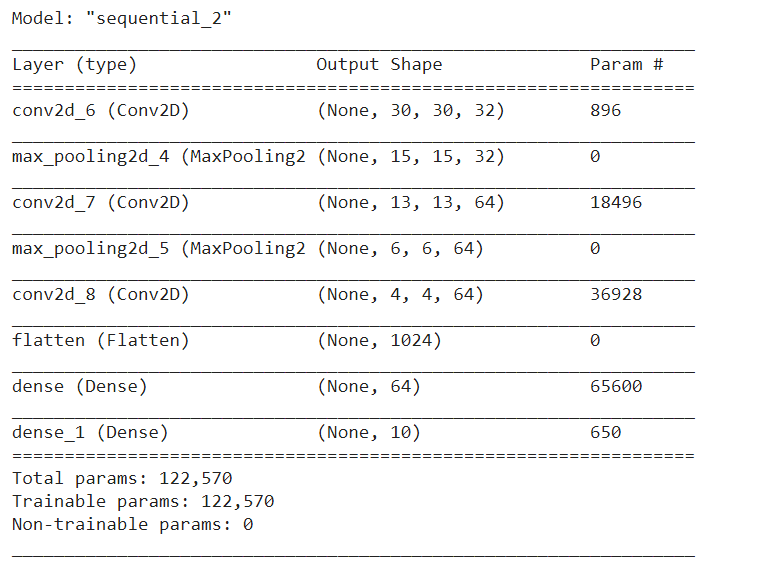
model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.Flatten())

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

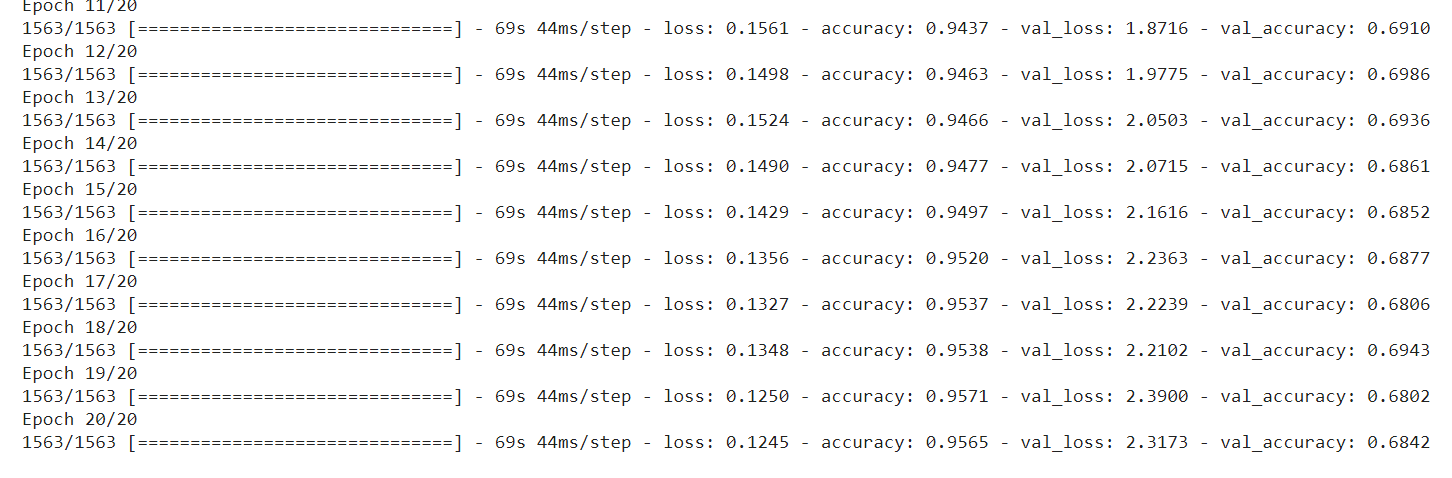
model.add(layers.Dense(10))

model.summary()

****

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=**True**),metrics=['accuracy'])

history = model.fit(train\_images,train\_labels,epochs=20,validation\_data=(test\_images,test\_labels))

****

plt.plot(history.history['accuracy'],label='accuracy')

plt.plot(history.history['val\_accuracy'],label='val\_accuracy')

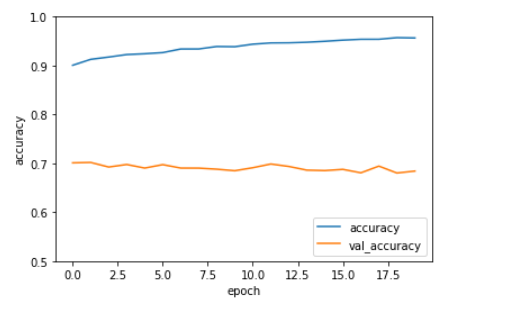
plt.xlabel('epoch')

plt.ylabel('accuracy')

plt.ylim([0.5,1])

plt.legend(loc='lower right')

plt.show()

****

test\_loss , test\_acc = model.evaluate(test\_images,test\_labels,verbose=2)

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

**2) MNIST**

***Importing the Dataset :-***

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **tensorflow** **as** **tf**

*# dataset preparation*

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

(train\_images,train\_labels) , (test\_images,test\_labels) = datasets.mnist.load\_data()

*# Normalize pixel values to be within 0 , 1*

train\_images , test\_images = train\_images/255.0 , test\_images/255.0

train\_images = np.reshape(train\_images, train\_images.shape + (1,))

test\_images = np.reshape(test\_images, test\_images.shape + (1,))

train\_images[0].shape

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))

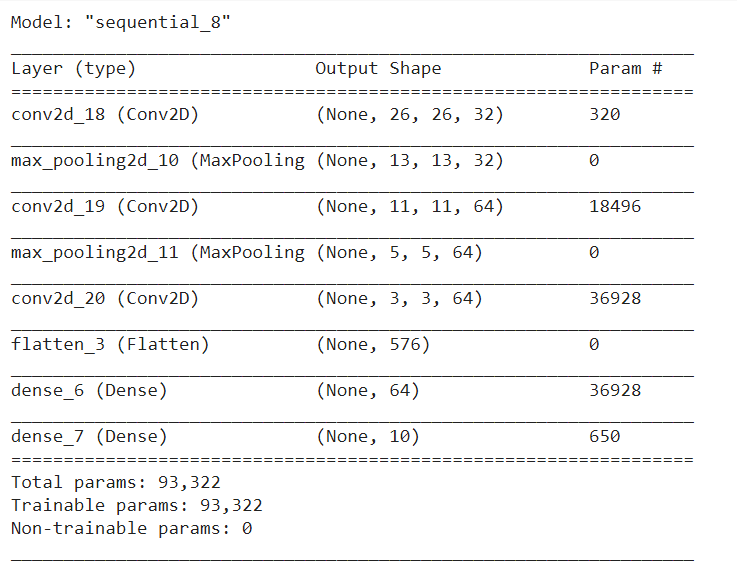
model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.Flatten())

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

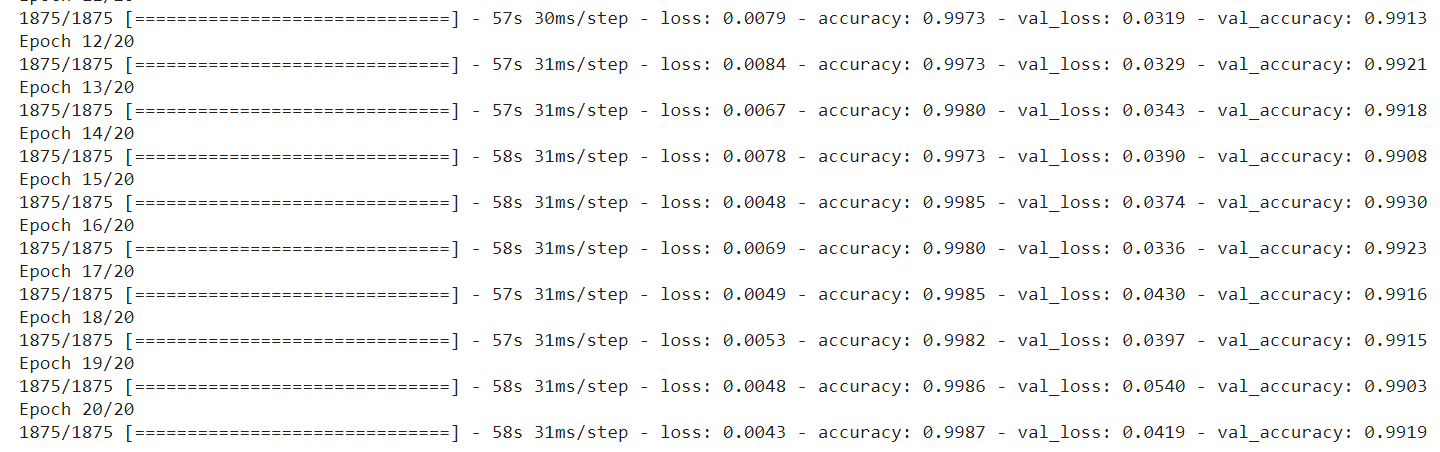
model.add(layers.Dense(10))

model.summary()

****

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=**True**),metrics=['accuracy'])

history = model.fit(train\_images,train\_labels,epochs=20,validation\_data=(test\_images,test\_labels))

****

plt.plot(history.history['accuracy'],label='accuracy')

plt.plot(history.history['val\_accuracy'],label='val\_accuracy')

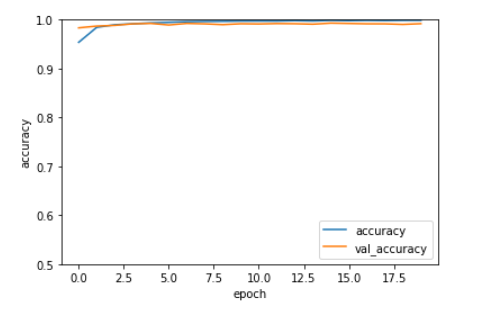
plt.xlabel('epoch')

plt.ylabel('accuracy')

plt.ylim([0.5,1])

plt.legend(loc='lower right')

plt.show()

****

test\_loss , test\_acc = model.evaluate(test\_images,test\_labels,verbose=2)

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

**3) SAVEE**

***Importing the Dataset :-***

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

**import** **librosa**

**import** **numpy** **as** **np**

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

**def** preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, *#log\_specgram*

step\_size=10, eps=1e-10):

mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

**return** mel\_db.T

**def** load\_audio\_file(file\_path, input\_length=input\_length):

data = librosa.core.load(file\_path, sr=16000)[0] *#, sr=16000*

**if** len(data)>input\_length:

max\_offset = len(data)-input\_length

offset = np.random.randint(max\_offset)

data = data[offset:(input\_length+offset)]

**else**:

**if** input\_length > len(data):

max\_offset = input\_length - len(data)

offset = np.random.randint(max\_offset)

**else**:

offset = 0

data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

data = preprocess\_audio\_mel\_T(data)

**return** data

**import** **os**

**from** **scipy.io** **import** wavfile

**import** **librosa**

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

rootDirectory = "/content/AudioData/"

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

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

X = list()

y = list()

**for** person **in** personNames:

directory = os.path.join(rootDirectory,person)

**for** filename **in** os.listdir(directory):

filePath = os.path.join(directory, filename)

data = load\_audio\_file(file\_path=filePath)

data = np.reshape(data, data.shape + (1,))

**if**(filename[0:1] **in** classes):

X.append(data)

y.append(classes.index(filename[0:1]))

**elif**(filename[0:2] **in** classes):

X.append(data)

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

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

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

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **tensorflow** **as** **tf**

*# dataset preparation*

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, train\_size= 0.7 ,random\_state=10)

model = models.Sequential()

model.add(layers.Conv2D(32,(3,3),activation='relu',input\_shape=(157,320,1)))

model.add(layers.MaxPool2D(2,2))

model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.MaxPool2D(2,2))

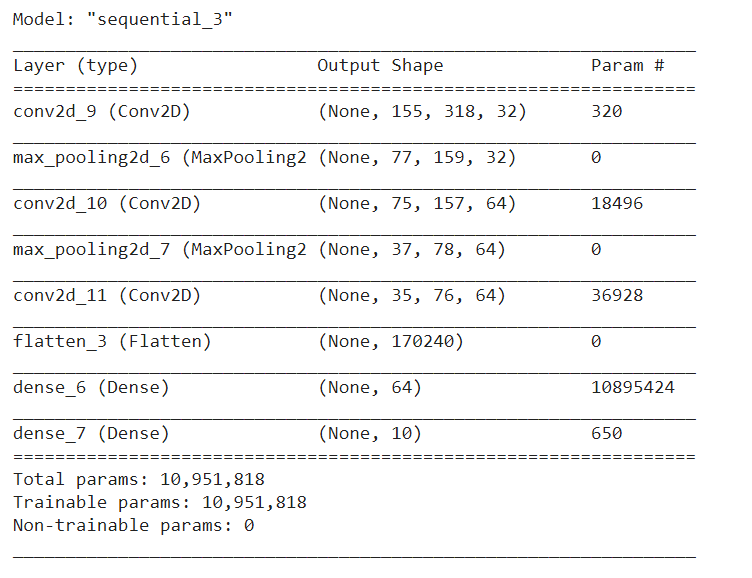
model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.Flatten())

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

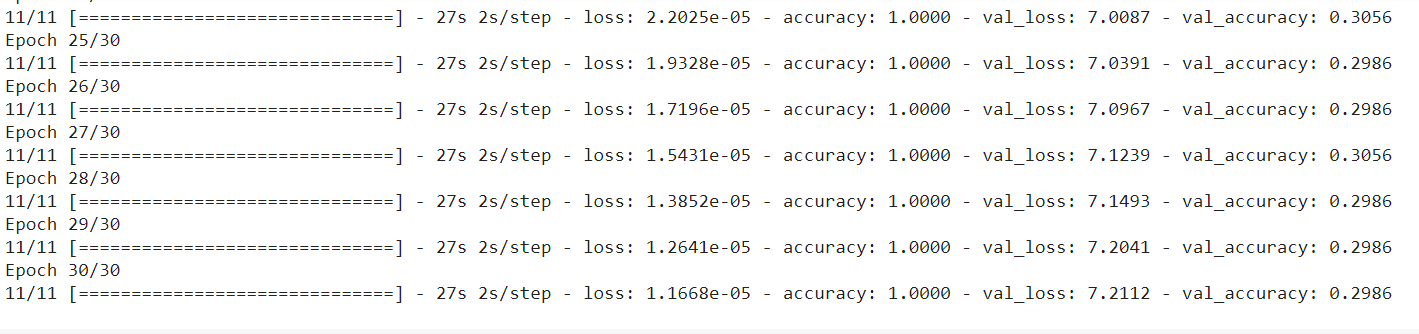
model.add(layers.Dense(10))

model.summary()

****

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=**True**),metrics=['accuracy'])

history = model.fit(X\_train,y\_train,epochs=30,validation\_data=(X\_test,y\_test))

****

plt.plot(history.history['accuracy'],label='accuracy')

plt.plot(history.history['val\_accuracy'],label='val\_accuracy')

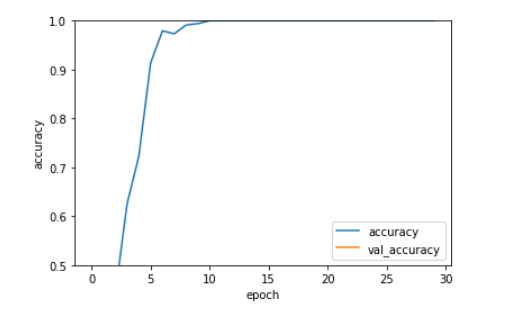
plt.xlabel('epoch')

plt.ylabel('accuracy')

plt.ylim([0.5,1])

plt.legend(loc='lower right')

plt.show()

****

test\_loss , test\_acc = model.evaluate(X\_test,y\_test,verbose=2)

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

**4) EmoDB**

***Importing the Dataset :-***

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

**import** **librosa**

**import** **numpy** **as** **np**

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

**def** preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, *#log\_specgram*

step\_size=10, eps=1e-10):

mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

**return** mel\_db.T

**def** load\_audio\_file(file\_path, input\_length=input\_length):

data = librosa.core.load(file\_path, sr=16000)[0] *#, sr=16000*

**if** len(data)>input\_length:

max\_offset = len(data)-input\_length

offset = np.random.randint(max\_offset)

data = data[offset:(input\_length+offset)]

**else**:

**if** input\_length > len(data):

max\_offset = input\_length - len(data)

offset = np.random.randint(max\_offset)

**else**:

offset = 0

data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

data = preprocess\_audio\_mel\_T(data)

**return** data

*# Preprocessing the dataset*

**import** **os**

**from** **scipy.io** **import** wavfile

**import** **librosa**

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

directory = "/content/wav/"

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

X = list()

y = list()

**for** filename **in** os.listdir(directory):

filePath = os.path.join(directory, filename)

data = load\_audio\_file(file\_path=filePath)

data = np.reshape(data, data.shape + (1,))

**if**(filename[5:6] **in** classes):

X.append(data)

y.append(classes.index(filename[5:6]))

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

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

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **tensorflow** **as** **tf**

*# dataset preparation*

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, train\_size= 0.7 ,random\_state=10)

model = models.Sequential()

model.add(layers.Conv2D(32,(3,3),activation='relu',input\_shape=(157,320,1)))

model.add(layers.MaxPool2D(2,2))

model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.MaxPool2D(2,2))

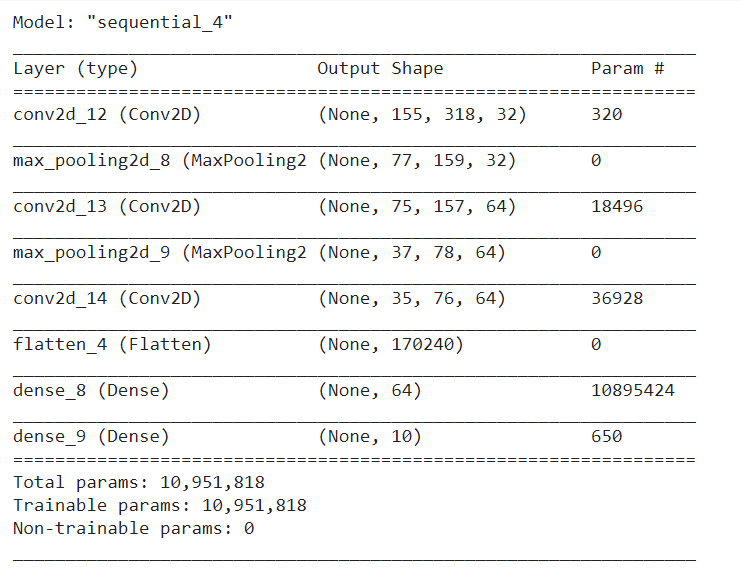
model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.Flatten())

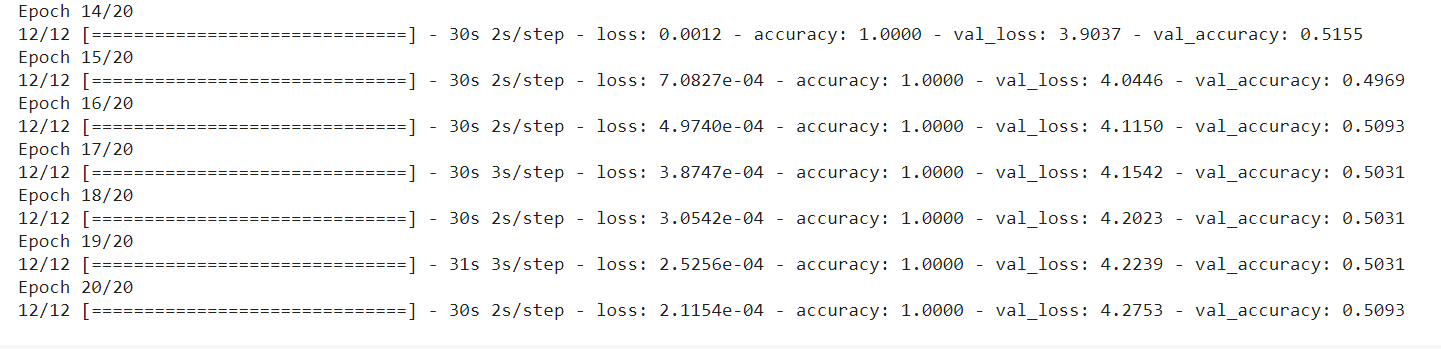
model.add(layers.Dense(64,activation='relu'))

model.add(layers.Dense(10))

model.summary()

****

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=**True**),metrics=['accuracy']) history = model.fit(X\_train,y\_train,epochs=20,validation\_data=(X\_test,y\_test))

****

plt.plot(history.history['accuracy'],label='accuracy')

plt.plot(history.history['val\_accuracy'],label='val\_accuracy')

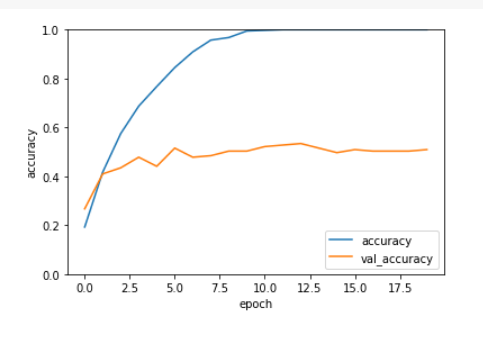
plt.xlabel('epoch')

plt.ylabel('accuracy')

plt.ylim([0,1])

plt.legend(loc='lower right')

plt.show()

****

test\_loss , test\_acc = model.evaluate(X\_test,y\_test,verbose=2)

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

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

1. **VGG-16**

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

**import** **tensorflow** **as** **tf**

**from** **tensorflow** **import** keras

**import** **matplotlib.pyplot** **as** **plt**

%matplotlib inline

**import** **numpy** **as** **np**

**import** **skimage.transform**

**from** **\_\_future\_\_** **import** print\_function

!pip install keras\_applications

**import** **numpy** **as** **np**

**import** **warnings**

**from** **keras.models** **import** Model

**from** **keras.layers** **import** Flatten

**from** **keras.layers** **import** Dense

**from** **keras.layers** **import** Input

**from** **keras.layers** **import** Conv2D

**from** **keras.layers** **import** MaxPooling2D

**from** **keras.layers** **import** GlobalMaxPooling2D

**from** **keras.layers** **import** GlobalAveragePooling2D

**from** **keras.preprocessing** **import** image

**from** **keras.utils** **import** layer\_utils

**from** **keras.utils.data\_utils** **import** get\_file

**from** **keras** **import** backend **as** K

**from** **keras.applications.imagenet\_utils** **import** decode\_predictions

**from** **keras.applications.imagenet\_utils** **import** preprocess\_input

**from** **keras\_applications.imagenet\_utils** **import** \_obtain\_input\_shape

**from** **keras.utils.layer\_utils** **import** get\_source\_inputs

**def** load\_preprocess\_training\_batch(X\_train):

new = []

**for** item **in** X\_train:

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

new.append(tmpFeature)

**return** new

**def** preprocess\_data(X\_train):

**for** item **in** X\_train:

item = np.expand\_dims(item, axis=0)

item = preprocess\_input(item)

**return** X\_train

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

**import** **tensorflow** **as** **tf**

**from** **tensorflow** **import** keras

**import** **matplotlib.pyplot** **as** **plt**

%matplotlib inline

**import** **numpy** **as** **np**

**import** **skimage.transform**

* 1. **CIFAR-10**

(X\_train, y\_train) , (X\_test, y\_test) = keras.datasets.cifar10.load\_data()

X\_train = X\_train[0:2000]

y\_train = y\_train[0:2000]

X\_test = X\_test[0:2000]

y\_test = y\_test[0:2000]

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = X\_train\_resized / 255

X\_test\_resized = X\_test\_resized / 255

X\_train\_resized = preprocess\_data(X\_train\_resized)

X\_test\_resized = preprocess\_data(X\_test\_resized)

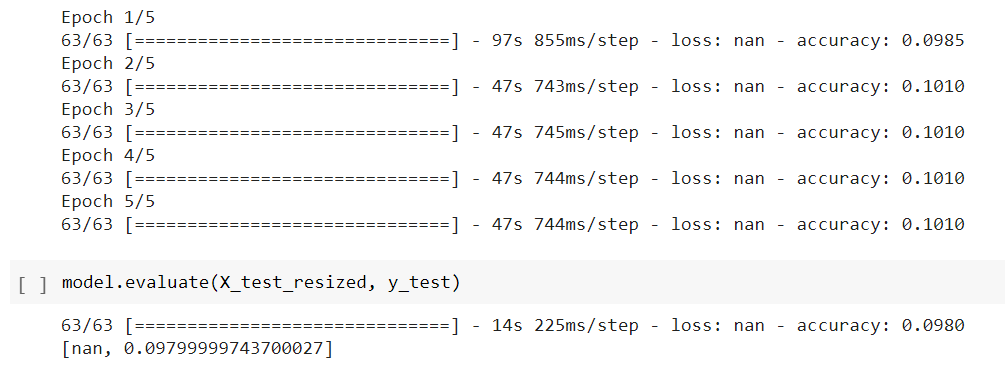
model = VGG16(include\_top=**True**, weights='imagenet')

model.compile(optimizer='SGD',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=5)

****

model.evaluate(X\_test\_resized, y\_test)

[nan, 0.09799999743700027]

* 1. **MNIST**

(X\_train, y\_train) , (X\_test, y\_test) = keras.datasets.mnist.load\_data()

X\_train = X\_train[0:2000]

y\_train = y\_train[0:2000]

X\_test = X\_test[0:2000]

y\_test = y\_test[0:2000]

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = X\_train\_resized / 255.0

X\_test\_resized = X\_test\_resized / 255.0

X\_train\_resized = preprocess\_data(X\_train\_resized)

X\_test\_resized = preprocess\_data(X\_test\_resized)

**import** **cv2**

X\_train\_new = list()

**for** i **in** range(len(X\_train\_resized)):

g = X\_train\_resized[i]

X\_train\_new.append(cv2.merge([g,g,g]))

X\_train\_new = np.asarray(X\_train\_new,dtype=np.float32)

X\_test\_new = list()

**for** i **in** range(len(X\_test\_resized)):

g = X\_test\_resized[i]

X\_test\_new.append(cv2.merge([g,g,g]))

X\_test\_new = np.asarray(X\_test\_new,dtype=np.float32)

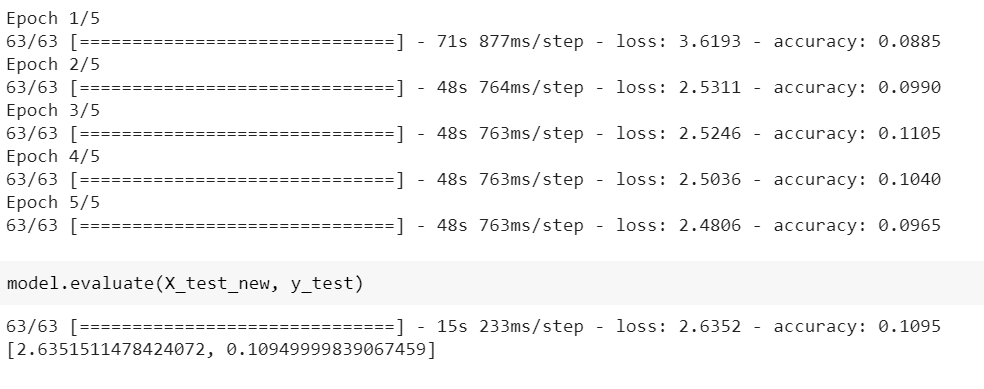
model = VGG16(include\_top=**True**, weights='imagenet')

model.compile(optimizer='SGD',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

history = model.fit(X\_train\_new, y\_train, epochs=5)

****

model.evaluate(X\_test\_new, y\_test)

[2.6351511478424072, 0.10949999839067459]

* 1. **SAVEE**

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

**import** **librosa**

**import** **numpy** **as** **np**

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

**def** preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, *#log\_specgram*

step\_size=10, eps=1e-10):

mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

**return** mel\_db.T

**def** load\_audio\_file(file\_path, input\_length=input\_length):

data = librosa.core.load(file\_path, sr=16000)[0] *#, sr=16000*

**if** len(data)>input\_length:

max\_offset = len(data)-input\_length

offset = np.random.randint(max\_offset)

data = data[offset:(input\_length+offset)]

**else**:

**if** input\_length > len(data):

max\_offset = input\_length - len(data)

offset = np.random.randint(max\_offset)

**else**:

offset = 0

data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

data = preprocess\_audio\_mel\_T(data)

**return** data

**import** **os**

**from** **scipy.io** **import** wavfile

**import** **librosa**

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

**import** **cv2**

rootDirectory = "/content/AudioData/"

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]))

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

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

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **tensorflow** **as** **tf**

*# dataset preparation*

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, train\_size= 0.5 ,random\_state=10)

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = preprocess\_data(X\_train\_resized)

X\_test\_resized = preprocess\_data(X\_test\_resized)

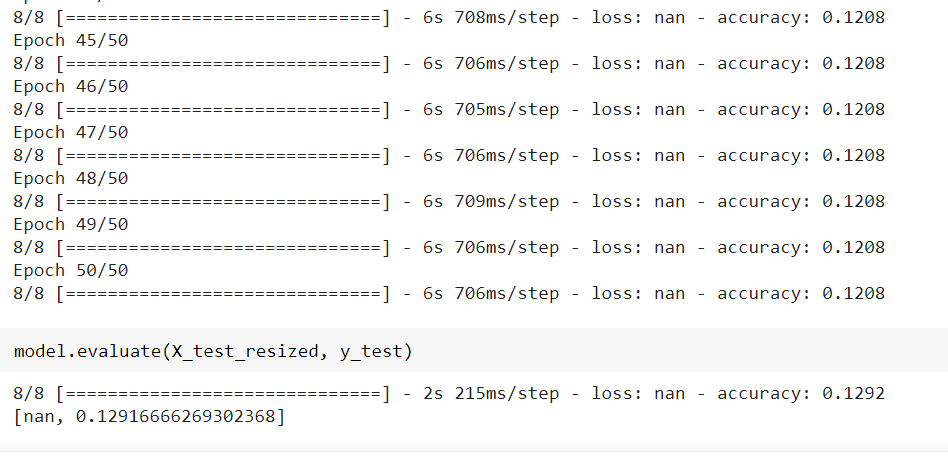
model = VGG16(include\_top=**True**, weights='imagenet')

model.compile(optimizer='SGD',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=50)

****

model.evaluate(X\_test\_resized, y\_test)

[nan, 0.12916666269302368]

* 1. **EmoDB**

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

**import** **librosa**

**import** **numpy** **as** **np**

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

**def** preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, *#log\_specgram*

step\_size=10, eps=1e-10):

mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

**return** mel\_db.T

**def** load\_audio\_file(file\_path, input\_length=input\_length):

data = librosa.core.load(file\_path, sr=16000)[0] *#, sr=16000*

**if** len(data)>input\_length:

max\_offset = len(data)-input\_length

offset = np.random.randint(max\_offset)

data = data[offset:(input\_length+offset)]

**else**:

**if** input\_length > len(data):

max\_offset = input\_length - len(data)

offset = np.random.randint(max\_offset)

**else**:

offset = 0

data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

data = preprocess\_audio\_mel\_T(data)

return data

**import** **os**

**from** **scipy.io** **import** wavfile

**import** **librosa**

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

**import** **cv2**

directory = "/content/wav/"

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

X = list()

y = list()

**for** filename **in** os.listdir(directory):

filePath = os.path.join(directory, filename)

a = load\_audio\_file(file\_path=filePath)

data = cv2.merge([a,a,a])

**if**(filename[5:6] **in** classes):

X.append(data)

y.append(classes.index(filename[5:6]))

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

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

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **tensorflow** **as** **tf**

*# dataset preparation*

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

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, train\_size= 0.5 ,random\_state=10)

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = preprocess\_data(X\_train\_resized)

X\_test\_resized = preprocess\_data(X\_test\_resized)

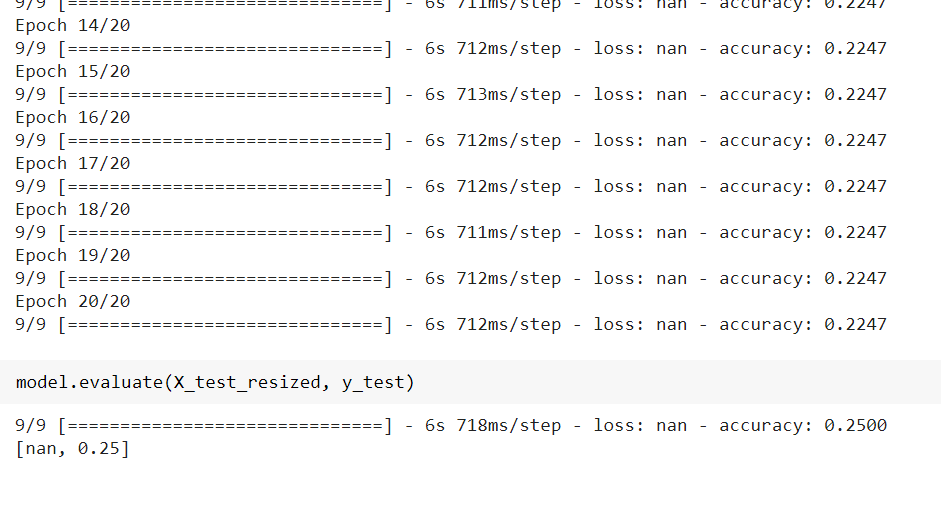
model = VGG16(include\_top=**True**, weights='imagenet')

model.compile(optimizer='SGD',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=20)

****

model.evaluate(X\_test\_resized, y\_test)

[nan, 0.25]

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

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

1. **ResNet-50**
2. from google.colab import drive
3. drive.mount('/content/drive')
4. from \_\_future\_\_ import print\_function
5. import numpy as np
6. import warnings
7. !pip install keras\_applications
8. from keras.layers import Input
9. from keras import layers
10. from keras.layers import Dense
11. from keras.layers import Activation
12. from keras.layers import Flatten
13. from keras.layers import Conv2D
14. from keras.layers import MaxPooling2D
15. from keras.layers import GlobalMaxPooling2D
16. from keras.layers import ZeroPadding2D
17. from keras.layers import AveragePooling2D
18. from keras.layers import GlobalAveragePooling2D
19. from keras.layers import BatchNormalization
20. from keras.models import Model
21. from keras.preprocessing import image
22. import keras.backend as K
23. from keras.utils import layer\_utils
24. from keras.utils.data\_utils import get\_file
25. from keras.applications.imagenet\_utils import decode\_predictions
26. from keras.applications.imagenet\_utils import preprocess\_input
27. from keras\_applications.imagenet\_utils import \_obtain\_input\_shape
28. from keras.utils.layer\_utils import get\_source\_inputs
29. import tensorflow as tf
30. from tensorflow import keras
31. import matplotlib.pyplot as plt
32. %matplotlib inline
33. import numpy as np
34. import skimage.transform

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

def load\_preprocess\_training\_batch(X\_train):

    new = []

    for item in X\_train:

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

        new.append(tmpFeature)

    return new

def preprocess\_data(X\_train):

    for item in X\_train:

      item = np.expand\_dims(item, axis=0)

      item = preprocess\_input(item)

    return X\_train

**2.1) CIFAR-10**

(X\_train, y\_train) , (X\_test, y\_test) = keras.datasets.cifar10.load\_data()

X\_train = X\_train[0:2000]

y\_train = y\_train[0:2000]

X\_test = X\_test[0:2000]

y\_test = y\_test[0:2000]

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = X\_train\_resized / 255

X\_test\_resized = X\_test\_resized / 255

X\_train\_resized = preprocess\_data(X\_train\_resized)

X\_test\_resized = preprocess\_data(X\_test\_resized)

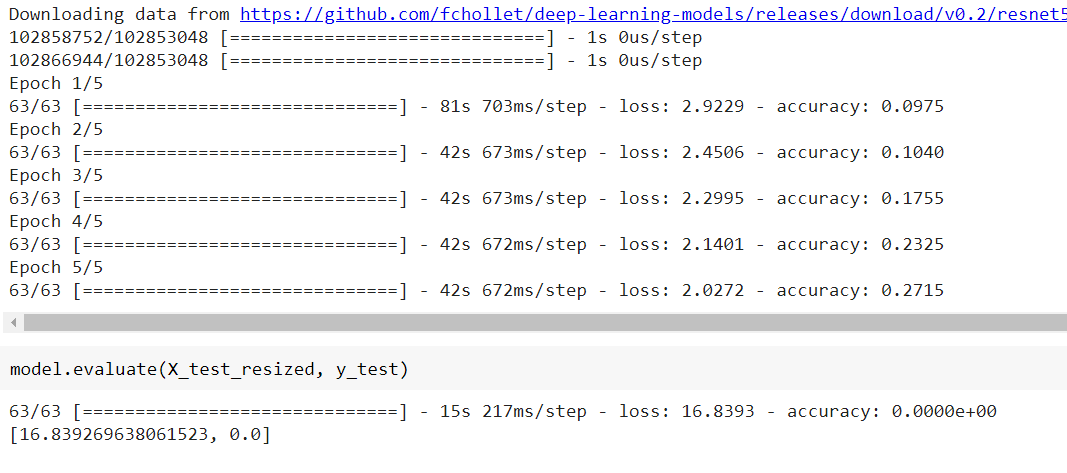
model = ResNet50(include\_top=True, weights='imagenet')

model.compile(optimizer='SGD',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=5)

****

**2.2) MNIST**

(X\_train, y\_train) , (X\_test, y\_test) = keras.datasets.mnist.load\_data()

X\_train = X\_train[0:2000]

y\_train = y\_train[0:2000]

X\_test = X\_test[0:2000]

y\_test = y\_test[0:2000]

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = X\_train\_resized / 255.0

X\_test\_resized = X\_test\_resized / 255.0

X\_train\_resized = preprocess\_data(X\_train\_resized)

X\_test\_resized = preprocess\_data(X\_test\_resized)

import cv2

X\_train\_new = list()

for i in range(len(X\_train\_resized)):

  g  = X\_train\_resized[i]

  X\_train\_new.append(cv2.merge([g,g,g]))

X\_train\_new = np.asarray(X\_train\_new,dtype=np.float32)

X\_test\_new = list()

for i in range(len(X\_test\_resized)):

  g  = X\_test\_resized[i]

  X\_test\_new.append(cv2.merge([g,g,g]))

X\_test\_new = np.asarray(X\_test\_new,dtype=np.float32)

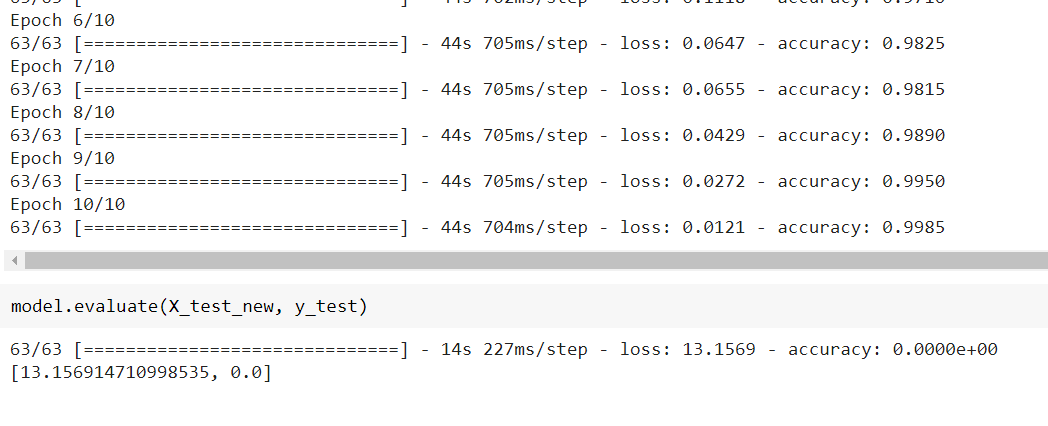
model = ResNet50(include\_top=True, weights='imagenet')

model.compile(optimizer='SGD',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train\_new, y\_train, epochs=10)

****

**2.3) SAVEE**

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

import librosa

import numpy as np

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

def preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, #log\_specgram

                 step\_size=10, eps=1e-10):

    mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

    mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

    return mel\_db.T

def load\_audio\_file(file\_path, input\_length=input\_length):

  data = librosa.core.load(file\_path, sr=16000)[0] #, sr=16000

  if len(data)>input\_length:

    max\_offset = len(data)-input\_length

    offset = np.random.randint(max\_offset)

    data = data[offset:(input\_length+offset)]

  else:

    if input\_length > len(data):

      max\_offset = input\_length - len(data)

      offset = np.random.randint(max\_offset)

    else:

      offset = 0

    data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

  data = preprocess\_audio\_mel\_T(data)

  return data

# Preprocessing the dataset

import os

from scipy.io import wavfile

import librosa

import matplotlib.pyplot as plt

import numpy as np

import cv2

rootDirectory = "/content/AudioData/"

personNames = ["DC","JE","JK","KL"]

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

X = list()

y = list()

for person in personNames:

  directory = os.path.join(rootDirectory,person)

  for filename in os.listdir(directory):

    filePath = os.path.join(directory, filename)

    a = load\_audio\_file(file\_path=filePath)

    data = cv2.merge([a,a,a])

    # data = np.reshape(data, data.shape + (1,))

    if(filename[0:1] in classes):

      X.append(data)

      y.append(classes.index(filename[0:1]))

    elif(filename[0:2] in classes):

      X.append(data)

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

X = np.asarray(X, dtype=np.float32)

y = np.asarray(y, dtype=np.float32)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, train\_size= 0.5 ,random\_state=10)

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = preprocess\_data(X\_train\_resized)

X\_test\_resized = preprocess\_data(X\_test\_resized)

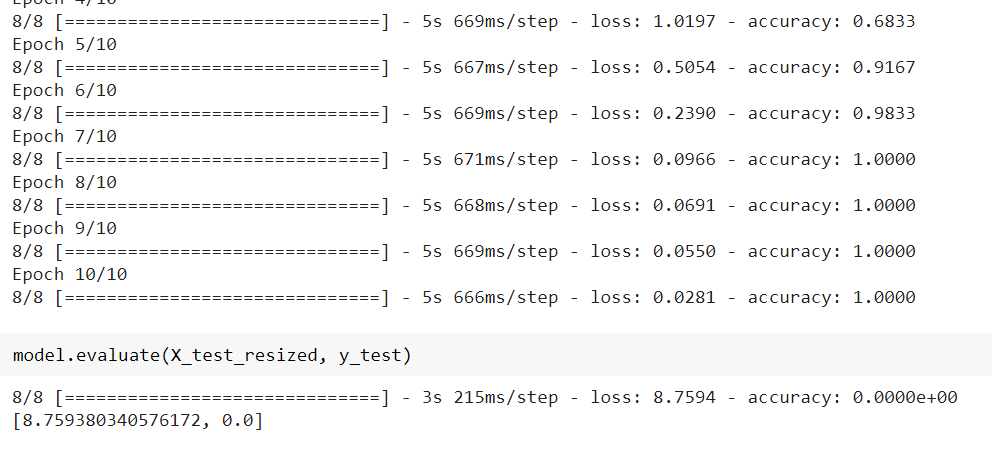
model = ResNet50(include\_top=True, weights='imagenet')

model.compile(optimizer='SGD',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=10)

****

**2.4) EmoDB**

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

import librosa

import numpy as np

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

def preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, #log\_specgram

                 step\_size=10, eps=1e-10):

    mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

    mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

    return mel\_db.T

def load\_audio\_file(file\_path, input\_length=input\_length):

  data = librosa.core.load(file\_path, sr=16000)[0] #, sr=16000

  if len(data)>input\_length:

    max\_offset = len(data)-input\_length

    offset = np.random.randint(max\_offset)

    data = data[offset:(input\_length+offset)]

  else:

    if input\_length > len(data):

      max\_offset = input\_length - len(data)

      offset = np.random.randint(max\_offset)

    else:

      offset = 0

    data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

  data = preprocess\_audio\_mel\_T(data)

  return data

# Preprocessing the dataset

import os

from scipy.io import wavfile

import librosa

import matplotlib.pyplot as plt

import numpy as np

import cv2

directory = "/content/wav/"

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

X = list()

y = list()

for filename in os.listdir(directory):

  filePath = os.path.join(directory, filename)

  a = load\_audio\_file(file\_path=filePath)

  data = cv2.merge([a,a,a])

  if(filename[5:6] in classes):

    X.append(data)

    y.append(classes.index(filename[5:6]))

X = np.asarray(X, dtype=np.float32)

y = np.asarray(y, dtype=np.float32)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, train\_size= 0.5 ,random\_state=10)

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = preprocess\_data(X\_train\_resized)

X\_test\_resized = preprocess\_data(X\_test\_resized)

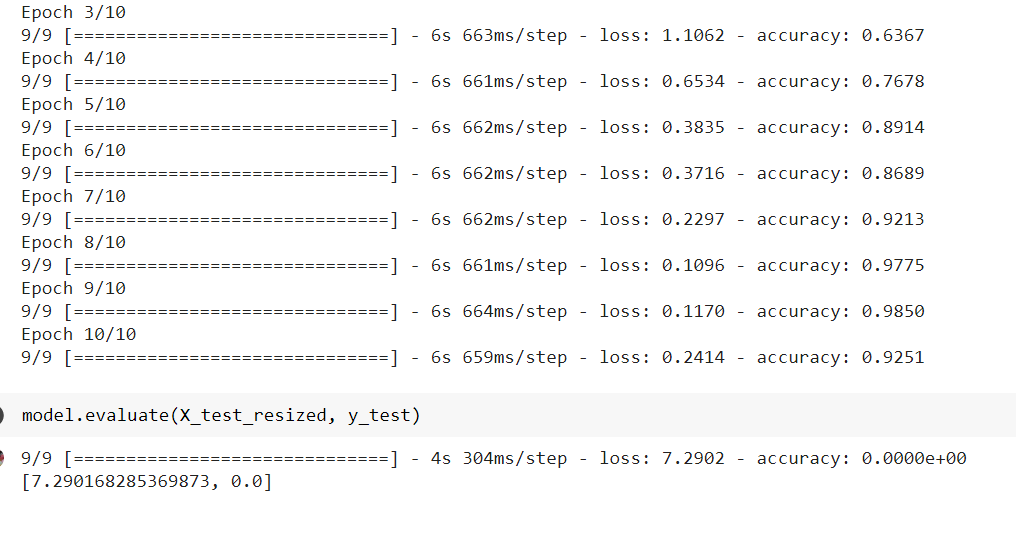
model = ResNet50(include\_top=True, weights='imagenet')

model.compile(optimizer='SGD',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=10)

****

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

1. **Recurrent Neural Networks (RNN)**

from google.colab import drive

drive.mount('/content/drive')

**3.1) CIFAR-10**

import os

import tensorflow as tf

import keras

from tensorflow.keras import layers

from tensorflow.keras import Model

from os import getcwd

cifar10 = tf.keras.datasets.cifar10

(training\_images, training\_labels), (test\_images, test\_labels) = cifar10.load\_data()

training\_images = training\_images.reshape(50000, 1024, 3)

training\_images = training\_images[0:10000]

training\_labels = training\_labels[0:10000]

training\_images = training\_images/255.0

test\_images = test\_images.reshape(10000, 1024, 3)

test\_images = test\_images[0:5000]

test\_labels = test\_labels[0:5000]

test\_images = test\_images/255.0

model = tf.keras.models.Sequential([

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input\_shape=(1024,3), return\_sequences=True)),

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(64, activation='relu'),

    tf.keras.layers.Dense(10, activation='softmax')

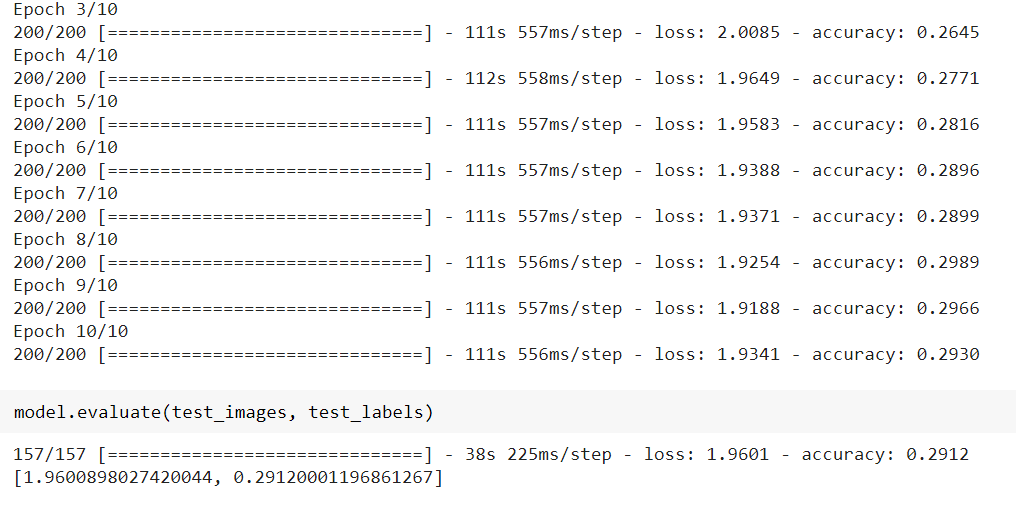
    ])

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(training\_images, training\_labels, batch\_size = 50, epochs=10)

****

**3.2) MNIST**

import torch

# Device configuration

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

device

from torchvision import datasets

from torchvision.transforms import ToTensor

train\_data = datasets.MNIST(

    root = 'data',

    train = True,

    transform = ToTensor(),

    download = True,

)

test\_data = datasets.MNIST(

    root = 'data',

    train = False,

    transform = ToTensor()

)

import matplotlib.pyplot as plt

plt.imshow(train\_data.data[0], cmap='gray')

plt.title('%i' % train\_data.targets[0])

plt.show()

figure = plt.figure(figsize=(10, 8))

cols, rows = 5, 5

for i in range(1, cols \* rows + 1):

    sample\_idx = torch.randint(len(train\_data), size=(1,)).item()

    img, label = train\_data[sample\_idx]

    figure.add\_subplot(rows, cols, i)

    plt.title(label)

    plt.axis("off")

    plt.imshow(img.squeeze(), cmap="gray")

plt.show()

from torch.utils.data import DataLoader

loaders = {

    'train' : torch.utils.data.DataLoader(train\_data,

                                          batch\_size=100,

                                          shuffle=True,

                                          num\_workers=1),

    'test'  : torch.utils.data.DataLoader(test\_data,

                                          batch\_size=100,

                                          shuffle=True,

                                          num\_workers=1),

}

loaders

from torch import nn

import torch.nn.functional as F

sequence\_length = 28

input\_size = 28

hidden\_size = 128

num\_layers = 2

num\_classes = 10

batch\_size = 100

num\_epochs = 2

learning\_rate = 0.01

class RNN(nn.Module):

    pass

model = RNN().to(device)

print(model)

class RNN(nn.Module):

    def \_\_init\_\_(self, input\_size, hidden\_size, num\_layers, num\_classes):

        super(RNN, self).\_\_init\_\_()

        self.hidden\_size = hidden\_size

        self.num\_layers = num\_layers

        self.lstm = nn.LSTM(input\_size, hidden\_size, num\_layers, batch\_first=True)

        self.fc = nn.Linear(hidden\_size, num\_classes)

        pass

    def forward(self, x):

        # Set initial hidden and cell states

        h0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(device)

        c0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(device)

        # Passing in the input and hidden state into the model and  obtaining outputs

        out, hidden = self.lstm(x, (h0, c0))  # out: tensor of shape (batch\_size, seq\_length, hidden\_size)

        #Reshaping the outputs such that it can be fit into the fully connected layer

        out = self.fc(out[:, -1, :])

        return out

        pass

pass

model = RNN(input\_size, hidden\_size, num\_layers, num\_classes).to(device)

print(model)

loss\_func = nn.CrossEntropyLoss()

loss\_func

from torch import optim

optimizer = optim.Adam(model.parameters(), lr = 0.01)

optimizer

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)

# Test the model

model.eval()

with torch.no\_grad():

    correct = 0

    total = 0

    for images, labels in loaders['test']:

        images = images.reshape(-1, sequence\_length, input\_size).to(device)

        labels = labels.to(device)

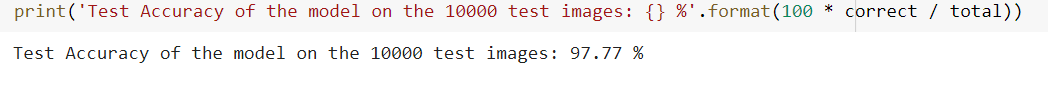
        outputs = model(images)

        \_, predicted = torch.max(outputs.data, 1)

        total = total + labels.size(0)

        correct = correct + (predicted == labels).sum().item()

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

****

**3.3) SAVEE**

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

import librosa

import numpy as np

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

def preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, #log\_specgram

                 step\_size=10, eps=1e-10):

    mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

    mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

    return mel\_db.T

def load\_audio\_file(file\_path, input\_length=input\_length):

  data = librosa.core.load(file\_path, sr=16000)[0] #, sr=16000

  if len(data)>input\_length:

    max\_offset = len(data)-input\_length

    offset = np.random.randint(max\_offset)

    data = data[offset:(input\_length+offset)]

  else:

    if input\_length > len(data):

      max\_offset = input\_length - len(data)

      offset = np.random.randint(max\_offset)

    else:

      offset = 0

    data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

  data = preprocess\_audio\_mel\_T(data)

  return data

# Preprocessing the dataset

import os

from scipy.io import wavfile

import librosa

import matplotlib.pyplot as plt

import numpy as np

import cv2

rootDirectory = "/content/AudioData/"

personNames = ["DC","JE","JK","KL"]

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

X = list()

y = list()

for person in personNames:

  directory = os.path.join(rootDirectory,person)

  for filename in os.listdir(directory):

    filePath = os.path.join(directory, filename)

    data = load\_audio\_file(file\_path=filePath)

    # data = cv2.merge([a,a,a])

    if(filename[0:1] in classes):

      X.append(data)

      y.append(classes.index(filename[0:1]))

    elif(filename[0:2] in classes):

      X.append(data)

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

X = np.asarray(X, dtype=np.float32)

y = np.asarray(y, dtype=np.float32)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, train\_size= 0.6 ,random\_state=10)

import os

import tensorflow as tf

import keras

from tensorflow.keras import layers

from tensorflow.keras import Model

from os import getcwd

model = tf.keras.models.Sequential([

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input\_shape=(157,320), return\_sequences=True)),

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(64, activation='relu'),

    tf.keras.layers.Dense(10, activation='softmax')

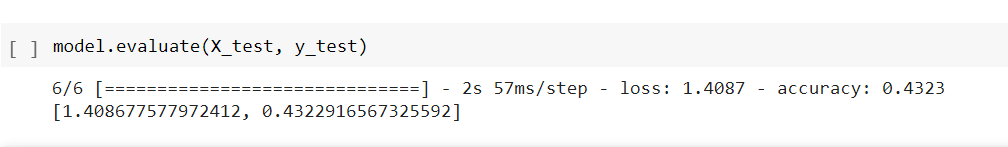
    ])

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train,y\_train, batch\_size = 50, epochs=50)

****

**3.4) EmoDB**

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

import librosa

import numpy as np

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

def preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, #log\_specgram

                 step\_size=10, eps=1e-10):

    mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

    mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

    return mel\_db.T

def load\_audio\_file(file\_path, input\_length=input\_length):

  data = librosa.core.load(file\_path, sr=16000)[0] #, sr=16000

  if len(data)>input\_length:

    max\_offset = len(data)-input\_length

    offset = np.random.randint(max\_offset)

    data = data[offset:(input\_length+offset)]

  else:

    if input\_length > len(data):

      max\_offset = input\_length - len(data)

      offset = np.random.randint(max\_offset)

    else:

      offset = 0

    data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

  data = preprocess\_audio\_mel\_T(data)

  return data

# Preprocessing the dataset

import os

from scipy.io import wavfile

import librosa

import matplotlib.pyplot as plt

import numpy as np

import cv2

directory = "/content/wav/"

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

X = list()

y = list()

for filename in os.listdir(directory):

  filePath = os.path.join(directory, filename)

  data = load\_audio\_file(file\_path=filePath)

  if(filename[5:6] in classes):

    X.append(data)

    y.append(classes.index(filename[5:6]))

X = np.asarray(X, dtype=np.float32)

y = np.asarray(y, dtype=np.float32)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, train\_size= 0.7 ,random\_state=10)

model = tf.keras.models.Sequential([

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input\_shape=(157,320), return\_sequences=True)),

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(64, activation='relu'),

    tf.keras.layers.Dense(10, activation='softmax')

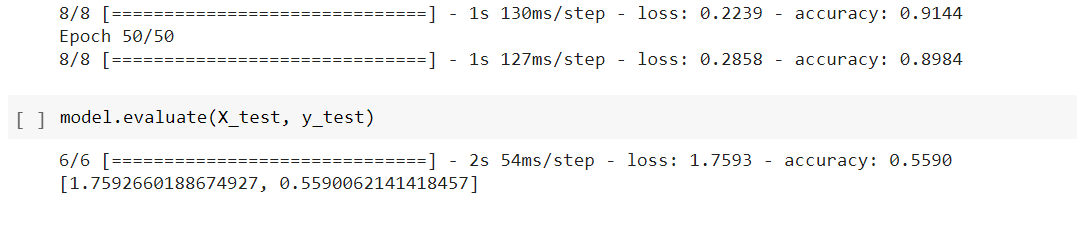
    ])

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train,y\_train, batch\_size = 50, epochs=50)

****

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

1. **AlexNet**

from google.colab import drive

drive.mount('/content/drive')

# Import necessary packages

import argparse

# Import necessary components to build LeNet

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation, Flatten

from keras.layers.convolutional import Conv2D, MaxPooling2D, ZeroPadding2D

from keras.layers import BatchNormalization

from keras.regularizers import l2

import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt

%matplotlib inline

import numpy as np

import skimage.transform

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()

def load\_preprocess\_training\_batch(X\_train):

    new = []

    for item in X\_train:

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

        new.append(tmpFeature)

    return new

**4.1) CIFAR-10**

model = alexnet\_model()

(X\_train, y\_train) , (X\_test, y\_test) = keras.datasets.cifar10.load\_data()

X\_train = X\_train[0:500]

y\_train = y\_train[0:500]

X\_test = X\_test[0:200]

y\_test = y\_test[0:200]

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = X\_train\_resized / 255

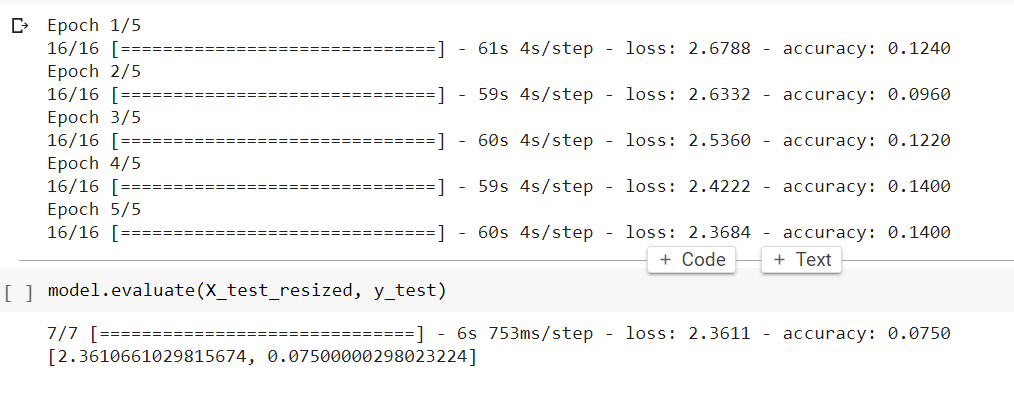
X\_test\_resized = X\_test\_resized / 255

model.compile(optimizer='SGD',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=5)

****

**4.2) MNIST**

(X\_train, y\_train) , (X\_test, y\_test) = keras.datasets.mnist.load\_data()

X\_train = X\_train[0:2000]

y\_train = y\_train[0:2000]

X\_test = X\_test[0:2000]

y\_test = y\_test[0:2000]

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

X\_train\_resized = X\_train\_resized / 255.0

X\_test\_resized = X\_test\_resized / 255.0

import cv2

X\_train\_new = list()

for i in range(len(X\_train\_resized)):

  g  = X\_train\_resized[i]

  X\_train\_new.append(cv2.merge([g,g,g]))

X\_train\_new = np.asarray(X\_train\_new,dtype=np.float32)

X\_test\_new = list()

for i in range(len(X\_test\_resized)):

  g  = X\_test\_resized[i]

  X\_test\_new.append(cv2.merge([g,g,g]))

X\_test\_new = np.asarray(X\_test\_new,dtype=np.float32)

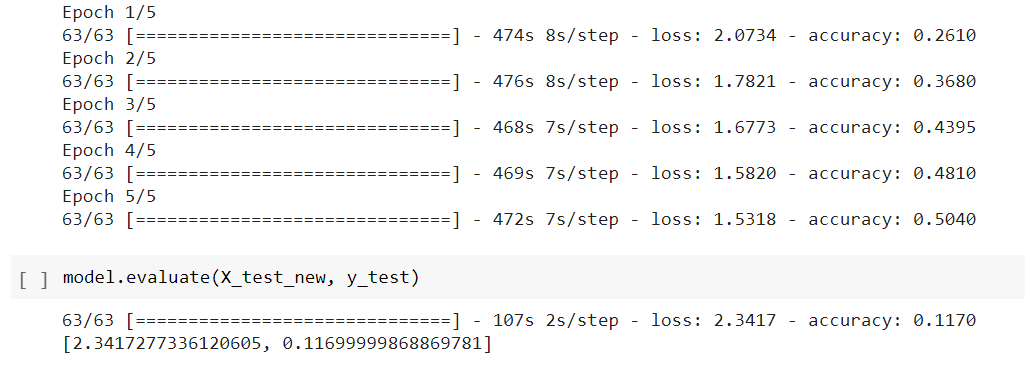
model = alexnet\_model()

model.compile(optimizer='SGD',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train\_new, y\_train, epochs=5)

****

**4.3) SAVEE**

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

import librosa

import numpy as np

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

def preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, #log\_specgram

                 step\_size=10, eps=1e-10):

    mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

    mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

    return mel\_db.T

def load\_audio\_file(file\_path, input\_length=input\_length):

  data = librosa.core.load(file\_path, sr=16000)[0] #, sr=16000

  if len(data)>input\_length:

    max\_offset = len(data)-input\_length

    offset = np.random.randint(max\_offset)

    data = data[offset:(input\_length+offset)]

  else:

    if input\_length > len(data):

      max\_offset = input\_length - len(data)

      offset = np.random.randint(max\_offset)

    else:

      offset = 0

    data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

  data = preprocess\_audio\_mel\_T(data)

  return data

# Preprocessing the dataset

import os

from scipy.io import wavfile

import librosa

import matplotlib.pyplot as plt

import numpy as np

import cv2

rootDirectory = "/content/AudioData/"

personNames = ["DC","JE","JK","KL"]

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

X = list()

y = list()

for person in personNames:

  directory = os.path.join(rootDirectory,person)

  for filename in os.listdir(directory):

    filePath = os.path.join(directory, filename)

    a = load\_audio\_file(file\_path=filePath)

    data = cv2.merge([a,a,a])

    if(filename[0:1] in classes):

      X.append(data)

      y.append(classes.index(filename[0:1]))

    elif(filename[0:2] in classes):

      X.append(data)

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

X = np.asarray(X, dtype=np.float32)

y = np.asarray(y, dtype=np.float32)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, train\_size= 0.5 ,random\_state=10)

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

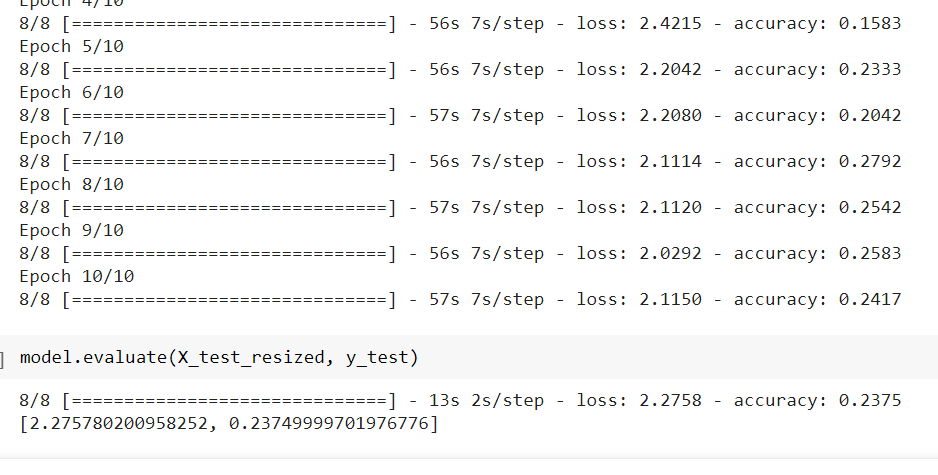
model = alexnet\_model()

model.compile(optimizer='SGD',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=10)

****

**4.4) EmoDB**

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

import librosa

import numpy as np

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

def preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, #log\_specgram

                 step\_size=10, eps=1e-10):

    mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

    mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

    return mel\_db.T

def load\_audio\_file(file\_path, input\_length=input\_length):

  data = librosa.core.load(file\_path, sr=16000)[0] #, sr=16000

  if len(data)>input\_length:

    max\_offset = len(data)-input\_length

    offset = np.random.randint(max\_offset)

    data = data[offset:(input\_length+offset)]

  else:

    if input\_length > len(data):

      max\_offset = input\_length - len(data)

      offset = np.random.randint(max\_offset)

    else:

      offset = 0

    data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

  data = preprocess\_audio\_mel\_T(data)

  return data

# Preprocessing the dataset

import os

from scipy.io import wavfile

import librosa

import matplotlib.pyplot as plt

import numpy as np

import cv2

directory = "/content/wav/"

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

X = list()

y = list()

for filename in os.listdir(directory):

  filePath = os.path.join(directory, filename)

  a = load\_audio\_file(file\_path=filePath)

  data = cv2.merge([a,a,a])

  if(filename[5:6] in classes):

    X.append(data)

    y.append(classes.index(filename[5:6]))

X = np.asarray(X, dtype=np.float32)

y = np.asarray(y, dtype=np.float32)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, train\_size= 0.6 ,random\_state=10)

X\_train\_resized = load\_preprocess\_training\_batch(X\_train)

X\_test\_resized = load\_preprocess\_training\_batch(X\_test)

X\_train\_resized = np.array(X\_train\_resized)

X\_test\_resized = np.array(X\_test\_resized)

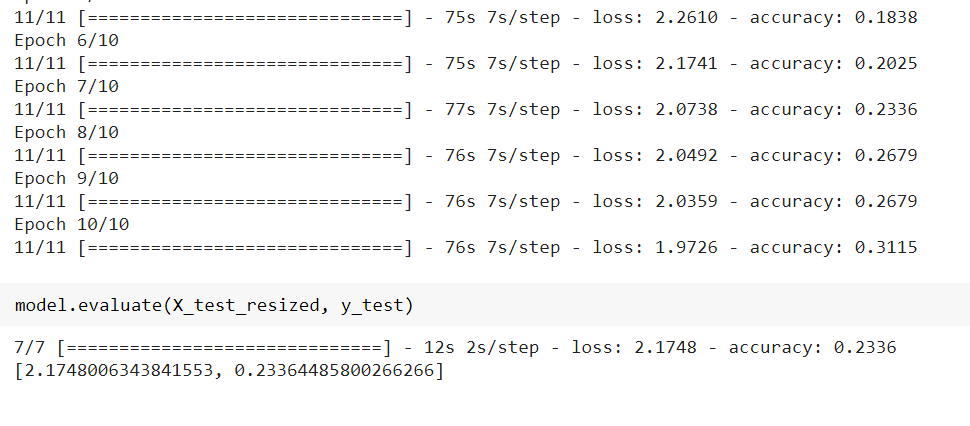
model = alexnet\_model()

model.compile(optimizer='SGD',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train\_resized, y\_train, epochs=10)

****

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

1. **GoogLeNet**

from google.colab import drive

drive.mount('/content/drive')

**5.1) CIFAR-10**

def inception\_module(x,

                     filters\_1x1,

                     filters\_3x3\_reduce,

                     filters\_3x3,

                     filters\_5x5\_reduce,

                     filters\_5x5,

                     filters\_pool\_proj,

                     name=None):

    conv\_1x1 = Conv2D(filters\_1x1, (1, 1), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(x)

    conv\_3x3 = Conv2D(filters\_3x3\_reduce, (1, 1), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(x)

    conv\_3x3 = Conv2D(filters\_3x3, (3, 3), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(conv\_3x3)

    conv\_5x5 = Conv2D(filters\_5x5\_reduce, (1, 1), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(x)

    conv\_5x5 = Conv2D(filters\_5x5, (5, 5), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(conv\_5x5)

    pool\_proj = MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)

    pool\_proj = Conv2D(filters\_pool\_proj, (1, 1), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(pool\_proj)

    output = concatenate([conv\_1x1, conv\_3x3, conv\_5x5, pool\_proj], axis=3, name=name)

    return output

kernel\_init = keras.initializers.glorot\_uniform()

bias\_init = keras.initializers.Constant(value=0.2)

input\_layer = Input(shape=(224, 224, 3))

x = Conv2D(64, (7, 7), padding='same', strides=(2, 2), activation='relu', name='conv\_1\_7x7/2', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(input\_layer)

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max\_pool\_1\_3x3/2')(x)

x = Conv2D(64, (1, 1), padding='same', strides=(1, 1), activation='relu', name='conv\_2a\_3x3/1')(x)

x = Conv2D(192, (3, 3), padding='same', strides=(1, 1), activation='relu', name='conv\_2b\_3x3/1')(x)

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max\_pool\_2\_3x3/2')(x)

x = inception\_module(x,

                     filters\_1x1=64,

                     filters\_3x3\_reduce=96,

                     filters\_3x3=128,

                     filters\_5x5\_reduce=16,

                     filters\_5x5=32,

                     filters\_pool\_proj=32,

                     name='inception\_3a')

x = inception\_module(x,

                     filters\_1x1=128,

                     filters\_3x3\_reduce=128,

                     filters\_3x3=192,

                     filters\_5x5\_reduce=32,

                     filters\_5x5=96,

                     filters\_pool\_proj=64,

                     name='inception\_3b')

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max\_pool\_3\_3x3/2')(x)

x = inception\_module(x,

                     filters\_1x1=192,

                     filters\_3x3\_reduce=96,

                     filters\_3x3=208,

                     filters\_5x5\_reduce=16,

                     filters\_5x5=48,

                     filters\_pool\_proj=64,

                     name='inception\_4a')

x1 = AveragePooling2D((5, 5), strides=3)(x)

x1 = Conv2D(128, (1, 1), padding='same', activation='relu')(x1)

x1 = Flatten()(x1)

x1 = Dense(1024, activation='relu')(x1)

x1 = Dropout(0.7)(x1)

x1 = Dense(10, activation='softmax', name='auxilliary\_output\_1')(x1)

x = inception\_module(x,

                     filters\_1x1=160,

                     filters\_3x3\_reduce=112,

                     filters\_3x3=224,

                     filters\_5x5\_reduce=24,

                     filters\_5x5=64,

                     filters\_pool\_proj=64,

                     name='inception\_4b')

x = inception\_module(x,

                     filters\_1x1=128,

                     filters\_3x3\_reduce=128,

                     filters\_3x3=256,

                     filters\_5x5\_reduce=24,

                     filters\_5x5=64,

                     filters\_pool\_proj=64,

                     name='inception\_4c')

x = inception\_module(x,

                     filters\_1x1=112,

                     filters\_3x3\_reduce=144,

                     filters\_3x3=288,

                     filters\_5x5\_reduce=32,

                     filters\_5x5=64,

                     filters\_pool\_proj=64,

                     name='inception\_4d')

x2 = AveragePooling2D((5, 5), strides=3)(x)

x2 = Conv2D(128, (1, 1), padding='same', activation='relu')(x2)

x2 = Flatten()(x2)

x2 = Dense(1024, activation='relu')(x2)

x2 = Dropout(0.7)(x2)

x2 = Dense(10, activation='softmax', name='auxilliary\_output\_2')(x2)

x = inception\_module(x,

                     filters\_1x1=256,

                     filters\_3x3\_reduce=160,

                     filters\_3x3=320,

                     filters\_5x5\_reduce=32,

                     filters\_5x5=128,

                     filters\_pool\_proj=128,

                     name='inception\_4e')

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max\_pool\_4\_3x3/2')(x)

x = inception\_module(x,

                     filters\_1x1=256,

                     filters\_3x3\_reduce=160,

                     filters\_3x3=320,

                     filters\_5x5\_reduce=32,

                     filters\_5x5=128,

                     filters\_pool\_proj=128,

                     name='inception\_5a')

x = inception\_module(x,

                     filters\_1x1=384,

                     filters\_3x3\_reduce=192,

                     filters\_3x3=384,

                     filters\_5x5\_reduce=48,

                     filters\_5x5=128,

                     filters\_pool\_proj=128,

                     name='inception\_5b')

x = GlobalAveragePooling2D(name='avg\_pool\_5\_3x3/1')(x)

x = Dropout(0.4)(x)

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

import keras

from keras.layers.core import Layer

import keras.backend as K

import tensorflow as tf

from keras.datasets import cifar10

from keras.models import Model

from keras.layers import Conv2D, MaxPool2D,  \

    Dropout, Dense, Input, concatenate,      \

    GlobalAveragePooling2D, AveragePooling2D,\

    Flatten

import cv2

import numpy as np

from keras.datasets import cifar10

from keras import backend as K

from keras.utils import np\_utils

import math

from tensorflow.keras.optimizers import SGD

from keras.callbacks import LearningRateScheduler

num\_classes = 10

def load\_cifar10\_data(img\_rows, img\_cols):

    # Load cifar10 training and validation sets

    (X\_train, Y\_train), (X\_valid, Y\_valid) = cifar10.load\_data()

    X\_train = X\_train[0:5000]

    Y\_train = Y\_train[0:5000]

    X\_valid = X\_valid[0:2000]

    Y\_valid = Y\_valid[0:2000]

    # Resize training images

    X\_train = np.array([cv2.resize(img, (img\_rows,img\_cols)) for img in X\_train[:,:,:,:]])

    X\_valid = np.array([cv2.resize(img, (img\_rows,img\_cols)) for img in X\_valid[:,:,:,:]])

    # Transform targets to keras compatible format

    Y\_train = np\_utils.to\_categorical(Y\_train, num\_classes)

    Y\_valid = np\_utils.to\_categorical(Y\_valid, num\_classes)

    X\_train = X\_train.astype('float32')

    X\_valid = X\_valid.astype('float32')

    # preprocess data

    X\_train = X\_train / 255.0

    X\_valid = X\_valid / 255.0

    return X\_train, Y\_train, X\_valid, Y\_valid

X\_train, y\_train, X\_test, y\_test = load\_cifar10\_data(224, 224)

model = Model(input\_layer, [x, x1, x2], name='inception\_v1')

model.summary()

epochs = 10

initial\_lrate = 0.01

def decay(epoch, steps=100):

    initial\_lrate = 0.01

    drop = 0.96

    epochs\_drop = 8

    lrate = initial\_lrate \* math.pow(drop, math.floor((1+epoch)/epochs\_drop))

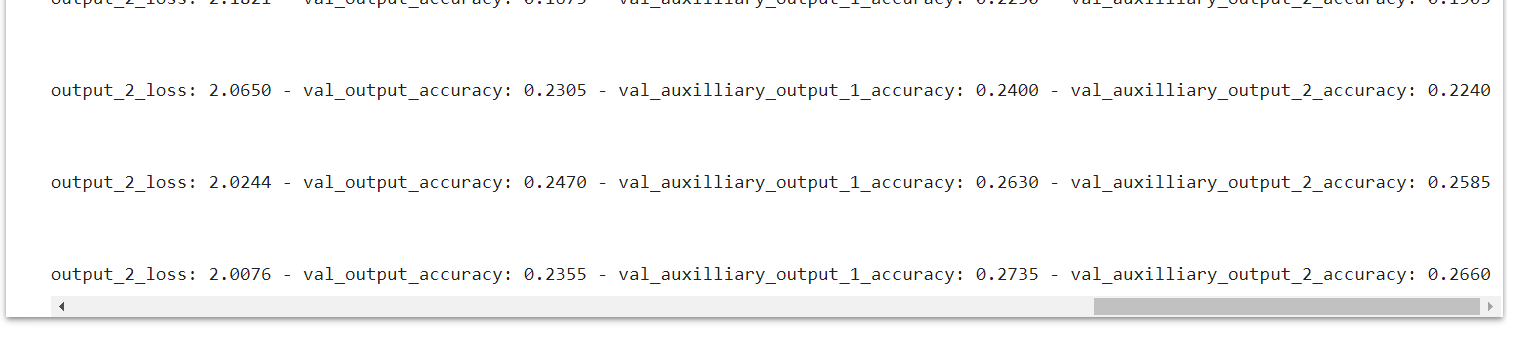
    return lrate

sgd = SGD(learning\_rate=initial\_lrate, momentum=0.9, nesterov=False)

lr\_sc = LearningRateScheduler(decay, verbose=1)

model.compile(loss=['categorical\_crossentropy', 'categorical\_crossentropy', 'categorical\_crossentropy'], loss\_weights=[1, 0.3, 0.3], optimizer=sgd, metrics=['accuracy'])

history = model.fit(X\_train, [y\_train, y\_train, y\_train], validation\_data=(X\_test, [y\_test, y\_test, y\_test]), epochs=epochs, batch\_size=256, callbacks=[lr\_sc])

****

**5.2) MNIST**

import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow.keras import datasets, layers, models, losses, Model

(x\_train, y\_train), (x\_test, y\_test)=tf.keras.datasets.mnist.load\_data()

x\_train = tf.pad(x\_train, [[0, 0], [2,2], [2,2]])/255

x\_test = tf.pad(x\_test, [[0, 0], [2,2], [2,2]])/255

x\_train = tf.expand\_dims(x\_train, axis=3, name=None)

x\_test = tf.expand\_dims(x\_test, axis=3, name=None)

x\_train = tf.repeat(x\_train, 3, axis=3)

x\_test = tf.repeat(x\_test, 3, axis=3)

x\_val = x\_train[-2000:,:,:]

y\_val = y\_train[-2000:]

x\_train = x\_train[:-2000,:,:]

y\_train = y\_train[:-2000]

def inception(x,

              filters\_1x1,

              filters\_3x3\_reduce,

              filters\_3x3,

              filters\_5x5\_reduce,

              filters\_5x5,

              filters\_pool):

  path1 = layers.Conv2D(filters\_1x1, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters\_3x3\_reduce, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters\_3x3, (1, 1), padding='same', activation='relu')(path2)

  path3 = layers.Conv2D(filters\_5x5\_reduce, (1, 1), padding='same', activation='relu')(x)

  path3 = layers.Conv2D(filters\_5x5, (1, 1), padding='same', activation='relu')(path3)

  path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)

  path4 = layers.Conv2D(filters\_pool, (1, 1), padding='same', activation='relu')(path4)

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

inp = layers.Input(shape=(32, 32, 3))

input\_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="bilinear", input\_shape=x\_train.shape[1:])(inp)

x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input\_tensor)

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

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

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

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

x = inception(x,

              filters\_1x1=64,

              filters\_3x3\_reduce=96,

              filters\_3x3=128,

              filters\_5x5\_reduce=16,

              filters\_5x5=32,

              filters\_pool=32)

x = inception(x,

              filters\_1x1=128,

              filters\_3x3\_reduce=128,

              filters\_3x3=192,

              filters\_5x5\_reduce=32,

              filters\_5x5=96,

              filters\_pool=64)

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

x = inception(x,

              filters\_1x1=192,

              filters\_3x3\_reduce=96,

              filters\_3x3=208,

              filters\_5x5\_reduce=16,

              filters\_5x5=48,

              filters\_pool=64)

aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)

aux1 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)

aux1 = layers.Flatten()(aux1)

aux1 = layers.Dense(1024, activation='relu')(aux1)

aux1 = layers.Dropout(0.7)(aux1)

aux1 = layers.Dense(10, activation='softmax')(aux1)

x = inception(x,

              filters\_1x1=160,

              filters\_3x3\_reduce=112,

              filters\_3x3=224,

              filters\_5x5\_reduce=24,

              filters\_5x5=64,

              filters\_pool=64)

x = inception(x,

              filters\_1x1=128,

              filters\_3x3\_reduce=128,

              filters\_3x3=256,

              filters\_5x5\_reduce=24,

              filters\_5x5=64,

              filters\_pool=64)

x = inception(x,

              filters\_1x1=112,

              filters\_3x3\_reduce=144,

              filters\_3x3=288,

              filters\_5x5\_reduce=32,

              filters\_5x5=64,

              filters\_pool=64)

aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)

aux2 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)

aux2 = layers.Flatten()(aux2)

aux2 = layers.Dense(1024, activation='relu')(aux2)

aux2 = layers.Dropout(0.7)(aux2)

aux2 = layers.Dense(10, activation='softmax')(aux2)

x = inception(x,

              filters\_1x1=256,

              filters\_3x3\_reduce=160,

              filters\_3x3=320,

              filters\_5x5\_reduce=32,

              filters\_5x5=128,

              filters\_pool=128)

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

x = inception(x,

              filters\_1x1=256,

              filters\_3x3\_reduce=160,

              filters\_3x3=320,

              filters\_5x5\_reduce=32,

              filters\_5x5=128,

              filters\_pool=128)

x = inception(x,

              filters\_1x1=384,

              filters\_3x3\_reduce=192,

              filters\_3x3=384,

              filters\_5x5\_reduce=48,

              filters\_5x5=128,

              filters\_pool=128)

x = layers.GlobalAveragePooling2D()(x)

x = layers.Dropout(0.4)(x)

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

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

model.compile(optimizer='adam', loss=[losses.sparse\_categorical\_crossentropy, losses.sparse\_categorical\_crossentropy, losses.sparse\_categorical\_crossentropy], loss\_weights=[1, 0.3, 0.3], metrics=['accuracy'])

history = model.fit(x\_train, [y\_train, y\_train, y\_train], validation\_data=(x\_val, [y\_val, y\_val, y\_val]), batch\_size=64, epochs=10)

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

axs[0].plot(history.history['loss'])

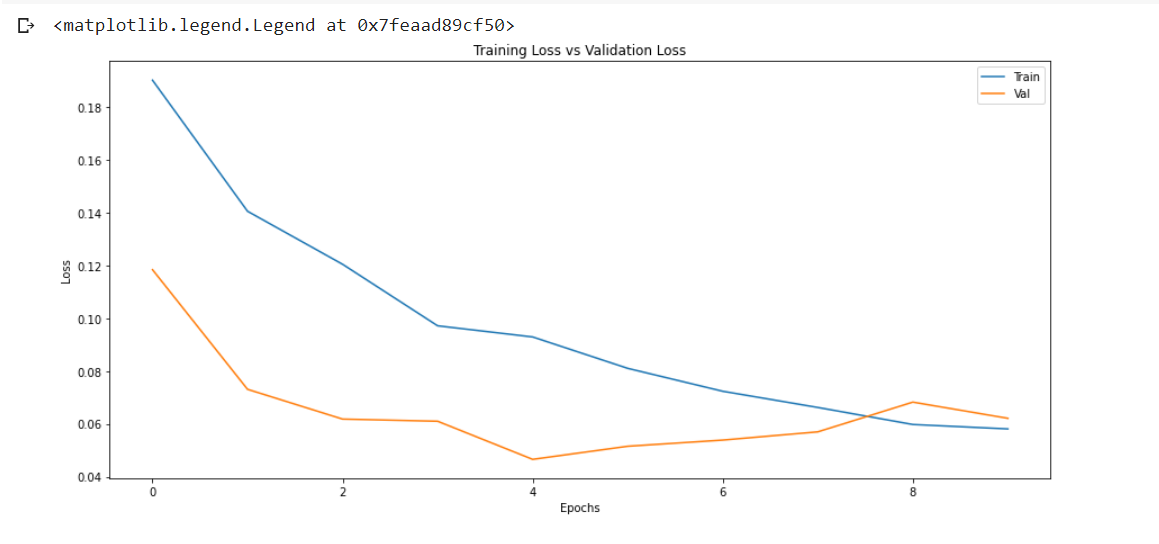
axs[0].plot(history.history['val\_loss'])

axs[0].title.set\_text('Training Loss vs Validation Loss')

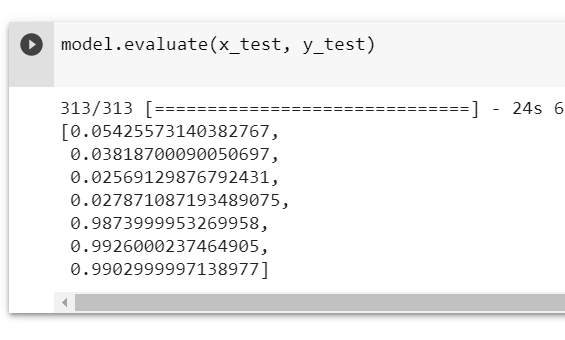
axs[0].set\_xlabel('Epochs')

axs[0].set\_ylabel('Loss')

axs[0].legend(['Train','Val'])

****

model.evaluate(x\_test, y\_test)

****

**5.3) SAVEE**

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

import librosa

import numpy as np

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

def preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, #log\_specgram

                 step\_size=10, eps=1e-10):

    mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

    mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

    return mel\_db.T

def load\_audio\_file(file\_path, input\_length=input\_length):

  data = librosa.core.load(file\_path, sr=16000)[0] #, sr=16000

  if len(data)>input\_length:

    max\_offset = len(data)-input\_length

    offset = np.random.randint(max\_offset)

    data = data[offset:(input\_length+offset)]

  else:

    if input\_length > len(data):

      max\_offset = input\_length - len(data)

      offset = np.random.randint(max\_offset)

    else:

      offset = 0

    data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

  data = preprocess\_audio\_mel\_T(data)

  return data

# Preprocessing the dataset

import os

from scipy.io import wavfile

import librosa

import matplotlib.pyplot as plt

import numpy as np

import cv2

rootDirectory = "/content/AudioData/"

personNames = ["DC","JE","JK","KL"]

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

X = list()

y = list()

for person in personNames:

  directory = os.path.join(rootDirectory,person)

  for filename in os.listdir(directory):

    filePath = os.path.join(directory, filename)

    a = load\_audio\_file(file\_path=filePath)

    data = cv2.merge([a,a,a])

    if(filename[0:1] in classes):

      X.append(data)

      y.append(classes.index(filename[0:1]))

    elif(filename[0:2] in classes):

      X.append(data)

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

X = np.asarray(X, dtype=np.float32)

y = np.asarray(y, dtype=np.float32)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, train\_size= 0.6 ,random\_state=10)

import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow.keras import datasets, layers, models, losses, Model

def inception(x,

              filters\_1x1,

              filters\_3x3\_reduce,

              filters\_3x3,

              filters\_5x5\_reduce,

              filters\_5x5,

              filters\_pool):

  path1 = layers.Conv2D(filters\_1x1, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters\_3x3\_reduce, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters\_3x3, (1, 1), padding='same', activation='relu')(path2)

  path3 = layers.Conv2D(filters\_5x5\_reduce, (1, 1), padding='same', activation='relu')(x)

  path3 = layers.Conv2D(filters\_5x5, (1, 1), padding='same', activation='relu')(path3)

  path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)

  path4 = layers.Conv2D(filters\_pool, (1, 1), padding='same', activation='relu')(path4)

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

inp = layers.Input(shape=(157, 320, 3))

input\_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="bilinear", input\_shape=X\_train.shape[1:])(inp)

x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input\_tensor)

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

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

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

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

x = inception(x,

              filters\_1x1=64,

              filters\_3x3\_reduce=96,

              filters\_3x3=128,

              filters\_5x5\_reduce=16,

              filters\_5x5=32,

              filters\_pool=32)

x = inception(x,

              filters\_1x1=128,

              filters\_3x3\_reduce=128,

              filters\_3x3=192,

              filters\_5x5\_reduce=32,

              filters\_5x5=96,

              filters\_pool=64)

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

x = inception(x,

              filters\_1x1=192,

              filters\_3x3\_reduce=96,

              filters\_3x3=208,

              filters\_5x5\_reduce=16,

              filters\_5x5=48,

              filters\_pool=64)

aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)

aux1 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)

aux1 = layers.Flatten()(aux1)

aux1 = layers.Dense(1024, activation='relu')(aux1)

aux1 = layers.Dropout(0.7)(aux1)

aux1 = layers.Dense(10, activation='softmax')(aux1)

x = inception(x,

              filters\_1x1=160,

              filters\_3x3\_reduce=112,

              filters\_3x3=224,

              filters\_5x5\_reduce=24,

              filters\_5x5=64,

              filters\_pool=64)

x = inception(x,

              filters\_1x1=128,

              filters\_3x3\_reduce=128,

              filters\_3x3=256,

              filters\_5x5\_reduce=24,

              filters\_5x5=64,

              filters\_pool=64)

x = inception(x,

              filters\_1x1=112,

              filters\_3x3\_reduce=144,

              filters\_3x3=288,

              filters\_5x5\_reduce=32,

              filters\_5x5=64,

              filters\_pool=64)

aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)

aux2 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)

aux2 = layers.Flatten()(aux2)

aux2 = layers.Dense(1024, activation='relu')(aux2)

aux2 = layers.Dropout(0.7)(aux2)

aux2 = layers.Dense(10, activation='softmax')(aux2)

x = inception(x,

              filters\_1x1=256,

              filters\_3x3\_reduce=160,

              filters\_3x3=320,

              filters\_5x5\_reduce=32,

              filters\_5x5=128,

              filters\_pool=128)

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

x = inception(x,

              filters\_1x1=256,

              filters\_3x3\_reduce=160,

              filters\_3x3=320,

              filters\_5x5\_reduce=32,

              filters\_5x5=128,

              filters\_pool=128)

x = inception(x,

              filters\_1x1=384,

              filters\_3x3\_reduce=192,

              filters\_3x3=384,

              filters\_5x5\_reduce=48,

              filters\_5x5=128,

              filters\_pool=128)

x = layers.GlobalAveragePooling2D()(x)

x = layers.Dropout(0.4)(x)

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

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

model.compile(optimizer='adam', loss=[losses.sparse\_categorical\_crossentropy, losses.sparse\_categorical\_crossentropy, losses.sparse\_categorical\_crossentropy], loss\_weights=[1, 0.3, 0.3], metrics=['accuracy'])

history = model.fit(X\_train, [y\_train, y\_train, y\_train], validation\_data=(X\_test, [y\_test, y\_test, y\_test]), batch\_size=64, epochs=30)

fig, axs = plt.subplots(figsize=(12,8))

axs.plot(history.history['loss'])

axs.plot(history.history['val\_loss'])

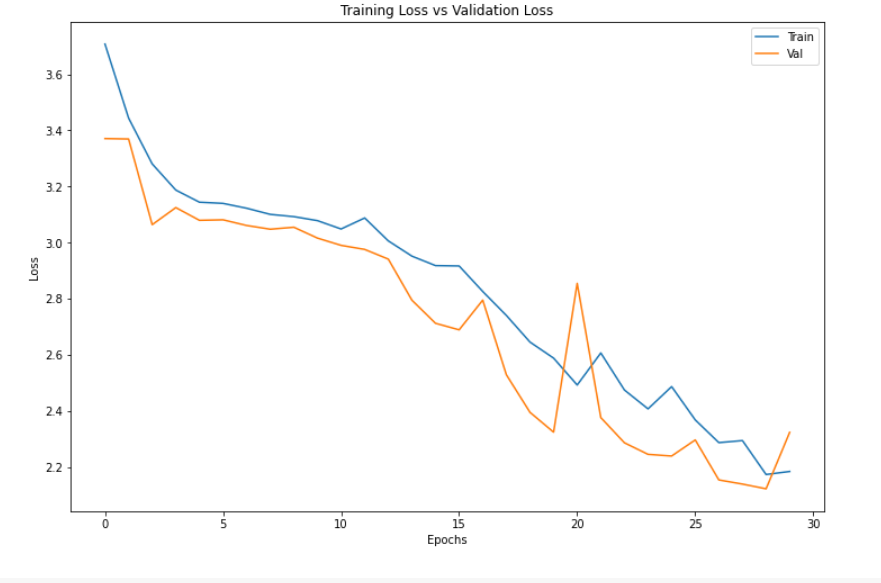
axs.title.set\_text('Training Loss vs Validation Loss')

axs.set\_xlabel('Epochs')

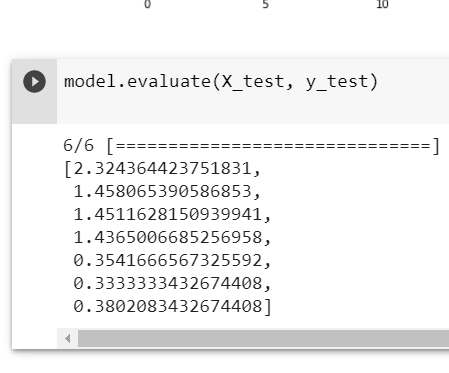
axs.set\_ylabel('Loss')

axs.legend(['Train','Val'])

plt.show()

****

model.evaluate(X\_test, y\_test)

****

**5.4) EmoDB**

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

import librosa

import numpy as np

input\_length = 16000\*5

batch\_size = 32

n\_mels = 320

def preprocess\_audio\_mel\_T(audio, sample\_rate=16000, window\_size=20, #log\_specgram

                 step\_size=10, eps=1e-10):

    mel\_spec = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels= n\_mels)

    mel\_db = (librosa.power\_to\_db(mel\_spec, ref=np.max) + 40)/40

    return mel\_db.T

def load\_audio\_file(file\_path, input\_length=input\_length):

  data = librosa.core.load(file\_path, sr=16000)[0] #, sr=16000

  if len(data)>input\_length:

    max\_offset = len(data)-input\_length

    offset = np.random.randint(max\_offset)

    data = data[offset:(input\_length+offset)]

  else:

    if input\_length > len(data):

      max\_offset = input\_length - len(data)

      offset = np.random.randint(max\_offset)

    else:

      offset = 0

    data = np.pad(data, (offset, input\_length - len(data) - offset), "constant")

  data = preprocess\_audio\_mel\_T(data)

  return data

# Preprocessing the dataset

import os

from scipy.io import wavfile

import librosa

import matplotlib.pyplot as plt

import numpy as np

import cv2

directory = "/content/wav/"

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

X = list()

y = list()

for filename in os.listdir(directory):

  filePath = os.path.join(directory, filename)

  a = load\_audio\_file(file\_path=filePath)

  data = cv2.merge([a,a,a])

  if(filename[5:6] in classes):

    X.append(data)

    y.append(classes.index(filename[5:6]))

X = np.asarray(X, dtype=np.float32)

y = np.asarray(y, dtype=np.float32)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

# dataset preparation

from tensorflow.keras import datasets,layers,models

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, train\_size= 0.6 ,random\_state=10)

def inception(x,

              filters\_1x1,

              filters\_3x3\_reduce,

              filters\_3x3,

              filters\_5x5\_reduce,

              filters\_5x5,

              filters\_pool):

  path1 = layers.Conv2D(filters\_1x1, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters\_3x3\_reduce, (1, 1), padding='same', activation='relu')(x)

  path2 = layers.Conv2D(filters\_3x3, (1, 1), padding='same', activation='relu')(path2)

  path3 = layers.Conv2D(filters\_5x5\_reduce, (1, 1), padding='same', activation='relu')(x)

  path3 = layers.Conv2D(filters\_5x5, (1, 1), padding='same', activation='relu')(path3)

  path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)

  path4 = layers.Conv2D(filters\_pool, (1, 1), padding='same', activation='relu')(path4)

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

inp = layers.Input(shape=(157, 320, 3))

input\_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="bilinear", input\_shape=X\_train.shape[1:])(inp)

x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input\_tensor)

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

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

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

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

x = inception(x,

              filters\_1x1=64,

              filters\_3x3\_reduce=96,

              filters\_3x3=128,

              filters\_5x5\_reduce=16,

              filters\_5x5=32,

              filters\_pool=32)

x = inception(x,

              filters\_1x1=128,

              filters\_3x3\_reduce=128,

              filters\_3x3=192,

              filters\_5x5\_reduce=32,

              filters\_5x5=96,

              filters\_pool=64)

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

x = inception(x,

              filters\_1x1=192,

              filters\_3x3\_reduce=96,

              filters\_3x3=208,

              filters\_5x5\_reduce=16,

              filters\_5x5=48,

              filters\_pool=64)

aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)

aux1 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)

aux1 = layers.Flatten()(aux1)

aux1 = layers.Dense(1024, activation='relu')(aux1)

aux1 = layers.Dropout(0.7)(aux1)

aux1 = layers.Dense(10, activation='softmax')(aux1)

x = inception(x,

              filters\_1x1=160,

              filters\_3x3\_reduce=112,

              filters\_3x3=224,

              filters\_5x5\_reduce=24,

              filters\_5x5=64,

              filters\_pool=64)

x = inception(x,

              filters\_1x1=128,

              filters\_3x3\_reduce=128,

              filters\_3x3=256,

              filters\_5x5\_reduce=24,

              filters\_5x5=64,

              filters\_pool=64)

x = inception(x,

              filters\_1x1=112,

              filters\_3x3\_reduce=144,

              filters\_3x3=288,

              filters\_5x5\_reduce=32,

              filters\_5x5=64,

              filters\_pool=64)

aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)

aux2 = layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)

aux2 = layers.Flatten()(aux2)

aux2 = layers.Dense(1024, activation='relu')(aux2)

aux2 = layers.Dropout(0.7)(aux2)

aux2 = layers.Dense(10, activation='softmax')(aux2)

x = inception(x,

              filters\_1x1=256,

              filters\_3x3\_reduce=160,

              filters\_3x3=320,

              filters\_5x5\_reduce=32,

              filters\_5x5=128,

              filters\_pool=128)

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

x = inception(x,

              filters\_1x1=256,

              filters\_3x3\_reduce=160,

              filters\_3x3=320,

              filters\_5x5\_reduce=32,

              filters\_5x5=128,

              filters\_pool=128)

x = inception(x,

              filters\_1x1=384,

              filters\_3x3\_reduce=192,

              filters\_3x3=384,

              filters\_5x5\_reduce=48,

              filters\_5x5=128,

              filters\_pool=128)

x = layers.GlobalAveragePooling2D()(x)

x = layers.Dropout(0.4)(x)

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

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

model.compile(optimizer='adam', loss=[losses.sparse\_categorical\_crossentropy, losses.sparse\_categorical\_crossentropy, losses.sparse\_categorical\_crossentropy], loss\_weights=[1, 0.3, 0.3], metrics=['accuracy'])

history = model.fit(X\_train, [y\_train, y\_train, y\_train], validation\_data=(X\_test, [y\_test, y\_test, y\_test]), batch\_size=64, epochs=30)

fig, axs = plt.subplots(figsize=(12,8))

axs.plot(history.history['loss'])

axs.plot(history.history['val\_loss'])

axs.title.set\_text('Training Loss vs Validation Loss')

axs.set\_xlabel('Epochs')

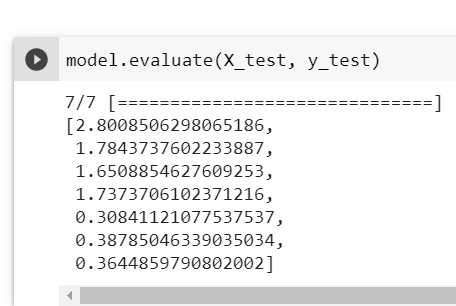
axs.set\_ylabel('Loss')

axs.legend(['Train','Val'])

plt.show()

****

model.evaluate(X\_test, y\_test)

****

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