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 [1]:

```
from google.colab import files
files.upload()
```

Choose Files No file selected Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[1]:

```
{'kaggle.json': b'{"username":"devilkar","key":"a2a33b5546abd8ee8ed097013e8fe26c"}'}
```

In [2]:

```
!mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d devilkar/speechrecogonition
```

```
Downloading speechrecogonition.zip to /content
 56% 5.00M/8.93M [00:00<00:00, 48.7MB/s]
100% 8.93M/8.93M [00:00<00:00, 57.3MB/s]
```

In [3]:

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

We shared recordings.zip, please unzip those.

In [4]:

```
base_path = "/content/speechrecogonition.zip"
```

```
speech_data = zipfile.ZipFile(base_path,'r')
#Mention the file Name
speech_data.extractall("Speech_recorded_data")
speech_data.close()
```

In [5]:

```
#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"
path = "/content/Speech_recorded_data"
print(os.listdir(path))
```

```
['recordings']
```

In [6]:

```
recording_path = os.path.join(path,'recordings')
print("Total No of Audio file : {0}".format(len(os.listdir(recording_path))))
```

```
Total No of Audio file : 2000
```

In [7]:

```
recording_path
```

Out[7]:

```
'/content/Speech_recorded_data/recordings'
```

In [8]:

```
print(os.listdir(recording_path)[10].split("_"))
```

```
['7', 'yweweler', '35.wav']
```

In [9]:

```
all_files = []
for path in os.listdir(recording_path):
    all_files.append(recording_path + "/"+path)
```

Grader function 1

In [10]:

```
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[10]:

```
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

Exploring the sound dataset

In [11]:

```
!git clone https://github.com/AllenDowney/ThinkDSP.git
```

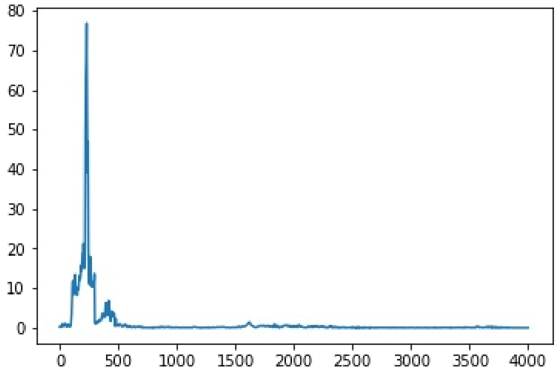
```
Cloning into 'ThinkDSP'...
remote: Enumerating objects: 2421, done.
remote: Total 2421 (delta 0), reused 0 (delta 0), pack-reused 2421
Receiving objects: 100% (2421/2421), 207.80 MiB | 6.92 MiB/s, done.
Resolving deltas: 100% (1320/1320), done.
```

In [12]:

```
#It is a good programming practise to explore the dataset that you are dealing with. This dataset is unique in it
#https://colab.research.google.com/github/Tyler-Hilbert/AudioProcessingInPythonWorkshop/blob/master/AudioProcessi
#visualize the data and write code to play 2-3 sound samples in the notebook for better understanding.
#please go through the following reference video https://www.youtube.com/watch?v=37zCgCdV468
```

In [13]:

```
import sys
sys.path.insert(0, 'ThinkDSP/code/')
import thinkdsp
import matplotlib.pyplot as pyplot
import IPython

# Read in audio file
wave = thinkdsp.read_wave(all_files[90])

# Grab first 10 seconds of audio (you can ignore me)
clipLength = 20 # in seconds
index = 0
while (index < wave.ts.size and wave.ts[index] < clipLength):
        index += 1
# Remove extras
wave.ts = wave.ts[:index]
wave.ys = wave.ys[:index]


# Filter
spectrum = wave.make_spectrum()
spectrum.low_pass(cutoff = 300, factor = .1)
#spectrum.high_pass(cutoff = 1500, factor = .1) # FIXME - Change back to low pass filter
filteredWave = spectrum.make_wave()

# Plot spectrum of audio file
spectrum = filteredWave.make_spectrum()
spectrum.plot()
pyplot.show()

# Play filtered audio file
filteredWave.play()
IPython.display.Audio('sound.wav')
```




```
Writing sound.wav
```

Out[13]: Your browser does not support the audio element.

Creating dataframe

In [10]:

```
#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
audio_labels = [float(audio_file.split("_")[0]) for audio_file in os.listdir(recording_path)]
df_audio = pd.DataFrame({"path" : all_files,"label":audio_labels})
```

In [11]:

```
#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   float64
dtypes: float64(1), object(1)
memory usage: 31.4+ KB
```

Grader function 2

Grader function 2

In [12]:
```
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[12]: True

In [13]:
```
from sklearn.utils import shuffle
df_audio = shuffle(df_audio, random_state=33)#don't change the random state
```

Train and Validation split

In [14]:
```
#split the data into train and validation and save in X_train, X_test, y_train, y_test
from sklearn.model_selection import  train_test_split

X = df_audio['path']
y = df_audio['label']
X_train, X_test, y_train,y_test = train_test_split(X,y,stratify = y,test_size = 0.3, random_state = 45)
#use stratify sampling
#use random state of 45
#use test size of 30%
```

Grader function 3

In [15]:
```
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[15]: True

Preprocessing

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

In [16]:
```
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 [17]:
```
X_train_processed = []
X_test_processed = []
```

In [18]:
```
#use load_wav function that was written above to get every wave.
#save it in X_train_processed and X_test_processed
def preprocess_audio_data(audio_files):
    raw_files,duration_list = [],[]
    for idx,audio_file in enumerate(audio_files):
        samples,duration = load_wav(audio_file)
        raw_files.append(samples)
        duration_list.append(duration)
```

```
    preprocessed_dataframe = pd.DataFrame({"raw_data" : raw_files,'duration' : duration_list})
    return preprocessed_dataframe

X_train_processed,X_test_processed = preprocess_audio_data(X_train.values),preprocess_audio_data(X_test.values)

#X_train_processed, X_test_processed = load_wav(X_train.values), load_wav(X_test.values)
# X_train_processed/X_test_processed should be dataframes with two columns(raw_data, duration) with same index o
```

In [19]:
```
X_train_processed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1400 entries, 0 to 1399
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   raw_data  1400 non-null   object
 1   duration  1400 non-null   float64
dtypes: float64(1), object(1)
memory usage: 22.0+ KB
```

In [24]:
```
X_train_processed.head(10)
```

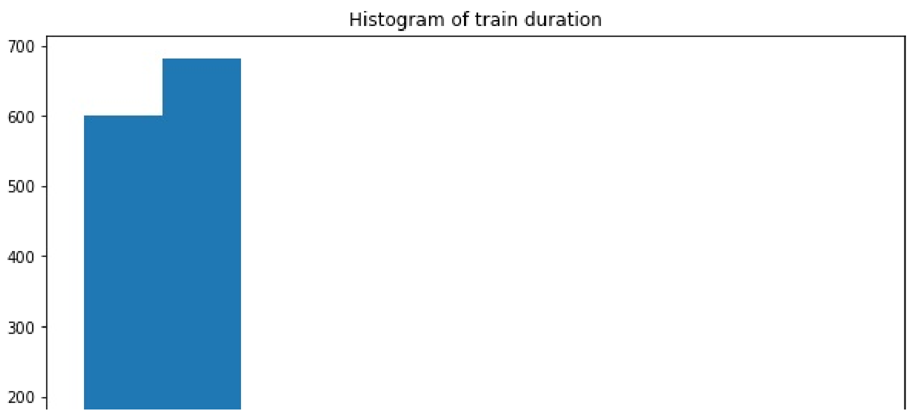
Out[24]:

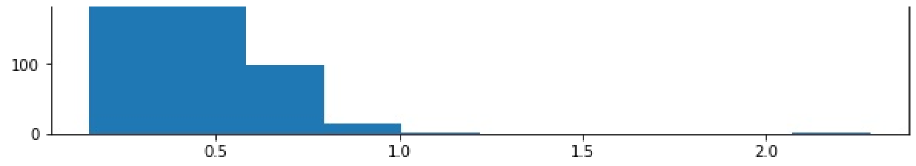
	raw_data	duration
0	[-0.00050216826, 2.7594652e-07, 0.00022539018,...	0.459138
1	[-0.0008239056, -0.00075821934, -0.0005198983,...	0.257143
2	[0.0016795197, -6.503213e-05, -0.005055008, -0...	0.230385
3	[-0.009725378, -0.011341786, -0.011049435, -0....	0.395420
4	[0.0005111587, 0.0006119106, 0.0006307227, 0.0...	0.387029
5	[-0.0005348968, 0.00029281023, 0.001044386, 0....	0.338912
6	[4.4097458e-05, 5.474639e-05, 3.8307837e-05, 9...	0.405397
7	[-0.00019068005, -0.00024691445, -0.0002540210...	0.241542
8	[-0.0069812275, -0.007846648, -0.008209535, -0...	0.402132
9	[1.7610597e-05, -8.146022e-05, -0.00022273406,...	0.378639

In [20]:
```
X_test_processed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   raw_data  600 non-null    object
 1   duration  600 non-null    float64
dtypes: float64(1), object(1)
memory usage: 9.5+ KB
```

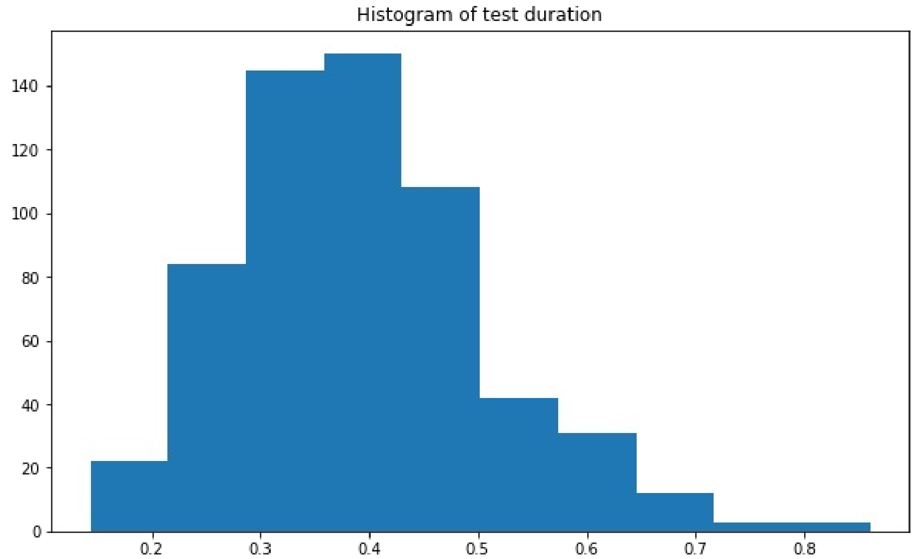
In [26]:
```
#plot the histogram of the duration for trian
import matplotlib.pyplot as plt
plt.figure(figsize = (10,6))
plt.hist(X_train_processed.duration.values)
plt.title("Histogram of train duration")
plt.show()
```





In [27]:

```
#plot the histogram of the duration for trian
plt.figure(figsize = (10,6))
plt.hist(X_test_processed.duration.values)
plt.title("Histogram of test duration")
plt.show()
```



In [28]:

```
#print 0 to 100 percentile values with step size of 10 for train data duration.
for i in range(0,100,10):
    print("{} th percentile is {}".format(i,np.percentile(X_train_processed.duration.values,i)))
```

```
0 th percentile is 0.1564172335600907
10 th percentile is 0.26312018140589566
20 th percentile is 0.3022222222222222
30 th percentile is 0.33380952380952383
40 th percentile is 0.3618684807256236
50 th percentile is 0.391156462585034
60 th percentile is 0.41955555555555557
70 th percentile is 0.4501995464852607
80 th percentile is 0.48639455782312924
90 th percentile is 0.5621632653061226
```

In [29]:

```
#print 90th percentile to 100 th percentil
for i in  range(90,101,1):
    print("{} th percentile is {}".format(i,np.percentile(X_train_processed.duration.values,i)))
```

```
90 th percentile is 0.5621632653061226
91 th percentile is 0.5757945578231292
92 th percentile is 0.5895238095238096
93 th percentile is 0.6080367346938776
94 th percentile is 0.6200163265306122
95 th percentile is 0.6355306122448978
96 th percentile is 0.6466866213151926
97 th percentile is 0.6652689342403627
98 th percentile is 0.7149841269841267
99 th percentile is 0.8209596371882085
100 th percentile is 2.282766439909297
```

Grader function 4

In [30]:

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

```
    return flag_columns and flag_shape
grader_processed()
```

Out[30]: True

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

While loading the audio files, we are using sampling rate of 22050 so one sec will give array 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 type must be bool.

In [31]:

```
max_length  = 17640
```

In [21]:

```
#https://www.tensorflow.org/guide/keras/masking_and_padding
#https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_length = 17640
## as discussed above, Pad with Zero if length of sequence is less than 17640 else Truncate the number.
## save in the X_train_pad_seq, X_test_pad_seq
X_train_pad_seq = pad_sequences(X_train_processed['raw_data'],maxlen = max_length, dtype = 'float32',padding= 'po
X_test_pad_seq = pad_sequences(X_test_processed['raw_data'],maxlen = max_length, dtype='float32',padding=  'post'
```

In [33]:

```
(X_train_pad_seq != 0.0)
```

```
Out[33]: array([[ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       ...,
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False]])
```

In [22]:

```
## as discussed above, Pad with Zero if length of sequence is less than 17640 else Truncate the number.
## save in the X_train_pad_seq, X_test_pad_seq
## also Create masking vector X_train_mask, X_test_mask
X_train_mask,X_test_mask = (X_train_pad_seq != 0.0),(X_test_pad_seq != 0.0)
## all the X_train_pad_seq, X_test_pad_seq, X_train_mask, X_test_mask will be numpy arrays mask vector dtype must
```

In [23]:

```
type(X_train_mask[0][0])
```

Out[23]: numpy.bool_

Grader function 5

In [36]:

```
def grader_padoutput():
    flag_padshape = (X_train_pad_seq.shape==(1400, 17640)) and (X_test_pad_seq.shape==(600, 17640)) and (y_train.
    flag_maskshape = (X_train_mask.shape==(1400, 17640)) and (X_test_mask.shape==(600, 17640)) and (y_test.shape=
    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[36]: True

1. Giving Raw data directly.

In [24]:

```
import tensorflow as tf
```

In [38]:
```
X_train_mask.shape
```

Out[38]: (1400, 17640)

In [39]:
```
input = tf.keras.layers.Input(shape=(X_train_pad_seq.shape[1],1))
mask_input = tf.keras.layers.Input(shape=(X_train_mask.shape[1]),dtype='bool')
lstm_layer = tf.keras.layers.LSTM(32)
lstm_output = lstm_layer(input,mask = mask_input)
dense = tf.keras.layers.Dense(16,activation='relu')(lstm_output)
output = tf.keras.layers.Dense(10,activation = 'softmax')(dense)
model_2 = tf.keras.models.Model(inputs = [input ,mask_input], outputs = output)
```

In [40]:
```
model_2.summary()
```

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

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

 lstm (LSTM)                    (None, 32)           4352        ['input_1[0][0]',
                                                                  'input_2[0][0]']

 dense (Dense)                  (None, 16)           528         ['lstm[0][0]']

 dense_1 (Dense)                (None, 10)           170         ['dense[0][0]']

==================================================================================================
Total params: 5,050
Trainable params: 5,050
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [25]:
```
from sklearn.metrics import f1_score
class F1Metrics(tf.keras.callbacks.Callback):

    def __init__(self,validation_data):

        super(F1Metrics,self).__init__()
        self.validation_data = validation_data


    def on_epoch_end(self,epochs,logs = {}):

        y_true = self.validation_data[1]
        y_pred = self.model.predict(self.validation_data[0])
        y_pred = np.argmax(y_pred,axis = 1
                           )
        f1_score_metrics = f1_score(y_true,y_pred,average='micro')

        logs['val_f1_score'] = f1_score_metrics

        print("f1 Score  : {0}".format(f1_score_metrics))
```

In [42]:
```
#lstm input dimention is -> number_of_batches,time_stamps,Features
X_train_input = [ np.expand_dims(X_train_pad_seq,axis =2),X_train_mask]
X_test_input = [np.expand_dims(X_test_pad_seq,axis = 2),X_test_mask]
```

In [ ]:
```
import datetime
#https://stackoverflow.com/questions/44583254/valueerror-input-0-is-incompatible-with-layer-lstm-13-expected-ndim
f1_score_callback = F1Metrics((X_test_input,y_test))
early_stopping_call_backs = tf.keras.callbacks.EarlyStopping(patience= 2)
log_dir= log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
                            log_dir=log_dir, histogram_freq=1)
call_backs = [f1_score_callback,early_stopping_call_backs,tensorboard_callback]

model_2.compile(optimizer= tf.keras.optimizers.Adam( learning_rate=0.001),loss = tf.keras.losses.SparseCategorica
              )
model_history = model_2.fit(X_train_input,y_train,epochs = 5,validation_data=(X_test_input,y_test),batch_size = 3
```

```
Epoch 1/5
44/44 [==============================] - 77s 2s/step - loss: 2.3033 - accuracy: 0.0957 - val_loss: 2.3027 - val_a
ccuracy: 0.0933
f1 Score  : 0.09333333333333334
Epoch 2/5
44/44 [==============================] - 64s 1s/step - loss: 2.3030 - accuracy: 0.0864 - val_loss: 2.3026 - val_a
ccuracy: 0.0983
f1 Score  : 0.09833333333333333
Epoch 3/5
44/44 [==============================] - 72s 2s/step - loss: 2.3028 - accuracy: 0.0864 - val_loss: 2.3026 - val_a
ccuracy: 0.1000
f1 Score  : 0.10000000000000002
Epoch 4/5
44/44 [==============================] - 71s 2s/step - loss: 2.3028 - accuracy: 0.0957 - val_loss: 2.3026 - val_a
ccuracy: 0.1033
f1 Score  : 0.10333333333333333
Epoch 5/5
44/44 [==============================] - 72s 2s/step - loss: 2.3027 - accuracy: 0.0936 - val_loss: 2.3026 - val_a
ccuracy: 0.1000
f1 Score  : 0.10000000000000002
```

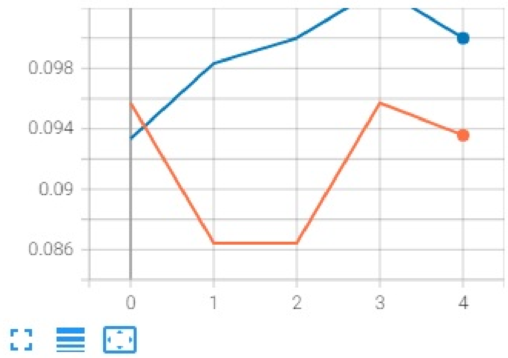
In [ ]:

```
%load_ext tensorboard
%tensorboard --logdir '/content/logs/fit/20211109-123458'
```

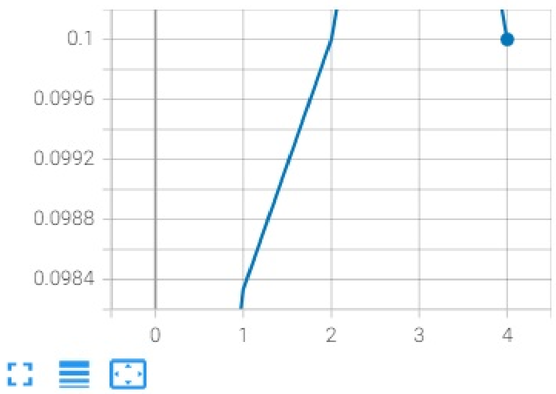
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Model-1 Accuracy,Loss and F1Score

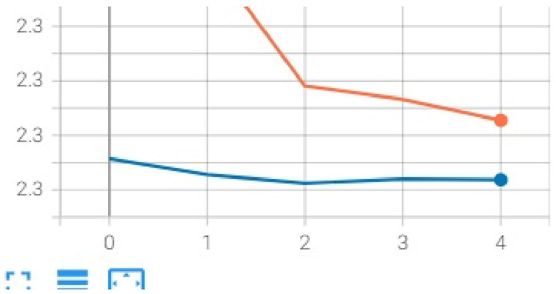
epoch_accuracy
tag: epoch_accuracy



epoch_f1_score
tag: epoch_f1_score



epoch_loss
tag: epoch_loss



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). Also check the datatype of class labels(y_values) and make sure that you convert your class labels to integer datatype before fitting in the model.
3. While defining your model make sure that you pass both the input layer and mask input layer as input to lstm layer as follows

```
lstm_output = self.lstm(input_layer, mask=masking_input_layer)
```

4. Use tensorboard to plot the graphs of loss and metric(use custom micro F1 score as metric) and histograms of gradients. You can write your code for computing F1 score using this link

5. make sure that it won't overfit.
6. You are free to include any regularization

In [ ]:
```
## as discussed above, please write the architecture of the model.
## you will have two input layers in your model (data input layer and mask input layer)
## make sure that you have defined the data type of masking layer as bool
```

In [ ]:
```
```

In [ ]:
```
#train your model
#model1.fit([X_train_pad_seq,X_train_mask],y_train_int,.........)
```

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 in 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 [26]:
```
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 [28]:
```
convert_to_spectrogram(X_train_pad_seq[0])
```

Out[28]:
```
array([[-54.270134, -52.505615, -51.841507, ..., -80.        , -80.        ,
         -80.        ],
       [-51.343956, -50.148666, -49.257126, ..., -80.        , -80.        ,
         -80.        ],
       [-51.26763 , -52.10228 , -51.384697, ..., -80.        , -80.        ,
         -80.        ],
       ...,
       [-80.        , -80.        , -80.        , ..., -80.        , -80.        ,
         -80.        ],
```

```
       [-80.      , -80.      , -80.      , ..., -80.      , -80.      ,
        -80.      ],
       [-80.      , -80.      , -80.      , ..., -80.      , -80.      ,
        -80.      ]], dtype=float32)
```

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 numpy arrays)

def convert_padding_to_spectogram(padding_data):
    spectrogram_list = []
    for padded_sequence in padding_data:
        spectrogram_list.append(convert_to_spectrogram(padded_sequence))
    return np.array(spectrogram_list)
```

In [30]:

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

In [31]:

```
X_train_spectrogram.shape
```

Out[31]: (1400, 64, 35)

Grader function 6

In [48]:

```
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[48]: 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.
(ex: Output from LSTM will be  (None, time_steps, features) average the output of every time step
i.e, you should get (None,time_steps)
and then pass to dense layer )
3. give the above output to Dense layer of size 10( output layer) and train the network with
sparse categorical cross entropy.
4. Use tensorboard to plot the graphs of loss and metric(use custom micro F1 score as metric) and
histograms of gradients. You can write your code for computing F1 score using this link
5. make sure that it won't overfit.
6. You are free to include any regularization
```

In [32]:

```
# write the architecture of the model
tf.keras.backend.clear_session()
input_spectrum = tf.keras.layers.Input(shape=(X_train_spectrogram.shape[1],X_train_spectrogram.shape[2]))
lstm_layer = tf.keras.layers.LSTM(units = 64, return_sequences= True)(input_spectrum)
lstm_layer = tf.keras.layers.LSTM(units = 64, return_sequences= True)(lstm_layer)
global_average_pooling = tf.keras.layers.GlobalAvgPool1D()(lstm_layer)
dense_layer = tf.keras.layers.Dense(units = 64, activation = 'relu')(global_average_pooling)
dense_layer = tf.keras.layers.Dense(units = 32 , activation = 'relu')(dense_layer)
output_layer = tf.keras.layers.Dense(units= 10,activation = 'softmax')(dense_layer)
model_3 = tf.keras.models.Model(inputs = input_spectrum,outputs= output_layer)
model_3.summary()
#print model.summary and make sure that it is following point 2 mentioned above
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
```

```
 input_1 (InputLayer)        [(None, 64, 35)]          0

 lstm (LSTM)                 (None, 64, 64)            25600

 lstm_1 (LSTM)               (None, 64, 64)            33024

 global_average_pooling1d (G  (None, 64)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 10)                330

=================================================================
Total params: 65,194
Trainable params: 65,194
Non-trainable params: 0
_________________________________________________________________
```

In [34]:

```
import datetime
#compile and fit your model.
#model2.fit([X_train_spectrogram],y_train_int,......)
f1_score_callback = F1Metrics((X_test_spectrogram,y_test))
#early_stopping_call_backs = tf.keras.callbacks.EarlyStopping(patience= 2)
log_dir= log_dir = "logs/fit/model_2" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
                            log_dir=log_dir, histogram_freq=1)
call_backs = [f1_score_callback,tensorboard_callback]

model_3.compile(optimizer= tf.keras.optimizers.Adam( learning_rate=0.001),loss = tf.keras.losses.SparseCategorica
model_history = model_3.fit(X_train_spectrogram,y_train,epochs = 100,validation_data=(X_test_spectrogram,y_test),
```

```
Epoch 1/100
 5/44 [==>...........................] - ETA: 1s - loss: 2.2978 - accuracy: 0.1063WARNING:tensorflow:Callback met
hod `on_train_batch_end` is slow compared to the batch time (batch time: 0.0139s vs `on_train_batch_end` time: 0.
0327s). Check your callbacks.
43/44 [============================>.] - ETA: 0s - loss: 2.1562 - accuracy: 0.1795f1 Score  : 0.21666666666666667
44/44 [==============================] - 13s 100ms/step - loss: 2.1542 - accuracy: 0.1807 - val_loss: 1.9882 - va
l_accuracy: 0.2167 - val_f1_score: 0.2167
Epoch 2/100
43/44 [============================>.] - ETA: 0s - loss: 1.8897 - accuracy: 0.2573f1 Score  : 0.38
44/44 [==============================] - 2s 45ms/step - loss: 1.8900 - accuracy: 0.2579 - val_loss: 1.8489 - val_
accuracy: 0.3800 - val_f1_score: 0.3800
Epoch 3/100
43/44 [============================>.] - ETA: 0s - loss: 1.7009 - accuracy: 0.3895f1 Score  : 0.36333333333333334
44/44 [==============================] - 2s 49ms/step - loss: 1.7052 - accuracy: 0.3886 - val_loss: 1.7534 - val_
accuracy: 0.3633 - val_f1_score: 0.3633
Epoch 4/100
43/44 [============================>.] - ETA: 0s - loss: 1.5922 - accuracy: 0.4172f1 Score  : 0.41999999999999993
44/44 [==============================] - 2s 45ms/step - loss: 1.5874 - accuracy: 0.4186 - val_loss: 1.5704 - val_
accuracy: 0.4200 - val_f1_score: 0.4200
Epoch 5/100
43/44 [============================>.] - ETA: 0s - loss: 1.4948 - accuracy: 0.4513f1 Score  : 0.4683333333333333
44/44 [==============================] - 2s 44ms/step - loss: 1.4937 - accuracy: 0.4529 - val_loss: 1.5191 - val_
accuracy: 0.4683 - val_f1_score: 0.4683
Epoch 6/100
43/44 [============================>.] - ETA: 0s - loss: 1.4040 - accuracy: 0.5051f1 Score  : 0.5166666666666667
44/44 [==============================] - 2s 47ms/step - loss: 1.4028 - accuracy: 0.5064 - val_loss: 1.4113 - val_
accuracy: 0.5167 - val_f1_score: 0.5167
Epoch 7/100
43/44 [============================>.] - ETA: 0s - loss: 1.3024 - accuracy: 0.5429f1 Score  : 0.4633333333333333
44/44 [==============================] - 2s 44ms/step - loss: 1.3059 - accuracy: 0.5400 - val_loss: 1.4439 - val_
accuracy: 0.4633 - val_f1_score: 0.4633
Epoch 8/100
43/44 [============================>.] - ETA: 0s - loss: 1.2420 - accuracy: 0.5828f1 Score  : 0.5233333333333333
44/44 [==============================] - 2s 49ms/step - loss: 1.2443 - accuracy: 0.5807 - val_loss: 1.3924 - val_
accuracy: 0.5233 - val_f1_score: 0.5233
Epoch 9/100
43/44 [============================>.] - ETA: 0s - loss: 1.2673 - accuracy: 0.5610f1 Score  : 0.5233333333333333
44/44 [==============================] - 2s 47ms/step - loss: 1.2606 - accuracy: 0.5650 - val_loss: 1.4079 - val_
accuracy: 0.5233 - val_f1_score: 0.5233
Epoch 10/100
43/44 [============================>.] - ETA: 0s - loss: 1.1377 - accuracy: 0.6090f1 Score  : 0.6
44/44 [==============================] - 2s 46ms/step - loss: 1.1339 - accuracy: 0.6093 - val_loss: 1.1622 - val_
accuracy: 0.6000 - val_f1_score: 0.6000
Epoch 11/100
43/44 [============================>.] - ETA: 0s - loss: 1.1058 - accuracy: 0.6010f1 Score  : 0.5966666666666667
44/44 [==============================] - 2s 45ms/step - loss: 1.1080 - accuracy: 0.6021 - val_loss: 1.1932 - val_
accuracy: 0.5967 - val_f1_score: 0.5967
```

```
Epoch 12/100
43/44 [============================>.] - ETA: 0s - loss: 0.9423 - accuracy: 0.6984f1 Score  : 0.6666666666666666
44/44 [==============================] - 2s 47ms/step - loss: 0.9396 - accuracy: 0.7000 - val_loss: 0.9768 - val_
accuracy: 0.6667 - val_f1_score: 0.6667
Epoch 13/100
43/44 [============================>.] - ETA: 0s - loss: 0.8781 - accuracy: 0.7006f1 Score  : 0.655
44/44 [==============================] - 2s 47ms/step - loss: 0.8740 - accuracy: 0.7021 - val_loss: 0.9736 - val_
accuracy: 0.6550 - val_f1_score: 0.6550
Epoch 14/100
43/44 [============================>.] - ETA: 0s - loss: 0.9242 - accuracy: 0.6781f1 Score  : 0.63
44/44 [==============================] - 2s 45ms/step - loss: 0.9211 - accuracy: 0.6800 - val_loss: 1.0535 - val_
accuracy: 0.6300 - val_f1_score: 0.6300
Epoch 15/100
43/44 [============================>.] - ETA: 0s - loss: 0.7825 - accuracy: 0.7362f1 Score  : 0.6783333333333333
44/44 [==============================] - 2s 48ms/step - loss: 0.7893 - accuracy: 0.7336 - val_loss: 0.8842 - val_
accuracy: 0.6783 - val_f1_score: 0.6783
Epoch 16/100
43/44 [============================>.] - ETA: 0s - loss: 0.7099 - accuracy: 0.7478f1 Score  : 0.6833333333333333
44/44 [==============================] - 2s 45ms/step - loss: 0.7036 - accuracy: 0.7507 - val_loss: 0.9613 - val_
accuracy: 0.6833 - val_f1_score: 0.6833
Epoch 17/100
43/44 [============================>.] - ETA: 0s - loss: 0.7637 - accuracy: 0.7398f1 Score  : 0.6866666666666666
44/44 [==============================] - 2s 45ms/step - loss: 0.7609 - accuracy: 0.7400 - val_loss: 0.8756 - val_
accuracy: 0.6867 - val_f1_score: 0.6867
Epoch 18/100
43/44 [============================>.] - ETA: 0s - loss: 0.7143 - accuracy: 0.7536f1 Score  : 0.69
44/44 [==============================] - 2s 45ms/step - loss: 0.7252 - accuracy: 0.7507 - val_loss: 0.8744 - val_
accuracy: 0.6900 - val_f1_score: 0.6900
Epoch 19/100
43/44 [============================>.] - ETA: 0s - loss: 0.6906 - accuracy: 0.7551f1 Score  : 0.6966666666666667
44/44 [==============================] - 2s 46ms/step - loss: 0.6879 - accuracy: 0.7564 - val_loss: 0.8429 - val_
accuracy: 0.6967 - val_f1_score: 0.6967
Epoch 20/100
43/44 [============================>.] - ETA: 0s - loss: 0.7292 - accuracy: 0.7369f1 Score  : 0.5983333333333334
44/44 [==============================] - 2s 48ms/step - loss: 0.7303 - accuracy: 0.7357 - val_loss: 1.0928 - val_
accuracy: 0.5983 - val_f1_score: 0.5983
Epoch 21/100
43/44 [============================>.] - ETA: 0s - loss: 0.6533 - accuracy: 0.7805f1 Score  : 0.7299999999999999
44/44 [==============================] - 2s 48ms/step - loss: 0.6519 - accuracy: 0.7814 - val_loss: 0.7666 - val_
accuracy: 0.7300 - val_f1_score: 0.7300
Epoch 22/100
43/44 [============================>.] - ETA: 0s - loss: 0.5924 - accuracy: 0.7783f1 Score  : 0.7616666666666667
44/44 [==============================] - 2s 47ms/step - loss: 0.6008 - accuracy: 0.7786 - val_loss: 0.7420 - val_
accuracy: 0.7617 - val_f1_score: 0.7617
Epoch 23/100
43/44 [============================>.] - ETA: 0s - loss: 0.5400 - accuracy: 0.8089f1 Score  : 0.6966666666666667
44/44 [==============================] - 2s 47ms/step - loss: 0.5485 - accuracy: 0.8050 - val_loss: 0.8678 - val_
accuracy: 0.6967 - val_f1_score: 0.6967
Epoch 24/100
43/44 [============================>.] - ETA: 0s - loss: 0.5573 - accuracy: 0.8096f1 Score  : 0.7166666666666667
44/44 [==============================] - 2s 48ms/step - loss: 0.5544 - accuracy: 0.8100 - val_loss: 0.8173 - val_
accuracy: 0.7167 - val_f1_score: 0.7167
Epoch 25/100
43/44 [============================>.] - ETA: 0s - loss: 0.5099 - accuracy: 0.8140f1 Score  : 0.7516666666666667
44/44 [==============================] - 2s 48ms/step - loss: 0.5135 - accuracy: 0.8121 - val_loss: 0.7081 - val_
accuracy: 0.7517 - val_f1_score: 0.7517
Epoch 26/100
43/44 [============================>.] - ETA: 0s - loss: 0.4491 - accuracy: 0.8416f1 Score  : 0.7816666666666666
44/44 [==============================] - 2s 46ms/step - loss: 0.4490 - accuracy: 0.8414 - val_loss: 0.5845 - val_
accuracy: 0.7817 - val_f1_score: 0.7817
Epoch 27/100
43/44 [============================>.] - ETA: 0s - loss: 0.3961 - accuracy: 0.8561f1 Score  : 0.815
44/44 [==============================] - 2s 47ms/step - loss: 0.4002 - accuracy: 0.8543 - val_loss: 0.5750 - val_
accuracy: 0.8150 - val_f1_score: 0.8150
Epoch 28/100
43/44 [============================>.] - ETA: 0s - loss: 0.4049 - accuracy: 0.8554f1 Score  : 0.7983333333333333
44/44 [==============================] - 2s 45ms/step - loss: 0.4042 - accuracy: 0.8557 - val_loss: 0.5932 - val_
accuracy: 0.7983 - val_f1_score: 0.7983
Epoch 29/100
43/44 [============================>.] - ETA: 0s - loss: 0.4299 - accuracy: 0.8525f1 Score  : 0.8000000000000002
44/44 [==============================] - 2s 50ms/step - loss: 0.4328 - accuracy: 0.8521 - val_loss: 0.6123 - val_
accuracy: 0.8000 - val_f1_score: 0.8000
Epoch 30/100
43/44 [============================>.] - ETA: 0s - loss: 0.3891 - accuracy: 0.8576f1 Score  : 0.8266666666666667
44/44 [==============================] - 2s 47ms/step - loss: 0.3861 - accuracy: 0.8586 - val_loss: 0.5227 - val_
accuracy: 0.8267 - val_f1_score: 0.8267
Epoch 31/100
43/44 [============================>.] - ETA: 0s - loss: 0.4218 - accuracy: 0.8481f1 Score  : 0.7966666666666665
44/44 [==============================] - 2s 45ms/step - loss: 0.4201 - accuracy: 0.8486 - val_loss: 0.6415 - val_
accuracy: 0.7967 - val_f1_score: 0.7967
Epoch 32/100
43/44 [============================>.] - ETA: 0s - loss: 0.4170 - accuracy: 0.8503f1 Score  : 0.8166666666666667
44/44 [==============================] - 2s 45ms/step - loss: 0.4201 - accuracy: 0.8486 - val_loss: 0.5761 - val_
```

```
accuracy: 0.8167 - val_f1_score: 0.8167
Epoch 33/100
43/44 [============================>.] - ETA: 0s - loss: 0.3045 - accuracy: 0.9041f1 Score  : 0.8433333333333335
44/44 [==============================] - 2s 46ms/step - loss: 0.3072 - accuracy: 0.9014 - val_loss: 0.4711 - val_
accuracy: 0.8433 - val_f1_score: 0.8433
Epoch 34/100
43/44 [============================>.] - ETA: 0s - loss: 0.2989 - accuracy: 0.8953f1 Score  : 0.8533333333333335
44/44 [==============================] - 2s 47ms/step - loss: 0.2966 - accuracy: 0.8964 - val_loss: 0.4863 - val_
accuracy: 0.8533 - val_f1_score: 0.8533
Epoch 35/100
43/44 [============================>.] - ETA: 0s - loss: 0.3995 - accuracy: 0.8496f1 Score  : 0.81
44/44 [==============================] - 2s 50ms/step - loss: 0.3979 - accuracy: 0.8514 - val_loss: 0.5858 - val_
accuracy: 0.8100 - val_f1_score: 0.8100
Epoch 36/100
43/44 [============================>.] - ETA: 0s - loss: 0.3095 - accuracy: 0.8837f1 Score  : 0.865
44/44 [==============================] - 2s 46ms/step - loss: 0.3113 - accuracy: 0.8836 - val_loss: 0.4405 - val_
accuracy: 0.8650 - val_f1_score: 0.8650
Epoch 37/100
43/44 [============================>.] - ETA: 0s - loss: 0.2542 - accuracy: 0.9092f1 Score  : 0.8533333333333335
44/44 [==============================] - 2s 45ms/step - loss: 0.2550 - accuracy: 0.9093 - val_loss: 0.4498 - val_
accuracy: 0.8533 - val_f1_score: 0.8533
Epoch 38/100
43/44 [============================>.] - ETA: 0s - loss: 0.3923 - accuracy: 0.8590f1 Score  : 0.8466666666666667
44/44 [==============================] - 2s 46ms/step - loss: 0.3914 - accuracy: 0.8600 - val_loss: 0.4825 - val_
accuracy: 0.8467 - val_f1_score: 0.8467
Epoch 39/100
43/44 [============================>.] - ETA: 0s - loss: 0.3260 - accuracy: 0.8808f1 Score  : 0.87
44/44 [==============================] - 2s 47ms/step - loss: 0.3227 - accuracy: 0.8829 - val_loss: 0.4487 - val_
accuracy: 0.8700 - val_f1_score: 0.8700
Epoch 40/100
43/44 [============================>.] - ETA: 0s - loss: 0.2184 - accuracy: 0.9266f1 Score  : 0.8766666666666667
44/44 [==============================] - 2s 47ms/step - loss: 0.2169 - accuracy: 0.9271 - val_loss: 0.4361 - val_
accuracy: 0.8767 - val_f1_score: 0.8767
Epoch 41/100
43/44 [============================>.] - ETA: 0s - loss: 0.2336 - accuracy: 0.9142f1 Score  : 0.8116666666666666
44/44 [==============================] - 2s 47ms/step - loss: 0.2372 - accuracy: 0.9129 - val_loss: 0.6010 - val_
accuracy: 0.8117 - val_f1_score: 0.8117
Epoch 42/100
43/44 [============================>.] - ETA: 0s - loss: 0.2700 - accuracy: 0.9121f1 Score  : 0.855
44/44 [==============================] - 2s 48ms/step - loss: 0.2710 - accuracy: 0.9114 - val_loss: 0.5011 - val_
accuracy: 0.8550 - val_f1_score: 0.8550
Epoch 43/100
43/44 [============================>.] - ETA: 0s - loss: 0.2954 - accuracy: 0.8888f1 Score  : 0.8483333333333335
44/44 [==============================] - 2s 46ms/step - loss: 0.2927 - accuracy: 0.8900 - val_loss: 0.4735 - val_
accuracy: 0.8483 - val_f1_score: 0.8483
Epoch 44/100
43/44 [============================>.] - ETA: 0s - loss: 0.2074 - accuracy: 0.9331f1 Score  : 0.8483333333333335
44/44 [==============================] - 2s 47ms/step - loss: 0.2082 - accuracy: 0.9336 - val_loss: 0.4748 - val_
accuracy: 0.8483 - val_f1_score: 0.8483
Epoch 45/100
43/44 [============================>.] - ETA: 0s - loss: 0.2225 - accuracy: 0.9215f1 Score  : 0.8466666666666667
44/44 [==============================] - 2s 48ms/step - loss: 0.2227 - accuracy: 0.9214 - val_loss: 0.5352 - val_
accuracy: 0.8467 - val_f1_score: 0.8467
Epoch 46/100
43/44 [============================>.] - ETA: 0s - loss: 0.1550 - accuracy: 0.9513f1 Score  : 0.89
44/44 [==============================] - 2s 47ms/step - loss: 0.1553 - accuracy: 0.9514 - val_loss: 0.3920 - val_
accuracy: 0.8900 - val_f1_score: 0.8900
Epoch 47/100
43/44 [============================>.] - ETA: 0s - loss: 0.1846 - accuracy: 0.9302f1 Score  : 0.8716666666666667
44/44 [==============================] - 2s 47ms/step - loss: 0.1844 - accuracy: 0.9307 - val_loss: 0.4155 - val_
accuracy: 0.8717 - val_f1_score: 0.8717
Epoch 48/100
43/44 [============================>.] - ETA: 0s - loss: 0.1985 - accuracy: 0.9310f1 Score  : 0.825
44/44 [==============================] - 2s 48ms/step - loss: 0.1990 - accuracy: 0.9307 - val_loss: 0.5472 - val_
accuracy: 0.8250 - val_f1_score: 0.8250
Epoch 49/100
44/44 [==============================] - ETA: 0s - loss: 0.2721 - accuracy: 0.9086f1 Score  : 0.8666666666666667
44/44 [==============================] - 2s 46ms/step - loss: 0.2721 - accuracy: 0.9086 - val_loss: 0.4470 - val_
accuracy: 0.8667 - val_f1_score: 0.8667
Epoch 50/100
43/44 [============================>.] - ETA: 0s - loss: 0.2601 - accuracy: 0.9077f1 Score  : 0.8766666666666667
44/44 [==============================] - 2s 48ms/step - loss: 0.2568 - accuracy: 0.9093 - val_loss: 0.4151 - val_
accuracy: 0.8767 - val_f1_score: 0.8767
Epoch 51/100
43/44 [============================>.] - ETA: 0s - loss: 0.2668 - accuracy: 0.9062f1 Score  : 0.8933333333333333
44/44 [==============================] - 2s 45ms/step - loss: 0.2683 - accuracy: 0.9043 - val_loss: 0.3570 - val_
accuracy: 0.8933 - val_f1_score: 0.8933
Epoch 52/100
43/44 [============================>.] - ETA: 0s - loss: 0.2336 - accuracy: 0.9164f1 Score  : 0.8916666666666667
44/44 [==============================] - 2s 45ms/step - loss: 0.2339 - accuracy: 0.9157 - val_loss: 0.3781 - val_
accuracy: 0.8917 - val_f1_score: 0.8917
Epoch 53/100
43/44 [============================>.] - ETA: 0s - loss: 0.1919 - accuracy: 0.9339f1 Score  : 0.8983333333333333
```

```
44/44 [==============================] - 2s 48ms/step - loss: 0.1919 - accuracy: 0.9336 - val_loss: 0.3788 - val_
accuracy: 0.8983 - val_f1_score: 0.8983
Epoch 54/100
43/44 [============================>.] - ETA: 0s - loss: 0.2117 - accuracy: 0.9259f1 Score  : 0.8866666666666667
44/44 [==============================] - 2s 48ms/step - loss: 0.2114 - accuracy: 0.9250 - val_loss: 0.3737 - val_
accuracy: 0.8867 - val_f1_score: 0.8867
Epoch 55/100
43/44 [============================>.] - ETA: 0s - loss: 0.1713 - accuracy: 0.9382f1 Score  : 0.8816666666666667
44/44 [==============================] - 2s 46ms/step - loss: 0.1720 - accuracy: 0.9379 - val_loss: 0.4304 - val_
accuracy: 0.8817 - val_f1_score: 0.8817
Epoch 56/100
43/44 [============================>.] - ETA: 0s - loss: 0.2624 - accuracy: 0.9026f1 Score  : 0.8833333333333333
44/44 [==============================] - 2s 47ms/step - loss: 0.2613 - accuracy: 0.9029 - val_loss: 0.3580 - val_
accuracy: 0.8833 - val_f1_score: 0.8833
Epoch 57/100
43/44 [============================>.] - ETA: 0s - loss: 0.2743 - accuracy: 0.9070f1 Score  : 0.775
44/44 [==============================] - 2s 48ms/step - loss: 0.2746 - accuracy: 0.9064 - val_loss: 0.6943 - val_
accuracy: 0.7750 - val_f1_score: 0.7750
Epoch 58/100
43/44 [============================>.] - ETA: 0s - loss: 0.2756 - accuracy: 0.9135f1 Score  : 0.8983333333333333
44/44 [==============================] - 2s 48ms/step - loss: 0.2724 - accuracy: 0.9150 - val_loss: 0.3407 - val_
accuracy: 0.8983 - val_f1_score: 0.8983
Epoch 59/100
43/44 [============================>.] - ETA: 0s - loss: 0.1718 - accuracy: 0.9469f1 Score  : 0.8783333333333333
44/44 [==============================] - 2s 46ms/step - loss: 0.1713 - accuracy: 0.9464 - val_loss: 0.3886 - val_
accuracy: 0.8783 - val_f1_score: 0.8783
Epoch 60/100
43/44 [============================>.] - ETA: 0s - loss: 0.1508 - accuracy: 0.9520f1 Score  : 0.9083333333333333
44/44 [==============================] - 2s 48ms/step - loss: 0.1496 - accuracy: 0.9521 - val_loss: 0.3173 - val_
accuracy: 0.9083 - val_f1_score: 0.9083
Epoch 61/100
43/44 [============================>.] - ETA: 0s - loss: 0.1152 - accuracy: 0.9615f1 Score  : 0.9166666666666666
44/44 [==============================] - 2s 50ms/step - loss: 0.1143 - accuracy: 0.9621 - val_loss: 0.3071 - val_
accuracy: 0.9167 - val_f1_score: 0.9167
Epoch 62/100
43/44 [============================>.] - ETA: 0s - loss: 0.0877 - accuracy: 0.9731f1 Score  : 0.8983333333333333
44/44 [==============================] - 2s 47ms/step - loss: 0.0880 - accuracy: 0.9729 - val_loss: 0.3467 - val_
accuracy: 0.8983 - val_f1_score: 0.8983
Epoch 63/100
43/44 [============================>.] - ETA: 0s - loss: 0.1037 - accuracy: 0.9680f1 Score  : 0.92
44/44 [==============================] - 2s 48ms/step - loss: 0.1041 - accuracy: 0.9671 - val_loss: 0.2975 - val_
accuracy: 0.9200 - val_f1_score: 0.9200
Epoch 64/100
43/44 [============================>.] - ETA: 0s - loss: 0.1170 - accuracy: 0.9658f1 Score  : 0.8833333333333333
44/44 [==============================] - 2s 48ms/step - loss: 0.1158 - accuracy: 0.9664 - val_loss: 0.4291 - val_
accuracy: 0.8833 - val_f1_score: 0.8833
Epoch 65/100
43/44 [============================>.] - ETA: 0s - loss: 0.1191 - accuracy: 0.9564f1 Score  : 0.9083333333333333
44/44 [==============================] - 2s 46ms/step - loss: 0.1188 - accuracy: 0.9571 - val_loss: 0.3341 - val_
accuracy: 0.9083 - val_f1_score: 0.9083
Epoch 66/100
43/44 [============================>.] - ETA: 0s - loss: 0.1224 - accuracy: 0.9571f1 Score  : 0.8866666666666667
44/44 [==============================] - 2s 48ms/step - loss: 0.1212 - accuracy: 0.9579 - val_loss: 0.3655 - val_
accuracy: 0.8867 - val_f1_score: 0.8867
Epoch 67/100
43/44 [============================>.] - ETA: 0s - loss: 0.1035 - accuracy: 0.9717f1 Score  : 0.91
44/44 [==============================] - 2s 45ms/step - loss: 0.1075 - accuracy: 0.9700 - val_loss: 0.3411 - val_
accuracy: 0.9100 - val_f1_score: 0.9100
Epoch 68/100
44/44 [==============================] - ETA: 0s - loss: 0.3471 - accuracy: 0.8886f1 Score  : 0.8000000000000002
44/44 [==============================] - 2s 51ms/step - loss: 0.3471 - accuracy: 0.8886 - val_loss: 0.5862 - val_
accuracy: 0.8000 - val_f1_score: 0.8000
Epoch 69/100
43/44 [============================>.] - ETA: 0s - loss: 0.3458 - accuracy: 0.8779f1 Score  : 0.885
44/44 [==============================] - 2s 48ms/step - loss: 0.3417 - accuracy: 0.8793 - val_loss: 0.3671 - val_
accuracy: 0.8850 - val_f1_score: 0.8850
Epoch 70/100
43/44 [============================>.] - ETA: 0s - loss: 0.1829 - accuracy: 0.9390f1 Score  : 0.8866666666666667
44/44 [==============================] - 2s 45ms/step - loss: 0.1817 - accuracy: 0.9393 - val_loss: 0.3693 - val_
accuracy: 0.8867 - val_f1_score: 0.8867
Epoch 71/100
43/44 [============================>.] - ETA: 0s - loss: 0.1786 - accuracy: 0.9310f1 Score  : 0.8533333333333335
44/44 [==============================] - 2s 45ms/step - loss: 0.1798 - accuracy: 0.9307 - val_loss: 0.5014 - val_
accuracy: 0.8533 - val_f1_score: 0.8533
Epoch 72/100
43/44 [============================>.] - ETA: 0s - loss: 0.1814 - accuracy: 0.9397f1 Score  : 0.88
44/44 [==============================] - 2s 45ms/step - loss: 0.1816 - accuracy: 0.9393 - val_loss: 0.3828 - val_
accuracy: 0.8800 - val_f1_score: 0.8800
Epoch 73/100
43/44 [============================>.] - ETA: 0s - loss: 0.1270 - accuracy: 0.9520f1 Score  : 0.9183333333333333
44/44 [==============================] - 2s 48ms/step - loss: 0.1314 - accuracy: 0.9507 - val_loss: 0.2767 - val_
accuracy: 0.9183 - val_f1_score: 0.9183
Epoch 74/100
```

```
43/44 [============================>.] - ETA: 0s - loss: 0.0970 - accuracy: 0.9680f1 Score  : 0.8983333333333333
44/44 [==============================] - 2s 46ms/step - loss: 0.0961 - accuracy: 0.9686 - val_loss: 0.3643 - val_
accuracy: 0.8983 - val_f1_score: 0.8983
Epoch 75/100
43/44 [============================>.] - ETA: 0s - loss: 0.4102 - accuracy: 0.8605f1 Score  : 0.8716666666666667
44/44 [==============================] - 2s 45ms/step - loss: 0.4164 - accuracy: 0.8571 - val_loss: 0.4879 - val_
accuracy: 0.8717 - val_f1_score: 0.8717
Epoch 76/100
43/44 [============================>.] - ETA: 0s - loss: 0.2605 - accuracy: 0.9142f1 Score  : 0.895
44/44 [==============================] - 2s 47ms/step - loss: 0.2589 - accuracy: 0.9143 - val_loss: 0.3985 - val_
accuracy: 0.8950 - val_f1_score: 0.8950
Epoch 77/100
43/44 [============================>.] - ETA: 0s - loss: 0.2036 - accuracy: 0.9273f1 Score  : 0.89
44/44 [==============================] - 2s 46ms/step - loss: 0.2028 - accuracy: 0.9271 - val_loss: 0.3537 - val_
accuracy: 0.8900 - val_f1_score: 0.8900
Epoch 78/100
43/44 [============================>.] - ETA: 0s - loss: 0.1559 - accuracy: 0.9440f1 Score  : 0.9066666666666666
44/44 [==============================] - 2s 49ms/step - loss: 0.1563 - accuracy: 0.9436 - val_loss: 0.3001 - val_
accuracy: 0.9067 - val_f1_score: 0.9067
Epoch 79/100
43/44 [============================>.] - ETA: 0s - loss: 0.1162 - accuracy: 0.9608f1 Score  : 0.9133333333333333
44/44 [==============================] - 2s 47ms/step - loss: 0.1152 - accuracy: 0.9614 - val_loss: 0.3106 - val_
accuracy: 0.9133 - val_f1_score: 0.9133
Epoch 80/100
43/44 [============================>.] - ETA: 0s - loss: 0.1022 - accuracy: 0.9644f1 Score  : 0.92
44/44 [==============================] - 2s 50ms/step - loss: 0.1024 - accuracy: 0.9643 - val_loss: 0.2745 - val_
accuracy: 0.9200 - val_f1_score: 0.9200
Epoch 81/100
43/44 [============================>.] - ETA: 0s - loss: 0.0659 - accuracy: 0.9767f1 Score  : 0.94
44/44 [==============================] - 2s 47ms/step - loss: 0.0680 - accuracy: 0.9757 - val_loss: 0.2565 - val_
accuracy: 0.9400 - val_f1_score: 0.9400
Epoch 82/100
43/44 [============================>.] - ETA: 0s - loss: 0.0994 - accuracy: 0.9637f1 Score  : 0.915
44/44 [==============================] - 2s 46ms/step - loss: 0.0983 - accuracy: 0.9643 - val_loss: 0.3028 - val_
accuracy: 0.9150 - val_f1_score: 0.9150
Epoch 83/100
43/44 [============================>.] - ETA: 0s - loss: 0.1075 - accuracy: 0.9622f1 Score  : 0.8983333333333333
44/44 [==============================] - 2s 50ms/step - loss: 0.1069 - accuracy: 0.9621 - val_loss: 0.3791 - val_
accuracy: 0.8983 - val_f1_score: 0.8983
Epoch 84/100
43/44 [============================>.] - ETA: 0s - loss: 0.1200 - accuracy: 0.9608f1 Score  : 0.9116666666666666
44/44 [==============================] - 2s 48ms/step - loss: 0.1195 - accuracy: 0.9607 - val_loss: 0.3322 - val_
accuracy: 0.9117 - val_f1_score: 0.9117
Epoch 85/100
43/44 [============================>.] - ETA: 0s - loss: 0.0914 - accuracy: 0.9702f1 Score  : 0.925
44/44 [==============================] - 2s 48ms/step - loss: 0.0900 - accuracy: 0.9707 - val_loss: 0.2866 - val_
accuracy: 0.9250 - val_f1_score: 0.9250
Epoch 86/100
43/44 [============================>.] - ETA: 0s - loss: 0.0820 - accuracy: 0.9789f1 Score  : 0.8983333333333333
44/44 [==============================] - 2s 50ms/step - loss: 0.0807 - accuracy: 0.9793 - val_loss: 0.3347 - val_
accuracy: 0.8983 - val_f1_score: 0.8983
Epoch 87/100
43/44 [============================>.] - ETA: 0s - loss: 0.1163 - accuracy: 0.9586f1 Score  : 0.8716666666666667
44/44 [==============================] - 2s 48ms/step - loss: 0.1146 - accuracy: 0.9593 - val_loss: 0.4377 - val_
accuracy: 0.8717 - val_f1_score: 0.8717
Epoch 88/100
43/44 [============================>.] - ETA: 0s - loss: 0.0969 - accuracy: 0.9702f1 Score  : 0.9133333333333333
44/44 [==============================] - 2s 48ms/step - loss: 0.0988 - accuracy: 0.9700 - val_loss: 0.3123 - val_
accuracy: 0.9133 - val_f1_score: 0.9133
Epoch 89/100
43/44 [============================>.] - ETA: 0s - loss: 0.1115 - accuracy: 0.9680f1 Score  : 0.9216666666666666
44/44 [==============================] - 2s 46ms/step - loss: 0.1099 - accuracy: 0.9686 - val_loss: 0.2961 - val_
accuracy: 0.9217 - val_f1_score: 0.9217
Epoch 90/100
43/44 [============================>.] - ETA: 0s - loss: 0.0503 - accuracy: 0.9840f1 Score  : 0.9133333333333333
44/44 [==============================] - 2s 48ms/step - loss: 0.0503 - accuracy: 0.9843 - val_loss: 0.3170 - val_
accuracy: 0.9133 - val_f1_score: 0.9133
Epoch 91/100
43/44 [============================>.] - ETA: 0s - loss: 0.0677 - accuracy: 0.9804f1 Score  : 0.92
44/44 [==============================] - 2s 48ms/step - loss: 0.0675 - accuracy: 0.9800 - val_loss: 0.2770 - val_
accuracy: 0.9200 - val_f1_score: 0.9200
Epoch 92/100
43/44 [============================>.] - ETA: 0s - loss: 0.0718 - accuracy: 0.9738f1 Score  : 0.9166666666666666
44/44 [==============================] - 2s 51ms/step - loss: 0.0722 - accuracy: 0.9736 - val_loss: 0.3271 - val_
accuracy: 0.9167 - val_f1_score: 0.9167
Epoch 93/100
43/44 [============================>.] - ETA: 0s - loss: 0.0926 - accuracy: 0.9666f1 Score  : 0.9383333333333334
44/44 [==============================] - 2s 49ms/step - loss: 0.0925 - accuracy: 0.9664 - val_loss: 0.2590 - val_
accuracy: 0.9383 - val_f1_score: 0.9383
Epoch 94/100
43/44 [============================>.] - ETA: 0s - loss: 0.1228 - accuracy: 0.9578f1 Score  : 0.8599999999999999
44/44 [==============================] - 2s 45ms/step - loss: 0.1249 - accuracy: 0.9564 - val_loss: 0.5153 - val_
accuracy: 0.8600 - val_f1_score: 0.8600
```

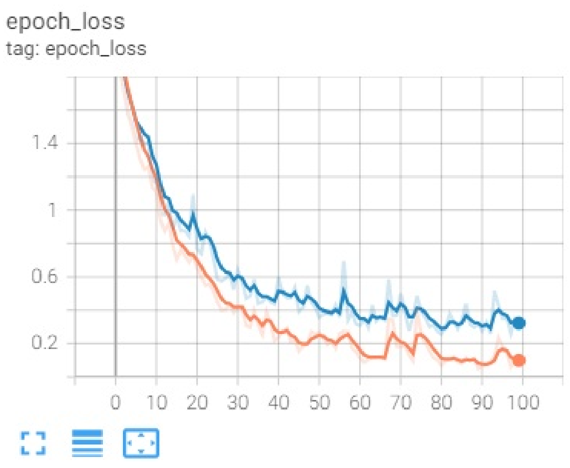
```
Epoch 95/100
43/44 [===========================>.] - ETA: 0s - loss: 0.2329 - accuracy: 0.9230f1 Score  : 0.885
44/44 [==============================] - 2s 49ms/step - loss: 0.2364 - accuracy: 0.9221 - val_loss: 0.4302 - val_
accuracy: 0.8850 - val_f1_score: 0.8850
Epoch 96/100
43/44 [===========================>.] - ETA: 0s - loss: 0.1852 - accuracy: 0.9360f1 Score  : 0.9116666666666666
44/44 [==============================] - 2s 48ms/step - loss: 0.1846 - accuracy: 0.9357 - val_loss: 0.3502 - val_
accuracy: 0.9117 - val_f1_score: 0.9117
Epoch 97/100
43/44 [===========================>.] - ETA: 0s - loss: 0.1458 - accuracy: 0.9440f1 Score  : 0.91
44/44 [==============================] - 2s 49ms/step - loss: 0.1437 - accuracy: 0.9450 - val_loss: 0.3494 - val_
accuracy: 0.9100 - val_f1_score: 0.9100
Epoch 98/100
43/44 [===========================>.] - ETA: 0s - loss: 0.0586 - accuracy: 0.9818f1 Score  : 0.9316666666666665
44/44 [==============================] - 2s 46ms/step - loss: 0.0599 - accuracy: 0.9807 - val_loss: 0.2595 - val_
accuracy: 0.9317 - val_f1_score: 0.9317
Epoch 99/100
43/44 [===========================>.] - ETA: 0s - loss: 0.0877 - accuracy: 0.9688f1 Score  : 0.9166666666666666
44/44 [==============================] - 2s 50ms/step - loss: 0.0880 - accuracy: 0.9679 - val_loss: 0.3471 - val_
accuracy: 0.9167 - val_f1_score: 0.9167
Epoch 100/100
43/44 [===========================>.] - ETA: 0s - loss: 0.0895 - accuracy: 0.9709f1 Score  : 0.9216666666666666
44/44 [==============================] - 2s 51ms/step - loss: 0.0888 - accuracy: 0.9714 - val_loss: 0.3063 - val_
accuracy: 0.9217 - val_f1_score: 0.9217
```

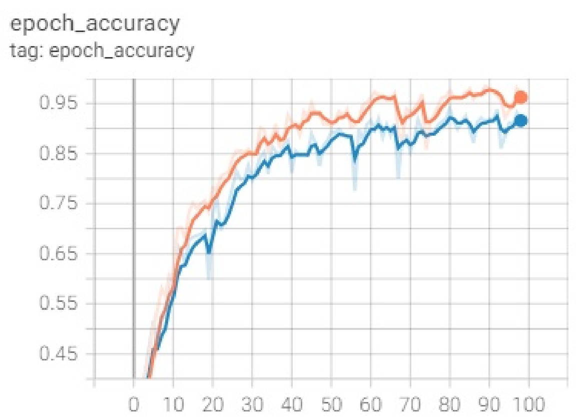
In [37]:

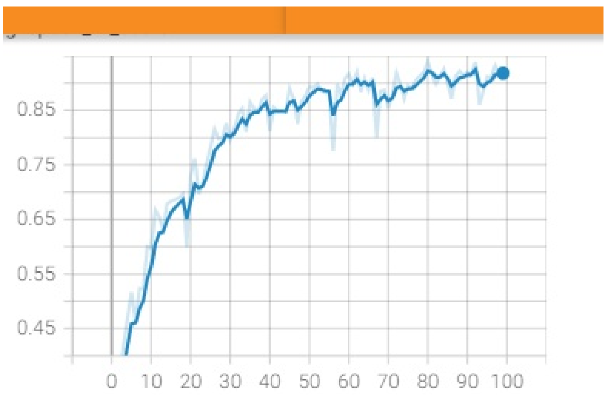
```
%load_ext tensorboard
%tensorboard --logdir "/content/logs/fit/model_220211110-103337"
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Model-2 Accuracy,F1Score and Loss







3. Data augmentation with raw features

```
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 [50]:
```
## 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=pitch_value)
            augmented_data.append(final_data)
    return augmented_data
```

In [51]:
```
temp_path = df_audio.iloc[0].path
aug_temp = generate_augmented_data(temp_path)
```

In [52]:
```
aug_temp[0]
```

Out[52]:
```
array([-0.01506544, -0.01777345, -0.01335487, ..., -0.00206719,
       -0.00166811, -0.00205901], dtype=float32)
```

In [53]:
```
type(aug_temp)
```

Out[53]:
```
list
```

In [54]:
```
len(aug_temp)
```

Out[54]:
```
9
```

Follow the steps

1. Split data 'df_audio' into train and test (80-20 split)

2. We have 2000 data points(1600 train points, 400 test points)

In [55]:
```
X_train, X_test, y_train, y_test=train_test_split(df_audio['path'],df_audio['label'],random_state=45,test_size=0.
```

1. Do augmentation only on X_train,pass each point of X_train to generate_augmented_data function.After augmentation we will get 14400 train points. Make sure that you are augmenting the corresponding class labels (y_train) also.
2. Preprocess your X_test using load_wav function.
3. Convert the augmented_train_data and test_data to numpy arrays.
4. Perform padding and masking on augmented_train_data and test_data.
5. After padding define the model similar to model 1 and fit the data

Note - While fitting your model on the augmented data for model 3 you might face Resource exhaust error. One simple hack to avoid that is save the augmented_train_data,augment_y_train,test_data and y_test to Drive or into your local system. Then restart the runtime so that now you can train your model with full RAM capacity. Upload these files again in the new runtime session perform padding and masking and then fit your model.

In [56]:
```
import tqdm
def data_agumentation(audio_data,labels):

    augmented_data,genetated_labels = [],[]

    for data in range(len(audio_data)):
        argu_generated_data = generate_augmented_data(audio_data[data])
        augmented_data.extend(argu_generated_data)
        genetated_labels.extend([labels[data]] * len(argu_generated_data))

    return (augmented_data,genetated_labels)
```

In [57]:
```
X_train.to_numpy()
```

Out[57]:
```
array(['/content/Speech_recorded_data/recordings/8_yweweler_46.wav',
       '/content/Speech_recorded_data/recordings/2_jackson_26.wav',
       '/content/Speech_recorded_data/recordings/7_yweweler_18.wav', ...,
       '/content/Speech_recorded_data/recordings/0_nicolas_12.wav',
       '/content/Speech_recorded_data/recordings/3_nicolas_11.wav',
       '/content/Speech_recorded_data/recordings/2_nicolas_38.wav'],
      dtype=object)
```

In [58]:
```
X_train,y_train = data_agumentation(X_train.to_numpy(),y_train.to_numpy())
X_train = np.array(X_train)
y_train = np.array(y_train)
print("After Agumentation dataset_length : {0}".format(len(X_train)))
```

```
After Agumentation dataset_length : 14400
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: VisibleDeprecationWarning: Creating an ndarray fr
om ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or sha
pes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
```

In [59]:
```
print("After Agumentation dataset_length : {0}".format(len(X_train)))
print("Shape of the agumeted Data : {}".format(X_train.shape))
```

```
After Agumentation dataset_length : 14400
Shape of the agumeted Data : (14400,)
```

In [65]:
```
X_train[0]
```

Out[65]:
```
array([ 1.09635206e-04,  1.83896977e-04,  2.19535970e-04, ...,
        3.21143627e-04, -2.62462981e-05, -2.69002427e-04], dtype=float32)
```

In [ ]:
```
X_test_spectrogram =
```

In [61]:
```
X_train_agumented = []
for data in X_train:
    X_train_agumented.append(data[0])
```

In [67]:
```
X_test_processed = preprocess_audio_data(X_test.values)
```

In [69]:
```
#https://www.tensorflow.org/guide/keras/masking_and_padding
#https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_length = 17640
## as discussed above, Pad with Zero if length of sequence is less than 17640 else Truncate the number.
## save in the X_train_pad_seq, X_test_pad_seq
X_train_pad_seq = pad_sequences(X_train,maxlen = max_length, dtype = 'float32',padding= 'post')
X_test_pad_seq = pad_sequences(X_test_processed['raw_data'].to_numpy(),maxlen = max_length, dtype='float32',paddi
X_train_mask,X_test_mask = (X_train_pad_seq != 0.0),(X_test_pad_seq != 0.0)
```

In [70]:
```
print("Shape of X_train Pad Sequence : {0}".format(X_train_pad_seq.shape))
```

```
Shape of X_train Pad Sequence : (14400, 17640)
```

In [71]:
```
import datetime
input = tf.keras.layers.Input(shape=(X_train_pad_seq.shape[1],1))
mask_input = tf.keras.layers.Input(shape=(X_train_mask.shape[1]),dtype='bool')
lstm_layer = tf.keras.layers.LSTM(32)
lstm_output = lstm_layer(input,mask = mask_input)
dense = tf.keras.layers.Dense(16,activation='relu')(lstm_output)
```

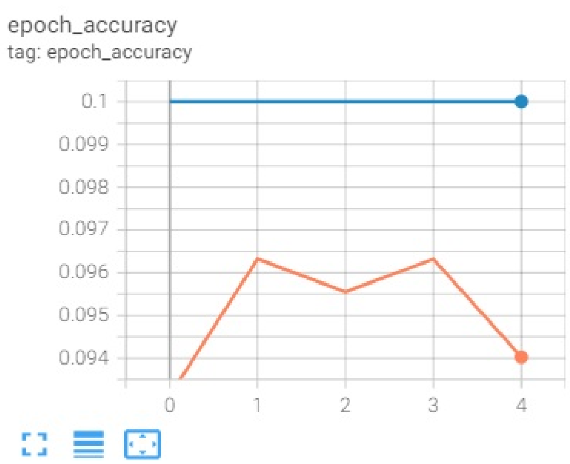
```
output = tf.keras.layers.Dense(10,activation = 'softmax')(dense)
model_3 = tf.keras.models.Model(inputs = [input ,mask_input], outputs = output)

#lstm input dimention is -> number_of_batches,time_stamps,Features
X_train_input = [ np.expand_dims(X_train_pad_seq,axis =2),X_train_mask]
X_test_input = [np.expand_dims(X_test_pad_seq,axis = 2),X_test_mask]


#https://stackoverflow.com/questions/44583254/valueerror-input-0-is-incompatible-with-layer-lstm-13-expected-ndim
f1_score_callback = F1Metrics((X_test_input,y_test))
early_stopping_call_backs = tf.keras.callbacks.EarlyStopping(patience= 2)
log_dir= log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
                            log_dir=log_dir, histogram_freq=1)
call_backs = [f1_score_callback,early_stopping_call_backs,tensorboard_callback]

model_3.compile(optimizer= tf.keras.optimizers.Adam( learning_rate=0.001),loss = tf.keras.losses.SparseCategorica
              )
model_history = model_3.fit(X_train_input,y_train,epochs = 5,validation_data=(X_test_input,y_test),batch_size = 3



#https://stackoverflow.com/questions/62839033/input-y-of-equal-op-has-type-bool-that-does-not-match-type-float32-
```

```
Epoch 1/5
450/450 [==============================] - ETA: 0s - loss: 2.3029 - accuracy: 0.0931f1 Score  : 0.100000000000000
02
450/450 [==============================] - 625s 1s/step - loss: 2.3029 - accuracy: 0.0931 - val_loss: 2.3026 - va
l_accuracy: 0.1000 - val_f1_score: 0.1000
Epoch 2/5
450/450 [==============================] - ETA: 0s - loss: 2.3028 - accuracy: 0.0963f1 Score  : 0.100000000000000
02
450/450 [==============================] - 617s 1s/step - loss: 2.3028 - accuracy: 0.0963 - val_loss: 2.3026 - va
l_accuracy: 0.1000 - val_f1_score: 0.1000
Epoch 3/5
450/450 [==============================] - ETA: 0s - loss: 2.3028 - accuracy: 0.0956f1 Score  : 0.100000000000000
02
450/450 [==============================] - 614s 1s/step - loss: 2.3028 - accuracy: 0.0956 - val_loss: 2.3026 - va
l_accuracy: 0.1000 - val_f1_score: 0.1000
Epoch 4/5
450/450 [==============================] - ETA: 0s - loss: 2.3028 - accuracy: 0.0963f1 Score  : 0.100000000000000
02
450/450 [==============================] - 614s 1s/step - loss: 2.3028 - accuracy: 0.0963 - val_loss: 2.3026 - va
l_accuracy: 0.1000 - val_f1_score: 0.1000
Epoch 5/5
450/450 [==============================] - ETA: 0s - loss: 2.3028 - accuracy: 0.0940f1 Score  : 0.100000000000000
02
450/450 [==============================] - 620s 1s/step - loss: 2.3028 - accuracy: 0.0940 - val_loss: 2.3026 - va
l_accuracy: 0.1000 - val_f1_score: 0.1000
```

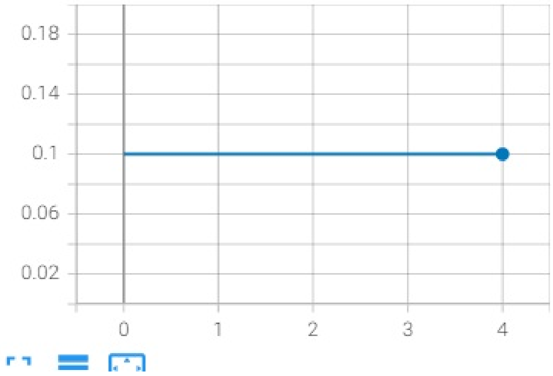
In [72]:

```
%load_ext tensorboard
%tensorboard --logdir "/content/logs/fit/20211110-081359"
```

Model - 3 Accuracy,Loss,F1 Score



epoch_f1_score
tag: epoch_f1_score



4. Data augmentation with spectogram data

1. use convert_to_spectrogram and convert the padded data from train and test data to spectogram data.
2. The shape of train data will be 14400 x 64 x 35 and shape of test_data will be 400 x 64 x35
3. Define the model similar to model 2 and fit the data

In [73]:

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

In [74]:

```
print("Shape of the X_train Spectrogram : {0}".format(X_train_spectrogram.shape))
print("Shape of the X_test Spectrogram : {0}".format(X_test_spectrogram.shape))
```

```
Shape of the X_train Spectrogram : (14400, 64, 35)
Shape of the X_test Spectrogram : (400, 64, 35)
```

In [77]:

```
# write the architecture of the model
tf.keras.backend.clear_session()
input_spectrum = tf.keras.layers.Input(shape=(X_train_spectrogram.shape[1],X_train_spectrogram.shape[2]))
lstm_layer = tf.keras.layers.LSTM(units = 64, return_sequences= True)(input_spectrum)
lstm_layer = tf.keras.layers.LSTM(units = 64, return_sequences= True)(lstm_layer)
global_average_pooling = tf.keras.layers.GlobalAvgPool1D()(lstm_layer)
dense_layer = tf.keras.layers.Dense(units = 64, activation = 'relu')(global_average_pooling)
dense_layer = tf.keras.layers.Dense(units = 32 , activation = 'relu')(dense_layer)
output_layer = tf.keras.layers.Dense(units= 10,activation = 'softmax')(dense_layer)
model_4 = tf.keras.models.Model(inputs = input_spectrum,outputs= output_layer)
model_4.summary()
#print model.summary and make sure that it is following point 2 mentioned above
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64, 35)]          0

 lstm (LSTM)                 (None, 64, 64)            25600

 lstm_1 (LSTM)               (None, 64, 64)            33024

 global_average_pooling1d (G  (None, 64)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 10)                330

=================================================================
Total params: 65,194
Trainable params: 65,194
Non-trainable params: 0
_________________________________________________________________
```

In [78]:

```
#compile and fit your model.
#model2.fit([X_train_spectrogram],y_train_int,......)
```

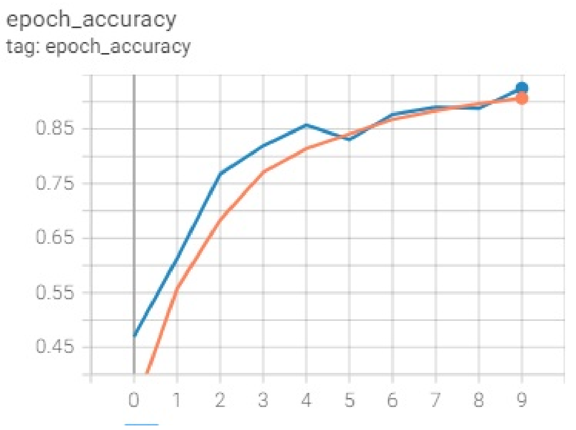
```
f1_score_callback = F1Metrics((X_test_spectrogram,y_test))
#early_stopping_call_backs = tf.keras.callbacks.EarlyStopping(patience= 2)
log_dir= log_dir = "logs/fit/model_4" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
                                log_dir=log_dir, histogram_freq=1)
call_backs = [f1_score_callback,tensorboard_callback]

model_4.compile(optimizer= tf.keras.optimizers.Adam( learning_rate=0.001),loss = tf.keras.losses.SparseCategorica
model_history = model_4.fit(X_train_spectrogram,y_train,epochs = 10,validation_data=(X_test_spectrogram,y_test),b
```

```
Epoch 1/10
  5/450 [..............................] - ETA: 15s - loss: 2.2965 - accuracy: 0.0875WARNING:tensorflow:Callback
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0097s vs `on_train_batch_end` time:
0.0405s). Check your callbacks.
449/450 [============================>.] - ETA: 0s - loss: 1.7627 - accuracy: 0.3339f1 Score  : 0.47
450/450 [==============================] - 22s 41ms/step - loss: 1.7619 - accuracy: 0.3344 - val_loss: 1.3490 - v
al_accuracy: 0.4700 - val_f1_score: 0.4700
Epoch 2/10
449/450 [============================>.] - ETA: 0s - loss: 1.1735 - accuracy: 0.5606f1 Score  : 0.615
450/450 [==============================] - 16s 36ms/step - loss: 1.1733 - accuracy: 0.5608 - val_loss: 1.0518 - v
al_accuracy: 0.6150 - val_f1_score: 0.6150
Epoch 3/10
449/450 [============================>.] - ETA: 0s - loss: 0.8380 - accuracy: 0.6851f1 Score  : 0.769999999999999
9
450/450 [==============================] - 17s 37ms/step - loss: 0.8378 - accuracy: 0.6851 - val_loss: 0.6744 - v
al_accuracy: 0.7700 - val_f1_score: 0.7700
Epoch 4/10
449/450 [============================>.] - ETA: 0s - loss: 0.6181 - accuracy: 0.7722f1 Score  : 0.82
450/450 [==============================] - 16s 36ms/step - loss: 0.6181 - accuracy: 0.7722 - val_loss: 0.5097 - v
al_accuracy: 0.8200 - val_f1_score: 0.8200
Epoch 5/10
449/450 [============================>.] - ETA: 0s - loss: 0.5160 - accuracy: 0.8142f1 Score  : 0.8575
450/450 [==============================] - 16s 36ms/step - loss: 0.5157 - accuracy: 0.8145 - val_loss: 0.4229 - v
al_accuracy: 0.8575 - val_f1_score: 0.8575
Epoch 6/10
449/450 [============================>.] - ETA: 0s - loss: 0.4472 - accuracy: 0.8413f1 Score  : 0.83
450/450 [==============================] - 16s 36ms/step - loss: 0.4472 - accuracy: 0.8414 - val_loss: 0.4702 - v
al_accuracy: 0.8300 - val_f1_score: 0.8300
Epoch 7/10
449/450 [============================>.] - ETA: 0s - loss: 0.3790 - accuracy: 0.8681f1 Score  : 0.8775
450/450 [==============================] - 16s 37ms/step - loss: 0.3792 - accuracy: 0.8681 - val_loss: 0.3204 - v
al_accuracy: 0.8775 - val_f1_score: 0.8775
Epoch 8/10
450/450 [==============================] - ETA: 0s - loss: 0.3411 - accuracy: 0.8838f1 Score  : 0.89
450/450 [==============================] - 17s 37ms/step - loss: 0.3411 - accuracy: 0.8838 - val_loss: 0.3577 - v
al_accuracy: 0.8900 - val_f1_score: 0.8900
Epoch 9/10
450/450 [==============================] - ETA: 0s - loss: 0.2949 - accuracy: 0.8962f1 Score  : 0.8875
450/450 [==============================] - 17s 37ms/step - loss: 0.2949 - accuracy: 0.8962 - val_loss: 0.3069 - v
al_accuracy: 0.8875 - val_f1_score: 0.8875
Epoch 10/10
449/450 [============================>.] - ETA: 0s - loss: 0.2721 - accuracy: 0.9067f1 Score  : 0.925
450/450 [==============================] - 17s 37ms/step - loss: 0.2721 - accuracy: 0.9067 - val_loss: 0.2686 - v
al_accuracy: 0.9250 - val_f1_score: 0.9250
```

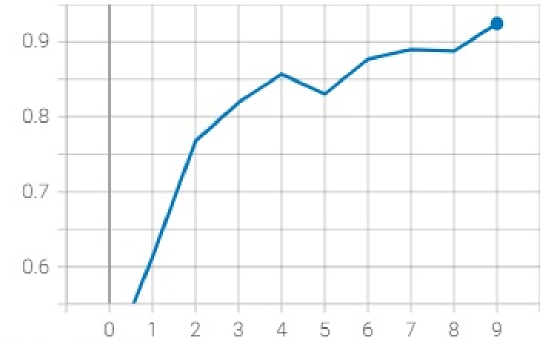
In [79]:

```
%tensorboard --logdir '/content/logs/fit/model_420211110-092309'
```

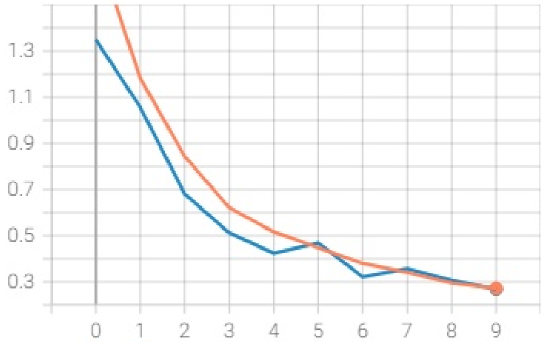
Model - 4 Accuracy , F1 Score and Loss



epoch_f1_score
tag: epoch_f1_score



epoch_loss
tag: epoch_loss



Conclusion

In [4]:

```
from prettytable import PrettyTable
table = PrettyTable(['Model','Optimizer','Accuracy/f1_Score'])
table.add_row(['RNN','ADAM','10.002'])
table.add_row(['RNN','ADAM','92.47'])
table.add_row(['RNN','ADAM','10.00'])
table.add_row(['RNN','ADAM','92.50'])
```

In [5]:

```
print(table)
```

```
+-------+-----------+-------------------+
| Model | Optimizer | Accuracy/f1_Score |
+-------+-----------+-------------------+
|  RNN  |    ADAM   |       10.002      |
|  RNN  |    ADAM   |       92.47       |
|  RNN  |    ADAM   |       10.00       |
|  RNN  |    ADAM   |       92.50       |
+-------+-----------+-------------------+
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js