# speech-detection-assignment

August 2, 2023

## 1 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.
2. You have to write the code in the same cell which contains the instrction.
3. Training the LSTM with RAW data
4. Converting to spectrogram and Training the LSTM network
5. Creating the augmented data and doing step 2 and 3 again.

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

```
Mounted at /content/drive
```

[ ]:
```
!unzip "/content/drive/MyDrive/projects_DS_ML/Spoken_Digit_recognition/
↪recordings.zip"
```

[ ]:
```
import numpy as np
import pandas as pd
import librosa
import os
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

We shared recordings.zip, please unzip those.

[ ]:
```
## defining a path for the dataset
path = "/content/recordings"
all_files = os.listdir(path)
```

[ ]:
```
int(all_files[0][0])
```

[ ]:
```
0
```

Grader function 1

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

## 1.1 Creating dataframe

[ ]:
```
label= [int(i[0]) for i in all_files]
df_audio= pd.DataFrame(list(zip(all_files, label)),
               columns =['path', 'label'])
```

[ ]:
```
df_audio
```

[ ]:
```
                    path  label
0          0_theo_29.wav      0
1          5_theo_25.wav      5
2           7_theo_6.wav      7
3          8_theo_31.wav      8
4       7_jackson_23.wav      7
…                    …    …
1995    6_nicolas_37.wav      6
1996       7_theo_24.wav      7
1997   8_yweweler_48.wav      8
1998     0_jackson_4.wav      0
1999        6_theo_9.wav      6

[2000 rows x 2 columns]
```

[ ]:
```
#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   int64
dtypes: int64(1), object(1)
memory usage: 31.4+ KB
```

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

[ ]:
```
df_audio['path'].shape
```

[ ]: (2000,)

```
[ ]: y=df_audio["label"]    ##this gives x and y as pandas series
X=df_audio["path"]
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.30,␣
 ↪stratify=y, random_state=33)
```

```
[ ]: type(X_train)    ##pandas series have value_counts functions
```

[ ]: pandas.core.series.Series

we will create two functions using which a wav file can be converted to spectrogram

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

```
[ ]: def preprocess_wav(arr):
  samples = []
  durations = []
  for i in arr.values:
    sample, duration = load_wav(i)
    samples.append(sample)
    durations.append(duration)
  dataframe_processed = pd.DataFrame(list(zip(samples, durations)),columns␣
 ↪=['raw_data', 'duration'])
  return dataframe_processed
```

```
[ ]: # train_samples , train_durations = preprocess_wav(X_train)
preprocessed_train = preprocess_wav(X_train)
```

```
<ipython-input-13-bfb723a3eac9>:8: FutureWarning: get_duration() keyword
argument 'filename' has been renamed to 'path' in version 0.10.0.
        This alias will be removed in version 1.0.
  duration = librosa.get_duration(filename = final)
```

```
[ ]: preprocessed_test = preprocess_wav(X_test)
```

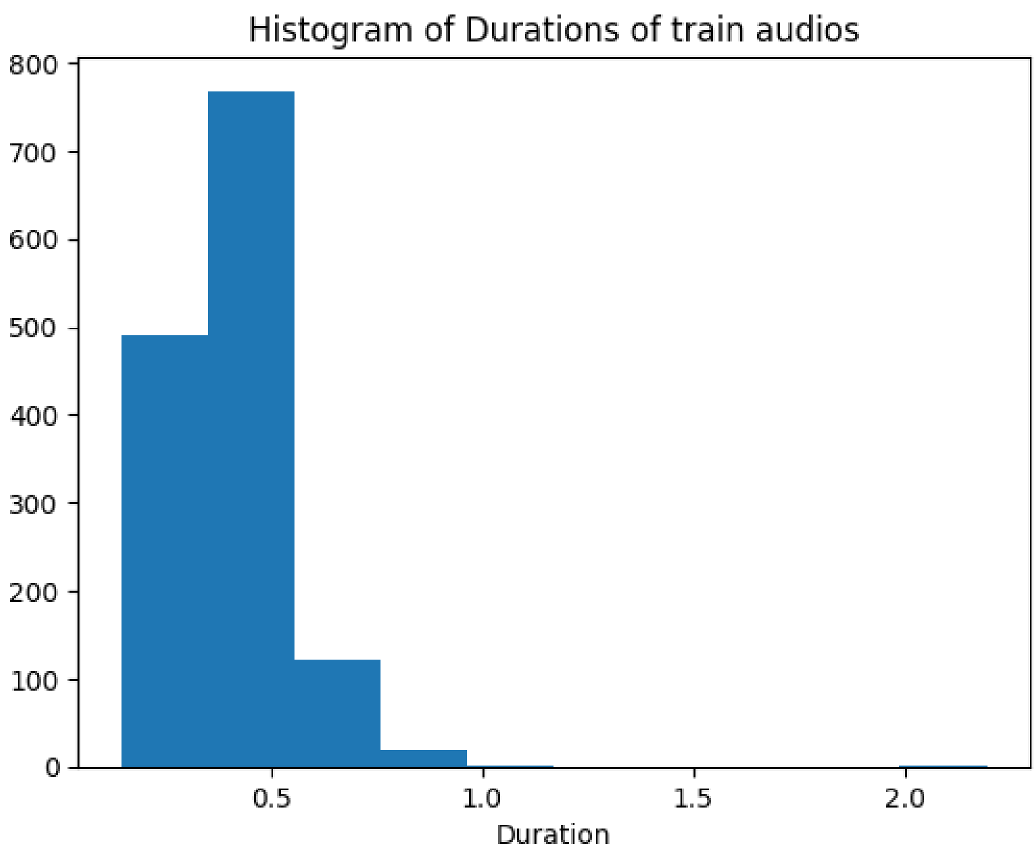
```
<ipython-input-13-bfb723a3eac9>:8: FutureWarning: get_duration() keyword
```

```
argument 'filename' has been renamed to 'path' in version 0.10.0.
        This alias will be removed in version 1.0.
  duration = librosa.get_duration(filename = final)
```

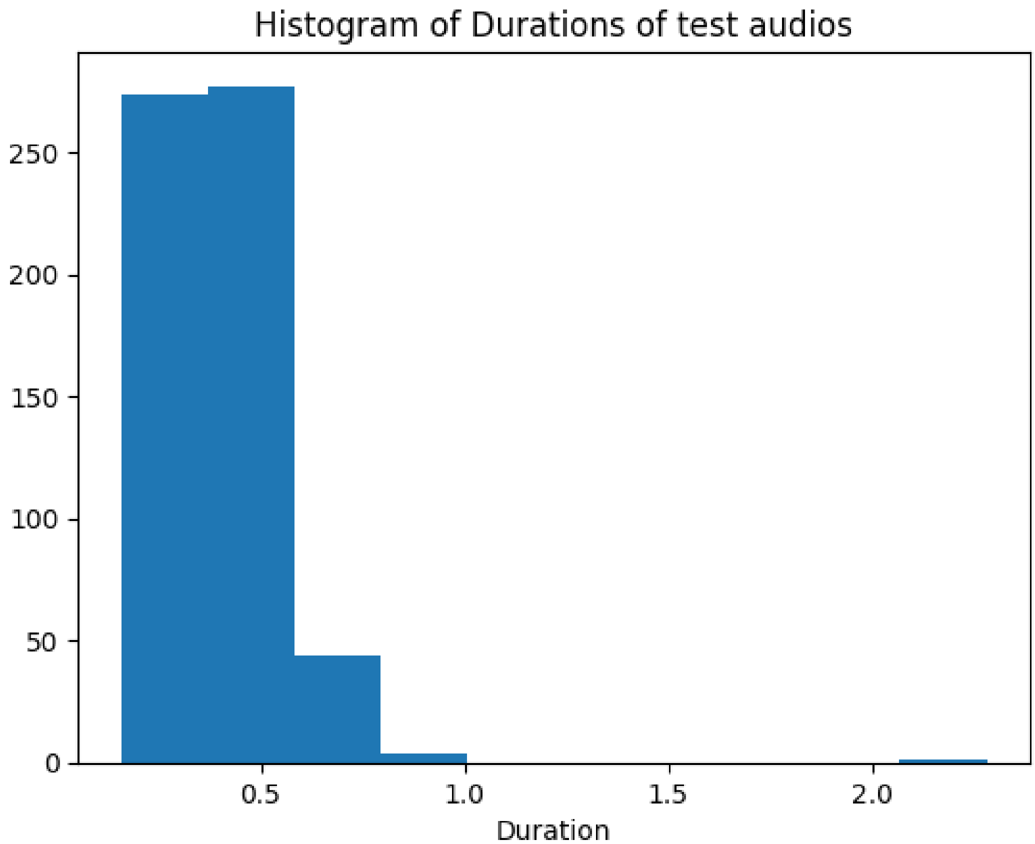
[ ]:
```python
print(np.percentile(preprocessed_train['duration'].values, 50))
```

```
0.390875
```

[ ]:
```python
#plot the histogram of the duration for trian
plt.hist(preprocessed_train.duration)
plt.xlabel('Duration')
plt.title('Histogram of Durations of train audios')
plt.show()
```




[ ]:
```python
#plot the histogram of the duration for test
plt.hist(preprocessed_test.duration)
plt.xlabel('Duration')
plt.title('Histogram of Durations of test audios')
plt.show()
```




[ ]:
```
#print 0 to 100 percentile values with step size of 10 for train data duration.
for i in range(0,110,10):
  print(i,"th percentile is ",np.percentile(preprocessed_train['duration'].
↪values, i))
```

```
0 th percentile is  0.1435
10 th percentile is  0.2627
20 th percentile is  0.30275
30 th percentile is  0.3360875
40 th percentile is  0.3615
50 th percentile is  0.390875
60 th percentile is  0.417575
70 th percentile is  0.4462
80 th percentile is  0.482625
90 th percentile is  0.5554
100 th percentile is  2.195875
```

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

```
print(i,"th percentile is ",np.percentile(preprocessed_train['duration'].
↪values, i))
```

```
90 th percentile is  0.5554
91 th percentile is  0.5707725
92 th percentile is  0.5811350000000001
93 th percentile is  0.5915062500000003
94 th percentile is  0.61042
95 th percentile is  0.6230937499999999
96 th percentile is  0.6435949999999999
97 th percentile is  0.6647649999999999
98 th percentile is  0.7050174999999999
99 th percentile is  0.8072624999999999
100 th percentile is  2.195875
```

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.

[ ]:
```
preprocessed_train['raw_data'][2].shape   ###every row has different number of␣
↪values in array depending on duration
```

[ ]:
```
(9992,)
```

[ ]:
```
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
## 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.
def pad_mask (df, max_length):
  new_padded=[]
  new_masks = []
  for i in range (df['raw_data'].shape[0]):
    length = df['raw_data'][i].shape[0]
    if length <= max_length:
      new_add = [0] * (max_length - length)
```

```
        new_one = np.concatenate((df['raw_data'][i],new_add), axis=0)
        mask = [True]* length + [False]* (max_length - length)

      else:
        new_one = df['raw_data'][i][: max_length]    ##https://www.geeksforgeeks.
↪org/python-remove-last-k-elements-of-list/#:~:
↪text=%23%20initializing%20list%C2%A0,str(res))
        mask = [True]*max_length
      # new_one = np.concatenate((X_train_processed['raw_data'][i],new_add),␣
↪axis=0)
      new_padded.append(new_one)
      new_masks.append(mask)
    return np.array(new_padded), np.array(new_masks)
```

[ ]:
```
X_train_pad_seq, X_train_mask = pad_mask(preprocessed_train, max_length)
X_test_pad_seq, X_test_mask = pad_mask(preprocessed_test, max_length)
```

[ ]:
```
X_train_mask[10]
```

[ ]:
```
array([ True,  True,  True, …, False, False, False])
```

saving the numpy aarays to disc to use them directly for further assessment

[ ]:
```
# np.save('X_train_pad_seq.npy', X_train_pad_seq )
# np.save('X_test_pad_seq.npy', X_test_pad_seq)
# np.save('X_train_mask.npy', X_train_mask )
# np.save('X_test_mask.npy', X_test_mask)
```

Loading the arrays

[ ]:
```
# import numpy as np


# X_train_pad_seq = np.load('/content/drive/MyDrive/projects_DS_ML/
↪Spoken_Digit_recognition/X_train_pad_seq.npy')
# X_test_pad_seq = np.load('/content/drive/MyDrive/projects_DS_ML/
↪Spoken_Digit_recognition/X_test_pad_seq.npy')
# X_train_mask = np.load('/content/drive/MyDrive/projects_DS_ML/
↪Spoken_Digit_recognition/X_train_mask.npy')
# X_test_mask = np.load('/content/drive/MyDrive/projects_DS_ML/
↪Spoken_Digit_recognition/X_test_mask.npy')
```

#### 1.1.1 1. Giving Raw data directly.

Now we have

Train data: X__train__pad__seq, X__train__mask and y__train
Test data: X__test__pad__seq, X__test__mask and y__test

We will create a LSTM model which takes this input.

Task:

1. Create an LSTM network which takes "X_train_pad_seq" as input, "X_train_mask" as mask input. You can use any number of LSTM cells. Please read LSTM documentation(https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM) in tensorflow to know more about mask and also https://www.tensorflow.org/guide/keras/masking_and_padding
2. Get the final output of the LSTM and give it to Dense layer of any size and then give it to Dense layer of size 10(because we have 10 outputs) and then compile with the sparse categorical cross entropy( because we are not converting it to one hot vectors). 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
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

[ ]:
```
from tensorflow.keras.layers import Input, LSTM, Dense
from tensorflow.keras.models import Model
import tensorflow as tf
from tensorflow.keras import layers
```

[ ]:
```
import tensorflow as tf
import numpy as np
from sklearn.metrics import f1_score

class Metrics(tf.keras.callbacks.Callback):
    def __init__(self, validation_data):
        super().__init__()
        self.x_test = validation_data[0]
        self.y_test = validation_data[1]

    def on_epoch_end(self, epoch, logs=None):
        val_predict = np.asarray(self.model.predict(self.x_test))
        val_label = np.argmax(val_predict, axis=1)
        val_targ = self.y_test
        val_f1 = f1_score(val_targ, val_label, average='micro')
        print("val_F1_score:", val_f1)
```

[ ]:
```
padded = Input(shape = (17640,1))
masked = Input(shape = (17640), dtype = 'bool')
layer = LSTM(25)(padded, mask = masked)
layer = Dense(50, activation = 'relu', kernel_initializer = 'he_normal')(layer)
```

```
out = Dense(10, activation = 'softmax', kernel_initializer =␣
 ↪'glorot_normal')(layer)

model1 = Model(inputs = [padded, masked], outputs = out)
#printing the summary of model
model1.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, 25)            2700
['input_1[0][0]',
'input_2[0][0]']

 dense (Dense)                  (None, 50)            1300        ['lstm[0][0]']

 dense_1 (Dense)                (None, 10)            510         ['dense[0][0]']

==================================================================================
==================
Total params: 4,510
Trainable params: 4,510
Non-trainable params: 0
__________________________________________________________________________________
__________________
```

[ ]:
```
import datetime
logdir = os.path.join('logs', datetime.datetime.now().strftime('%Y%m%d-%H%M%S'))
```

[ ]:
```
tf.keras.backend.clear_session()
model1.compile(optimizer = tf.keras.optimizers.Adam(0.001), loss =␣
 ↪'sparse_categorical_crossentropy')

custom_calls = [tf.keras.callbacks.TensorBoard(logdir, histogram_freq = 1,␣
 ↪write_graph = True),
      Metrics(([X_test_pad_seq, X_test_mask], y_test)),
      tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', min_delta = 0.1,␣
 ↪patience = 3)]
```

```
model1.fit([X_train_pad_seq, X_train_mask], y_train, validation_data =␣
↪([X_test_pad_seq, X_test_mask], y_test),
          batch_size = 16, epochs = 10, callbacks = custom_calls)
```

```
Epoch 1/10
19/19 [==============================] - 4s 164ms/step
val_F1_score: 0.10499999999999998
88/88 [==============================] - 51s 448ms/step - loss: 2.3035 -
val_loss: 2.3026
Epoch 2/10
19/19 [==============================] - 4s 195ms/step
val_F1_score: 0.10000000000000002
88/88 [==============================] - 40s 458ms/step - loss: 2.3035 -
val_loss: 2.3026
Epoch 3/10
19/19 [==============================] - 3s 133ms/step
val_F1_score: 0.10000000000000002
88/88 [==============================] - 37s 426ms/step - loss: 2.3034 -
val_loss: 2.3026
Epoch 4/10
19/19 [==============================] - 3s 134ms/step
val_F1_score: 0.10000000000000002
88/88 [==============================] - 34s 392ms/step - loss: 2.3030 -
val_loss: 2.3026
```

[ ]:
```
<keras.callbacks.History at 0x7a9a80b0ec20>
```

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

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

### 1.1.2 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

[ ]:
```
X_train_pad_seq.shape
```

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

[ ]:
```
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
```

[ ]:
```
def create_spectrogram(arr):
  spect_list = []
  for i in arr:
    spect = convert_to_spectrogram(i)
    spect_list.append(spect)
  return(np.array(spect_list))
```

[ ]:
```
X_train_spectrogram = create_spectrogram(X_train_pad_seq)
```

[ ]:
```
X_test_spectrogram = create_spectrogram(X_test_pad_seq)
```

[ ]:
```
X_test_spectrogram.shape
```

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

shape we getis 1400, 64,35 here 64 is n_mels and 35 is no of frames There is a formula to calculate no of frames

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

[ ]:
```
# write the architecture of the model
#print model.summary and make sure that it is following point 2 mentioned above
input = Input(shape = (64,35,))
layer = LSTM(256, return_sequences = True)(input)
layer = tf.math.reduce_mean(layer, axis = -1)
layer = Dense(256, activation = 'relu', kernel_initializer = 'he_normal')(layer)
layer = Dense(128, activation = 'relu', kernel_initializer =␣
 ↪'glorot_normal')(layer)
```

```python
output = Dense(10, activation = 'softmax', kernel_initializer = 'glorot_normal')(layer)

model2 = Model(inputs = input, outputs = output)
#printing the model summary
model2.summary()
```

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

 lstm (LSTM)                 (None, 64, 256)           299008

 tf.math.reduce_mean (TFOpLa  (None, 64)               0
 mbda)

 dense (Dense)               (None, 256)               16640

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 349,834
Trainable params: 349,834
Non-trainable params: 0
_________________________________________________________________
```

1. input layer = batch_size * 64*35
2. in 2nd layer, 64 *128 i.e. each row passes with 128 cells
3. as return seq is true we get output 64*128 otherwise it would be 64*1
4. now we use global pooling i.e. mean value of 128 cells so output = 64*1
5.

[ ]:
```python
!rm -rf ./logs/
```

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

model2.compile(optimizer = tf.keras.optimizers.Adam(0.001), loss = 'sparse_categorical_crossentropy')
callback_list = [tf.keras.callbacks.TensorBoard(logdir, histogram_freq = 1, write_graph = True),
      Metrics([X_test_spectrogram, y_test]),
```

```
    tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', min_delta = 0.001,␣
↪patience = 5)]

model2.fit(X_train_spectrogram, y_train, validation_data = (X_test_spectrogram,␣
↪y_test),
           batch_size = 16, epochs = 50, callbacks = callback_list)
```

```
Epoch 1/50
 1/88 […] - ETA: 4:17 - loss: 2.3034

WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the
batch time (batch time: 0.0068s vs `on_train_batch_end` time: 0.0112s). Check
your callbacks.

19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.49833333333333335
88/88 [==============================] - 5s 23ms/step - loss: 1.9386 - val_loss:
1.3828
Epoch 2/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.6116666666666667
88/88 [==============================] - 1s 10ms/step - loss: 1.2669 - val_loss:
1.0695
Epoch 3/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.6833333333333333
88/88 [==============================] - 1s 10ms/step - loss: 0.9862 - val_loss:
0.8880
Epoch 4/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.6816666666666666
88/88 [==============================] - 1s 10ms/step - loss: 0.8373 - val_loss:
0.7923
Epoch 5/50
19/19 [==============================] - 0s 4ms/step
val_F1_score: 0.805
88/88 [==============================] - 1s 16ms/step - loss: 0.7399 - val_loss:
0.6203
Epoch 6/50
19/19 [==============================] - 0s 5ms/step
val_F1_score: 0.8000000000000002
88/88 [==============================] - 1s 15ms/step - loss: 0.6340 - val_loss:
0.5824
Epoch 7/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8066666666666665
88/88 [==============================] - 1s 16ms/step - loss: 0.5646 - val_loss:
0.5453
```

```
Epoch 8/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8483333333333335
88/88 [==============================] - 1s 11ms/step - loss: 0.5429 - val_loss:
0.4749
Epoch 9/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8266666666666667
88/88 [==============================] - 1s 11ms/step - loss: 0.4822 - val_loss:
0.4744
Epoch 10/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8483333333333335
88/88 [==============================] - 1s 11ms/step - loss: 0.4657 - val_loss:
0.4788
Epoch 11/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.865
88/88 [==============================] - 1s 11ms/step - loss: 0.4089 - val_loss:
0.3832
Epoch 12/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.875
88/88 [==============================] - 1s 11ms/step - loss: 0.3916 - val_loss:
0.3887
Epoch 13/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8866666666666667
88/88 [==============================] - 1s 11ms/step - loss: 0.3758 - val_loss:
0.3759
Epoch 14/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8716666666666667
88/88 [==============================] - 1s 10ms/step - loss: 0.3530 - val_loss:
0.3797
Epoch 15/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.87
88/88 [==============================] - 1s 10ms/step - loss: 0.3365 - val_loss:
0.3738
Epoch 16/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8883333333333333
88/88 [==============================] - 1s 11ms/step - loss: 0.3188 - val_loss:
0.3603
Epoch 17/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8666666666666667
```

```
88/88 [==============================] - 1s 10ms/step - loss: 0.3258 - val_loss:
0.3783
Epoch 18/50
19/19 [==============================] - 0s 4ms/step
val_F1_score: 0.8933333333333333
88/88 [==============================] - 1s 14ms/step - loss: 0.3111 - val_loss:
0.3271
Epoch 19/50
19/19 [==============================] - 0s 4ms/step
val_F1_score: 0.9016666666666667
88/88 [==============================] - 1s 16ms/step - loss: 0.2869 - val_loss:
0.3048
Epoch 20/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8866666666666667
88/88 [==============================] - 1s 16ms/step - loss: 0.2873 - val_loss:
0.3137
Epoch 21/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.9183333333333333
88/88 [==============================] - 1s 11ms/step - loss: 0.2583 - val_loss:
0.2613
Epoch 22/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8783333333333333
88/88 [==============================] - 1s 11ms/step - loss: 0.2644 - val_loss:
0.3548
Epoch 23/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.9033333333333333
88/88 [==============================] - 1s 10ms/step - loss: 0.2579 - val_loss:
0.3052
Epoch 24/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8933333333333333
88/88 [==============================] - 1s 10ms/step - loss: 0.2492 - val_loss:
0.3203
Epoch 25/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.8916666666666667
88/88 [==============================] - 1s 11ms/step - loss: 0.2588 - val_loss:
0.3074
Epoch 26/50
19/19 [==============================] - 0s 3ms/step
val_F1_score: 0.9116666666666666
88/88 [==============================] - 1s 11ms/step - loss: 0.2740 - val_loss:
0.2633
```

```
[ ]: <keras.callbacks.History at 0x7a9a4c70cc70>
```

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

```
Reusing TensorBoard on port 6006 (pid 3892), started 0:02:16 ago. (Use '!kill␣
↪3892' to kill it.)

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

### 1.1.3 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.

```
[ ]: ## 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]:  ## 30% change
        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
```

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

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

```
[ ]: 9
```

9 datapoints for one point

## 1.2 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)

```
[ ]: X_train, X_test, y_train,␣
↪y_test=train_test_split(df_audio['path'],df_audio['label'],random_state=45,test_size=0.
↪2,stratify=df_audio['label'])
```

```
[ ]: X_train.shape
```

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

3. 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.
4. Preprocess your X_test using load_wav function.
5. Convert the augmented_train_data and test_data to numpy arrays.
6. Perform padding and masking on augmented_train_data and test_data.
7. After padding define the model similar to model 1 and fit the data

```
[ ]: def create_aug_samples (arr,label):
       aug_data = []
       aug_label = []
       for pt , label in zip(arr.values, label.values):

         new_data = generate_augmented_data(pt)
         aug_data.extend(new_data)
         aug_label.extend([label]*9)
       # return aug_data , aug_label
       X_train_processed = pd.DataFrame(list(zip(aug_data, aug_label)),columns␣
      ↪=['raw_data', 'label'])
       return X_train_processed
```

```
[ ]: X_train_processed = create_aug_samples(X_train, y_train)
```

```
[ ]: test_data =[]
     for i in X_test.values:
         test_data.append(load_wav(i, get_duration = False))
     X_test_processed = pd.DataFrame({'raw_data' : test_data, 'label' : y_test.
      ↪values})
```

```
[ ]: X_test_processed
```

```
[ ]:                                              raw_data  label
     0    [-8.44707e-05, -8.357633e-05, -9.395467e-05, -…      1
     1    [-0.012328528, -0.021660486, -0.024156429, -0…      8
     2    [-0.012697448, -0.017050935, -0.017266486, -0…      0
     3    [0.008832167, 0.0115452595, 0.011243898, 0.011…      0
     4    [-0.00033190163, -0.0002754184, -0.00017003811…      0
     ..                                                …    …
     395  [-0.00018083575, -0.00025713706, -0.0002576629…      6
     396  [-8.487413e-05, -0.00027589238, -0.00045009854…      4
     397  [-0.008284398, -0.013819212, -0.015761238, -0…      8
     398  [0.009452392, 0.012606097, 0.01237566, 0.01192…      1
     399  [-0.00015275163, -0.00018364502, -0.0001810008…      4
```

```
[400 rows x 2 columns]
```

[ ]:
```
max_length=17640
```

[ ]:
```
def pad_mask (df,max_length):
  new_padded=[]
  new_masks = []
  for i in range (df['raw_data'].shape[0]):
    length = df['raw_data'][i].shape[0]
    if length <= max_length:
      new_add = [0] * (max_length - length)
      new_one = np.concatenate((df['raw_data'][i],new_add), axis=0)
      mask = [True]* length + [False]* (max_length - length)

    else:
      new_one = df['raw_data'][i][: max_length]   ##https://www.geeksforgeeks.
↪org/python-remove-last-k-elements-of-list/#:~:
↪text=%23%20initializing%20list%C2%A0,str(res))
      mask = [True]*max_length
    # new_one = np.concatenate((X_train_processed['raw_data'][i],new_add),␣
↪axis=0)
    new_padded.append(new_one)
    new_masks.append(mask)
  return np.array(new_padded), np.array(new_masks)
```

[ ]:
```
X_train_pad_seq, X_train_mask = pad_mask(X_train_processed, max_length)
X_test_pad_seq, X_test_mask = pad_mask(X_test_processed, max_length)
```

[ ]:
```
X_test_pad_seq.shape
```

[ ]:
```
(400, 17640)
```

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.

[ ]:
```
padded = Input(shape = (17640,1))
masked = Input(shape = (17640), dtype = 'bool')

layer = LSTM(50)(padded, mask = masked)
layer = Dense(128, activation = 'relu', kernel_initializer = 'he_normal')(layer)
output = Dense(10, activation = 'softmax', kernel_initializer =␣
↪'glorot_normal')(layer)
```

```
model3 = Model(inputs = [padded,masked], outputs = output)
```

[ ]:
```
model3.summary()
```

```
Model: "model_1"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape          Param #     Connected to
==================================================================================
==================
 input_2 (InputLayer)           [(None, 17640, 1)]    0           []

 input_3 (InputLayer)           [(None, 17640)]       0           []

 lstm_1 (LSTM)                  (None, 50)            10400
['input_2[0][0]',
'input_3[0][0]']

 dense_3 (Dense)                (None, 128)           6528
['lstm_1[0][0]']

 dense_4 (Dense)                (None, 10)            1290
['dense_3[0][0]']

==================================================================================
==================
Total params: 18,218
Trainable params: 18,218
Non-trainable params: 0
__________________________________________________________________________________
__________________
```

[ ]:
```
!rm -rf ./logs/
```

[ ]:
```
model3.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001),
               loss = 'sparse_categorical_crossentropy')

callback_list = [tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1,␣
 ↪write_graph = True),
      tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', min_delta = 0.05,␣
 ↪patience = 5),
      Metrics(([X_test_pad_seq, X_test_mask], y_test))]

model3.fit([X_train_pad_seq, X_train_mask], X_train_processed.label,␣
 ↪validation_data = ([X_test_pad_seq, X_test_mask], y_test),
           epochs = 25, callbacks = callback_list, batch_size = 128)
```

```
Epoch 1/25
13/13 [==============================] - 3s 145ms/step
val_F1_score: 0.0825
113/113 [==============================] - 72s 584ms/step - loss: 2.3030 -
val_loss: 2.3026
Epoch 2/25
13/13 [==============================] - 3s 201ms/step
val_F1_score: 0.10749999999999998
113/113 [==============================] - 62s 548ms/step - loss: 2.3030 -
val_loss: 2.3027
Epoch 3/25
13/13 [==============================] - 2s 144ms/step
val_F1_score: 0.10000000000000002
113/113 [==============================] - 59s 523ms/step - loss: 2.3029 -
val_loss: 2.3027
Epoch 4/25
13/13 [==============================] - 2s 144ms/step
val_F1_score: 0.10000000000000002
113/113 [==============================] - 59s 519ms/step - loss: 2.3028 -
val_loss: 2.3027
Epoch 5/25
13/13 [==============================] - 3s 195ms/step
val_F1_score: 0.0975
113/113 [==============================] - 60s 529ms/step - loss: 2.3028 -
val_loss: 2.3027
Epoch 6/25
13/13 [==============================] - 2s 142ms/step
val_F1_score: 0.10000000000000002
113/113 [==============================] - 58s 514ms/step - loss: 2.3028 -
val_loss: 2.3026
```

[ ]:
```
<keras.callbacks.History at 0x7a99e00939a0>
```

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

```
Reusing TensorBoard on port 6006 (pid 3892), started 0:13:44 ago. (Use '!kill
3892' to kill it.)
```

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

### 1.2.1 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

[ ]:
```
X_train_spectrogram = create_spectrogram(X_train_pad_seq)
```

[ ]:
```
X_test_spectrogram = create_spectrogram(X_test_pad_seq)
```

[ ]:
```
X_test_spectrogram.shape
```

[ ]: (400, 64, 35)

[ ]:
```
# write the architecture of the model
#print model.summary and make sure that it is following point 2 mentioned above
input = Input(shape = (64,35,))
layer = LSTM(256, return_sequences = True)(input)
layer = tf.math.reduce_mean(layer, axis = -1)
layer = Dense(256, activation = 'relu', kernel_initializer = 'he_normal')(layer)
layer = Dense(128, activation = 'relu', kernel_initializer =␣
 ↪'glorot_normal')(layer)
output = Dense(10, activation = 'softmax', kernel_initializer =␣
 ↪'glorot_normal')(layer)

model4 = Model(inputs = input, outputs = output)
#printing the model summary
model4.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 64, 35)]          0

 lstm_2 (LSTM)               (None, 64, 256)           299008

 tf.math.reduce_mean_1 (TFOp  (None, 64)               0
 Lambda)

 dense_5 (Dense)             (None, 256)               16640

 dense_6 (Dense)             (None, 128)               32896

 dense_7 (Dense)             (None, 10)                1290

=================================================================
Total params: 349,834
Trainable params: 349,834
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
!rm -rf ./logs/
```

[ ]:
```
model4.compile(optimizer = tf.keras.optimizers.Adam(0.001), loss =␣
 ↪'sparse_categorical_crossentropy')
callback_list = [tf.keras.callbacks.TensorBoard(logdir, histogram_freq = 1,␣
 ↪write_graph = True),
      Metrics([X_test_spectrogram, y_test]),
      tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', min_delta = 0.001,␣
 ↪patience = 3)]

model4.fit(X_train_spectrogram, X_train_processed.label, validation_data =␣
 ↪(X_test_spectrogram, y_test),
           batch_size = 128, epochs = 50, callbacks = callback_list)
```

```
Epoch 1/50
  4/113 [>…] - ETA: 2s - loss: 0.0984

WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the
batch time (batch time: 0.0101s vs `on_train_batch_end` time: 0.1024s). Check
your callbacks.

13/13 [==============================] - 1s 4ms/step
val_F1_score: 0.9475
113/113 [==============================] - 6s 25ms/step - loss: 0.0695 -
val_loss: 0.1337
Epoch 2/50
13/13 [==============================] - 0s 4ms/step
val_F1_score: 0.9675
113/113 [==============================] - 2s 15ms/step - loss: 0.0622 -
val_loss: 0.1041
Epoch 3/50
13/13 [==============================] - 0s 3ms/step
val_F1_score: 0.965
113/113 [==============================] - 1s 13ms/step - loss: 0.0683 -
val_loss: 0.1081
Epoch 4/50
13/13 [==============================] - 0s 3ms/step
val_F1_score: 0.965
113/113 [==============================] - 1s 12ms/step - loss: 0.0578 -
val_loss: 0.1204
Epoch 5/50
13/13 [==============================] - 0s 3ms/step
val_F1_score: 0.9725
113/113 [==============================] - 1s 13ms/step - loss: 0.0614 -
val_loss: 0.0761
Epoch 6/50
13/13 [==============================] - 0s 3ms/step
val_F1_score: 0.9575
113/113 [==============================] - 1s 13ms/step - loss: 0.0619 -
val_loss: 0.0972
```

```
Epoch 7/50
13/13 [==============================] - 0s 3ms/step
val_F1_score: 0.96
113/113 [==============================] - 1s 13ms/step - loss: 0.0671 -
val_loss: 0.0979
Epoch 8/50
13/13 [==============================] - 0s 3ms/step
val_F1_score: 0.955
113/113 [==============================] - 1s 13ms/step - loss: 0.0642 -
val_loss: 0.1164
```

[ ]: <keras.callbacks.History at 0x7a9a74de6140>

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

```
Reusing TensorBoard on port 6006 (pid 3892), started 0:19:57 ago. (Use '!kill␣
↪3892' to kill it.)
```

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

[ ]:
```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ['S. No.', 'Model', 'Train Loss', 'Val Loss', 'Val F1␣
↪Score']
table.add_row([1,'Raw Data', 2.3028, 2.3026, 0.1])
table.add_row([2, 'Spectrogram', 0.2740, 0.2633, 0.917])
table.add_row([3, 'Raw Data with Data Augmentation', 2.3028, 2.3026, 0.1])
table.add_row([4, 'Spectrogram with Data Augmentation', 0.0642, 0.1164, 0.955])
print(table)
```

```
+--------+------------------------------------+------------+----------+---------
-----+
| S. No. |               Model                | Train Loss | Val Loss | Val F1
Score |
+--------+------------------------------------+------------+----------+---------
-----+
|   1    |              Raw Data              |   2.3028   |  2.3026  |     0.1
|
|   2    |            Spectrogram             |   0.274    |  0.2633  |    0.917
|
|   3    |  Raw Data with Data Augmentation   |   2.3028   |  2.3026  |     0.1
|
|   4    | Spectrogram with Data Augmentation |   0.0642   |  0.1164  |    0.955
|
+--------+------------------------------------+------------+----------+---------
-----+
```

**CONCLUSION :** 1. with only raw data model is performing very poorly. Even with the augmented data, val_F1_score is not increasing. 2. With spectrogram, val_f1_score increased tremendously and with the addition of data augmentation, f1_score went to 0.955.