Speech_detection_Assignment

April 22, 2021

1 Modules

[99]:

```
import numpy as np
import pandas as pd
import librosa
import os
from sklearn.model_selection import train_test_split
import seaborn
import tensorboard
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import sklearn
from tensorflow.keras.layers import Input, LSTM, Dense
from tensorflow.keras.models import Model
import tensorflow as tf
from tensorflow.keras.initializers import he_normal, he_uniform
from tensorflow.keras.regularizers import l1, l2
from sklearn.preprocessing  import Normalizer, MinMaxScaler
from sklearn.utils import shuffle

from google.colab import drive
drive.mount('/content/drive')
##if you need any imports you can do that here.
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

We shared recordings.zip, please unzip those.

2 Reading Data

[100]:

```
recording_path = "/content/drive/MyDrive/Colab Notebooks/Spoken Digit␣
 ↪Recognization/recordings/"
```

[101]:

```
#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"
```

```
all_files=[]
all_files_Names = []
files = os.listdir(recording_path)

for file in files:
  all_files_Names.append(file)
  file = recording_path + file
  all_files.append(file)
```

Grader function 1

[102]:
```
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()
```

[102]:
```
True
```

[103]:
```
all_files[0], all_files_Names[0]
```

[103]:
```
('/content/drive/MyDrive/Colab Notebooks/Spoken Digit
Recognization/recordings/4_theo_20.wav',
 '4_theo_20.wav')
```

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

[104]:
```
#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
labels = []

for file in all_files_Names:
  labels.append(file.split("_")[0])


data = {"path":all_files, "label":labels}
df_audio = pd.DataFrame(data)
```

[105]:
```
#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

[106]:

```
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()
```

[106]:

```
True
```

[107]:

```
df_audio = shuffle(df_audio, random_state=33)#don't change the random state
```

3 With out augmentaion

3.0.1 Train and Test Split

[108]:

```
#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%
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

[109]:

```
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()
```

[109]:
```
True
```

3.1 Preprocessing

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

3.1.1 Raw audio file extraction

[110]:
```
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
```

[ ]:
```
#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) with same index of X_train/y_train

X_train_preprocessed_samples = []
X_train_preprocessed_duration = []

X_test_preprocessed_samples = []
X_test_preprocessed_duration = []

for row in X_train:
  samples, duration = load_wav(row, get_duration=True)
  X_train_preprocessed_samples.append(samples)
  X_train_preprocessed_duration.append(duration)
  print(row)

for row in X_test:
  samples, duration = load_wav(row, get_duration=True)
  X_test_preprocessed_samples.append(samples)
  X_test_preprocessed_duration.append(duration)
  print(row)
```

```
[112]: data_train = {"raw_data":X_train_preprocessed_samples, "duration":
        ↪X_train_preprocessed_duration}
       X_train_preprocessed = pd.DataFrame(data_train)

       data_test = {"raw_data":X_test_preprocessed_samples, "duration":
        ↪X_test_preprocessed_duration}
       X_test_preprocessed = pd.DataFrame(data_test)
```

```
[113]: X_train_preprocessed.head(1)
```

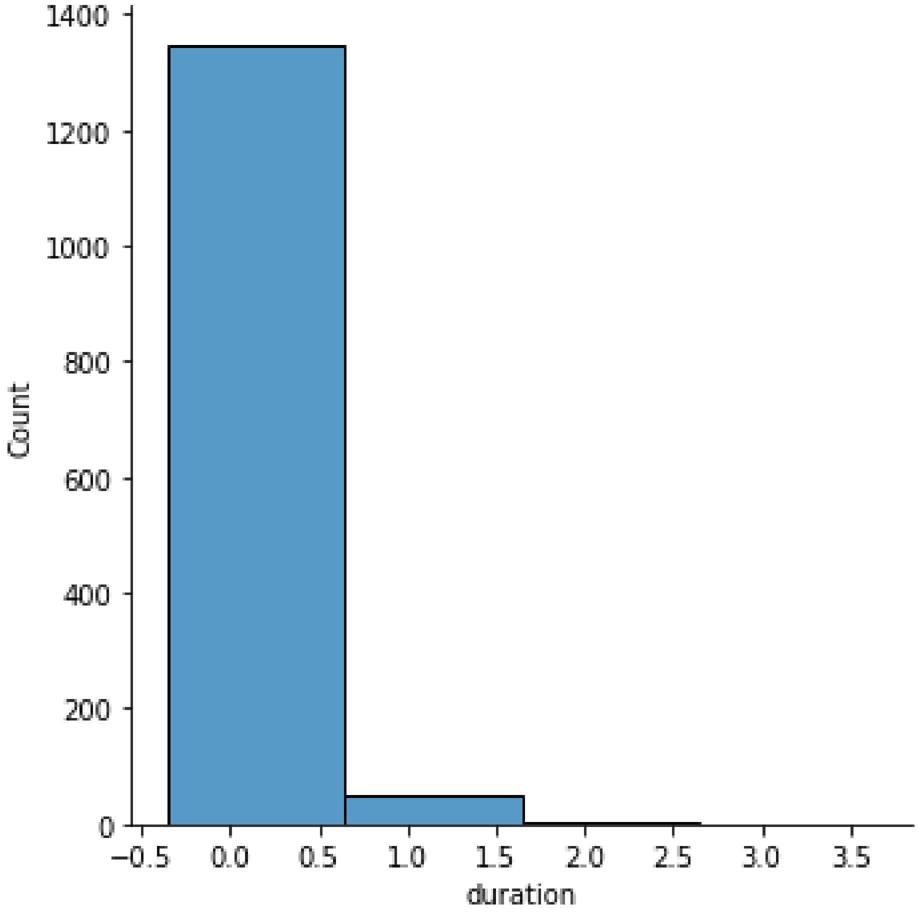
```
[113]:                                             raw_data  duration
       0  [-6.593454e-05, -5.2746916e-05, -1.7588934e-05…  0.271882
```

```
[114]: X_test_preprocessed.head(1)
```

```
[114]:                                             raw_data  duration
       0  [-3.8278453e-05, -0.00015818808, -0.0002546222…  0.293152
```

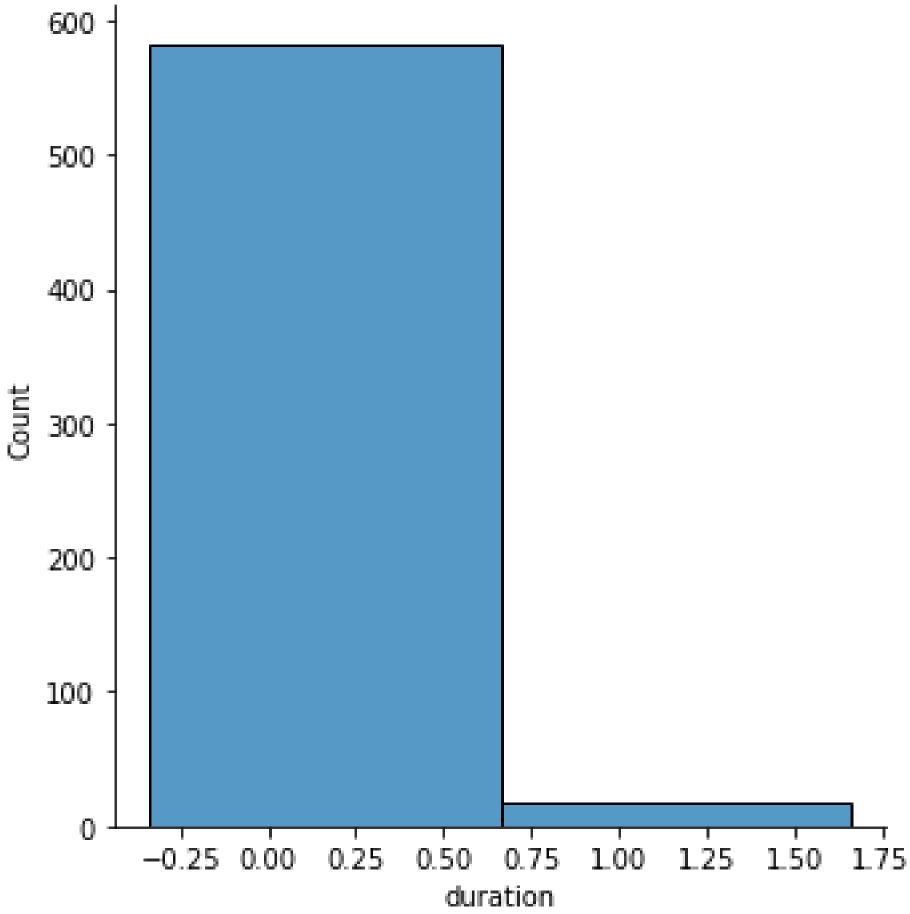
```
[115]: #plot the histogram of the duration for trian
       a = X_train_preprocessed
       seaborn.displot(a, x="duration", discrete=True)
```

```
[115]: <seaborn.axisgrid.FacetGrid at 0x7f39e2456f28>
```



```
[116]: #plot the histogram of the duration for trian
       a = X_test_preprocessed
       seaborn.displot(a, x="duration", discrete=True)
```

```
[116]: <seaborn.axisgrid.FacetGrid at 0x7f39e22d2668>
```



[117]:

```
#print 0 to 100 percentile values with step size of 10 for train data duration.
for i in range(0,101,10):
  print(i,np.percentile(X_train_preprocessed['duration'], i))
```

```
0 0.1435374149659864
10 0.25958730158730153
20 0.2984308390022676
30 0.33215419501133786
40 0.3596371882086168
50 0.39034013605442175
60 0.4151746031746032
70 0.4448027210884353
80 0.48297505668934243
90 0.5517777777777779
100 2.282766439909297
```

[118]:

```
##print 90 to 100 percentile values with step size of 1.
for i in range(90,101,1):
  print(i,np.percentile(X_train_preprocessed['duration'], i))
```

```
90 0.5517777777777779
91 0.5654417233560091
92 0.5790349206349206
93 0.5941251700680278
94 0.6101723356009069
95 0.6230884353741496
96 0.6388244897959183
97 0.6611179138321994
98 0.6956090702947844
99 0.7961165532879818
100 2.282766439909297
```

Grader function 4

[119]:
```
X_train_processed = X_train_preprocessed
X_test_processed = X_test_preprocessed
```

[120]:
```
def grader_processed():
    flag_columns = (all(X_train_processed.columns==['raw_data', 'duration']))␣
 ↪and (all(X_test_processed.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()
```

[120]:
```
True
```

3.1.2 Mask and Pad raw audio data

[121]:
```
max_length  = 17640
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Masking, Embedding
```

[122]:
```
def mask_seq(seq):
  seq_masked_list = []
  for row in seq:
    masks = []
    for item in row:
      if item == 0:
        masks.append(False)
      else:
        masks.append(True)
    seq_masked_list.append(masks)

  return np.array(seq_masked_list)
```

[123]:

```
## 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

## all the X_train_pad_seq, X_test_pad_seq, X_train_mask, X_test_mask will be␣
 ↪numpy arrays mask vector dtype must be bool.

X_train_pad_seq = pad_sequences(sequences=X_train_processed['raw_data'].values,␣
 ↪maxlen=max_length, dtype='float32', padding="post")
X_train_mask = mask_seq(X_train_pad_seq)

X_test_pad_seq = pad_sequences(sequences=X_test_processed['raw_data'].values,␣
 ↪maxlen=max_length, dtype='float32', padding="post")
X_test_mask = mask_seq(X_test_pad_seq)

X_train_pad_seq.shape, np.array(X_train_mask).shape, X_test_pad_seq.shape,␣
 ↪X_test_mask.shape
```

[123]:

```
((1400, 17640), (1400, 17640), (600, 17640), (600, 17640))
```

Grader function 5

[124]:

```
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()
```

[124]:

```
True
```

3.1.3 Giving Raw data directly. (Train LSTM model with raw audio data) (MODEL - 1)

[ ]:

```

```

[ ]:

```
# X_train_pad_seq_normalized = sklearn.preprocessing.normalize(X_train_pad_seq)
# X_test_pad_seq_normalized = sklearn.preprocessing.normalize(X_test_pad_seq)

X_train_pad_seq_normalized = (X_train_pad_seq - np.min(X_train_pad_seq)) / (np.
 ↪max(X_train_pad_seq) - np.min(X_train_pad_seq))
X_test_pad_seq_normalized = (X_test_pad_seq - np.min(X_test_pad_seq)) / (np.
 ↪max(X_test_pad_seq) - np.min(X_test_pad_seq))
```

[ ]:
```
X_train_pad_seq_expanded = np.expand_dims(X_train_pad_seq, axis=2)
X_test_pad_seq_expanded = np.expand_dims(X_test_pad_seq, axis=2)
```

[ ]:
```
X_train_pad_seq_expanded.shape, X_test_pad_seq_expanded.shape, y_train.shape,␣
 ↪y_test.shape
```

[ ]:
```
((1400, 17640, 1), (600, 17640, 1), (1400,), (600,))
```

[ ]:
```
class LSTM_Layer(tf.keras.layers.Layer):

    def __init__(self, **kwargs):
        super(LSTM_Layer, self).__init__(**kwargs)
        self.lstm          = LSTM(25) # recurrent_activation=tf.keras.layers.
 ↪LeakyReLU(alpha=0.5)


    def call(self, inputs, masked_inputs):
      output= self.lstm(inputs, mask=masked_inputs)
      return output




lstm_layer = LSTM_Layer()
```

[ ]:
```
tf.keras.backend.clear_session()

input_padded = Input(shape=(17640,1), name="padded_input_layer",␣
 ↪dtype="float32")
input_masked = Input(shape=(17640,), name="masked_input_layer", dtype="bool")
lstm_out     = lstm_layer(input_padded, input_masked)
dense_1_out  = Dense(units=50,
                     activation=tf.keras.layers.LeakyReLU(alpha=0.5),
                     kernel_initializer=he_normal(),
                     kernel_regularizer=l1(0.0001),
                     activity_regularizer=l1(0.0001)) (lstm_out)
output       = Dense(10, activation='softmax') (dense_1_out)

model = Model(inputs=[input_padded, input_masked], outputs=output)
model.summary()
```

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

```
--------------------------------------------------------------------------------
------------------
masked_input_layer (InputLayer) [(None, 17640)]      0

--------------------------------------------------------------------------------
------------------
lstm__layer (LSTM_Layer)        (None, 25)           2700
padded_input_layer[0][0]
masked_input_layer[0][0]

--------------------------------------------------------------------------------
------------------
dense (Dense)                   (None, 50)           1300
lstm__layer[2][0]

--------------------------------------------------------------------------------
------------------
dense_1 (Dense)                 (None, 10)           510         dense[0][0]
================================================================================
==================
Total params: 4,510
Trainable params: 4,510
Non-trainable params: 0

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

[ ]:

```

```

[ ]:

```python
label_encoder = LabelEncoder()
y_train_label_encoded = label_encoder.fit_transform(y_train)
y_test_label_encoded = label_encoder.transform(y_test)
```

[ ]:

```python
class CustomCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs=None):
        keys = list(logs.keys())
        y_pred = model.predict([X_test_pad_seq_expanded, X_test_mask])
        y_pred_list = []
        for i in y_pred:
          y_pred_list.append(tf.argmax(i))

        print("\nEpoch {}  micro-F1 score {}\n".format(epoch, sklearn.metrics.
 ↪f1_score(y_test_label_encoded, y_pred_list, average='micro')) )
```

[ ]:

```python
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              metrics=['accuracy'],
              loss=tf.keras.losses.SparseCategoricalCrossentropy())
```

```
!rm -rf logs/fit/model_1
log_dir="logs/fit/model_1"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                                                      histogram_freq=1,
                                                      write_graph=True,
                                                      write_grads=True)

earlystopping_callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss",␣
 ↪patience=2, verbose=1, mode='min')

myCallback = CustomCallback()

model.fit(x=[X_train_pad_seq_expanded, X_train_mask],
          y=y_train_label_encoded,
          batch_size=100,
          epochs=70,
          verbose=1,
          validation_data=([X_test_pad_seq_expanded, X_test_mask],␣
 ↪y_test_label_encoded),
          callbacks=[myCallback, tensorboard_callback, earlystopping_callback]
          )
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the
`TensorBoard` Callback.
Epoch 1/70
 2/14 [===>…] - ETA: 28s - loss: 2.3364 - accuracy:
0.0850WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.9491s vs `on_train_batch_end` time: 3.7974s).
Check your callbacks.
14/14 [==============================] - ETA: 0s - loss: 2.3332 - accuracy:
0.0857
Epoch 0  micro-F1 score 0.10000000000000002

14/14 [==============================] - 22s 2s/step - loss: 2.3332 - accuracy:
0.0857 - val_loss: 2.3318 - val_accuracy: 0.1000
Epoch 2/70
14/14 [==============================] - ETA: 0s - loss: 2.3310 - accuracy:
0.0986
Epoch 1  micro-F1 score 0.095

14/14 [==============================] - 16s 1s/step - loss: 2.3310 - accuracy:
0.0986 - val_loss: 2.3308 - val_accuracy: 0.0950
Epoch 3/70
14/14 [==============================] - ETA: 0s - loss: 2.3298 - accuracy:
0.1036
Epoch 2  micro-F1 score 0.09666666666666666
```

```
14/14 [==============================] - 16s 1s/step - loss: 2.3298 - accuracy:
0.1036 - val_loss: 2.3301 - val_accuracy: 0.0967
Epoch 4/70
14/14 [==============================] - ETA: 0s - loss: 2.3289 - accuracy:
0.1000
Epoch 3  micro-F1 score 0.095

14/14 [==============================] - 16s 1s/step - loss: 2.3289 - accuracy:
0.1000 - val_loss: 2.3293 - val_accuracy: 0.0950
Epoch 5/70
14/14 [==============================] - ETA: 0s - loss: 2.3282 - accuracy:
0.1071
Epoch 4  micro-F1 score 0.08666666666666667

14/14 [==============================] - 16s 1s/step - loss: 2.3282 - accuracy:
0.1071 - val_loss: 2.3286 - val_accuracy: 0.0867
Epoch 6/70
14/14 [==============================] - ETA: 0s - loss: 2.3272 - accuracy:
0.1021
Epoch 5  micro-F1 score 0.09333333333333334

14/14 [==============================] - 16s 1s/step - loss: 2.3272 - accuracy:
0.1021 - val_loss: 2.3281 - val_accuracy: 0.0933
Epoch 7/70
14/14 [==============================] - ETA: 0s - loss: 2.3263 - accuracy:
0.1000
Epoch 6  micro-F1 score 0.09166666666666666

14/14 [==============================] - 16s 1s/step - loss: 2.3263 - accuracy:
0.1000 - val_loss: 2.3276 - val_accuracy: 0.0917
Epoch 8/70
14/14 [==============================] - ETA: 0s - loss: 2.3252 - accuracy:
0.1036
Epoch 7  micro-F1 score 0.08333333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3252 - accuracy:
0.1036 - val_loss: 2.3289 - val_accuracy: 0.0833
Epoch 9/70
14/14 [==============================] - ETA: 0s - loss: 2.3240 - accuracy:
0.1021
Epoch 8  micro-F1 score 0.10333333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3240 - accuracy:
0.1021 - val_loss: 2.3261 - val_accuracy: 0.1033
Epoch 10/70
14/14 [==============================] - ETA: 0s - loss: 2.3237 - accuracy:
0.0986
Epoch 9  micro-F1 score 0.11166666666666666
```

```
14/14 [==============================] - 16s 1s/step - loss: 2.3237 - accuracy:
0.0986 - val_loss: 2.3256 - val_accuracy: 0.1117
Epoch 11/70
14/14 [==============================] - ETA: 0s - loss: 2.3225 - accuracy:
0.1107
Epoch 10  micro-F1 score 0.10000000000000002

14/14 [==============================] - 16s 1s/step - loss: 2.3225 - accuracy:
0.1107 - val_loss: 2.3255 - val_accuracy: 0.1000
Epoch 12/70
14/14 [==============================] - ETA: 0s - loss: 2.3203 - accuracy:
0.1021
Epoch 11  micro-F1 score 0.10666666666666669

14/14 [==============================] - 16s 1s/step - loss: 2.3203 - accuracy:
0.1021 - val_loss: 2.3279 - val_accuracy: 0.1067
Epoch 13/70
14/14 [==============================] - ETA: 0s - loss: 2.3193 - accuracy:
0.1079
Epoch 12  micro-F1 score 0.08833333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3193 - accuracy:
0.1079 - val_loss: 2.3241 - val_accuracy: 0.0883
Epoch 14/70
14/14 [==============================] - ETA: 0s - loss: 2.3208 - accuracy:
0.1014
Epoch 13  micro-F1 score 0.09166666666666666

14/14 [==============================] - 16s 1s/step - loss: 2.3208 - accuracy:
0.1014 - val_loss: 2.3233 - val_accuracy: 0.0917
Epoch 15/70
14/14 [==============================] - ETA: 0s - loss: 2.3199 - accuracy:
0.1043
Epoch 14  micro-F1 score 0.11666666666666667

14/14 [==============================] - 16s 1s/step - loss: 2.3199 - accuracy:
0.1043 - val_loss: 2.3227 - val_accuracy: 0.1167
Epoch 16/70
14/14 [==============================] - ETA: 0s - loss: 2.3185 - accuracy:
0.0971
Epoch 15  micro-F1 score 0.10333333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3185 - accuracy:
0.0971 - val_loss: 2.3217 - val_accuracy: 0.1033
Epoch 17/70
14/14 [==============================] - ETA: 0s - loss: 2.3210 - accuracy:
0.1079
```

```
Epoch 16  micro-F1 score 0.08833333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3210 - accuracy:
0.1079 - val_loss: 2.3225 - val_accuracy: 0.0883
Epoch 18/70
14/14 [==============================] - ETA: 0s - loss: 2.3202 - accuracy:
0.1014
Epoch 17  micro-F1 score 0.10000000000000002

14/14 [==============================] - 16s 1s/step - loss: 2.3202 - accuracy:
0.1014 - val_loss: 2.3216 - val_accuracy: 0.1000
Epoch 19/70
14/14 [==============================] - ETA: 0s - loss: 2.3198 - accuracy:
0.0964
Epoch 18  micro-F1 score 0.08666666666666667

14/14 [==============================] - 16s 1s/step - loss: 2.3198 - accuracy:
0.0964 - val_loss: 2.3206 - val_accuracy: 0.0867
Epoch 20/70
14/14 [==============================] - ETA: 0s - loss: 2.3189 - accuracy:
0.1043
Epoch 19  micro-F1 score 0.09

14/14 [==============================] - 16s 1s/step - loss: 2.3189 - accuracy:
0.1043 - val_loss: 2.3200 - val_accuracy: 0.0900
Epoch 21/70
14/14 [==============================] - ETA: 0s - loss: 2.3180 - accuracy:
0.1100
Epoch 20  micro-F1 score 0.08833333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3180 - accuracy:
0.1100 - val_loss: 2.3195 - val_accuracy: 0.0883
Epoch 22/70
14/14 [==============================] - ETA: 0s - loss: 2.3176 - accuracy:
0.1150
Epoch 21  micro-F1 score 0.09666666666666666

14/14 [==============================] - 16s 1s/step - loss: 2.3176 - accuracy:
0.1150 - val_loss: 2.3191 - val_accuracy: 0.0967
Epoch 23/70
14/14 [==============================] - ETA: 0s - loss: 2.3170 - accuracy:
0.1157
Epoch 22  micro-F1 score 0.09666666666666666

14/14 [==============================] - 16s 1s/step - loss: 2.3170 - accuracy:
0.1157 - val_loss: 2.3187 - val_accuracy: 0.0967
Epoch 24/70
14/14 [==============================] - ETA: 0s - loss: 2.3165 - accuracy:
```

```
0.1079
Epoch 23  micro-F1 score 0.10499999999999998

14/14 [==============================] - 16s 1s/step - loss: 2.3165 - accuracy:
0.1079 - val_loss: 2.3184 - val_accuracy: 0.1050
Epoch 25/70
14/14 [==============================] - ETA: 0s - loss: 2.3158 - accuracy:
0.1043
Epoch 24  micro-F1 score 0.09833333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3158 - accuracy:
0.1043 - val_loss: 2.3181 - val_accuracy: 0.0983
Epoch 26/70
14/14 [==============================] - ETA: 0s - loss: 2.3154 - accuracy:
0.1107
Epoch 25  micro-F1 score 0.10666666666666669

14/14 [==============================] - 16s 1s/step - loss: 2.3154 - accuracy:
0.1107 - val_loss: 2.3178 - val_accuracy: 0.1067
Epoch 27/70
14/14 [==============================] - ETA: 0s - loss: 2.3149 - accuracy:
0.1021
Epoch 26  micro-F1 score 0.095

14/14 [==============================] - 16s 1s/step - loss: 2.3149 - accuracy:
0.1021 - val_loss: 2.3175 - val_accuracy: 0.0950
Epoch 28/70
14/14 [==============================] - ETA: 0s - loss: 2.3143 - accuracy:
0.1029
Epoch 27  micro-F1 score 0.10333333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3143 - accuracy:
0.1029 - val_loss: 2.3173 - val_accuracy: 0.1033
Epoch 29/70
14/14 [==============================] - ETA: 0s - loss: 2.3138 - accuracy:
0.1071
Epoch 28  micro-F1 score 0.10333333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3138 - accuracy:
0.1071 - val_loss: 2.3170 - val_accuracy: 0.1033
Epoch 30/70
14/14 [==============================] - ETA: 0s - loss: 2.3134 - accuracy:
0.1036
Epoch 29  micro-F1 score 0.10666666666666669

14/14 [==============================] - 16s 1s/step - loss: 2.3134 - accuracy:
0.1036 - val_loss: 2.3168 - val_accuracy: 0.1067
Epoch 31/70
```

```
14/14 [==============================] - ETA: 0s - loss: 2.3127 - accuracy:
0.1021
Epoch 30  micro-F1 score 0.10833333333333334

14/14 [==============================] - 16s 1s/step - loss: 2.3127 - accuracy:
0.1021 - val_loss: 2.3165 - val_accuracy: 0.1083
Epoch 32/70
14/14 [==============================] - ETA: 0s - loss: 2.3120 - accuracy:
0.1057
Epoch 31  micro-F1 score 0.11333333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3120 - accuracy:
0.1057 - val_loss: 2.3163 - val_accuracy: 0.1133
Epoch 33/70
14/14 [==============================] - ETA: 0s - loss: 2.3114 - accuracy:
0.1050
Epoch 32  micro-F1 score 0.11333333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3114 - accuracy:
0.1050 - val_loss: 2.3161 - val_accuracy: 0.1133
Epoch 34/70
14/14 [==============================] - ETA: 0s - loss: 2.3108 - accuracy:
0.1086
Epoch 33  micro-F1 score 0.11166666666666666

14/14 [==============================] - 16s 1s/step - loss: 2.3108 - accuracy:
0.1086 - val_loss: 2.3160 - val_accuracy: 0.1117
Epoch 35/70
14/14 [==============================] - ETA: 0s - loss: 2.3099 - accuracy:
0.1186
Epoch 34  micro-F1 score 0.11833333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3099 - accuracy:
0.1186 - val_loss: 2.3158 - val_accuracy: 0.1183
Epoch 36/70
14/14 [==============================] - ETA: 0s - loss: 2.3093 - accuracy:
0.1086
Epoch 35  micro-F1 score 0.10499999999999998

14/14 [==============================] - 16s 1s/step - loss: 2.3093 - accuracy:
0.1086 - val_loss: 2.3157 - val_accuracy: 0.1050
Epoch 37/70
14/14 [==============================] - ETA: 0s - loss: 2.3085 - accuracy:
0.1064
Epoch 36  micro-F1 score 0.11

14/14 [==============================] - 16s 1s/step - loss: 2.3085 - accuracy:
0.1064 - val_loss: 2.3155 - val_accuracy: 0.1100
```

```
Epoch 38/70
14/14 [==============================] - ETA: 0s - loss: 2.3077 - accuracy:
0.0993
Epoch 37  micro-F1 score 0.11

14/14 [==============================] - 16s 1s/step - loss: 2.3077 - accuracy:
0.0993 - val_loss: 2.3152 - val_accuracy: 0.1100
Epoch 39/70
14/14 [==============================] - ETA: 0s - loss: 2.3068 - accuracy:
0.1093
Epoch 38  micro-F1 score 0.095

14/14 [==============================] - 16s 1s/step - loss: 2.3068 - accuracy:
0.1093 - val_loss: 2.3143 - val_accuracy: 0.0950
Epoch 40/70
14/14 [==============================] - ETA: 0s - loss: 2.3052 - accuracy:
0.1107
Epoch 39  micro-F1 score 0.10833333333333334

14/14 [==============================] - 16s 1s/step - loss: 2.3052 - accuracy:
0.1107 - val_loss: 2.3140 - val_accuracy: 0.1083
Epoch 41/70
14/14 [==============================] - ETA: 0s - loss: 2.3052 - accuracy:
0.1186
Epoch 40  micro-F1 score 0.125

14/14 [==============================] - 16s 1s/step - loss: 2.3052 - accuracy:
0.1186 - val_loss: 2.3137 - val_accuracy: 0.1250
Epoch 42/70
14/14 [==============================] - ETA: 0s - loss: 2.3033 - accuracy:
0.1171
Epoch 41  micro-F1 score 0.10666666666666669

14/14 [==============================] - 16s 1s/step - loss: 2.3033 - accuracy:
0.1171 - val_loss: 2.3134 - val_accuracy: 0.1067
Epoch 43/70
14/14 [==============================] - ETA: 0s - loss: 2.3037 - accuracy:
0.1121
Epoch 42  micro-F1 score 0.09833333333333333

14/14 [==============================] - 16s 1s/step - loss: 2.3037 - accuracy:
0.1121 - val_loss: 2.3163 - val_accuracy: 0.0983
Epoch 44/70
14/14 [==============================] - ETA: 0s - loss: 2.3115 - accuracy:
0.0964
Epoch 43  micro-F1 score 0.10000000000000002

14/14 [==============================] - 16s 1s/step - loss: 2.3115 - accuracy:
```

```
0.0964 - val_loss: 2.3157 - val_accuracy: 0.1000
Epoch 00044: early stopping
```

```
[ ]: <tensorflow.python.keras.callbacks.History at 0x7f39e7058908>
```

```
[ ]: %load_ext tensorboard
     %tensorboard --logdir logs/fit
```

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

```
Reusing TensorBoard on port 6006 (pid 26053), started 0:16:26 ago. (Use '!kill␣
↪26053' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

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

```
[ ]: 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
```

```
[ ]: ##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)
     X_train_spectrogram = []
     X_test_spectrogram = []

     for row in X_train_pad_seq:
       spectrum = convert_to_spectrogram(row)
       X_train_spectrogram.append(spectrum)

     for row in X_test_pad_seq:
       spectrum = convert_to_spectrogram(row)
       X_test_spectrogram.append(spectrum)


     X_train_spectrogram = np.array(X_train_spectrogram)
     X_test_spectrogram = np.array(X_test_spectrogram)

     X_test_spectrogram.shape, X_train_spectrogram.shape
```

```
[ ]: ((600, 64, 35), (1400, 64, 35))
```

Grader function 6

[ ]:
```
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()
```

[ ]:
```
True
```

3.1.5 Train LSTM Model (Spectrogram converted audio data) (MODEL - 2)

[ ]:
```
class LSTM_Layer(tf.keras.layers.Layer):

    def __init__(self, **kwargs):
        super(LSTM_Layer, self).__init__(**kwargs)
        self.lstm          = LSTM(80, return_sequences=True, return_state=True)␣
 ↪# recurrent_activation=tf.keras.layers.LeakyReLU(alpha=0.5)
        self.avg           = tf.keras.layers.
 ↪GlobalAveragePooling1D(data_format="channels_first")


    def call(self, inputs):
      lstm_output, _, _= self.lstm(inputs)
      output = self.avg(lstm_output)
      return output



lstm_layer = LSTM_Layer()
```

[ ]:
```
tf.keras.backend.clear_session()

input_spectro = Input(shape=(64,35), name="spectro_input_layer",␣
 ↪dtype="float32")
lstm_out      = lstm_layer(input_spectro)
dense_1_out   = Dense(units=50,
                      activation=tf.keras.layers.LeakyReLU(alpha=0.5),
                      kernel_initializer=he_normal(),
                      kernel_regularizer=l1(0.0001),
                      activity_regularizer=l1(0.0001)) (lstm_out)
output        = Dense(10, activation='softmax') (dense_1_out)

model = Model(inputs=input_spectro, outputs=output)
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
spectro_input_layer (InputLa [(None, 64, 35)]          0

_________________________________________________________________
lstm__layer (LSTM_Layer)     (None, 64)                37120

_________________________________________________________________
dense (Dense)                (None, 50)                3250

_________________________________________________________________
dense_1 (Dense)              (None, 10)                510
=================================================================
Total params: 40,880
Trainable params: 40,880
Non-trainable params: 0

_________________________________________________________________
```

[ ]:

```
class CustomCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs=None):
        keys = list(logs.keys())
        y_pred = model.predict(X_test_spectrogram)
        y_pred_list = []
        for i in y_pred:
          y_pred_list.append(tf.argmax(i))

        print("\nEpoch {}  micro-F1 score {}\n".format(epoch, sklearn.metrics.
 ↪f1_score(y_test_label_encoded, y_pred_list, average='micro')))
```

[ ]:

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              metrics=['accuracy'],
              loss=tf.keras.losses.SparseCategoricalCrossentropy())

!rm -rf logs/fit/model_2
log_dir="logs/fit/model_2"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                                                      histogram_freq=1,
                                                      write_graph=True,
                                                      write_grads=True)

earlystopping_callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss",␣
 ↪patience=2, verbose=1, mode='min')

myCallback = CustomCallback()

model.fit(x=X_train_spectrogram,
          y=y_train_label_encoded,
          batch_size=100,
          epochs=70,
          verbose=1,
          validation_data=(X_test_spectrogram, y_test_label_encoded),
          callbacks=[myCallback, tensorboard_callback, earlystopping_callback]
          )
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the
`TensorBoard` Callback.
Epoch 1/70
 2/14 [===>…] - ETA: 0s - loss: 2.3444 - accuracy:
0.1200WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0106s vs `on_train_batch_end` time: 0.0773s).
Check your callbacks.
 9/14 [==================>…] - ETA: 0s - loss: 2.3414 - accuracy:
0.1522
Epoch 0  micro-F1 score 0.165

14/14 [==============================] - 1s 86ms/step - loss: 2.3354 - accuracy:
0.1671 - val_loss: 2.3196 - val_accuracy: 0.1650
Epoch 2/70
 9/14 [==================>…] - ETA: 0s - loss: 2.2930 - accuracy:
0.1678
Epoch 1  micro-F1 score 0.20833333333333334

14/14 [==============================] - 0s 27ms/step - loss: 2.2837 - accuracy:
0.1664 - val_loss: 2.2384 - val_accuracy: 0.2083
Epoch 3/70
 8/14 [================>…] - ETA: 0s - loss: 2.2022 - accuracy:
0.1887
Epoch 2  micro-F1 score 0.20833333333333334

14/14 [==============================] - 0s 26ms/step - loss: 2.1680 - accuracy:
0.2036 - val_loss: 2.1046 - val_accuracy: 0.2083
Epoch 4/70
 9/14 [==================>…] - ETA: 0s - loss: 2.0610 - accuracy:
0.2411
Epoch 3  micro-F1 score 0.245

14/14 [==============================] - 0s 26ms/step - loss: 2.0271 - accuracy:
0.2414 - val_loss: 1.9816 - val_accuracy: 0.2450
Epoch 5/70
 9/14 [==================>…] - ETA: 0s - loss: 1.9407 - accuracy:
0.2856
Epoch 4  micro-F1 score 0.36

14/14 [==============================] - 0s 27ms/step - loss: 1.9053 - accuracy:
0.3100 - val_loss: 1.8619 - val_accuracy: 0.3600
```

```
Epoch 6/70
 9/14 [==================>…] - ETA: 0s - loss: 1.8104 - accuracy:
0.3856
Epoch 5  micro-F1 score 0.39833333333333326

14/14 [==============================] - 0s 25ms/step - loss: 1.7899 - accuracy:
0.4071 - val_loss: 1.7722 - val_accuracy: 0.3983
Epoch 7/70
 9/14 [==================>…] - ETA: 0s - loss: 1.6743 - accuracy:
0.4456
Epoch 6  micro-F1 score 0.465

14/14 [==============================] - 0s 25ms/step - loss: 1.6875 - accuracy:
0.4386 - val_loss: 1.6526 - val_accuracy: 0.4650
Epoch 8/70
 9/14 [==================>…] - ETA: 0s - loss: 1.5878 - accuracy:
0.4978
Epoch 7  micro-F1 score 0.5183333333333333

14/14 [==============================] - 0s 26ms/step - loss: 1.5857 - accuracy:
0.4964 - val_loss: 1.5570 - val_accuracy: 0.5183
Epoch 9/70
 9/14 [==================>…] - ETA: 0s - loss: 1.5174 - accuracy:
0.5244
Epoch 8  micro-F1 score 0.5316666666666666

14/14 [==============================] - 0s 25ms/step - loss: 1.4967 - accuracy:
0.5457 - val_loss: 1.4798 - val_accuracy: 0.5317
Epoch 10/70
 9/14 [==================>…] - ETA: 0s - loss: 1.4427 - accuracy:
0.5767
Epoch 9  micro-F1 score 0.5816666666666667

14/14 [==============================] - 0s 24ms/step - loss: 1.4166 - accuracy:
0.5857 - val_loss: 1.4028 - val_accuracy: 0.5817
Epoch 11/70
 9/14 [==================>…] - ETA: 0s - loss: 1.3427 - accuracy:
0.6067
Epoch 10  micro-F1 score 0.5866666666666667

14/14 [==============================] - 0s 26ms/step - loss: 1.3375 - accuracy:
0.6021 - val_loss: 1.3649 - val_accuracy: 0.5867
Epoch 12/70
 9/14 [==================>…] - ETA: 0s - loss: 1.3055 - accuracy:
0.6244
Epoch 11  micro-F1 score 0.6116666666666667

14/14 [==============================] - 0s 26ms/step - loss: 1.2931 - accuracy:
```

```
0.6386 - val_loss: 1.3153 - val_accuracy: 0.6117
Epoch 13/70
 9/14 [==================>…] - ETA: 0s - loss: 1.2240 - accuracy:
0.6533
Epoch 12  micro-F1 score 0.6416666666666667

14/14 [==============================] - 0s 25ms/step - loss: 1.2210 - accuracy:
0.6543 - val_loss: 1.2309 - val_accuracy: 0.6417
Epoch 14/70
 9/14 [==================>…] - ETA: 0s - loss: 1.1704 - accuracy:
0.6822
Epoch 13  micro-F1 score 0.6583333333333333

14/14 [==============================] - 0s 25ms/step - loss: 1.1759 - accuracy:
0.6679 - val_loss: 1.1961 - val_accuracy: 0.6583
Epoch 15/70
 9/14 [==================>…] - ETA: 0s - loss: 1.1337 - accuracy:
0.7044
Epoch 14  micro-F1 score 0.665

14/14 [==============================] - 0s 25ms/step - loss: 1.1320 - accuracy:
0.6929 - val_loss: 1.1629 - val_accuracy: 0.6650
Epoch 16/70
 9/14 [==================>…] - ETA: 0s - loss: 1.0931 - accuracy:
0.6989
Epoch 15  micro-F1 score 0.6833333333333333

14/14 [==============================] - 0s 27ms/step - loss: 1.0902 - accuracy:
0.6914 - val_loss: 1.1174 - val_accuracy: 0.6833
Epoch 17/70
 9/14 [==================>…] - ETA: 0s - loss: 1.0456 - accuracy:
0.7189
Epoch 16  micro-F1 score 0.67

14/14 [==============================] - 0s 25ms/step - loss: 1.0589 - accuracy:
0.7057 - val_loss: 1.1183 - val_accuracy: 0.6700
Epoch 18/70
 9/14 [==================>…] - ETA: 0s - loss: 1.0592 - accuracy:
0.7022
Epoch 17  micro-F1 score 0.6983333333333334

14/14 [==============================] - 0s 25ms/step - loss: 1.0325 - accuracy:
0.7164 - val_loss: 1.0621 - val_accuracy: 0.6983
Epoch 19/70
 9/14 [==================>…] - ETA: 0s - loss: 0.9922 - accuracy:
0.7222
Epoch 18  micro-F1 score 0.7033333333333334
```

```
14/14 [==============================] - 0s 25ms/step - loss: 1.0058 - accuracy:
0.7071 - val_loss: 1.0389 - val_accuracy: 0.7033
Epoch 20/70
 9/14 [==================>…] - ETA: 0s - loss: 0.9854 - accuracy:
0.7278
Epoch 19  micro-F1 score 0.7083333333333334

14/14 [==============================] - 0s 25ms/step - loss: 0.9657 - accuracy:
0.7321 - val_loss: 1.0068 - val_accuracy: 0.7083
Epoch 21/70
 9/14 [==================>…] - ETA: 0s - loss: 0.9570 - accuracy:
0.7289
Epoch 20  micro-F1 score 0.7183333333333334

14/14 [==============================] - 0s 24ms/step - loss: 0.9391 - accuracy:
0.7321 - val_loss: 0.9849 - val_accuracy: 0.7183
Epoch 22/70
 9/14 [==================>…] - ETA: 0s - loss: 0.9146 - accuracy:
0.7622
Epoch 21  micro-F1 score 0.715

14/14 [==============================] - 0s 25ms/step - loss: 0.9185 - accuracy:
0.7457 - val_loss: 0.9557 - val_accuracy: 0.7150
Epoch 23/70
 9/14 [==================>…] - ETA: 0s - loss: 0.9101 - accuracy:
0.7522
Epoch 22  micro-F1 score 0.7116666666666667

14/14 [==============================] - 0s 25ms/step - loss: 0.8852 - accuracy:
0.7543 - val_loss: 0.9394 - val_accuracy: 0.7117
Epoch 24/70
 9/14 [==================>…] - ETA: 0s - loss: 0.8388 - accuracy:
0.7700
Epoch 23  micro-F1 score 0.7283333333333334

14/14 [==============================] - 0s 25ms/step - loss: 0.8677 - accuracy:
0.7607 - val_loss: 0.9214 - val_accuracy: 0.7283
Epoch 25/70
 7/14 [==============>…] - ETA: 0s - loss: 0.8447 - accuracy:
0.7557
Epoch 24  micro-F1 score 0.7216666666666668

14/14 [==============================] - 0s 25ms/step - loss: 0.8309 - accuracy:
0.7736 - val_loss: 0.9034 - val_accuracy: 0.7217
Epoch 26/70
 9/14 [==================>…] - ETA: 0s - loss: 0.8236 - accuracy:
0.7778
Epoch 25  micro-F1 score 0.7283333333333334
```

```
14/14 [==============================] - 0s 25ms/step - loss: 0.8275 - accuracy:
0.7764 - val_loss: 0.8888 - val_accuracy: 0.7283
Epoch 27/70
 9/14 [==================>…] - ETA: 0s - loss: 0.7978 - accuracy:
0.7767
Epoch 26  micro-F1 score 0.7433333333333333

14/14 [==============================] - 0s 26ms/step - loss: 0.8014 - accuracy:
0.7707 - val_loss: 0.8466 - val_accuracy: 0.7433
Epoch 28/70
 9/14 [==================>…] - ETA: 0s - loss: 0.7596 - accuracy:
0.7967
Epoch 27  micro-F1 score 0.7416666666666667

14/14 [==============================] - 0s 25ms/step - loss: 0.7690 - accuracy:
0.7943 - val_loss: 0.8374 - val_accuracy: 0.7417
Epoch 29/70
 9/14 [==================>…] - ETA: 0s - loss: 0.7573 - accuracy:
0.7989
Epoch 28  micro-F1 score 0.7516666666666667

14/14 [==============================] - 0s 25ms/step - loss: 0.7456 - accuracy:
0.7979 - val_loss: 0.8156 - val_accuracy: 0.7517
Epoch 30/70
14/14 [==============================] - ETA: 0s - loss: 0.7341 - accuracy:
0.8021
Epoch 29  micro-F1 score 0.755

14/14 [==============================] - 0s 26ms/step - loss: 0.7341 - accuracy:
0.8021 - val_loss: 0.7947 - val_accuracy: 0.7550
Epoch 31/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6938 - accuracy:
0.8178
Epoch 30  micro-F1 score 0.7566666666666667

14/14 [==============================] - 0s 25ms/step - loss: 0.7132 - accuracy:
0.8036 - val_loss: 0.7911 - val_accuracy: 0.7567
Epoch 32/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6935 - accuracy:
0.8022
Epoch 31  micro-F1 score 0.7516666666666667

14/14 [==============================] - 0s 25ms/step - loss: 0.7091 - accuracy:
0.8007 - val_loss: 0.7996 - val_accuracy: 0.7517
Epoch 33/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6982 - accuracy:
0.8056
```

```
Epoch 32  micro-F1 score 0.7733333333333333

14/14 [==============================] - 0s 25ms/step - loss: 0.6992 - accuracy:
0.8079 - val_loss: 0.7569 - val_accuracy: 0.7733
Epoch 34/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6661 - accuracy:
0.8267
Epoch 33  micro-F1 score 0.7733333333333333

14/14 [==============================] - 0s 25ms/step - loss: 0.6746 - accuracy:
0.8193 - val_loss: 0.7533 - val_accuracy: 0.7733
Epoch 35/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6948 - accuracy:
0.8278
Epoch 34  micro-F1 score 0.7699999999999999

14/14 [==============================] - 0s 25ms/step - loss: 0.6642 - accuracy:
0.8321 - val_loss: 0.7394 - val_accuracy: 0.7700
Epoch 36/70
14/14 [==============================] - ETA: 0s - loss: 0.6468 - accuracy:
0.8264
Epoch 35  micro-F1 score 0.7866666666666666

14/14 [==============================] - 0s 26ms/step - loss: 0.6468 - accuracy:
0.8264 - val_loss: 0.7239 - val_accuracy: 0.7867
Epoch 37/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6209 - accuracy:
0.8356
Epoch 36  micro-F1 score 0.785

14/14 [==============================] - 0s 26ms/step - loss: 0.6314 - accuracy:
0.8357 - val_loss: 0.7106 - val_accuracy: 0.7850
Epoch 38/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6278 - accuracy:
0.8244
Epoch 37  micro-F1 score 0.79

14/14 [==============================] - 0s 25ms/step - loss: 0.6296 - accuracy:
0.8264 - val_loss: 0.6905 - val_accuracy: 0.7900
Epoch 39/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6124 - accuracy:
0.8389
Epoch 38  micro-F1 score 0.8016666666666666

14/14 [==============================] - 0s 25ms/step - loss: 0.6182 - accuracy:
0.8400 - val_loss: 0.6815 - val_accuracy: 0.8017
Epoch 40/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6199 - accuracy:
```

```
0.8467
Epoch 39  micro-F1 score 0.7933333333333333

14/14 [==============================] - 0s 25ms/step - loss: 0.6032 - accuracy:
0.8457 - val_loss: 0.6868 - val_accuracy: 0.7933
Epoch 41/70
 9/14 [==================>…] - ETA: 0s - loss: 0.5992 - accuracy:
0.8333
Epoch 40  micro-F1 score 0.7966666666666665

14/14 [==============================] - 0s 25ms/step - loss: 0.5956 - accuracy:
0.8400 - val_loss: 0.6673 - val_accuracy: 0.7967
Epoch 42/70
 9/14 [==================>…] - ETA: 0s - loss: 0.6047 - accuracy:
0.8500
Epoch 41  micro-F1 score 0.8016666666666666

14/14 [==============================] - 0s 25ms/step - loss: 0.5890 - accuracy:
0.8479 - val_loss: 0.6639 - val_accuracy: 0.8017
Epoch 43/70
 9/14 [==================>…] - ETA: 0s - loss: 0.5646 - accuracy:
0.8478
Epoch 42  micro-F1 score 0.8116666666666666

14/14 [==============================] - 0s 25ms/step - loss: 0.5686 - accuracy:
0.8543 - val_loss: 0.6448 - val_accuracy: 0.8117
Epoch 44/70
14/14 [==============================] - ETA: 0s - loss: 0.5583 - accuracy:
0.8536
Epoch 43  micro-F1 score 0.81

14/14 [==============================] - 0s 27ms/step - loss: 0.5583 - accuracy:
0.8536 - val_loss: 0.6406 - val_accuracy: 0.8100
Epoch 45/70
 9/14 [==================>…] - ETA: 0s - loss: 0.5337 - accuracy:
0.8633
Epoch 44  micro-F1 score 0.81

14/14 [==============================] - 0s 26ms/step - loss: 0.5524 - accuracy:
0.8636 - val_loss: 0.6370 - val_accuracy: 0.8100
Epoch 46/70
 9/14 [==================>…] - ETA: 0s - loss: 0.5361 - accuracy:
0.8711
Epoch 45  micro-F1 score 0.81

14/14 [==============================] - 0s 25ms/step - loss: 0.5511 - accuracy:
0.8657 - val_loss: 0.6293 - val_accuracy: 0.8100
Epoch 47/70
```

```
 9/14 [==================>…] - ETA: 0s - loss: 0.5322 - accuracy:
0.8711
Epoch 46  micro-F1 score 0.8000000000000002

14/14 [==============================] - 0s 27ms/step - loss: 0.5424 - accuracy:
0.8614 - val_loss: 0.6482 - val_accuracy: 0.8000
Epoch 48/70
 9/14 [==================>…] - ETA: 0s - loss: 0.5607 - accuracy:
0.8456
Epoch 47  micro-F1 score 0.8066666666666665

14/14 [==============================] - 0s 25ms/step - loss: 0.5419 - accuracy:
0.8536 - val_loss: 0.6100 - val_accuracy: 0.8067
Epoch 49/70
 9/14 [==================>…] - ETA: 0s - loss: 0.5173 - accuracy:
0.8700
Epoch 48  micro-F1 score 0.8016666666666666

14/14 [==============================] - 0s 26ms/step - loss: 0.5257 - accuracy:
0.8600 - val_loss: 0.6317 - val_accuracy: 0.8017
Epoch 50/70
 9/14 [==================>…] - ETA: 0s - loss: 0.5022 - accuracy:
0.8733
Epoch 49  micro-F1 score 0.8183333333333332

14/14 [==============================] - 0s 25ms/step - loss: 0.5168 - accuracy:
0.8693 - val_loss: 0.6128 - val_accuracy: 0.8183
Epoch 00050: early stopping
```

[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f39e3e20860>
```

[ ]:

```
%reload_ext tensorboard
%tensorboard --logdir logs/fit
```

4 With augmentation

[125]:

```
## 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
```

[128]:
```
temp_path = df_audio.iloc[0].path
aug_temp = generate_augmented_data(temp_path)
```

[129]:
```
len(aug_temp)
```

[129]:
```
9
```

[126]:
```
df_audio.head()
```

[126]:
```
                                                     path label
766   /content/drive/MyDrive/Colab Notebooks/Spoken …     4
182   /content/drive/MyDrive/Colab Notebooks/Spoken …     5
1763  /content/drive/MyDrive/Colab Notebooks/Spoken …     2
1814  /content/drive/MyDrive/Colab Notebooks/Spoken …     1
596   /content/drive/MyDrive/Colab Notebooks/Spoken …     9
```

4.1 Train test split (audio data frame)

[127]:
```
X_train_aug, X_test_aug, y_train_aug, y_test_aug =␣
 ↪train_test_split(df_audio['path'],
                                                    df_audio['label'],
                                                    test_size=0.20,
                                                    random_state=45,
                                                    stratify=df_audio['label'])
```

4.2 Augmenting Train data

[130]:
```
X_train_augmented = []

for row in X_train_aug:
  X_train_augmented.append(generate_augmented_data(row))

X_train_augmented = np.array(X_train_augmented)
```

[138]:
```
X_train_augmented.shape, X_train_aug.shape
```

[138]:
```
((1600, 9), (1600,))
```

[139]:
```
X_train_augmented_final = []
y_train_augmented_final = []

for x, y in zip(X_train_augmented, y_train_aug):
  for row in x:
    X_train_augmented_final.append(row)
    y_train_augmented_final.append(y)

X_train_augmented_final = np.array(X_train_augmented_final)
y_train_augmented_final = np.array(y_train_augmented_final)

X_train_augmented_final.shape, y_train_augmented_final.shape
```

```
[139]: ((14400,), (14400,))
```

```
[140]: X_train_augmented_final.shape, y_train_aug.shape
```

```
[140]: ((14400,), (1600,))
```

4.3 Getting raw data from test audio files

```
[134]: X_test_aug_preprocessed = []

       for row in X_test_aug:
         samples, duration = load_wav(row, get_duration=True)
         X_test_aug_preprocessed.append(samples)
```

```
[135]: X_test_aug_preprocessed = np.array(X_test_aug_preprocessed)
       X_test_aug_preprocessed.shape
```

```
[135]: (400,)
```

4.4 Pad and Mask sequence

```
[146]: def mask_seq(seq):
         seq_masked_list = []
         for row in seq:
           masks = []
           for item in row:
             if item == 0:
               masks.append(False)
             else:
               masks.append(True)
           seq_masked_list.append(masks)

         return np.array(seq_masked_list)
```

```
[ ]: # Padding sequence --> (batch, max_seq_len)
     X_train_aug_pad_seq = pad_sequences(sequences=X_train_augmented_final,␣
      ↪maxlen=max_length, dtype='float32', padding="post")
```

[147]:

```
# Masking padded sequence --> (batch, max_seq_len) (0=False, other=True)
X_train_aug_mask = mask_seq(X_train_aug_pad_seq)
```

[148]:

```
X_train_aug_pad_seq.shape, np.array(X_train_aug_mask).shape
```

[148]:

```
((14400, 17640), (14400, 17640))
```

[149]:

```
X_test_aug_pad_seq = pad_sequences(sequences=X_test_aug_preprocessed,␣
 ↪maxlen=max_length, dtype='float32', padding="post")
X_test_aug_mask = mask_seq(X_test_aug_pad_seq)

X_test_aug_pad_seq.shape, np.array(X_test_aug_mask).shape
```

[149]:

```
((400, 17640), (400, 17640))
```

[152]:

```
# reshaping padded_sequence --> (batch, seqlen, 1)
X_train_aug_pad_seq_expanded = np.expand_dims(X_train_aug_pad_seq, axis=2)
X_test_aug_pad_seq_expanded = np.expand_dims(X_test_aug_pad_seq, axis=2)
```

[153]:

```
X_train_aug_pad_seq_expanded.shape, X_test_aug_pad_seq_expanded.shape
```

[153]:

```
((14400, 17640, 1), (400, 17640, 1))
```

[164]:

```
# label encoding targets
label_encoder = LabelEncoder()
y_train_aug_label_encoded = label_encoder.fit_transform(y_train_augmented_final)
y_test_aug_label_encoded = label_encoder.transform(y_test_aug)
```

[165]:

```
y_train_aug_label_encoded.shape, y_test_aug.shape
```

[165]:

```
((14400,), (400,))
```

4.5 Train Model with augmented Train Data (RAW audio data) (MODEL - 3)

[166]:

```
class LSTM_Layer(tf.keras.layers.Layer):

    def __init__(self, **kwargs):
        super(LSTM_Layer, self).__init__(**kwargs)
        self.lstm         = LSTM(25) # recurrent_activation=tf.keras.layers.
 ↪LeakyReLU(alpha=0.5)


    def call(self, inputs, masked_inputs):
      output= self.lstm(inputs, mask=masked_inputs)
      return output
```

```

lstm_layer = LSTM_Layer()
```

[172]:
```
tf.keras.backend.clear_session()

input_padded = Input(shape=(17640,1), name="padded_input_layer_aug",␣
 ↪dtype="float32")
input_masked = Input(shape=(17640,), name="masked_input_layer_aug",␣
 ↪dtype="bool")
lstm_out     = lstm_layer(input_padded, input_masked)
dense_1_out  = Dense(units=50,
                     activation=tf.keras.layers.LeakyReLU(alpha=0.5),
                     kernel_initializer=he_normal(),
                     kernel_regularizer=l1(0.0001),
                     activity_regularizer=l1(0.0001)) (lstm_out)
output       = Dense(10, activation='softmax') (dense_1_out)

model = Model(inputs=[input_padded, input_masked], outputs=output)
model.summary()
```

```
Model: "functional_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
padded_input_layer_aug (InputLa [(None, 17640, 1)]   0

________________________________________________________________________________
__________________
masked_input_layer_aug (InputLa [(None, 17640)]      0

________________________________________________________________________________
__________________
lstm__layer (LSTM_Layer)        (None, 25)           2700
padded_input_layer_aug[0][0]
masked_input_layer_aug[0][0]

________________________________________________________________________________
__________________
dense (Dense)                   (None, 50)           1300
lstm__layer[1][0]

________________________________________________________________________________
__________________
dense_1 (Dense)                 (None, 10)           510         dense[0][0]
================================================================================
==================
Total params: 4,510
Trainable params: 4,510
Non-trainable params: 0
```

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

[ ]:

```
# X_train_aug_pad_seq_expanded, X_test_aug_pad_seq_expanded, X_train_aug_mask,␣
↪X_test_aug_mask, y_train_aug_label_encoded, y_test_aug_label_encoded
```

[168]:

```
class CustomCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs=None):
        keys = list(logs.keys())
        y_pred = model.predict([X_test_aug_pad_seq_expanded, X_test_aug_mask])
        y_pred_list = []
        for i in y_pred:
          y_pred_list.append(tf.argmax(i))

        print("\nEpoch {}  micro-F1 score {}\n".format(epoch, sklearn.metrics.
↪f1_score(y_test_aug_label_encoded, y_pred_list, average='micro')) )
```

[173]:

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              metrics=['accuracy'],
              loss=tf.keras.losses.SparseCategoricalCrossentropy())

!rm -rf logs/fit/model_3
log_dir="logs/fit/model_3"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                                                      histogram_freq=1,
                                                      write_graph=True,
                                                      write_grads=True)

earlystopping_callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss",␣
↪patience=2, verbose=1, mode='min')

myCallback = CustomCallback()

model.fit(x=[X_train_aug_pad_seq_expanded, X_train_aug_mask],
          y=y_train_aug_label_encoded,
          batch_size=100,
          epochs=70,
          verbose=1,
          validation_data=([X_test_aug_pad_seq_expanded, X_test_aug_mask],␣
↪y_test_aug_label_encoded),
          callbacks=[myCallback, tensorboard_callback, earlystopping_callback]
          )
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the
```

```
`TensorBoard` Callback.
Epoch 1/70
144/144 [==============================] - ETA: 0s - loss: 2.3255 - accuracy:
0.0969
Epoch 0  micro-F1 score 0.10000000000000002

144/144 [==============================] - 108s 752ms/step - loss: 2.3255 -
accuracy: 0.0969 - val_loss: 2.3187 - val_accuracy: 0.1000
Epoch 2/70
144/144 [==============================] - ETA: 0s - loss: 2.3143 - accuracy:
0.0954
Epoch 1  micro-F1 score 0.10000000000000002

144/144 [==============================] - 105s 726ms/step - loss: 2.3143 -
accuracy: 0.0954 - val_loss: 2.3100 - val_accuracy: 0.1000
Epoch 3/70
144/144 [==============================] - ETA: 0s - loss: 2.3076 - accuracy:
0.0935
Epoch 2  micro-F1 score 0.10000000000000002

144/144 [==============================] - 105s 730ms/step - loss: 2.3076 -
accuracy: 0.0935 - val_loss: 2.3051 - val_accuracy: 0.1000
Epoch 4/70
144/144 [==============================] - ETA: 0s - loss: 2.3041 - accuracy:
0.0963
Epoch 3  micro-F1 score 0.10000000000000002

144/144 [==============================] - 105s 731ms/step - loss: 2.3041 -
accuracy: 0.0963 - val_loss: 2.3030 - val_accuracy: 0.1000
Epoch 5/70
144/144 [==============================] - ETA: 0s - loss: 2.3030 - accuracy:
0.0961
Epoch 4  micro-F1 score 0.10000000000000002

144/144 [==============================] - 104s 722ms/step - loss: 2.3030 -
accuracy: 0.0961 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 6/70
144/144 [==============================] - ETA: 0s - loss: 2.3030 - accuracy:
0.0962
Epoch 5  micro-F1 score 0.10000000000000002

144/144 [==============================] - 103s 719ms/step - loss: 2.3030 -
accuracy: 0.0962 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 7/70
144/144 [==============================] - ETA: 0s - loss: 2.3029 - accuracy:
0.0987
Epoch 6  micro-F1 score 0.10000000000000002
```

```
144/144 [==============================] - 103s 716ms/step - loss: 2.3029 -
accuracy: 0.0987 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 8/70
144/144 [==============================] - ETA: 0s - loss: 2.3029 - accuracy:
0.0948
Epoch 7  micro-F1 score 0.10000000000000002

144/144 [==============================] - 103s 715ms/step - loss: 2.3029 -
accuracy: 0.0948 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 00008: early stopping
```

[173]:
```
<tensorflow.python.keras.callbacks.History at 0x7f39deb8df98>
```

[174]:
```
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

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

```
<IPython.core.display.Javascript object>
```

4.6 Augmented data to spectrogram conversion

[175]:
```
X_train_aug_spectrogram = []
X_test_aug_spectrogram = []

for row in X_train_aug_pad_seq:
  spectrum = convert_to_spectrogram(row)
  X_train_aug_spectrogram.append(spectrum)

for row in X_test_aug_pad_seq:
  spectrum = convert_to_spectrogram(row)
  X_test_aug_spectrogram.append(spectrum)


X_train_aug_spectrogram = np.array(X_train_aug_spectrogram)
X_test_aug_spectrogram = np.array(X_test_aug_spectrogram)

X_train_aug_spectrogram.shape, X_test_aug_spectrogram.shape
```

[175]:
```
((14400, 64, 35), (400, 64, 35))
```

4.7 Train augmented spectrogram converted (MODEL - 4)

[177]:
```
class LSTM_Layer(tf.keras.layers.Layer):

    def __init__(self, **kwargs):
        super(LSTM_Layer, self).__init__(**kwargs)
        self.lstm         = LSTM(80, return_sequences=True, return_state=True)␣
↪# recurrent_activation=tf.keras.layers.LeakyReLU(alpha=0.5)
        self.avg          = tf.keras.layers.
↪GlobalAveragePooling1D(data_format="channels_first")


    def call(self, inputs):
      lstm_output, _, _= self.lstm(inputs)
      output = self.avg(lstm_output)
      return output



lstm_layer = LSTM_Layer()
```

[179]:

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

input_spectro = Input(shape=(64,35), name="spectro_aug_input_layer",␣
 ↪dtype="float32")
lstm_out     = lstm_layer(input_spectro)
dense_1_out  = Dense(units=50,
                     activation=tf.keras.layers.LeakyReLU(alpha=0.5),
                     kernel_initializer=he_normal(),
                     kernel_regularizer=l1(0.0001),
                     activity_regularizer=l1(0.0001)) (lstm_out)
output       = Dense(10, activation='softmax') (dense_1_out)

model = Model(inputs=input_spectro, outputs=output)
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
spectro_aug_input_layer (Inp [(None, 64, 35)]          0
_________________________________________________________________
lstm__layer (LSTM_Layer)     (None, 64)                37120
_________________________________________________________________
dense (Dense)                (None, 50)                3250
_________________________________________________________________
dense_1 (Dense)              (None, 10)                510
=================================================================
Total params: 40,880
Trainable params: 40,880
Non-trainable params: 0
_________________________________________________________________
```

[180]:
```
class CustomCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs=None):
        keys = list(logs.keys())
        y_pred = model.predict(X_test_aug_spectrogram)
        y_pred_list = []
        for i in y_pred:
          y_pred_list.append(tf.argmax(i))

        print("\nEpoch {}  micro-F1 score {}\n".format(epoch, sklearn.metrics.
 ↪f1_score(y_test_aug_label_encoded, y_pred_list, average='micro')))
```

[181]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              metrics=['accuracy'],
              loss=tf.keras.losses.SparseCategoricalCrossentropy())

!rm -rf logs/fit/model_4
log_dir="logs/fit/model_4"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                                                      histogram_freq=1,
                                                      write_graph=True,
                                                      write_grads=True)

earlystopping_callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss",␣
 ↪patience=2, verbose=1, mode='min')

myCallback = CustomCallback()

model.fit(x=X_train_aug_spectrogram,
          y=y_train_aug_label_encoded,
          batch_size=100,
          epochs=70,
          verbose=1,
          validation_data=(X_test_aug_spectrogram, y_test_aug_label_encoded),
          callbacks=[myCallback, tensorboard_callback, earlystopping_callback]
          )
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the
`TensorBoard` Callback.
Epoch 1/70
  2/144 […] - ETA: 9s - loss: 2.3499 - accuracy:
0.0750WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0108s vs `on_train_batch_end` time: 0.1216s).
Check your callbacks.
144/144 [==============================] - ETA: 0s - loss: 2.0003 - accuracy:
```

```
0.3142
Epoch 0  micro-F1 score 0.5075

144/144 [==============================] - 2s 14ms/step - loss: 2.0003 -
accuracy: 0.3142 - val_loss: 1.5436 - val_accuracy: 0.5075
Epoch 2/70
141/144 [============================>.] - ETA: 0s - loss: 1.3669 - accuracy:
0.5685
Epoch 1  micro-F1 score 0.6575

144/144 [==============================] - 1s 8ms/step - loss: 1.3634 -
accuracy: 0.5701 - val_loss: 1.1324 - val_accuracy: 0.6575
Epoch 3/70
138/144 [===========================>..] - ETA: 0s - loss: 1.0468 - accuracy:
0.6842
Epoch 2  micro-F1 score 0.7625

144/144 [==============================] - 1s 8ms/step - loss: 1.0412 -
accuracy: 0.6860 - val_loss: 0.8724 - val_accuracy: 0.7625
Epoch 4/70
140/144 [============================>.] - ETA: 0s - loss: 0.8506 - accuracy:
0.7431
Epoch 3  micro-F1 score 0.7825

144/144 [==============================] - 1s 8ms/step - loss: 0.8483 -
accuracy: 0.7439 - val_loss: 0.7231 - val_accuracy: 0.7825
Epoch 5/70
140/144 [============================>.] - ETA: 0s - loss: 0.7476 - accuracy:
0.7754
Epoch 4  micro-F1 score 0.7925

144/144 [==============================] - 1s 8ms/step - loss: 0.7444 -
accuracy: 0.7766 - val_loss: 0.6517 - val_accuracy: 0.7925
Epoch 6/70
136/144 [===========================>..] - ETA: 0s - loss: 0.6733 - accuracy:
0.8002
Epoch 5  micro-F1 score 0.8075

144/144 [==============================] - 1s 8ms/step - loss: 0.6730 -
accuracy: 0.7998 - val_loss: 0.6156 - val_accuracy: 0.8075
Epoch 7/70
139/144 [===========================>..] - ETA: 0s - loss: 0.6380 - accuracy:
0.8125
Epoch 6  micro-F1 score 0.8675

144/144 [==============================] - 1s 8ms/step - loss: 0.6361 -
accuracy: 0.8133 - val_loss: 0.5317 - val_accuracy: 0.8675
Epoch 8/70
```

```
141/144 [============================>.] - ETA: 0s - loss: 0.6031 - accuracy:
0.8228
Epoch 7  micro-F1 score 0.8599999999999999

144/144 [==============================] - 1s 8ms/step - loss: 0.6051 -
accuracy: 0.8220 - val_loss: 0.5338 - val_accuracy: 0.8600
Epoch 9/70
144/144 [==============================] - ETA: 0s - loss: 0.5733 - accuracy:
0.8303
Epoch 8  micro-F1 score 0.865

144/144 [==============================] - 1s 8ms/step - loss: 0.5733 -
accuracy: 0.8303 - val_loss: 0.4949 - val_accuracy: 0.8650
Epoch 10/70
142/144 [============================>.] - ETA: 0s - loss: 0.5444 - accuracy:
0.8439
Epoch 9  micro-F1 score 0.865

144/144 [==============================] - 1s 8ms/step - loss: 0.5448 -
accuracy: 0.8435 - val_loss: 0.4734 - val_accuracy: 0.8650
Epoch 11/70
142/144 [============================>.] - ETA: 0s - loss: 0.5183 - accuracy:
0.8523
Epoch 10  micro-F1 score 0.8875

144/144 [==============================] - 1s 8ms/step - loss: 0.5187 -
accuracy: 0.8519 - val_loss: 0.4404 - val_accuracy: 0.8875
Epoch 12/70
143/144 [============================>.] - ETA: 0s - loss: 0.5015 - accuracy:
0.8577
Epoch 11  micro-F1 score 0.88

144/144 [==============================] - 1s 8ms/step - loss: 0.5014 -
accuracy: 0.8578 - val_loss: 0.4305 - val_accuracy: 0.8800
Epoch 13/70
142/144 [============================>.] - ETA: 0s - loss: 0.4921 - accuracy:
0.8607
Epoch 12  micro-F1 score 0.885

144/144 [==============================] - 1s 8ms/step - loss: 0.4927 -
accuracy: 0.8605 - val_loss: 0.4370 - val_accuracy: 0.8850
Epoch 14/70
138/144 [===========================>..] - ETA: 0s - loss: 0.4976 - accuracy:
0.8612
Epoch 13  micro-F1 score 0.89

144/144 [==============================] - 1s 8ms/step - loss: 0.4955 -
accuracy: 0.8617 - val_loss: 0.4214 - val_accuracy: 0.8900
```

```
Epoch 15/70
139/144 [===========================>..] - ETA: 0s - loss: 0.4780 - accuracy:
0.8686
Epoch 14  micro-F1 score 0.895

144/144 [==============================] - 1s 8ms/step - loss: 0.4784 -
accuracy: 0.8684 - val_loss: 0.4314 - val_accuracy: 0.8950
Epoch 16/70
138/144 [===========================>..] - ETA: 0s - loss: 0.4625 - accuracy:
0.8759
Epoch 15  micro-F1 score 0.895

144/144 [==============================] - 1s 8ms/step - loss: 0.4639 -
accuracy: 0.8757 - val_loss: 0.4259 - val_accuracy: 0.8950
Epoch 00016: early stopping
```

```
[181]: <tensorflow.python.keras.callbacks.History at 0x7f39cb858f98>
```

```
[182]: %reload_ext tensorboard
       %tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 52854), started 0:10:32 ago. (Use '!kill␣
 ↪52854' to kill it.)

<IPython.core.display.Javascript object>
```

5 Instruction

As discussed above, for one data point, we will get 9 augmented data points.

Split data into train and test (80-20 split)

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

Do augmentation only on train data, after augmentation we will get 14400 train points.

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

6 Procedure

Data preprocessing

```
[ ]: """

     1. we are only considering audio file, those have duration less that equals to␣
      ↪(0.8).
     2. we are extracting audio file data using (LIBROSA).
     3. We are paddig the audio sequence data.
```

```
"""
```

Model - 1

[ ]:

```
"""
1. WE are giving (padded_seq, mask_seq) as INPUT to model - 1.

"""
```

Model - 2

[ ]:

```
"""
1. For model 2 we are converting raw data into spectrogram. (BOTH train and Test data)
2. feeding spectrogram data to model 2.
3. In Model 2 we are averaging all the sequence output.

"""
```

Model - 3

[ ]:

```
"""
1. we are augmenting all train data.
2. same as model-1, data is then padded and masked.
3. we are giving padded and masked data to model-3

""
```

Model - 4

[ ]:

```
"""

1. We are feeding spectrogram converted augmented data to Model - 4

"""
```

7 Observation

[ ]:

```
"""

1. spectrogram conversion of data, giving good F1 score.
2. more data we will get more accurate the model will be.

"""
```

[ ]:

```

```