# Spoken Digit Recognition

In this notebook, You will do Spoken Digit Recognition.

Input - speech signal, output - digit number

It contains

1. Reading the dataset. and Preprocess the data set. Detailed instrctions are given below. You have to write the code in the same cell which contains the instrction.
2. Training the LSTM with RAW data
3. Converting to spectrogram and Training the LSTM network
4. Creating the augmented data and doing step 2 and 3 again.

## instructions:

1. Don't change any Grader Functions. Don't manipulate any Grader functions. If you manipulate any, it will be considered as plagiarised.

2. Please read the instructions on the code cells and markdown cells. We will explain what to write.

3. please return outputs in the same format what we asked. Eg. Don't return List of we are asking for a numpy array.

4. Please read the external links that we are given so that you will learn the concept behind the code that you are writing.

5. We are giving instructions at each section if necessary, please follow them.

## Every Grader function has to return True.

In [5]:

```
import numpy as np
import pandas as pd
import librosa
import os
##if you need any imports you can do that here.
```

We shared recordings.zip, please unzip those.

In [ ]:

```
!wget --header="Host: doc-04-5k-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75
Safari/537.36" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
ation/signed-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-
US;q=0.8,en;q=0.7" --header="Cookie: AUTH_gnb78hdmdiks9t0b8kec09hpa7nncs5e_nonce=elsrji0nn9pme" --
header="Connection: keep-alive" "https://doc-04-5k-
docs.googleusercontent.com/docs/securesc/lcn000d4f5ncb3531bgn3uus2eb0i5pv/8u37gp18371 8ae9o6jhrog7k
g6k/1602681225000/03515051603858730688/03515051603858730688/1DjgjodZeX48koAeXwH-zBFYhcHlQohmo?e=do
wnload&authuser=0&nonce=elsrji0nn9pme&user=03515051603858730688&hash=9f6j1ugrhpk0rhd8n8cpv4l0vuj5b
-c -O 'recordings.zip'
```

```
--2020-10-14 13:15:06--  https://doc-04-5k-
```

```
docs.googleusercontent.com/docs/securesc/lcn000d4f5ncb3531bgn3uus2eb0i5pv/8u37gp18371 8ae9o6jhrog7k8
g6k/1602681225000/03515051603858730688/03515051603858730688/1DjgjodZeX48koAeXwH-zBFYhcHlQohmo?
e=download&authuser=0&nonce=elsrji0nn9pme&user=03515051603858730688&hash=9f6j1ugrhpk0rhd8n8cpv4l0vu
8a
Resolving doc-04-5k-docs.googleusercontent.com (doc-04-5k-docs.googleusercontent.com)...
64.233.166.132, 2a00:1450:400c:c09::84
Connecting to doc-04-5k-docs.googleusercontent.com (doc-04-5k-
docs.googleusercontent.com)|64.233.166.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/x-zip-compressed]
Saving to: 'recordings.zip'

recordings.zip          [ <=>                ]   8.85M  --.-KB/s    in 0.1s

2020-10-14 13:15:07 (76.9 MB/s) - 'recordings.zip' saved [9282934]
```

In [3]:

```
#read the all file names in the recordings folder given by us
#(if you get entire path, it is very useful in future)
#save those files names as list in "all_files"

import os

all_files = os.listdir("/content/recordings")
```

## Grader function 1

In [ ]:

```
def grader_files():
    temp = len(all_files)==2000
    temp1 = all([x[-3:]=="wav" for x in all_files])
    temp = temp and temp1
    return temp
grader_files()
```

Out[ ]:

```
True
```

Create a dataframe(name=df_audio) with two columns(path, label).
You can get the label from the first letter of name.
Eg: 0_jackson_0 --> 0
0_jackson_43 --> 0

In [6]:

```
#Create a dataframe(name=df_audio) with two columns(path, label).
#You can get the label from the first letter of name.
#Eg: 0_jackson_0 --> 0
#0_jackson_43 --> 0
df_audio=[]
for file in all_files:
  file_split=file.split("_")
  label=file_split[0]
  path="recordings/"+str(file)
  df_audio.append([path,label])

df_audio = pd.DataFrame(df_audio, columns = ['path', 'label'])
```

In [ ]:

```
#info
df_audio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 2 columns):
```

```
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   path    2000 non-null   object
 1   label   2000 non-null   object
dtypes: object(2)
memory usage: 31.4+ KB
```

### Grader function 2

In [ ]:

```
def grader_df():
    flag_shape = df_audio.shape==(2000,2)
    flag_columns = all(df_audio.columns==['path', 'label'])
    list_values = list(df_audio.label.value_counts())
    flag_label = len(list_values)==10
    flag_label2 = all([i==200 for i in list_values])
    final_flag = flag_shape and flag_columns and flag_label and flag_label2
    return final_flag
grader_df()
```

Out[ ]:

```
True
```

In [ ]:

```
from sklearn.utils import shuffle
df_audio = shuffle(df_audio, random_state=33)#don't change the random state
```

```
Train and Validation split
```

In [ ]:

```
#split the data into train and validation and save in X_train, X_test, y_train, y_test
#use stratify sampling
#use random state of 45
#use test size of 30%
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_audio['path'], df_audio['label'], test_size=
0.30, random_state=45,stratify=df_audio['label'])
```

### Grader function 3

In [ ]:

```
def grader_split():
    flag_len = (len(X_train)==1400) and (len(X_test)==600) and (len(y_train)==1400) and (len(y_test
)==600)
    values_ytrain = list(y_train.value_counts())
    flag_ytrain = (len(values_ytrain)==10) and (all([i==140 for i in values_ytrain]))
    values_ytest = list(y_test.value_counts())
    flag_ytest = (len(values_ytest)==10) and (all([i==60 for i in values_ytest]))
    final_flag = flag_len and flag_ytrain and flag_ytest
    return final_flag
grader_split()
```

Out[ ]:

```
True
```

```
Preprocessing

All files are in the "WAV" format. We will read those raw data files using the librosa
```

In [10]:

```
sample_rate = 22050
def load_wav(x, get_duration=True):
    '''This return the array values of audio with sampling rate of 22050 and Duration'''
    #loading the wav file with sampling rate of 22050
    samples, sample_rate = librosa.load(x, sr=22050)
    if get_duration:
        duration = librosa.get_duration(samples, sample_rate)
        return [samples, duration]
    else:
        return samples
```

In [ ]:

```
#use load_wav function that was written above to get every wave.
#save it in X_train_processed and X_test_processed
# X_train_processed/X_test_processed should be dataframes with two columns(raw_data, duration) wit
h same index of X_train/y_train
X_train_processed=[]
X_test_processed=[]

for file in X_train:
  row=load_wav(file)
  X_train_processed.append(row)

for file in X_test:
  row=load_wav(file)
  X_test_processed.append(row)

#https://www.geeksforgeeks.org/creating-pandas-dataframe-using-list-of-lists/

X_train_processed = pd.DataFrame(X_train_processed, columns = ['raw_data', 'duration'])
X_test_processed = pd.DataFrame(X_test_processed, columns = ['raw_data', 'duration'])
```

In [ ]:

```
X_train_processed.head()
```

Out[ ]:

| | raw_data | duration |
|---|---|---|
| **0** | [-0.0010944874, -0.0008889665, 0.00018223508, ... | 0.473379 |
| **1** | [0.009576598, 0.011238026, 0.011156514, 0.0102... | 0.481134 |
| **2** | [-0.006540089, -0.005709454, -0.0065910434, -0... | 0.230385 |
| **3** | [-0.0122581925, -0.01575047, -0.0175306, -0.01... | 0.428889 |
| **4** | [-0.010983801, -0.011517354, -0.009270695, -0.... | 0.532154 |

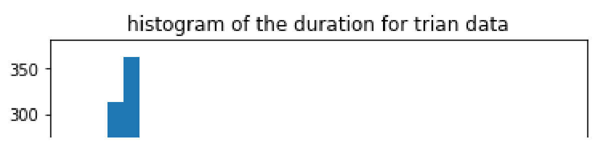
In [ ]:

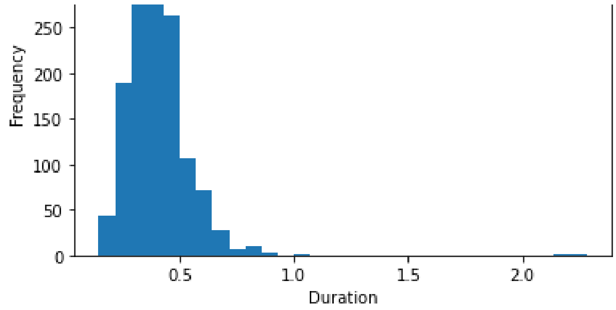
```
#plot the histogram of the duration for trian
import matplotlib.pyplot as plt

plt.hist(x=X_train_processed['duration'], bins=30)
plt.xlabel('Duration')
plt.ylabel('Frequency')
plt.title('histogram of the duration for trian data')
```

Out[ ]:

```
Text(0.5, 1.0, 'histogram of the duration for trian data')
```






In [ ]:

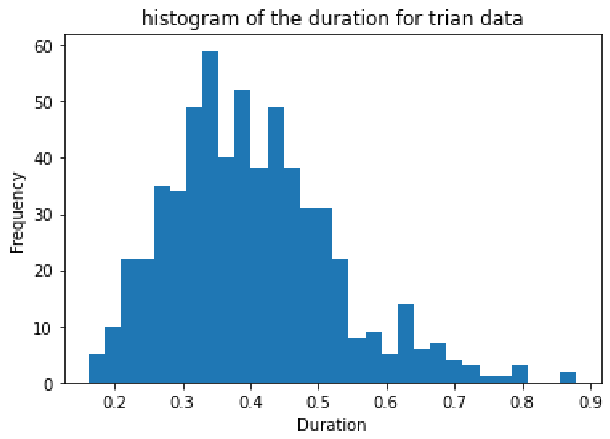
```
#plot the histogram of the duration for test

import matplotlib.pyplot as plt

plt.hist(x=X_test_processed['duration'], bins=30)
plt.xlabel('Duration')
plt.ylabel('Frequency')
plt.title('histogram of the duration for trian data')
```

Out[ ]:

```
Text(0.5, 1.0, 'histogram of the duration for trian data')
```




In [ ]:

```
#print 0 to 100 percentile values with step size of 10 for train data duration.
##print 90 to 100 percentile values with step size of 1.
for i in range(0,101,10):
  print("{0}th percentile is {1}".format(i,np.percentile(X_train_processed['duration'], i)))
  if i == 100:
    print("---------------------------------------------")
    for i in range(90,101,1):
      print("{0}th percentile is {1}".format(i,np.percentile(X_train_processed['duration'], i)))
```

```
0th percentile is 0.1435374149659864
10th percentile is 0.2606938775510204
20th percentile is 0.2993015873015873
30th percentile is 0.33215419501133786
40th percentile is 0.36093424036281185
50th percentile is 0.39034013605442175
60th percentile is 0.416172335600907
70th percentile is 0.4451972789115646
80th percentile is 0.48027210884353744
90th percentile is 0.5549297052154195
100th percentile is 2.282766439909297
---------------------------------------------
90th percentile is 0.5549297052154195
91th percentile is 0.569807709750567
92th percentile is 0.5815492063492064
93th percentile is 0.594125170068027 8
94th percentile is 0.6078993197278912
95th percentile is 0.622421768707483
```

```
96th percentile is 0.6379972789115645
97th percentile is 0.6582807256235828
98th percentile is 0.689717006802721
99th percentile is 0.8183029478458049
100th percentile is 2.282766439909297
```

## Grader function 4

In [ ]:

```
def grader_processed():
    flag_columns = (all(X_train_processed.columns==['raw_data', 'duration'])) and (all(X_test_proce
ssed.columns==['raw_data', 'duration']))
    flag_shape = (X_train_processed.shape ==(1400, 2)) and (X_test_processed.shape==(600,2))
    return flag_columns and flag_shape
grader_processed()
```

Out[ ]:

```
True
```

```
Based on our analysis 99 percentile values are less than 0.8sec so we will limit maximum le
ngth of X_train_processed and X_test_processed to 0.8 sec. It is similar to pad_sequence fo
r a text dataset.

While loading the audio files, we are using sampling rate of 22050 so one sec will give arr
ay of length 22050. so, our maximum length is 0.8*22050 = 17640

Pad with Zero if length of sequence is less than 17640 else Truncate the number.

Also create a masking vector for train and test.

masking vector value = 1 if it is real value, 0 if it is pad value. Masking vector data typ
e must be bool.
```

In [ ]:

```
max_length  = 17640
```

In [ ]:

```
## as discussed above, Pad with Zero if length of sequence is less than 17640 else Truncate the nu
mber.
## save in the X_train_pad_seq, X_test_pad_seq
## also Create masking vector X_train_mask, X_test_mask

## all the X_train_pad_seq, X_test_pad_seq, X_train_mask, X_test_mask will be numpy arrays mask ve
ctor dtype must be bool.
from keras.preprocessing.sequence import pad_sequences
from keras import layers


X_train_pad_seq=pad_sequences(X_train_processed["raw_data"],padding='post',maxlen=17640,truncating
='post',value=0.00,dtype='float32')
X_test_pad_seq=pad_sequences(X_test_processed["raw_data"],padding='post',maxlen=17640,truncating='p
ost',value=0.00,dtype='float32')
```

In [ ]:

```
def masking(data):
  X_mask=np.empty((0,17640), dtype='bool')
  for i,e in enumerate(data):
    preprocessing_len=len(e)
    difference  =17640-preprocessing_len
    if (difference>0):
      mask_vector=np.array([1]*(preprocessing_len)+[0]*difference,dtype='bool')
    elif (difference==0):
```

```
        mask_vector=np.array([1]*17640)
      X_mask = np.append(X_mask, [mask_vector], axis=0)
  return X_mask
```

In [ ]:

```
X_train_mask=masking(X_train_processed["raw_data"])
X_test_mask=masking(X_test_processed["raw_data"])
```

### Grader function 5

In [ ]:

```
def grader_padoutput():
    flag_padshape = (X_train_pad_seq.shape==(1400, 17640)) and (X_test_pad_seq.shape==(600, 17640))
and (y_train.shape==(1400,))
    flag_maskshape = (X_train_mask.shape==(1400, 17640)) and (X_test_mask.shape==(600, 17640)) and
(y_test.shape==(600,))
    flag_dtype = (X_train_mask.dtype==bool) and (X_test_mask.dtype==bool)
    return flag_padshape and flag_maskshape and flag_dtype
grader_padoutput()
```

Out[ ]:

```
True
```

## 1. Giving Raw data directly.

```
Now we have

Train data: X_train_pad_seq, X_train_mask and y_train
Test data: X_test_pad_seq, X_test_mask and y_test

We will create a LSTM model which takes this input.

Task:

1. Create an LSTM network which takes "X_train_pad_seq" as input, "X_train_mask" as mask
input. You can use any number of LSTM cells. Please read LSTM
documentation(https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM) in
tensorflow to know more about mask and also
https://www.tensorflow.org/guide/keras/masking_and_padding
2. Get the final output of the LSTM and give it to Dense layer of any size and then give it
to Dense layer of size 10(because we have 10 outputs) and then compile with the sparse
categorical cross entropy( because we are not converting it to one hot vectors).
3. Use tensorboard to plot the graphs of loss and metric(use micro F1 score as metric) and
histograms of gradients.
4. make sure that it won't overfit.
5. You are free to include any regularization
```

In [ ]:

```
from keras.layers import Input, LSTM, Dense,GlobalAveragePooling1D
from keras.models import Model
import tensorflow as tf
```

In [ ]:

```
## as discussed above, please write the LSTM

input_shape=X_train_pad_seq.shape
mask_shape=X_train_mask.shape

lstm_input = Input(shape=(input_shape[1],1), dtype='float32')
mask_input = Input(shape=(mask_shape[1]),dtype='bool')
```

```
x = LSTM(16  , return_sequences=False)(lstm_input,mask=mask_input)
x = Dense(24, activation="relu")(x)
output = Dense(10, activation="softmax")(x)

model1 = Model(inputs=[lstm_input,mask_input],outputs=[output])
```

In [ ]:

```
from sklearn.metrics import f1_score
import tensorflow_addons as tfa

from tensorflow_addons.metrics import F1Score


model1.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
```

In [ ]:

```
model1.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 17640, 1)]   0

__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 17640)]      0

__________________________________________________________________________________________________
lstm (LSTM)                     (None, 16)           1152        input_1[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 24)           408         lstm[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 10)           250         dense[0][0]
==================================================================================================
Total params: 1,810
Trainable params: 1,810
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [40]:

```
from keras.callbacks import TensorBoard

class CustomCallback(tf.keras.callbacks.Callback):
  def __init__(self, threshold,validation_data=()):
    super(CustomCallback, self).__init__()
    self.X_val, self.y_val = validation_data
    self.threshold = threshold

  def on_train_begin(self, logs={}):
        self.f1Score_List = []
  def on_epoch_end(self, epoch,validation_data=(), logs={}):
    y_targ = self.y_val
    y_pred_array = np.array((self.model.predict(self.X_val)))
    y_predict = np.argmax(y_pred_array, axis=1)

    f1Score = f1_score(y_targ, y_predict,average='micro')
    print(" - F1 Score:{0}".format(f1Score))
    self.f1Score_List.append(f1Score)
    if f1Score >= self.threshold:
        print("Stopping training,since we reach {0} F1 Score.".format(f1Score))
        self.model.stop_training = True
```

In [ ]:

```
!mkdir model_1
```

```
time: 134 ms
```

In [55]:

```
%load_ext tensorboard
```

In [ ]:

```
y_train= y_train.astype(int)

y_test= y_test.astype(int)
```

```
time: 1.43 ms
```

In [ ]:

```
#train your model

tensorboard_callback = TensorBoard(log_dir='model_1',histogram_freq=1)


model1.fit([X_train_pad_seq,X_train_mask], y_train, batch_size=64, epochs=10,
           validation_data=([X_test_pad_seq,X_test_mask], y_test),
           callbacks=[tensorboard_callback,CustomCallback(threshold=0.20,validation_data=([X_test_p
ad_seq,X_test_mask], y_test))]
                                    )
```

```
Epoch 1/10
 1/22 [>.............................] - ETA: 0s - loss: 2.3026 - accuracy:
0.1406WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after 2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
 2/22 [=>............................] - ETA: 36s - loss: 2.3025 - accuracy:
0.1484WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time
(batch time: 0.8363s vs `on_train_batch_end` time: 2.7817s). Check your callbacks.
22/22 [==============================] - ETA: 0s - loss: 2.3032 - accuracy: 0.0971 - F1
Score:0.10166666666666667
22/22 [==============================] - 27s 1s/step - loss: 2.3032 - accuracy: 0.0971 - val_loss:
2.3026 - val_accuracy: 0.1017
Epoch 2/10
22/22 [==============================] - ETA: 0s - loss: 2.3029 - accuracy: 0.0871 - F1
Score:0.09833333333333333
22/22 [==============================] - 22s 1s/step - loss: 2.3029 - accuracy: 0.0871 - val_loss:
2.3026 - val_accuracy: 0.0983
Epoch 3/10
22/22 [==============================] - ETA: 0s - loss: 2.3028 - accuracy: 0.1007 - F1
Score:0.10166666666666667
22/22 [==============================] - 22s 1s/step - loss: 2.3028 - accuracy: 0.1007 - val_loss:
2.3026 - val_accuracy: 0.1017
Epoch 4/10
22/22 [==============================] - ETA: 0s - loss: 2.3028 - accuracy: 0.1014 - F1
Score:0.10000000000000002
22/22 [==============================] - 22s 1s/step - loss: 2.3028 - accuracy: 0.1014 - val_loss:
2.3026 - val_accuracy: 0.1000
Epoch 5/10
22/22 [==============================] - ETA: 0s - loss: 2.3028 - accuracy: 0.1021 - F1
Score:0.10000000000000002
22/22 [==============================] - 22s 997ms/step - loss: 2.3028 - accuracy: 0.1021 - val_lo
ss: 2.3026 - val_accuracy: 0.1000
Epoch 6/10
22/22 [==============================] - ETA: 0s - loss: 2.3027 - accuracy: 0.0993 - F1
Score:0.10000000000000002
22/22 [==============================] - 22s 1s/step - loss: 2.3027 - accuracy: 0.0993 - val_loss:
2.3026 - val_accuracy: 0.1000
Epoch 7/10
22/22 [==============================] - ETA: 0s - loss: 2.3027 - accuracy: 0.0971 - F1
Score:0.10000000000000002
22/22 [==============================] - 22s 995ms/step - loss: 2.3027 - accuracy: 0.0971 - val_lo
ss: 2.3026 - val_accuracy: 0.1000
Epoch 8/10
22/22 [==============================] - ETA: 0s - loss: 2.3027 - accuracy: 0.0986 - F1
Score:0.10000000000000002
22/22 [==============================] - 22s 995ms/step - loss: 2.3027 - accuracy: 0.0986 - val_lo
```

```
ss: 2.3026 - val_accuracy: 0.1000
Epoch 9/10
22/22 [==============================] - ETA: 0s - loss: 2.3027 - accuracy: 0.1029 - F1
Score:0.10000000000000002
22/22 [==============================] - 22s 1000ms/step - loss: 2.3027 - accuracy: 0.1029 - val_l
oss: 2.3026 - val_accuracy: 0.1000
Epoch 10/10
22/22 [==============================] - ETA: 0s - loss: 2.3026 - accuracy: 0.1021 - F1
Score:0.10166666666666667
22/22 [==============================] - 22s 1s/step - loss: 2.3026 - accuracy: 0.1021 - val_loss:
2.3026 - val_accuracy: 0.1017
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7fd505fdc6d8>
```

```
time: 4min 1s
```

In [ ]:

```
%tensorboard --logdir 'model_1'
```

```
time: 3.59 s
```

### 2. Converting into spectrogram and giving spectrogram data as input

```
We can use librosa to convert raw data into spectrogram. A spectrogram shows the features i
n a two-dimensional representation with the
intensity of a frequency at a point in time i.e we are converting Time domain to frequency
domain. you can read more about this in https://pnsn.org/spectrograms/what-is-a-spectrogram
```

In [ ]:

```
def convert_to_spectrogram(raw_data):
    '''converting to spectrogram'''
    spectrum = librosa.feature.melspectrogram(y=raw_data, sr=sample_rate, n_mels=64)
    logmel_spectrum = librosa.power_to_db(S=spectrum, ref=np.max)
    return logmel_spectrum
```

In [ ]:

```
##use convert_to_spectrogram and convert every raw sequence in X_train_pad_seq and X_test_pad-seq.
## save those all in the X_train_spectrogram and X_test_spectrogram ( These two arrays must be num
py arrays)

def spectrogram(data):
  spectrogram = []
  for i in data:
    data_spectrogram =convert_to_spectrogram(i)
    spectrogram.append(data_spectrogram)
  return np.array(spectrogram)
```

In [ ]:

```
X_train_spectrogram=spectrogram(X_train_pad_seq)
X_test_spectrogram=spectrogram(X_test_pad_seq)
```

Grader function 6

In [ ]:

```
def grader_spectrogram():
    flag_shape = (X_train_spectrogram.shape==(1400,64, 35)) and (X_test_spectrogram.shape == (600,
64, 35))
    return flag_shape
grader_spectrogram()
```

Out[ ]:

```
True
```

```
Now we have

Train data: X_train_spectrogram and y_train
Test data: X_test_spectrogram and y_test

We will create a LSTM model which takes this input.

Task:

1. Create an LSTM network which takes "X_train_spectrogram" as input and has to return
output at every time step.
2. Average the output of every time step and give this to the Dense layer of any size.
3. give the above output to Dense layer of size 10( output layer) and train the network wit
h sparse categorical cross entropy.
4. Use tensorboard to plot the graphs of loss and metric(use micro F1 score as metric) and
histograms of gradients.
5. make sure that it won't overfit.
6. You are free to include any regularization
```

In [ ]:

```
## as discussed above, please write the LSTM

input_shape=X_train_spectrogram.shape

lstm_input = Input(shape=(input_shape[1],input_shape[2]), dtype='float32')
mask_input = Input(shape=(mask_shape[1]),dtype='bool')

x = LSTM(64   , return_sequences=True)(lstm_input)
x=GlobalAveragePooling1D()(x)
x = Dense(32, activation="relu")(x)
output = Dense(10, activation="softmax")(x)

model2 = Model(inputs=[lstm_input],outputs=[output])

model2.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

model2.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 64, 35)]          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 64, 64)            25600
_________________________________________________________________
global_average_pooling1d (Gl (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 10)                330
=================================================================
Total params: 28,010
Trainable params: 28,010
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
!mkdir model2
```

```
time: 174 ms
```

In [ ]:

```
#train your model

tensorboard_callback = TensorBoard(log_dir='model2',histogram_freq=1)


model2.fit(X_train_spectrogram, y_train, batch_size=64, epochs=200, verbose=1,
           validation_data=(X_test_spectrogram, y_test),
           callbacks=[tensorboard_callback,CustomCallback(threshold=0.80,validation_data=(X_test_sp
ectrogram, y_test))]
                                    )
```

```
Epoch 1/200
 2/22 [=>............................] - ETA: 1s - loss: 2.4032 - accuracy:
0.0703WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time
(batch time: 0.0138s vs `on_train_batch_end` time: 0.1354s). Check your callbacks.
21/22 [===========================>..] - ETA: 0s - loss: 2.2728 - accuracy: 0.1243 - F1
Score:0.16666666666666666
22/22 [==============================] - 1s 52ms/step - loss: 2.2699 - accuracy: 0.1243 -
val_loss: 2.2200 - val_accuracy: 0.1667
Epoch 2/200
22/22 [==============================] - ETA: 0s - loss: 2.1914 - accuracy: 0.2150 - F1 Score:0.29
22/22 [==============================] - 0s 15ms/step - loss: 2.1914 - accuracy: 0.2150 -
val_loss: 2.1575 - val_accuracy: 0.2900
Epoch 3/200
22/22 [==============================] - ETA: 0s - loss: 2.1329 - accuracy: 0.2800 - F1
Score:0.31333333333333335
22/22 [==============================] - 0s 14ms/step - loss: 2.1329 - accuracy: 0.2800 -
val_loss: 2.0958 - val_accuracy: 0.3133
Epoch 4/200
16/22 [====================>.........] - ETA: 0s - loss: 2.0717 - accuracy: 0.3027 - F1 Score:0.3
22/22 [==============================] - 0s 14ms/step - loss: 2.0530 - accuracy: 0.3057 -
val_loss: 2.0391 - val_accuracy: 0.3000
Epoch 5/200
17/22 [======================>.......] - ETA: 0s - loss: 2.0083 - accuracy: 0.2923 - F1
Score:0.32166666666666666
22/22 [==============================] - 0s 14ms/step - loss: 1.9995 - accuracy: 0.3050 -
val_loss: 1.9842 - val_accuracy: 0.3217
Epoch 6/200
22/22 [==============================] - ETA: 0s - loss: 1.9330 - accuracy: 0.3457 - F1
Score:0.34833333333333333
22/22 [==============================] - 0s 14ms/step - loss: 1.9330 - accuracy: 0.3457 -
val_loss: 1.9113 - val_accuracy: 0.3483
Epoch 7/200
17/22 [======================>.......] - ETA: 0s - loss: 1.8802 - accuracy: 0.3824 - F1
Score:0.395
22/22 [==============================] - 0s 13ms/step - loss: 1.8680 - accuracy: 0.3907 -
val_loss: 1.8373 - val_accuracy: 0.3950
Epoch 8/200
17/22 [======================>.......] - ETA: 0s - loss: 1.8164 - accuracy: 0.4099 - F1
Score:0.3633333333333333
22/22 [==============================] - 0s 14ms/step - loss: 1.8074 - accuracy: 0.4093 -
val_loss: 1.8266 - val_accuracy: 0.3633
Epoch 9/200
16/22 [====================>.........] - ETA: 0s - loss: 1.7987 - accuracy: 0.3770 - F1
Score:0.4000000000000001
22/22 [==============================] - 0s 14ms/step - loss: 1.7884 - accuracy: 0.3736 -
val_loss: 1.7563 - val_accuracy: 0.4000
Epoch 10/200
16/22 [====================>.........] - ETA: 0s - loss: 1.7188 - accuracy: 0.4150 - F1
Score:0.41833333333333333
22/22 [==============================] - 0s 13ms/step - loss: 1.7095 - accuracy: 0.4221 -
val_loss: 1.6919 - val_accuracy: 0.4183
Epoch 11/200
16/22 [====================>.........] - ETA: 0s - loss: 1.6904 - accuracy: 0.4463 - F1
Score:0.38333333333333336
22/22 [==============================] - 0s 15ms/step - loss: 1.6643 - accuracy: 0.4736 -
val_loss: 1.7170 - val_accuracy: 0.3833
Epoch 12/200
22/22 [==============================] - ETA: 0s - loss: 1.6261 - accuracy: 0.4521 - F1
Score:0.43333333333333335
22/22 [==============================] - 0s 14ms/step - loss: 1.6261 - accuracy: 0.4521 -
val_loss: 1.5977 - val_accuracy: 0.4333
Epoch 13/200
```

```
15/22 [===================>..........] - ETA: 0s - loss: 1.5330 - accuracy: 0.5063 - F1
Score:0.515
22/22 [==============================] - 0s 14ms/step - loss: 1.5485 - accuracy: 0.4929 -
val_loss: 1.5482 - val_accuracy: 0.5150
Epoch 14/200
15/22 [===================>..........] - ETA: 0s - loss: 1.5264 - accuracy: 0.5177 - F1
Score:0.49166666666666664
22/22 [==============================] - 0s 14ms/step - loss: 1.5302 - accuracy: 0.5114 -
val_loss: 1.5269 - val_accuracy: 0.4917
Epoch 15/200
21/22 [===========================>..] - ETA: 0s - loss: 1.4860 - accuracy: 0.5156 - F1
Score:0.5083333333333333
22/22 [==============================] - 0s 15ms/step - loss: 1.4775 - accuracy: 0.5221 -
val_loss: 1.4862 - val_accuracy: 0.5083
Epoch 16/200
22/22 [==============================] - ETA: 0s - loss: 1.4406 - accuracy: 0.5186 - F1
Score:0.4866666666666667
22/22 [==============================] - 0s 14ms/step - loss: 1.4406 - accuracy: 0.5186 -
val_loss: 1.5341 - val_accuracy: 0.4867
Epoch 17/200
22/22 [==============================] - ETA: 0s - loss: 1.4134 - accuracy: 0.5436 - F1
Score:0.5333333333333333
22/22 [==============================] - 0s 14ms/step - loss: 1.4134 - accuracy: 0.5436 -
val_loss: 1.3913 - val_accuracy: 0.5333
Epoch 18/200
22/22 [==============================] - ETA: 0s - loss: 1.3734 - accuracy: 0.5736 - F1
Score:0.5616666666666666
22/22 [==============================] - 0s 15ms/step - loss: 1.3734 - accuracy: 0.5736 -
val_loss: 1.3967 - val_accuracy: 0.5617
Epoch 19/200
16/22 [====================>.........] - ETA: 0s - loss: 1.3228 - accuracy: 0.5918 - F1
Score:0.5666666666666667
22/22 [==============================] - 0s 13ms/step - loss: 1.3312 - accuracy: 0.5900 -
val_loss: 1.3683 - val_accuracy: 0.5667
Epoch 20/200
22/22 [==============================] - ETA: 0s - loss: 1.3023 - accuracy: 0.5929 - F1
Score:0.5733333333333334
22/22 [==============================] - 0s 14ms/step - loss: 1.3023 - accuracy: 0.5929 -
val_loss: 1.3430 - val_accuracy: 0.5733
Epoch 21/200
21/22 [===========================>..] - ETA: 0s - loss: 1.2799 - accuracy: 0.5841 - F1
Score:0.5583333333333333
22/22 [==============================] - 0s 15ms/step - loss: 1.2836 - accuracy: 0.5843 -
val_loss: 1.3322 - val_accuracy: 0.5583
Epoch 22/200
15/22 [===================>..........] - ETA: 0s - loss: 1.2773 - accuracy: 0.5771 - F1
Score:0.545
22/22 [==============================] - 0s 14ms/step - loss: 1.2806 - accuracy: 0.5764 -
val_loss: 1.3500 - val_accuracy: 0.5450
Epoch 23/200
17/22 [======================>.......] - ETA: 0s - loss: 1.2568 - accuracy: 0.6002 - F1
Score:0.5416666666666666
22/22 [==============================] - 0s 14ms/step - loss: 1.2400 - accuracy: 0.6029 -
val_loss: 1.3204 - val_accuracy: 0.5417
Epoch 24/200
22/22 [==============================] - ETA: 0s - loss: 1.2236 - accuracy: 0.6057 - F1
Score:0.6016666666666667
22/22 [==============================] - 0s 14ms/step - loss: 1.2236 - accuracy: 0.6057 -
val_loss: 1.2393 - val_accuracy: 0.6017
Epoch 25/200
22/22 [==============================] - ETA: 0s - loss: 1.1991 - accuracy: 0.6186 - F1
Score:0.5716666666666667
22/22 [==============================] - 0s 14ms/step - loss: 1.1991 - accuracy: 0.6186 -
val_loss: 1.2486 - val_accuracy: 0.5717
Epoch 26/200
17/22 [======================>.......] - ETA: 0s - loss: 1.1781 - accuracy: 0.6314 - F1
Score:0.6066666666666667
22/22 [==============================] - 0s 14ms/step - loss: 1.1657 - accuracy: 0.6321 -
val_loss: 1.1936 - val_accuracy: 0.6067
Epoch 27/200
22/22 [==============================] - ETA: 0s - loss: 1.1399 - accuracy: 0.6336 - F1
Score:0.6016666666666667
22/22 [==============================] - 0s 15ms/step - loss: 1.1399 - accuracy: 0.6336 -
val_loss: 1.2234 - val_accuracy: 0.6017
Epoch 28/200
16/22 [====================>.........] - ETA: 0s - loss: 1.1382 - accuracy: 0.6309 - F1
Score:0.6283333333333333
```

```
22/22 [==============================] - 0s 13ms/step - loss: 1.1527 - accuracy: 0.6271 -
val_loss: 1.1601 - val_accuracy: 0.6283
Epoch 29/200
17/22 [======================>.......] - ETA: 0s - loss: 1.0765 - accuracy: 0.6618 - F1 Score:0.61
22/22 [==============================] - 0s 13ms/step - loss: 1.0958 - accuracy: 0.6429 -
val_loss: 1.1561 - val_accuracy: 0.6100
Epoch 30/200
16/22 [=====================>.........] - ETA: 0s - loss: 1.0572 - accuracy: 0.6592 - F1 Score:0.63
22/22 [==============================] - 0s 15ms/step - loss: 1.0795 - accuracy: 0.6500 -
val_loss: 1.1338 - val_accuracy: 0.6300
Epoch 31/200
15/22 [====================>..........] - ETA: 0s - loss: 1.0920 - accuracy: 0.6385 - F1
Score:0.5966666666666667
22/22 [==============================] - 0s 13ms/step - loss: 1.0770 - accuracy: 0.6450 -
val_loss: 1.1955 - val_accuracy: 0.5967
Epoch 32/200
17/22 [======================>.......] - ETA: 0s - loss: 1.0833 - accuracy: 0.6434 - F1 Score:0.65
22/22 [==============================] - 0s 13ms/step - loss: 1.0591 - accuracy: 0.6529 -
val_loss: 1.0991 - val_accuracy: 0.6500
Epoch 33/200
22/22 [==============================] - ETA: 0s - loss: 1.0378 - accuracy: 0.6629 - F1 Score:0.65
22/22 [==============================] - 0s 14ms/step - loss: 1.0378 - accuracy: 0.6629 -
val_loss: 1.0783 - val_accuracy: 0.6500
Epoch 34/200
21/22 [============================>..] - ETA: 0s - loss: 1.0215 - accuracy: 0.6525 - F1
Score:0.6616666666666666
22/22 [==============================] - 0s 15ms/step - loss: 1.0267 - accuracy: 0.6507 -
val_loss: 1.0826 - val_accuracy: 0.6617
Epoch 35/200
17/22 [======================>.......] - ETA: 0s - loss: 1.0121 - accuracy: 0.6829 - F1
Score:0.6283333333333333
22/22 [==============================] - 0s 14ms/step - loss: 1.0152 - accuracy: 0.6743 -
val_loss: 1.0883 - val_accuracy: 0.6283
Epoch 36/200
21/22 [============================>..] - ETA: 0s - loss: 0.9914 - accuracy: 0.6749 - F1
Score:0.6733333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.9840 - accuracy: 0.6764 -
val_loss: 1.0283 - val_accuracy: 0.6733
Epoch 37/200
21/22 [============================>..] - ETA: 0s - loss: 0.9494 - accuracy: 0.6868 - F1
Score:0.6583333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.9513 - accuracy: 0.6850 -
val_loss: 1.0615 - val_accuracy: 0.6583
Epoch 38/200
17/22 [======================>.......] - ETA: 0s - loss: 0.9479 - accuracy: 0.6875 - F1
Score:0.6333333333333333
22/22 [==============================] - 0s 13ms/step - loss: 0.9426 - accuracy: 0.6929 -
val_loss: 1.0802 - val_accuracy: 0.6333
Epoch 39/200
17/22 [======================>.......] - ETA: 0s - loss: 0.9639 - accuracy: 0.6737 - F1
Score:0.6666666666666666
22/22 [==============================] - 0s 14ms/step - loss: 0.9718 - accuracy: 0.6714 -
val_loss: 1.0176 - val_accuracy: 0.6667
Epoch 40/200
22/22 [==============================] - ETA: 0s - loss: 0.9152 - accuracy: 0.7050 - F1
Score:0.6816666666666666
22/22 [==============================] - 0s 14ms/step - loss: 0.9152 - accuracy: 0.7050 -
val_loss: 0.9725 - val_accuracy: 0.6817
Epoch 41/200
17/22 [======================>.......] - ETA: 0s - loss: 0.8776 - accuracy: 0.7123 - F1
Score:0.6933333333333334
22/22 [==============================] - 0s 13ms/step - loss: 0.8959 - accuracy: 0.7029 -
val_loss: 0.9798 - val_accuracy: 0.6933
Epoch 42/200
17/22 [======================>.......] - ETA: 0s - loss: 0.8604 - accuracy: 0.7252 - F1
Score:0.6516666666666666
22/22 [==============================] - 0s 14ms/step - loss: 0.8905 - accuracy: 0.7129 -
val_loss: 1.0238 - val_accuracy: 0.6517
Epoch 43/200
16/22 [=====================>.........] - ETA: 0s - loss: 0.9380 - accuracy: 0.6992 - F1
Score:0.685
22/22 [==============================] - 0s 14ms/step - loss: 0.9077 - accuracy: 0.7121 -
val_loss: 0.9808 - val_accuracy: 0.6850
Epoch 44/200
22/22 [==============================] - ETA: 0s - loss: 0.8800 - accuracy: 0.7200 - F1 Score:0.7
22/22 [==============================] - 0s 14ms/step - loss: 0.8800 - accuracy: 0.7200 -
val_loss: 0.9201 - val_accuracy: 0.7000
```

```
Epoch 45/200
22/22 [==============================] - ETA: 0s - loss: 0.8790 - accuracy: 0.7107 - F1
Score:0.6833333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.8790 - accuracy: 0.7107 -
val_loss: 0.9541 - val_accuracy: 0.6833
Epoch 46/200
16/22 [====================>.........] - ETA: 0s - loss: 0.8592 - accuracy: 0.7266 - F1
Score:0.695
22/22 [==============================] - 0s 14ms/step - loss: 0.8543 - accuracy: 0.7286 -
val_loss: 0.9407 - val_accuracy: 0.6950
Epoch 47/200
16/22 [====================>.........] - ETA: 0s - loss: 0.8540 - accuracy: 0.7139 - F1 Score:0.68
22/22 [==============================] - 0s 13ms/step - loss: 0.8364 - accuracy: 0.7257 -
val_loss: 0.9618 - val_accuracy: 0.6800
Epoch 48/200
22/22 [==============================] - ETA: 0s - loss: 0.8364 - accuracy: 0.7336 - F1
Score:0.7133333333333335
22/22 [==============================] - 0s 14ms/step - loss: 0.8364 - accuracy: 0.7336 -
val_loss: 0.9053 - val_accuracy: 0.7133
Epoch 49/200
22/22 [==============================] - ETA: 0s - loss: 0.8199 - accuracy: 0.7486 - F1
Score:0.7116666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.8199 - accuracy: 0.7486 -
val_loss: 0.8973 - val_accuracy: 0.7117
Epoch 50/200
15/22 [===================>..........] - ETA: 0s - loss: 0.7914 - accuracy: 0.7479 - F1
Score:0.695
22/22 [==============================] - 0s 13ms/step - loss: 0.8108 - accuracy: 0.7436 -
val_loss: 0.9516 - val_accuracy: 0.6950
Epoch 51/200
17/22 [======================>.......] - ETA: 0s - loss: 0.7906 - accuracy: 0.7454 - F1
Score:0.7116666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.7878 - accuracy: 0.7493 -
val_loss: 0.8603 - val_accuracy: 0.7117
Epoch 52/200
15/22 [===================>..........] - ETA: 0s - loss: 0.7926 - accuracy: 0.7344 - F1
Score:0.7166666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.7906 - accuracy: 0.7471 -
val_loss: 0.8803 - val_accuracy: 0.7167
Epoch 53/200
22/22 [==============================] - ETA: 0s - loss: 0.7665 - accuracy: 0.7614 - F1
Score:0.7233333333333334
22/22 [==============================] - 0s 14ms/step - loss: 0.7665 - accuracy: 0.7614 -
val_loss: 0.8894 - val_accuracy: 0.7233
Epoch 54/200
16/22 [====================>.........] - ETA: 0s - loss: 0.7674 - accuracy: 0.7627 - F1
Score:0.7233333333333334
22/22 [==============================] - 0s 13ms/step - loss: 0.7733 - accuracy: 0.7550 -
val_loss: 0.8976 - val_accuracy: 0.7233
Epoch 55/200
22/22 [==============================] - ETA: 0s - loss: 0.7608 - accuracy: 0.7564 - F1
Score:0.7299999999999999
22/22 [==============================] - 0s 15ms/step - loss: 0.7608 - accuracy: 0.7564 -
val_loss: 0.8792 - val_accuracy: 0.7300
Epoch 56/200
22/22 [==============================] - ETA: 0s - loss: 0.7847 - accuracy: 0.7414 - F1
Score:0.7299999999999999
22/22 [==============================] - 0s 14ms/step - loss: 0.7847 - accuracy: 0.7414 -
val_loss: 0.8392 - val_accuracy: 0.7300
Epoch 57/200
17/22 [======================>.......] - ETA: 0s - loss: 0.7776 - accuracy: 0.7436 - F1 Score:0.7
22/22 [==============================] - 0s 14ms/step - loss: 0.7498 - accuracy: 0.7579 -
val_loss: 0.9012 - val_accuracy: 0.7000
Epoch 58/200
21/22 [===========================>..] - ETA: 0s - loss: 0.7279 - accuracy: 0.7656 - F1
Score:0.7016666666666667
22/22 [==============================] - 0s 15ms/step - loss: 0.7276 - accuracy: 0.7636 -
val_loss: 0.9031 - val_accuracy: 0.7017
Epoch 59/200
22/22 [==============================] - ETA: 0s - loss: 0.7334 - accuracy: 0.7621 - F1
Score:0.755
22/22 [==============================] - 0s 14ms/step - loss: 0.7334 - accuracy: 0.7621 -
val_loss: 0.8029 - val_accuracy: 0.7550
Epoch 60/200
22/22 [==============================] - ETA: 0s - loss: 0.7478 - accuracy: 0.7564 - F1
Score:0.7366666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.7478 - accuracy: 0.7564 -
```

```
val_loss: 0.8263 - val_accuracy: 0.7367
Epoch 61/200
21/22 [===========================>..] - ETA: 0s - loss: 0.7257 - accuracy: 0.7582 - F1
Score:0.7333333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.7174 - accuracy: 0.7621 -
val_loss: 0.8080 - val_accuracy: 0.7333
Epoch 62/200
22/22 [==============================] - ETA: 0s - loss: 0.7259 - accuracy: 0.7700 - F1
Score:0.7433333333333333
22/22 [==============================] - 0s 15ms/step - loss: 0.7259 - accuracy: 0.7700 -
val_loss: 0.8026 - val_accuracy: 0.7433
Epoch 63/200
17/22 [======================>.......] - ETA: 0s - loss: 0.6739 - accuracy: 0.7831 - F1
Score:0.715
22/22 [==============================] - 0s 14ms/step - loss: 0.6876 - accuracy: 0.7786 -
val_loss: 0.8694 - val_accuracy: 0.7150
Epoch 64/200
22/22 [==============================] - ETA: 0s - loss: 0.7348 - accuracy: 0.7707 - F1
Score:0.7333333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.7348 - accuracy: 0.7707 -
val_loss: 0.8108 - val_accuracy: 0.7333
Epoch 65/200
22/22 [==============================] - ETA: 0s - loss: 0.7139 - accuracy: 0.7786 - F1
Score:0.7283333333333334
22/22 [==============================] - 0s 14ms/step - loss: 0.7139 - accuracy: 0.7786 -
val_loss: 0.8296 - val_accuracy: 0.7283
Epoch 66/200
17/22 [======================>.......] - ETA: 0s - loss: 0.7414 - accuracy: 0.7509 - F1
Score:0.7299999999999999
22/22 [==============================] - 0s 13ms/step - loss: 0.7187 - accuracy: 0.7614 -
val_loss: 0.8054 - val_accuracy: 0.7300
Epoch 67/200
22/22 [==============================] - ETA: 0s - loss: 0.6976 - accuracy: 0.7729 - F1 Score:0.75
22/22 [==============================] - 0s 15ms/step - loss: 0.6976 - accuracy: 0.7729 -
val_loss: 0.7844 - val_accuracy: 0.7500
Epoch 68/200
22/22 [==============================] - ETA: 0s - loss: 0.6591 - accuracy: 0.8014 - F1
Score:0.7416666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.6591 - accuracy: 0.8014 -
val_loss: 0.7913 - val_accuracy: 0.7417
Epoch 69/200
16/22 [====================>.........] - ETA: 0s - loss: 0.6682 - accuracy: 0.7852 - F1
Score:0.7183333333333334
22/22 [==============================] - 0s 13ms/step - loss: 0.6800 - accuracy: 0.7800 -
val_loss: 0.9000 - val_accuracy: 0.7183
Epoch 70/200
22/22 [==============================] - ETA: 0s - loss: 0.6667 - accuracy: 0.7843 - F1
Score:0.7533333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.6667 - accuracy: 0.7843 -
val_loss: 0.7660 - val_accuracy: 0.7533
Epoch 71/200
15/22 [===================>..........] - ETA: 0s - loss: 0.6470 - accuracy: 0.7948 - F1
Score:0.7416666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.6510 - accuracy: 0.7964 -
val_loss: 0.7960 - val_accuracy: 0.7417
Epoch 72/200
22/22 [==============================] - ETA: 0s - loss: 0.6566 - accuracy: 0.7943 - F1
Score:0.7566666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.6566 - accuracy: 0.7943 -
val_loss: 0.7713 - val_accuracy: 0.7567
Epoch 73/200
22/22 [==============================] - ETA: 0s - loss: 0.6511 - accuracy: 0.7893 - F1
Score:0.7583333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.6511 - accuracy: 0.7893 -
val_loss: 0.8020 - val_accuracy: 0.7583
Epoch 74/200
16/22 [====================>.........] - ETA: 0s - loss: 0.6362 - accuracy: 0.7998 - F1
Score:0.7716666666666666
22/22 [==============================] - 0s 14ms/step - loss: 0.6418 - accuracy: 0.7929 -
val_loss: 0.7408 - val_accuracy: 0.7717
Epoch 75/200
22/22 [==============================] - ETA: 0s - loss: 0.6446 - accuracy: 0.7950 - F1
Score:0.7683333333333333
22/22 [==============================] - 0s 15ms/step - loss: 0.6446 - accuracy: 0.7950 -
val_loss: 0.7373 - val_accuracy: 0.7683
Epoch 76/200
15/22 [===================>..........] - ETA: 0s - loss: 0.5907 - accuracy: 0.8177 - F1
```

```
15/22 [===================>..........] - ETA: 0s - loss: 0.5907 - accuracy: 0.8177 - F1
Score:0.7683333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.6137 - accuracy: 0.8007 -
val_loss: 0.7295 - val_accuracy: 0.7683
Epoch 77/200
22/22 [==============================] - ETA: 0s - loss: 0.6096 - accuracy: 0.8064 - F1 Score:0.75
22/22 [==============================] - 0s 15ms/step - loss: 0.6096 - accuracy: 0.8064 -
val_loss: 0.7626 - val_accuracy: 0.7500
Epoch 78/200
21/22 [===========================>..] - ETA: 0s - loss: 0.6031 - accuracy: 0.8080 - F1
Score:0.7566666666666667
22/22 [==============================] - 0s 15ms/step - loss: 0.6100 - accuracy: 0.8071 -
val_loss: 0.7462 - val_accuracy: 0.7567
Epoch 79/200
16/22 [====================>.........] - ETA: 0s - loss: 0.6059 - accuracy: 0.8135 - F1 Score:0.75
22/22 [==============================] - 0s 13ms/step - loss: 0.6052 - accuracy: 0.8129 -
val_loss: 0.7395 - val_accuracy: 0.7500
Epoch 80/200
22/22 [==============================] - ETA: 0s - loss: 0.6166 - accuracy: 0.8021 - F1
Score:0.7699999999999999
22/22 [==============================] - 0s 14ms/step - loss: 0.6166 - accuracy: 0.8021 -
val_loss: 0.7263 - val_accuracy: 0.7700
Epoch 81/200
22/22 [==============================] - ETA: 0s - loss: 0.6083 - accuracy: 0.8093 - F1
Score:0.745
22/22 [==============================] - 0s 15ms/step - loss: 0.6083 - accuracy: 0.8093 -
val_loss: 0.7531 - val_accuracy: 0.7450
Epoch 82/200
17/22 [======================>.......] - ETA: 0s - loss: 0.6173 - accuracy: 0.8015 - F1
Score:0.7533333333333333
22/22 [==============================] - 0s 13ms/step - loss: 0.6007 - accuracy: 0.8057 -
val_loss: 0.7618 - val_accuracy: 0.7533
Epoch 83/200
22/22 [==============================] - ETA: 0s - loss: 0.6042 - accuracy: 0.7957 - F1 Score:0.76
22/22 [==============================] - 0s 15ms/step - loss: 0.6042 - accuracy: 0.7957 -
val_loss: 0.7505 - val_accuracy: 0.7600
Epoch 84/200
22/22 [==============================] - ETA: 0s - loss: 0.5733 - accuracy: 0.8157 - F1
Score:0.7666666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.5733 - accuracy: 0.8157 -
val_loss: 0.7177 - val_accuracy: 0.7667
Epoch 85/200
15/22 [===================>..........] - ETA: 0s - loss: 0.5694 - accuracy: 0.8156 - F1
Score:0.7633333333333333
22/22 [==============================] - 0s 13ms/step - loss: 0.5907 - accuracy: 0.8043 -
val_loss: 0.7209 - val_accuracy: 0.7633
Epoch 86/200
20/22 [==========================>...] - ETA: 0s - loss: 0.5797 - accuracy: 0.8156 - F1
Score:0.7566666666666667
22/22 [==============================] - 0s 15ms/step - loss: 0.5835 - accuracy: 0.8157 -
val_loss: 0.7484 - val_accuracy: 0.7567
Epoch 87/200
22/22 [==============================] - ETA: 0s - loss: 0.5976 - accuracy: 0.8079 - F1 Score:0.75
22/22 [==============================] - 0s 15ms/step - loss: 0.5976 - accuracy: 0.8079 -
val_loss: 0.7484 - val_accuracy: 0.7500
Epoch 88/200
17/22 [======================>.......] - ETA: 0s - loss: 0.5928 - accuracy: 0.8088 - F1
Score:0.7783333333333333
22/22 [==============================] - 0s 13ms/step - loss: 0.5868 - accuracy: 0.8086 -
val_loss: 0.7062 - val_accuracy: 0.7783
Epoch 89/200
22/22 [==============================] - ETA: 0s - loss: 0.5610 - accuracy: 0.8236 - F1
Score:0.7816666666666666
22/22 [==============================] - 0s 14ms/step - loss: 0.5610 - accuracy: 0.8236 -
val_loss: 0.6986 - val_accuracy: 0.7817
Epoch 90/200
21/22 [===========================>..] - ETA: 0s - loss: 0.5679 - accuracy: 0.8251 - F1 Score:0.74
22/22 [==============================] - 0s 14ms/step - loss: 0.5637 - accuracy: 0.8257 -
val_loss: 0.8114 - val_accuracy: 0.7400
Epoch 91/200
16/22 [====================>.........] - ETA: 0s - loss: 0.5634 - accuracy: 0.8301 - F1
Score:0.7916666666666666
22/22 [==============================] - 0s 13ms/step - loss: 0.5876 - accuracy: 0.8236 -
val_loss: 0.7071 - val_accuracy: 0.7917
Epoch 92/200
22/22 [==============================] - ETA: 0s - loss: 0.5601 - accuracy: 0.8207 - F1
Score:0.775
22/22 [==============================] - 0s 14ms/step - loss: 0.5601 - accuracy: 0.8207 -
```

```
22/22 [==============================] - 0s 14ms/step - loss: 0.5601 - accuracy: 0.8207 -
val_loss: 0.7097 - val_accuracy: 0.7750
Epoch 93/200
15/22 [===================>..........] - ETA: 0s - loss: 0.5793 - accuracy: 0.8042 - F1
Score:0.7766666666666666
22/22 [==============================] - 0s 13ms/step - loss: 0.5899 - accuracy: 0.8064 -
val_loss: 0.7197 - val_accuracy: 0.7767
Epoch 94/200
22/22 [==============================] - ETA: 0s - loss: 0.5942 - accuracy: 0.8107 - F1
Score:0.7883333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.5942 - accuracy: 0.8107 -
val_loss: 0.6643 - val_accuracy: 0.7883
Epoch 95/200
17/22 [======================>.......] - ETA: 0s - loss: 0.5626 - accuracy: 0.7996 - F1
Score:0.7866666666666666
22/22 [==============================] - 0s 14ms/step - loss: 0.5402 - accuracy: 0.8157 -
val_loss: 0.6832 - val_accuracy: 0.7867
Epoch 96/200
20/22 [==========================>...] - ETA: 0s - loss: 0.5316 - accuracy: 0.8211 - F1 Score:0.78
22/22 [==============================] - 0s 15ms/step - loss: 0.5374 - accuracy: 0.8207 -
val_loss: 0.6870 - val_accuracy: 0.7800
Epoch 97/200
16/22 [====================>.........] - ETA: 0s - loss: 0.5719 - accuracy: 0.8213 - F1
Score:0.7716666666666666
22/22 [==============================] - 0s 14ms/step - loss: 0.5522 - accuracy: 0.8293 -
val_loss: 0.7038 - val_accuracy: 0.7717
Epoch 98/200
22/22 [==============================] - ETA: 0s - loss: 0.5359 - accuracy: 0.8250 - F1
Score:0.7666666666666667
22/22 [==============================] - 0s 14ms/step - loss: 0.5359 - accuracy: 0.8250 -
val_loss: 0.7243 - val_accuracy: 0.7667
Epoch 99/200
16/22 [====================>.........] - ETA: 0s - loss: 0.5332 - accuracy: 0.8291 - F1
Score:0.7766666666666666
22/22 [==============================] - 0s 14ms/step - loss: 0.5290 - accuracy: 0.8279 -
val_loss: 0.7146 - val_accuracy: 0.7767
Epoch 100/200
22/22 [==============================] - ETA: 0s - loss: 0.5156 - accuracy: 0.8386 - F1
Score:0.7699999999999999
22/22 [==============================] - 0s 14ms/step - loss: 0.5156 - accuracy: 0.8386 -
val_loss: 0.6859 - val_accuracy: 0.7700
Epoch 101/200
16/22 [====================>.........] - ETA: 0s - loss: 0.5354 - accuracy: 0.8213 - F1
Score:0.7933333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.4963 - accuracy: 0.8414 -
val_loss: 0.6561 - val_accuracy: 0.7933
Epoch 102/200
21/22 [===========================>..] - ETA: 0s - loss: 0.5059 - accuracy: 0.8363 - F1
Score:0.7766666666666666
22/22 [==============================] - 0s 15ms/step - loss: 0.5097 - accuracy: 0.8371 -
val_loss: 0.7191 - val_accuracy: 0.7767
Epoch 103/200
17/22 [======================>.......] - ETA: 0s - loss: 0.5166 - accuracy: 0.8401 - F1
Score:0.7883333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.5242 - accuracy: 0.8364 -
val_loss: 0.6668 - val_accuracy: 0.7883
Epoch 104/200
16/22 [====================>.........] - ETA: 0s - loss: 0.4998 - accuracy: 0.8408 - F1
Score:0.7783333333333333
22/22 [==============================] - 0s 14ms/step - loss: 0.5079 - accuracy: 0.8371 -
val_loss: 0.6904 - val_accuracy: 0.7783
Epoch 105/200
21/22 [===========================>..] - ETA: 0s - loss: 0.5207 - accuracy: 0.8363 - F1 Score:0.78
22/22 [==============================] - 0s 15ms/step - loss: 0.5178 - accuracy: 0.8379 -
val_loss: 0.6660 - val_accuracy: 0.7800
Epoch 106/200
21/22 [===========================>..] - ETA: 0s - loss: 0.5595 - accuracy: 0.8207 - F1
Score:0.785
22/22 [==============================] - 0s 14ms/step - loss: 0.5573 - accuracy: 0.8229 -
val_loss: 0.6683 - val_accuracy: 0.7850
Epoch 107/200
22/22 [==============================] - ETA: 0s - loss: 0.5657 - accuracy: 0.8157 - F1
Score:0.7816666666666666
22/22 [==============================] - 0s 15ms/step - loss: 0.5657 - accuracy: 0.8157 -
val_loss: 0.6913 - val_accuracy: 0.7817
Epoch 108/200
22/22 [==============================] - ETA: 0s - loss: 0.5317 - accuracy: 0.8229 - F1
Score:0.785
```

```
Score:0.785
22/22 [==============================] - 0s 14ms/step - loss: 0.5317 - accuracy: 0.8229 -
val_loss: 0.6698 - val_accuracy: 0.7850
Epoch 109/200
22/22 [==============================] - ETA: 0s - loss: 0.4861 - accuracy: 0.8386 - F1
Score:0.8033333333333333
Stopping training,since we reach 0.8033333333333333 F1 Score.
22/22 [==============================] - 0s 15ms/step - loss: 0.4861 - accuracy: 0.8386 -
val_loss: 0.6346 - val_accuracy: 0.8033
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f74b6d076a0>
```

```
time: 37.7 s
```

In [ ]:

```
%tensorboard --logdir 'model2'
```

```
time: 3.08 s
```

### 3. data augmentation

```
Till now we have done with 2000 samples only. It is very less data. We are giving the
process of generating augmented data below.

There are two types of augmentation:
1. time stretching - Time stretching either increases or decreases the length of the file.
For time stretching we move the file 30% faster or slower
2. pitch shifting - pitch shifting moves the frequencies higher or lower. For pitch
shifting we shift up or down one half-step.
```

In [8]:

```
## generating augmented data.
def generate_augmented_data(file_path):
    augmented_data = []
    samples = load_wav(file_path,get_duration=False)
    for time_value in [0.7, 1, 1.3]:
        for pitch_value in [-1, 0, 1]:
            time_stretch_data = librosa.effects.time_stretch(samples, rate=time_value)
            final_data = librosa.effects.pitch_shift(time_stretch_data, sr=sample_rate, n_steps=pit
ch_value)
            augmented_data.append(final_data)
    return augmented_data
```

As discussed above, for one data point, we will get 9 augmented data points.
We have 2000 data points(train plus test) so, after augmentation we will get 18000 ( train - 12600, test - 5400).

do the above steps i.e training with raw data and spectrogram data with augmentation.

In [ ]:

```
aug_data=[]
aug_labels=[]
for i in range(0,len(df_audio)):
  temp_path = df_audio.iloc[i].path
  aug_temp = generate_augmented_data(temp_path)
  label = df_audio.iloc[i].label
  for individual_sample in aug_temp:
    aug_data.append(individual_sample)
    aug_labels.append(label)
```

In [ ]:

```
aug_data=np.array(aug_data)
```

```
aug_labels=np.array(aug_labels)
```

In [ ]:

```
#split the data into train and validation and save in X_train, X_test, y_train, y_test


X_train, X_test, y_train, y_test = train_test_split(aug_data, aug_labels, test_size=0.30, random_st
ate=45,stratify=aug_labels)
```

**Model 3:**

In [ ]:

```
X_train_pad_seq=pad_sequences(X_train,padding='post',truncating='post',value=0.00,dtype='float32')
X_train_pad_seq.shape
```

Out[ ]:

```
(12600, 72192)
```

In [ ]:

```
X_test_pad_seq=pad_sequences(X_test,padding='post',maxlen=72192,truncating='post',value=0.00,dtype=
'float32')
```

In [ ]:

```
## all the X_train_pad_seq, X_test_pad_seq, X_train_mask, X_test_mask will be numpy arrays mask ve
ctor dtype must be bool.

def masking(data_pre):
  X_mask=np.empty((0,17640), dtype='bool')
  for i,e in enumerate(data_pre):
    #pre_ele=data_pre["raw_data"][i]
    len_pre=len(e)
    diff=17640-len_pre
    if (diff>0):
      mask_vec=np.array([1]*(len_pre)+[0]*diff,dtype='bool')
    elif (diff==0):
      mask_vec=np.array([1]*17640)
    X_mask = np.append(X_mask, [mask_vec], axis=0)
  return X_mask

X_train_mask=masking(X_train)
X_test_mask=masking(X_test)
```

In [ ]:

```
#model 3 architecture

input_shape=X_train_pad_seq.shape
mask_shape=X_train_mask.shape

lstm_input = Input(shape=(input_shape[1],1), dtype='float32')
mask_input = Input(shape=(mask_shape[1]),dtype='bool')

x = LSTM(16  , return_sequences=False)(lstm_input,mask=mask_input)
x = Dense(32, activation="relu")(x)
output = Dense(10, activation="softmax")(x)

model_3 = Model(inputs=[lstm_input,mask_input],outputs=[output])

model_3.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

model_3.summary()
```

```
Model: "functional_7"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
```

```
=====================================================================================================
input_7 (InputLayer)            [(None, 72192, 1)]   0
_____________________________________________________________________________________________________
input_8 (InputLayer)            [(None, 17640)]      0
_____________________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 16)           1152        input_7[0][0]
                                                                 input_8[0][0]
_____________________________________________________________________________________________________
dense_6 (Dense)                 (None, 32)           544         lstm_3[0][0]
_____________________________________________________________________________________________________
dense_7 (Dense)                 (None, 10)           330         dense_6[0][0]
=====================================================================================================
Total params: 2,026
Trainable params: 2,026
Non-trainable params: 0
_____________________________________________________________________________________________________
time: 849 ms
```

In [ ]:

```
y_train= y_train.astype(int)
y_test= y_test.astype(int)
```

```
time: 9.26 ms
```

In [ ]:

```
!mkdir model__3
```

```
time: 226 ms
```

In [ ]:

```
tensorboard_callback = TensorBoard(log_dir='model__3',histogram_freq=1)


model_3.fit([X_train_pad_seq,X_train_mask], y_train, batch_size=64, epochs=10, verbose=1,
           validation_data=([X_test_pad_seq,X_test_mask], y_test),
           callbacks=[tensorboard_callback,CustomCallback(threshold=0.15,validation_data=([X_test_p
ad_seq,X_test_mask], y_test))]
                                     )
```

```
Epoch 1/10
  2/197 [..............................] - ETA: 7:02 - loss: 2.3028 - accuracy:
0.0469WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time
(batch time: 0.9607s vs `on_train_batch_end` time: 3.3738s). Check your callbacks.
197/197 [==============================] - ETA: 0s - loss: 2.3028 - accuracy: 0.0999 - F1
Score:0.10000000000000002
197/197 [==============================] - 232s 1s/step - loss: 2.3028 - accuracy: 0.0999 - val_lo
ss: 2.3026 - val_accuracy: 0.1000
Epoch 2/10
197/197 [==============================] - ETA: 0s - loss: 2.3024 - accuracy: 0.0989 - F1
Score:0.09722222222222222
197/197 [==============================] - 226s 1s/step - loss: 2.3024 - accuracy: 0.0989 - val_lo
ss: 2.3021 - val_accuracy: 0.0972
Epoch 3/10
197/197 [==============================] - ETA: 0s - loss: 2.3111 - accuracy: 0.0996 - F1
Score:0.10185185185185185
197/197 [==============================] - 225s 1s/step - loss: 2.3111 - accuracy: 0.0996 - val_lo
ss: 2.3290 - val_accuracy: 0.1019
Epoch 4/10
197/197 [==============================] - ETA: 0s - loss: 2.3070 - accuracy: 0.0987 - F1
Score:0.09833333333333333
197/197 [==============================] - 220s 1s/step - loss: 2.3070 - accuracy: 0.0987 - val_lo
ss: 2.3031 - val_accuracy: 0.0983
Epoch 5/10
197/197 [==============================] - ETA: 0s - loss: 2.3023 - accuracy: 0.0993 - F1
Score:0.10629629629629629
197/197 [==============================] - 216s 1s/step - loss: 2.3023 - accuracy: 0.0993 - val_lo
ss: 2.3020 - val_accuracy: 0.1063
Epoch 6/10
197/197 [==============================] - ETA: 0s - loss: 2.3013 - accuracy: 0.1056 - F1
```

```
Score:0.10592592592592592
197/197 [==============================] - 214s 1s/step - loss: 2.3013 - accuracy: 0.1056 - val_lo
ss: 2.3013 - val_accuracy: 0.1059
Epoch 7/10
197/197 [==============================] - ETA: 0s - loss: 2.3004 - accuracy: 0.1050 - F1
Score:0.11370370370370371
197/197 [==============================] - 212s 1s/step - loss: 2.3004 - accuracy: 0.1050 - val_lo
ss: 2.3007 - val_accuracy: 0.1137
Epoch 8/10
197/197 [==============================] - ETA: 0s - loss: 2.2995 - accuracy: 0.1112 - F1
Score:0.10629629629629629
197/197 [==============================] - 210s 1s/step - loss: 2.2995 - accuracy: 0.1112 - val_lo
ss: 2.3001 - val_accuracy: 0.1063
Epoch 9/10
197/197 [==============================] - ETA: 0s - loss: 2.2986 - accuracy: 0.0998 - F1
Score:0.10481481481481483
197/197 [==============================] - 210s 1s/step - loss: 2.2986 - accuracy: 0.0998 - val_lo
ss: 2.2992 - val_accuracy: 0.1048
Epoch 10/10
197/197 [==============================] - ETA: 0s - loss: 2.2970 - accuracy: 0.0956 - F1
Score:0.11462962962962962
197/197 [==============================] - 210s 1s/step - loss: 2.2970 - accuracy: 0.0956 - val_lo
ss: 2.2982 - val_accuracy: 0.1146
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7fd344c4a7b8>
```

```
time: 36min 25s
```

In [ ]:

```
%tensorboard --logdir 'model__3'
```

```
time: 3.12 s
```

**Model 4:**

In [11]:

```
aug_data=[]
aug_labels=[]
for i in range(0,len(df_audio)):
  temp_path = df_audio.iloc[i].path
  aug_temp = generate_augmented_data(temp_path)
  label = df_audio.iloc[i].label
  for individual_sample in aug_temp:
    aug_data.append(individual_sample)
    aug_labels.append(label)
```

In [16]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(aug_data, aug_labels, test_size=0.30, random_st
ate=45,stratify=aug_labels)
```

In [26]:

```
import tensorflow as tf
X_train_pad_seq=tf.keras.preprocessing.sequence.pad_sequences(X_train,padding='post',maxlen=17500,t
runcating='post',value=0.00,dtype='float32')
X_train_pad_seq.shape
```

Out[26]:

```
(12600, 17500)
```

In [27]:

```
X_test_pad_seq=tf.keras.preprocessing.sequence.pad_sequences(X_test,padding='post',maxlen=17500,tru
ncating='post',value=0.00,dtype='float32')
```

In [28]:

```
def convert_to_spectrogram(raw_data):
    '''converting to spectrogram'''
    spectrum = librosa.feature.melspectrogram(y=raw_data, sr=sample_rate, n_mels=64)
    logmel_spectrum = librosa.power_to_db(S=spectrum, ref=np.max)
    return logmel_spectrum
```

In [29]:

```
##use convert_to_spectrogram and convert every raw sequence in X_train_pad_seq and X_test_pad-seq.
## save those all in the X_train_spectrogram and X_test_spectrogram ( These two arrays must be num
py arrays)

def spectrogram(data):
  spectrogram = []
  for i in data:
    data_spectrogram =convert_to_spectrogram(i)
    spectrogram.append(data_spectrogram)
  return np.array(spectrogram)
```

In [30]:

```
X_traidel X_train_pad_seq ,X_test_pad_seqn_spectrogram=spectrogram(X_train_pad_seq)
X_test_spectrogram=spectrogram(X_test_pad_seq)
```

In [47]:

```
from tensorflow.keras.layers import Dropout
from keras.layers import Input, LSTM, Dense,GlobalAveragePooling1D
from keras.models import Model
import tensorflow as tf
from sklearn.metrics import f1_score
import tensorflow_addons as tfa
from tensorflow_addons.metrics import F1Score
```

In [57]:

```
## as discussed above, please write the LSTM

input_shape=X_train_spectrogram.shape
lstm_input = Input(shape=(input_shape[1],input_shape[2]), dtype='float32')
x = LSTM(256, return_sequences=True,kernel_regularizer='l2')(lstm_input)
averaged_output= (tf.math.reduce_mean(x, axis=1))

x = Dense(64, activation="relu")(averaged_output)


output = Dense(10, activation="softmax")(x)

model_4 = Model(inputs=[lstm_input],outputs=[output])

model_4.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

model_4.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, 35)]          0
_________________________________________________________________
lstm (LSTM)                  (None, 64, 256)           299008
_________________________________________________________________
tf_op_layer_Mean (TensorFlow [(None, 256)]             0
_________________________________________________________________
dense (Dense)                (None, 64)                16448
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
```

```
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 316,106
Trainable params: 316,106
Non-trainable params: 0
_________________________________________________________________
```

In [58]:

```
tf.keras.backend.clear_session()
```

In [59]:

```
!mkdir model_4
```

In [52]:

```
y_train= y_train.astype(int)

y_test= y_test.astype(int)
```

In [60]:

```
#train your model

tensorboard_callback = TensorBoard(log_dir='model_4',histogram_freq=1)


model_4.fit(X_train_spectrogram, y_train, batch_size=64, epochs=200, verbose=1,
           validation_data=(X_test_spectrogram, y_test),
           callbacks=[tensorboard_callback,CustomCallback(threshold=0.80,validation_data=(X_test_sp
ectrogram, y_test))]
                                   )
```

```
Epoch 1/200
  2/197 [..............................] - ETA: 11s - loss: 3.0992 - accuracy:
0.0859WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time
(batch time: 0.0166s vs `on_train_batch_end` time: 0.0970s). Check your callbacks.
196/197 [============================>.] - ETA: 0s - loss: 2.1262 - accuracy: 0.3076 - F1
Score:0.43166666666666664
197/197 [==============================] - 3s 17ms/step - loss: 2.1245 - accuracy: 0.3081 - val_lo
ss: 1.7500 - val_accuracy: 0.4317
Epoch 2/200
196/197 [============================>.] - ETA: 0s - loss: 1.6019 - accuracy: 0.4855 - F1
Score:0.5492592592592592
197/197 [==============================] - 2s 13ms/step - loss: 1.6003 - accuracy: 0.4863 - val_lo
ss: 1.4829 - val_accuracy: 0.5493
Epoch 3/200
194/197 [============================>.] - ETA: 0s - loss: 1.3765 - accuracy: 0.5717 - F1
Score:0.5687037037037037
197/197 [==============================] - 3s 13ms/step - loss: 1.3766 - accuracy: 0.5709 - val_lo
ss: 1.3289 - val_accuracy: 0.5687
Epoch 4/200
196/197 [============================>.] - ETA: 0s - loss: 1.2238 - accuracy: 0.6134 - F1
Score:0.6411111111111111
197/197 [==============================] - 2s 13ms/step - loss: 1.2238 - accuracy: 0.6133 - val_lo
ss: 1.1467 - val_accuracy: 0.6411
Epoch 5/200
194/197 [============================>.] - ETA: 0s - loss: 1.1368 - accuracy: 0.6401 - F1
Score:0.6268518518518519
197/197 [==============================] - 3s 13ms/step - loss: 1.1383 - accuracy: 0.6398 - val_lo
ss: 1.1361 - val_accuracy: 0.6269
Epoch 6/200
197/197 [==============================] - ETA: 0s - loss: 1.0611 - accuracy: 0.6644 - F1
Score:0.6831481481481482
197/197 [==============================] - 3s 13ms/step - loss: 1.0611 - accuracy: 0.6644 - val_lo
ss: 1.0142 - val_accuracy: 0.6831
Epoch 7/200
196/197 [============================>.] - ETA: 0s - loss: 0.9773 - accuracy: 0.6923 - F1
Score:0.6990740740740741
197/197 [==============================] - 2s 13ms/step - loss: 0.9765 - accuracy: 0.6929 - val_lo
ss: 0.9474 - val_accuracy: 0.6991
Epoch 8/200
```

```
197/197 [==============================] - ETA: 0s - loss: 0.9459 - accuracy: 0.7057 - F1
Score:0.6835185185185185
197/197 [==============================] - 2s 13ms/step - loss: 0.9459 - accuracy: 0.7057 - val_lo
ss: 1.0165 - val_accuracy: 0.6835
Epoch 9/200
196/197 [============================>.] - ETA: 0s - loss: 0.9167 - accuracy: 0.7132 - F1
Score:0.7287037037037037
197/197 [==============================] - 3s 13ms/step - loss: 0.9156 - accuracy: 0.7137 - val_lo
ss: 0.8730 - val_accuracy: 0.7287
Epoch 10/200
193/197 [============================>.] - ETA: 0s - loss: 0.9031 - accuracy: 0.7170 - F1
Score:0.7222222222222222
197/197 [==============================] - 3s 13ms/step - loss: 0.9045 - accuracy: 0.7163 - val_lo
ss: 0.8732 - val_accuracy: 0.7222
Epoch 11/200
196/197 [============================>.] - ETA: 0s - loss: 0.8651 - accuracy: 0.7249 - F1
Score:0.6627777777777778
197/197 [==============================] - 2s 13ms/step - loss: 0.8655 - accuracy: 0.7248 - val_lo
ss: 1.0577 - val_accuracy: 0.6628
Epoch 12/200
196/197 [============================>.] - ETA: 0s - loss: 0.9544 - accuracy: 0.7121 - F1
Score:0.7144444444444443
197/197 [==============================] - 3s 13ms/step - loss: 0.9544 - accuracy: 0.7120 - val_lo
ss: 0.9415 - val_accuracy: 0.7144
Epoch 13/200
194/197 [============================>.] - ETA: 0s - loss: 0.8904 - accuracy: 0.7331 - F1
Score:0.7553703703703704
197/197 [==============================] - 3s 13ms/step - loss: 0.8892 - accuracy: 0.7337 - val_lo
ss: 0.8185 - val_accuracy: 0.7554
Epoch 14/200
197/197 [==============================] - ETA: 0s - loss: 0.8478 - accuracy: 0.7403 - F1
Score:0.7635185185185186
197/197 [==============================] - 2s 13ms/step - loss: 0.8478 - accuracy: 0.7403 - val_lo
ss: 0.7939 - val_accuracy: 0.7635
Epoch 15/200
192/197 [============================>.] - ETA: 0s - loss: 0.8276 - accuracy: 0.7438 - F1
Score:0.7581481481481481
197/197 [==============================] - 3s 13ms/step - loss: 0.8259 - accuracy: 0.7444 - val_lo
ss: 0.8083 - val_accuracy: 0.7581
Epoch 16/200
194/197 [============================>.] - ETA: 0s - loss: 0.8127 - accuracy: 0.7539 - F1
Score:0.7240740740740741
197/197 [==============================] - 2s 13ms/step - loss: 0.8123 - accuracy: 0.7539 - val_lo
ss: 0.8562 - val_accuracy: 0.7241
Epoch 17/200
192/197 [============================>.] - ETA: 0s - loss: 0.7985 - accuracy: 0.7497 - F1
Score:0.762962962962963
197/197 [==============================] - 3s 13ms/step - loss: 0.7975 - accuracy: 0.7506 - val_lo
ss: 0.7578 - val_accuracy: 0.7630
Epoch 18/200
195/197 [============================>.] - ETA: 0s - loss: 0.7630 - accuracy: 0.7683 - F1
Score:0.768148148148148
197/197 [==============================] - 3s 13ms/step - loss: 0.7634 - accuracy: 0.7682 - val_lo
ss: 0.7374 - val_accuracy: 0.7681
Epoch 19/200
193/197 [============================>.] - ETA: 0s - loss: 0.7217 - accuracy: 0.7795 - F1
Score:0.7757407407407407
197/197 [==============================] - 3s 13ms/step - loss: 0.7241 - accuracy: 0.7789 - val_lo
ss: 0.7339 - val_accuracy: 0.7757
Epoch 20/200
192/197 [============================>.] - ETA: 0s - loss: 0.7059 - accuracy: 0.7816 - F1
Score:0.7155555555555555
197/197 [==============================] - 3s 13ms/step - loss: 0.7087 - accuracy: 0.7803 - val_lo
ss: 0.8871 - val_accuracy: 0.7156
Epoch 21/200
192/197 [============================>.] - ETA: 0s - loss: 0.6953 - accuracy: 0.7814 - F1
Score:0.7918518518518518
197/197 [==============================] - 2s 13ms/step - loss: 0.6948 - accuracy: 0.7817 - val_lo
ss: 0.6795 - val_accuracy: 0.7919
Epoch 22/200
194/197 [============================>.] - ETA: 0s - loss: 0.7033 - accuracy: 0.7841 - F1
Score:0.7694444444444445
197/197 [==============================] - 3s 13ms/step - loss: 0.7048 - accuracy: 0.7843 - val_lo
ss: 0.7340 - val_accuracy: 0.7694
Epoch 23/200
192/197 [============================>.] - ETA: 0s - loss: 0.7493 - accuracy: 0.7659 - F1
Score:0.797962962962963
```

```
197/197 [==============================] - 3s 13ms/step - loss: 0.7487 - accuracy: 0.7657 - val_lo
ss: 0.6819 - val_accuracy: 0.7980
Epoch 24/200
195/197 [============================>.] - ETA: 0s - loss: 0.6711 - accuracy: 0.7933 - F1
Score:0.7914814814814815
197/197 [==============================] - 3s 13ms/step - loss: 0.6710 - accuracy: 0.7936 - val_lo
ss: 0.7029 - val_accuracy: 0.7915
Epoch 25/200
195/197 [============================>.] - ETA: 0s - loss: 0.6763 - accuracy: 0.7908 - F1
Score:0.8075925925925926
Stopping training,since we reach 0.8075925925925926 F1 Score.
197/197 [==============================] - 2s 13ms/step - loss: 0.6760 - accuracy: 0.7913 - val_lo
ss: 0.6425 - val_accuracy: 0.8076
```

Out[60]:

```
<tensorflow.python.keras.callbacks.History at 0x7f24bd6dd668>
```

In [61]:

```
%tensorboard --logdir 'model_4'
```

## Model Comparison

In [62]:

```
from prettytable import PrettyTable

Model_Comparion = PrettyTable(['Model', 'Accuracy','F1-Score','val Accuracy', 'loss', 'val loss'])

Model_Comparion.add_row(['Model-1', 0.1021,0.10166666666666667,0.1017,2.3026,2.3026])
Model_Comparion.add_row(['Model-2', 0.8386,0.8033333333333333,0.8033,0.4861,0.6346])
Model_Comparion.add_row(['Model-3', 0.0956,0.11462962962962962,0.1146,2.2970,2.2982])
Model_Comparion.add_row(['Model-4', 0.7913,0.8075925925925926,0.8076,0.6760,0.6425])

print(Model_Comparion)
```

```
+---------+----------+---------------------+--------------+--------+----------+
|  Model  | Accuracy |       F1-Score      | val Accuracy |  loss  | val loss |
+---------+----------+---------------------+--------------+--------+----------+
| Model-1 |  0.1021  | 0.10166666666666667 |    0.1017    | 2.3026 |  2.3026  |
| Model-2 |  0.8386  |  0.8033333333333333 |    0.8033    | 0.4861 |  0.6346  |
| Model-3 |  0.0956  | 0.11462962962962962 |    0.1146    | 2.297  |  2.2982  |
| Model-4 |  0.7913  |  0.8075925925925926 |    0.8076    | 0.676  |  0.6425  |
+---------+----------+---------------------+--------------+--------+----------+
```

## Observations

1. In all models accuracy is almost equal to micro F1 Score.In classification tasks for which every test case is guaranteed to be assigned to exactly one class, micro-F is equivalent to accuracy.It won't be the case in multi-label classification.
2. A simple way to see this is by looking at the formulas precision=TP/(TP+FP) and recall=TP/(TP+FN).The numerators are the same, and every FN for one class is another classes's FP, which makes the denominators the same as well.
3. Model-1 , 2 and 3 took only few minutes to converge. But model-4 took more time compared with first 3 models.
4. Used regularization to avoid overfitting.

## Referrence / Source

1. Grader funtions and comments helped me a lot.
2. Slack convertions.
3. [https://stackoverflow.com/questions/37358496/is-f1-micro-the-same-as-accuracy](https://stackoverflow.com/questions/37358496/is-f1-micro-the-same-as-accuracy)