Spoken Digit Recognition

In this notebook, You will do Spoken Digit Recognition.

Input - speech signal, output - digit number

It contains

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

Instructions:

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

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

Saving...

ormat what we asked. Eg. Don't return List of we are asking for a numpy array.

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

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

Every Grader function has to return True.

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

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

```
!pip install gdown
!pip install librosa
import numpy as np
import pandas as pd
import librosa
import os
import gdown
import zipfile
import os
import sklearn
from sklearn.model_selection import train_test_split
import librosa
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import tensorflow as tf
from tqdm import tqdm
from sklearn.utils import shuffle
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, LSTM, Dense,Dropout
from tensorflow.keras.models import Model
from sklearn.metrics import roc_auc_score,f1_score
from keras.callbacks import ModelCheckpoint,TensorBoard
##if you need any imports you can do that here.
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gdown in /usr/local/lib/python3.8/dist-packages (4.4.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.8/dist-packages (from gdown) (4.6.3)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.8/dist-packages (from gdown) (2.25.1)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from gdown) (1.15.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from gdown) (4.64.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from gdown) (3.9.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown) (1.26.14)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown) (2022.12.7)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown) (2.10)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown) (1.7.1)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: librosa in /usr/local/lib/python3.8/dist-packages (0.8.1)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.8/dist-packages (from librosa) (1.22.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages (from librosa) (23.0)
Requirement already satisfied: numba>=0.43.0 in /usr/local/lib/python3.8/dist-packages (from librosa) (0.56.4)
Requirement already satisfied: joblib>=0.14 in /usr/local/lib/python3.8/dist-packages (from librosa) (1.2.0)
Requirement already satisfied: resampy>=0.2.2 in /usr/local/lib/python3.8/dist-packages (from librosa) (0.4.2)
Requirement already satisfied: soundfile>=0.10.2 in /usr/local/lib/python3.8/dist-packages (from librosa) (0.12.1)
Requirement already satisfied: pooch>=1.0 in /usr/local/lib/python3.8/dist-packages (from librosa) (1.7.0)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from librosa) (1.10.1)
Requirement already satisfied: decorator>=3.0.0 in /usr/local/lib/python3.8/dist-packages (from librosa) (4.4.2)
Requirement already satisfied: scikit-learn!=0.19.0,>=0.14.0 in /usr/local/lib/python3.8/dist-packages (from librosa) (1.2.1)
Requirement already satisfied: audioread>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from librosa) (3.0.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.8/dist-packages (from numba>=0.43.0->librosa) (6.0.0)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.8/dist-packages (from numba>=0.43.0->librosa) (0.39.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from numba>=0.43.0->librosa) (57.4.0)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.8/dist-packages (from pooch>=1.0->librosa) (2.25.1)
Requirement already satisfied: platformdirs>=2.5.0 in /usr/local/lib/python3.8/dist-packages (from pooch>=1.0->librosa) (3.0.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn!=0.19.0,>=0.14.0->librosa) (3.1.0)
```

```
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.8/dist-packages (from soundfile>=0.10.2->librosa) (1.15.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from cffi>=1.0->soundfile>=0.10.2->librosa) (2.21)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests>=2.19.0->pooch>=1.0->librosa) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2022.12.7)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests>=2.19.0->pooch>=1.0->librosa) (1.26.14)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.8/dist-packages (from importlib-metadata->numba>=0.43.0->librosa) (3.15.0)
```

We shared recordings.zip, please unzip those.

```
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

```
!gdown 1BqrqsLqWvKa2m1Dw7KeWlME723VRpkTO
```

```
Downloading...
From: https://drive.google.com/uc?id=1BqrqsLqWvKa2m1Dw7KeWlME723VRpkTO
To: /content/Copy of recordings.zip
100% 9.28M/9.28M [00:00<00:00, 14.4MB/s]
```

```
zip_ref = zipfile.ZipFile("/content/drive/MyDrive/Copy of recordings.zip", "r")
zip_ref.extractall()
zip_ref.close()
```

Saving...

```
#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 = [file for file in tqdm(os.listdir("/content/recordings"))]
```

```
100%|██████████| 2000/2000 [00:00<00:00, 840120.98it/s]
```

Grader function 1

```
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()
```

```
True
```

Create a dataframe(name=df_audio) with two columns(path, label).

You can get the label from the first letter of name.

Eg: 0_jackson_0 --> 0

0_jackson_43 --> 0

```
audio_data = [{'path':path,'label':path[0]} for path in tqdm(all_files)]
df_audio = pd.DataFrame(audio_data)
```

```
100%|██████████| 2000/2000 [00:00<00:00, 575626.71it/s]
```

```
df_audio.head()
```

	path	label
0	1_yweweler_6.wav	1
1	8_yweweler_48.wav	8
2	6_yweweler_35.wav	6
3	7_theo_16.wav	7
4	7_nicolas_40.wav	7

Exploring the sound dataset

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

```
# download the thinkdsp library
!git clone https://github.com/AllenDowney/ThinkDSP.git
```

```
Cloning into 'ThinkDSP'...
remote: Enumerating objects: 2469, done.
remote: Total 2469 (delta 0), reused 0 (delta 0), pack-reused 2469
Receiving objects: 100% (2469/2469), 208.82 MiB | 20.26 MiB/s, done.
Resolving deltas: 100% (1353/1353), done.
```

```
import sys
sys.path.insert(0, 'ThinkDSP/code/')
import thinkdsp
```

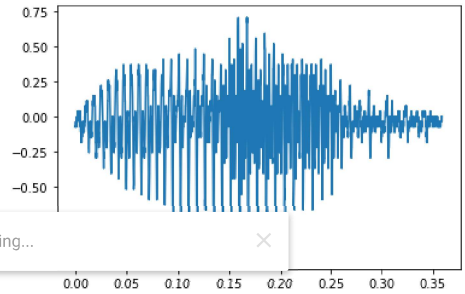
```
import matplotlib.pyplot as pyplot
import IPython
```

```
wave = thinkdsp.read_wave('/content/recordings/0_nicolas_32.wav')
wave.play()
IPython.display.Audio('sound.wav')
```

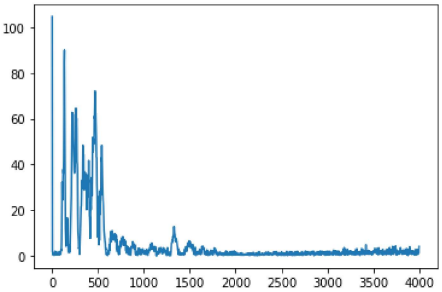
```
Writing sound.wav
```

0:00 / 0:00

```
segment = wave.segment()
segment.plot()
pyplot.show()
```




```
# Plot spectrum of wave
spectrum = wave.make_spectrum()
spectrum.plot()
pyplot.show()
```

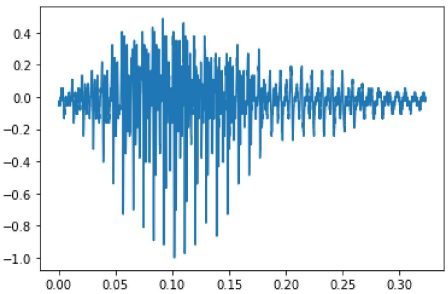



```
wave = thinkdsp.read_wave('/content/recordings/1_nicolas_32.wav')
wave.play()
IPython.display.Audio('sound.wav')
```

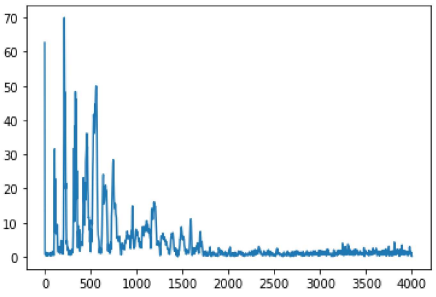
```
Writing sound.wav
```

0:00 / 0:00

```
segment = wave.segment(start=0)
segment.plot()
pyplot.show()
```




```
# Plot spectrum of wave
spectrum = wave.make_spectrum()
spectrum.plot()
pyplot.show()
```



Creating dataframe

```
#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

```

```
#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


```
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()
```

```
True
```

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

Train and Validation split

```
#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 = df_audio.path
y = df_audio.label
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=45,stratify=y)
```

Grader function 3

```
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()
```

```
True
```

Preprocessing

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

```
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
def get_full_path(x):
```

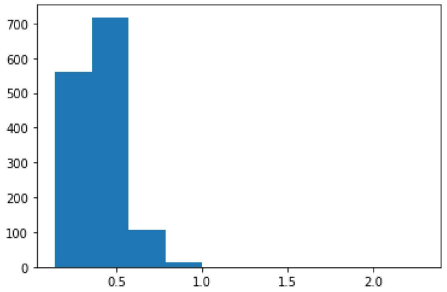
```
    return "/content/recordings/"+x
#data[['c','d']] = [givetup(a) for a in data['b']]
X_train_processed = pd.DataFrame()
X_test_processed = pd.DataFrame()
X_train_processed[['raw_data', 'duration']] = [load_wav(get_full_path(x)) for x in X_train]
X_test_processed[['raw_data', 'duration']] = [load_wav(get_full_path(x)) for x in X_test]
```

```
X_train_processed.head()
```

	raw_data	duration
0	[-0.006193983, -0.0053070816, -0.0017616726, 0...	0.58127
1	[-0.00013747877, -0.0002816826, -0.00045083734...	0.24966
2	[-0.004207985, -0.0012800224, 0.002231387, 0.0...	0.714785
3	[-0.000104876926, -9.907236e-05, -6.296669e-05...	0.303401
4	[-0.0005594357, -0.0001103753, 0.00031156163, ...	0.330522

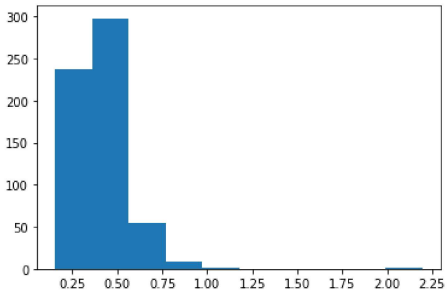
```
#plot the histogram of the duration for trian
plt.hist(X_train_processed.duration)
```

```
(array([561., 718., 107.,  12.,   1.,   0.,   0.,   0.,   0.,   1.]),
 array([0.1435374149659864, 0.3574603174603175, 0.5713832199546486,
        [illegible]992290249433107, 1.2131519274376419,
        [illegible]40997732426304, 1.854920634920635,
        [illegible]2766439909297], dtype=object),
 <BarContainer object of 10 artists>)
```




```
#plot the histogram of the duration for test
plt.hist(X_test_processed.duration)
```

```
(array([237., 298.,  55.,   8.,   1.,   0.,   0.,   0.,   0.,   1.]),
 array([0.1564172335600907, 0.36036734693877553, 0.5643174603174603,
        0.7682675736961452, 0.97221768707483, 1.1761678004535148,
        1.3801179138321995, 1.5840680272108845, 1.7880181405895692,
        1.991968253968254, 2.195918367346939], dtype=object),
 <BarContainer object of 10 artists>)
```




```
#print 0 to 100 percentile values with step size of 10 for train data duration.
for i in range(0,11,1):
    print(f"{i*10}th percentile is {np.percentile(X_train_processed.duration.values, i*10)}")
```

```
0th percentile is 0.1435374149659864
10th percentile is 0.25997732426303855
20th percentile is 0.298140589569161
30th percentile is 0.32884807256235826
40th percentile is 0.3573061224489796
50th percentile is 0.38770975056689344
60th percentile is 0.41365986394557824
70th percentile is 0.4454104308390022
80th percentile is 0.4823401360544218
90th percentile is 0.5528934240362812
100th percentile is 2.282766439909297
```

```
##print 90 to 100 percentile values with step size of 1.
for i in range(0,11,1):
    print(f"{i+90}th percentile is {np.percentile(X_train_processed.duration.values, i+90)}")
```

```
90th percentile is 0.5528934240362812
91th percentile is 0.5619047619047619
92th percentile is 0.5790748299319729
93th percentile is 0.5974403628117915
94th percentile is 0.610172335600907
95th percentile is 0.624079365079365
96th percentile is 0.6379374149659863
97th percentile is 0.6528367346938775
98th percentile is 0.6823900226757368
99th percentile is 0.7832580498866212
100th percentile is 2.282766439909297
```

Grader function 4

```
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()
```

```
True
```

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

While loading the audio files, we are using sampling rate of 22050 so one sec will give array of length 22050. so, our maximum length is 0.8*22050 = 17640 Pad with Zero if length of sequence is less than 17640 else Truncate the number.

Also create a masking vector for train and test.

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

```
max_length  = 17640
```



```
s(X_train_processed.raw_data, maxlen = max_length, padding = 'post',dtype = 'float32',truncating = 'post')
(X_test_processed.raw_data, maxlen = max_length,padding = 'post',dtype = 'float32',truncating = 'post')

X_train_mask = X_train_pad_seq != 0
X_test_mask = X_test_pad_seq != 0
```

```
## 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.
```

Grader function 5

```
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()
```

```
True
```

1. Giving Raw data directly.

Now we have

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

We will create a LSTM model which takes this input.

Task:

1. Create an LSTM network which takes "X_train_pad_seq" as input, "X_train_mask" as mask input. You can use any number of LSTM cells. Please read LSTM documentation(https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM) in tensorflow to know more about mask and also https://www.tensorflow.org/guide/keras/masking_and_padding
2. Get the final output of the LSTM and give it to Dense layer of any size and then give it to Dense layer of size 10(because we have 10 outputs) and then compile with the sparse categorical cross entropy(because we are not converting it to one hot vectors). Also check the datatype of class labels(y_values) and make sure that you convert your class labels to integer datatype before fitting in the model.
3. While defining your model make sure that you pass both the input layer and mask input layer as input to lstm layer as follows `lstm_output = self.lstm(input_layer, mask=masking_input_layer)`
4. Use tensorboard to plot the graphs of loss and metric(use custom micro F1 score as metric) and histograms of gradients. You can write your code for computing F1 score using this link
5. make sure that it won't overfit.
6. You are free to include any regularization

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

```

```
X_train_pad_seq = np.expand_dims(X_train_pad_seq,-1)
X_test_pad_seq = np.expand_dims(X_test_pad_seq,-1)
y_train = y_train.values.astype('int')
y_test = y_test.values.astype('int')
```

```
class F1ScoreCB(tf.keras.callbacks.Callback):
    def __init__(self, train_data, test_data):
        super().__init__()
        self.train_data = train_data
        self.test_data = test_data
        self.history = {}
        self.history['val_f1_score'] = []

    def on_epoch_end(self, epochs, logs = {}):
        train_preds = np.argmax(self.model.predict(self.train_data[0]), axis = -1)
        train_f1_score = f1_score(self.train_data[1], train_preds, average='micro')
        train_f1_score = np.round(train_f1_score, 4)

        test_preds = np.argmax(self.model.predict(self.test_data[0]), axis = -1)
        test_f1_score = f1_score(self.test_data[1], test_preds, average='micro')
        test_f1_score = np.round(test_f1_score, 4)
        self.history['val_f1_score'].append(test_f1_score)

        print(f" - f1_score: {train_f1_score} - val_f1_score: {test_f1_score}")

        writer1 = tf.summary.create_file_writer(logdir + '/train_f1_score')
        writer2 = tf.summary.create_file_writer(logdir + '/validation_f1_score')

        #writing to tensoboard
        with writer1.as_default():
            tf.summary.scalar('F1_Score', train_f1_score, step=epochs)

        with writer2.as_default():
            tf.summary.scalar('F1_Score', test_f1_score, step=epochs)
        writer2.flush()
```



```
f1_score_cb = F1ScoreCB(([X_train_pad_seq, X_train_mask], y_train), ([X_test_pad_seq, X_test_mask], y_test))
```

```
#creating the first model
x= Input(shape = (17640,1)) # this will take inputs of 17640 time steps, each having dimension = 1
mask_indicator = Input(shape =(17640),dtype='bool') # to tell the model,ignore the pads
lstm_1 = LSTM(units=256)(x,mask=mask_indicator)
dense_1 = Dense(100,activation='relu')(lstm_1)
dense_2 = Dense(50,activation='relu')(dense_1)
dense_3 = Dense(20,activation='relu')(dense_2)
output = Dense(10, activation='softmax')(dense_3)
model_1 = Model(inputs=[x, mask_indicator], outputs=output)
opt = tf.keras.optimizers.Adam(learning_rate=0.00001)
model_1.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
model_1.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, 256)          264192      ['input_1[0][0]',
                                                                  'input_2[0][0]']

 dense (Dense)                  (None, 100)          25700       ['lstm[0][0]']

 dense_1 (Dense)                (None, 50)           5050        ['dense[0][0]']

 dense_2 (Dense)                (None, 20)           1020        ['dense_1[0][0]']

 dense_3 (Dense)                (None, 10)           210         ['dense_2[0][0]']

==================================================================================================
Total params: 296,172
Trainable params: 296,172
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
tensorboard = TensorBoard(log_dir = './logs')

checkpoint = ModelCheckpoint('best_model_1.h',save_best_only=True,mode='auto')
```

```
!rm -rf ./logs/
import datetime

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

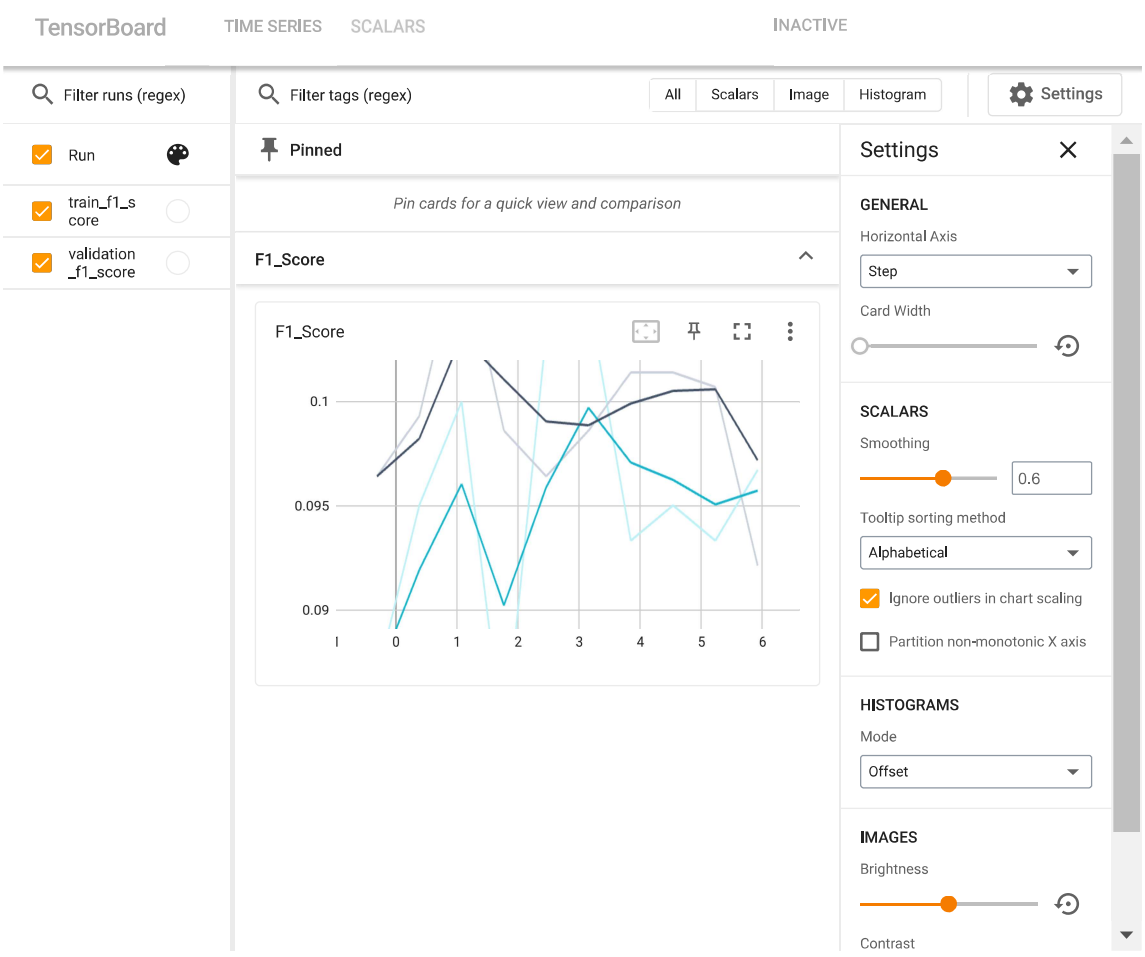
```
history = model_1.fit([X_train_pad_seq, X_train_mask], y_train,
            validation_data = ([X_test_pad_seq, X_test_mask], y_test),
            batch_size = 16 , epochs = 10,callbacks=[f1_score_cb,checkpoint,tensorboard],verbose=1)
```

```
Epoch 1/10
44/44 [==============================] - 11s 239ms/step
19/19 [==============================] - 5s 243ms/step
 - f1_score: 0.0964 - val_f1_score: 0.0867
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These func
88/88 [==============================] - 88s 1s/step - loss: 2.3026 - accuracy: 0.0929 - val_loss: 2.3026 - val_accuracy: 0.0867
Epoch 2/10
44/44 [==============================] - 11s 238ms/step
19/19 [==============================] - 5s 257ms/step
 - f1_score: 0.0993 - val_f1_score: 0.095
```

```
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These func
88/88 [==============================] - 86s 975ms/step - loss: 2.3026 - accuracy: 0.1014 - val_loss: 2.3026 - val_accuracy: 0.0950
Epoch 3/10
44/44 [==============================] - 11s 239ms/step
19/19 [==============================] - 5s 244ms/step
 - f1_score: 0.1079 - val_f1_score: 0.1
88/88 [==============================] - 75s 861ms/step - loss: 2.3026 - accuracy: 0.0950 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 4/10
44/44 [==============================] - 11s 238ms/step
19/19 [==============================] - 5s 239ms/step
 - f1_score: 0.0986 - val_f1_score: 0.0833
88/88 [==============================] - 76s 864ms/step - loss: 2.3025 - accuracy: 0.0964 - val_loss: 2.3026 - val_accuracy: 0.0833
Epoch 5/10
44/44 [==============================] - 11s 240ms/step
19/19 [==============================] - 5s 241ms/step
 - f1_score: 0.0964 - val_f1_score: 0.1033
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These func
88/88 [==============================] - 75s 855ms/step - loss: 2.3025 - accuracy: 0.0929 - val_loss: 2.3026 - val_accuracy: 0.1033
Epoch 6/10
44/44 [==============================] - 11s 242ms/step
19/19 [==============================] - 5s 246ms/step
 - f1_score: 0.0986 - val_f1_score: 0.105
88/88 [==============================] - 64s 731ms/step - loss: 2.3025 - accuracy: 0.0900 - val_loss: 2.3026 - val_accuracy: 0.1050
Epoch 7/10
44/44 [==============================] - 11s 239ms/step
19/19 [==============================] - 5s 244ms/step
 - f1_score: 0.1014 - val_f1_score: 0.0933
88/88 [==============================] - 66s 747ms/step - loss: 2.3025 - accuracy: 0.0857 - val_loss: 2.3026 - val_accuracy: 0.0933

Saving...

==============] - 11s 238ms/step
==============] - 5s 243ms/step
 - f1_score: 0.1014 - val_f1_score: 0.095
88/88 [==============================] - 66s 750ms/step - loss: 2.3025 - accuracy: 0.0986 - val_loss: 2.3026 - val_accuracy: 0.0950
Epoch 9/10
44/44 [==============================] - 11s 239ms/step
19/19 [==============================] - 5s 239ms/step
 - f1_score: 0.1007 - val_f1_score: 0.0933
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These func
88/88 [==============================] - 85s 971ms/step - loss: 2.3025 - accuracy: 0.0986 - val_loss: 2.3026 - val_accuracy: 0.0933
Epoch 10/10
44/44 [==============================] - 11s 235ms/step
19/19 [==============================] - 5s 247ms/step
 - f1_score: 0.0921 - val_f1_score: 0.0967
88/88 [==============================] - 76s 869ms/step - loss: 2.3025 - accuracy: 0.0936 - val_loss: 2.3026 - val_accuracy: 0.0967
```

```
1 %reload_ext tensorboard
2 %tensorboard --logdir $logdir
```

```
Reusing TensorBoard on port 6006 (pid 7926), started 0:00:06 ago. (Use '!kill 7926' to kill it.)
```



We get a good fit model at epoch 5

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

```
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
```

```
X_train_pad_seq = np.squeeze(X_train_pad_seq, axis = -1)
X_test_pad_seq = np.squeeze(X_test_pad_seq, axis = -1)
```

```
##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 = np.array([convert_to_spectrogram(seq) for seq in X_train_pad_seq])
X_test_spectrogram = np.array([convert_to_spectrogram(seq) for seq in X_test_pad_seq])
```

Grader function 6

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

```
True
```

Now we have

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

We will create a LSTM model which takes this input.

Task:

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

```
# write the architecture of the model
#print model.summary and make sure that it is following point 2 mentioned above
x= Input(shape = (64,35,)) # this will take inputs of 64 time steps, each 35 dimensional
lstm_1 = LSTM(units=256,return_sequences=True)(x)
lstm_1_averaged = tf.math.reduce_mean(lstm_1, axis = -1)
dense_1 = Dense(256,activation='relu',kernel_initializer = 'he_normal')(lstm_1_averaged)
dense_2 = Dense(100,activation='relu')(dense_1)
dense_3 = Dense(50,activation='relu',kernel_initializer = 'he_normal')(dense_2)
output = Dense(10, activation='softmax')(dense_3)
model_2 = Model(inputs=x, outputs=output)
opt = tf.keras.optimizers.Adam() # usign the default learning rate
model_2.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
model_2.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, 100)               25700

 dense_2 (Dense)             (None, 50)                5050

 dense_3 (Dense)             (None, 10)                510

=================================================================
Total params: 346,908
Trainable params: 346,908
Non-trainable params: 0
_________________________________________________________________
```

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

```
import datetime
!rm -rf ./logs/
```

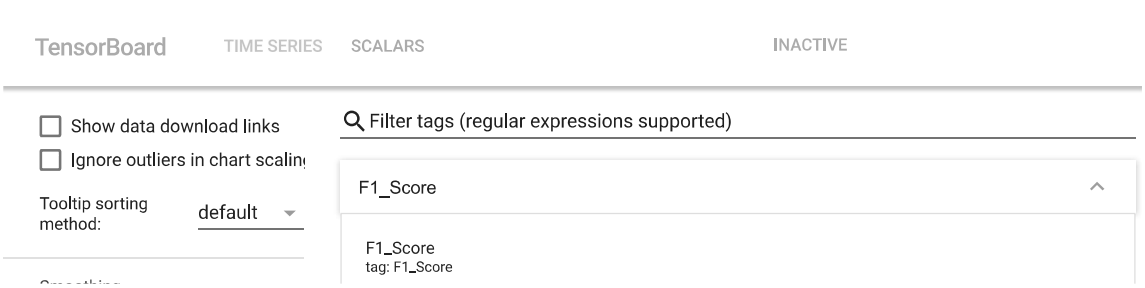
```
logdir = os.path.join('logs', datetime.datetime.now().strftime('%Y%m%d-%H%M%S'))
tensorboard = TensorBoard(log_dir = './logs')
checkpoint = ModelCheckpoint('best_model_2.h',save_best_only=True,mode='auto')
f1_score_cb = F1ScoreCB((X_train_spectrogram, y_train), (X_test_spectrogram, y_test))
```

```
history = model_2.fit(X_train_spectrogram, y_train,
            validation_data = (X_test_spectrogram, y_test),
            batch_size = 16 , epochs = 30,verbose=1,callbacks=[f1_score_cb,checkpoint,tensorboard])
```

```
Epoch 1/30
 6/88 [=>............................] - ETA: 0s - loss: 2.3025 - accuracy: 0.0625       WARNING:tensorflow:Callback method `on_train_batch_end` is slow compare
44/44 [==============================] - 1s 3ms/step
19/19 [==============================] - 0s 3ms/step
 - f1_score: 0.3407 - val_f1_score: 0.2933
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 23s 98ms/step - loss: 2.1544 - accuracy: 0.1829 - val_loss: 1.8738 - val_accuracy: 0.2933
Epoch 2/30
44/44 [==============================] - 0s 3ms/step
19/19 [==============================] - 0s 3ms/step
 - f1_score: 0.4693 - val_f1_score: 0.445
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 6s 72ms/step - loss: 1.6410 - accuracy: 0.4043 - val_loss: 1.4439 - val_accuracy: 0.4450
Epoch 3/30
44/44 [==============================] - 0s 3ms/step
19/19 [==============================] - 0s 3ms/step
 - f1_score: 0.5879 - val_f1_score: 0.5933
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 7s 84ms/step - loss: 1.2649 - accuracy: 0.5464 - val_loss: 1.1156 - val_accuracy: 0.5933
Epoch 4/30
44/44 [==============================] - 0s 3ms/step
19/19 [==============================] - 0s 3ms/step
 - f1_score: 0.6743 - val_f1_score: 0.6717
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 7s 81ms/step - loss: 1.0782 - accuracy: 0.6036 - val_loss: 1.0012 - val_accuracy: 0.6717
Epoch 5/30
44/44 [==============================] - 0s 3ms/step
19/19 [==============================] - 0s 5ms/step
 - f1_score: 0.7393 - val_f1_score: 0.7283
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 7s 80ms/step - loss: 0.8854 - accuracy: 0.6971 - val_loss: 0.8146 - val_accuracy: 0.7283
Epoch 6/30
44/44 [==============================] - 0s 3ms/step
19/19 [==============================] - 0s 3ms/step
 - f1_score: 0.7293 - val_f1_score: 0.7183
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 6s 73ms/step - loss: 0.8287 - accuracy: 0.6936 - val_loss: 0.7695 - val_accuracy: 0.7183
Epoch 7/30
44/44 [==============================] - 0s 3ms/step
19/19 [==============================] - 0s 4ms/step
 - f1_score: 0.7721 - val_f1_score: 0.7533
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 7s 82ms/step - loss: 0.7451 - accuracy: 0.7271 - val_loss: 0.6852 - val_accuracy: 0.7533
Epoch 8/30
44/44 [==============================] - 0s 3ms/step
19/19 [==============================] - 0s 3ms/step
 - f1_score: 0.7993 - val_f1_score: 0.7467
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 7s 75ms/step - loss: 0.7105 - accuracy: 0.7350 - val_loss: 0.6593 - val_accuracy: 0.7467
Epoch 9/30
44/44 [==============================] - 0s 5ms/step
19/19 [==============================] - 0s 4ms/step
 - f1_score: 0.8029 - val_f1_score: 0.78
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These f
88/88 [==============================] - 8s 97ms/step - loss: 0.6537 - accuracy: 0.7657 - val_loss: 0.5610 - val_accuracy: 0.7800
Epoch 10/30
44/44 [==============================] - 0s 3ms/step
```



```
%reload_ext tensorboard
%tensorboard --logdir $logdir
```



After converting the time domain speech recodrdings data to frequency domain, we are able to learn the classification in a much better manner. We trained it for 30 epochs. Clearly it can be trained for an even larger epochs, we stopped due to compuational limitation.

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

Saving...

es the frequencies higher or lower. For pitch shifting we shift up or down one half-step.

train f1 score

```
## 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
```

```
temp_path = df_audio.iloc[0].path
aug_temp = generate_augmented_data('/content/recordings/'+temp_path)
```

```
np.array(aug_temp).shape
```

```
(9,)
```

Follow the steps

1. Split data 'df_audio' into train and test (80-20 split)
2. We have 2000 data points(1600 train points, 400 test points)

```
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'])
```

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

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.

```
X_train_augmented = []
y_train_augmented = []
for path,label in tqdm(zip(X_train.values,y_train.values),total=1600):
    path = "/content/recordings/"+path
    augmented_datapoints = generate_augmented_data(path)
    augmented_labels = [label]*9
    X_train_augmented.extend(augmented_datapoints)
    y_train_augmented.extend(augmented_labels)
```

```
100%|██████████| 1600/1600 [05:41<00:00,  4.68it/s]
```

```
X_test_augmented = []
y_test_augmented = []
for path,label in tqdm(zip(X_test.values,y_test.values),total=400):
    path = "/content/recordings/"+path
    augmented_datapoints = generate_augmented_data(path)
    augmented_labels = [label]*9
    X_test_augmented.extend(augmented_datapoints)
    y_test_augmented.extend(augmented_labels)
```

```
100%|██████████| 400/400 [01:32<00:00,  4.32it/s]
```

```
X_train_augmented = np.array(X_train_augmented)
X_test_augmented = np.array(X_test_augmented)
```

```
y_train_augmented = np.array(y_train_augmented,dtype='int')
y_test_augmented = np.array(y_test_augmented,dtype='int')
```

```
x= Input(shape = (17640,1)) # this will take inputs of 17640 time steps, each having dimension = 1
mask_indicator = Input(shape =(17640),dtype='bool') # to tell the model,ignore the pads
lstm_1 = LSTM(units=256)(x,mask=mask_indicator)
dense_1 = Dense(100,activation='relu')(lstm_1)
dense_2 = Dense(50,activation='relu')(dense_1)
dense_3 = Dense(20,activation='relu')(dense_2)
                'softmax')(dense_3)
                _indicator], outputs=output)
opt = tf.keras.optimizers.Adam(learning_rate=0.00001)
model_3.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
model_3.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, 256)          264192      ['input_1[0][0]',
                                                                  'input_2[0][0]']

 dense (Dense)                  (None, 100)          25700       ['lstm[0][0]']

 dense_1 (Dense)                (None, 50)           5050        ['dense[0][0]']

 dense_2 (Dense)                (None, 20)           1020        ['dense_1[0][0]']

 dense_3 (Dense)                (None, 10)           210         ['dense_2[0][0]']

==================================================================================================
Total params: 296,172
Trainable params: 296,172
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
!rm -rf ./logs/
import datetime
logdir = os.path.join('logs', datetime.datetime.now().strftime('%Y%m%d-%H%M%S'))
```

```
import pickle
X_train_augmented = open("/content/drive/MyDrive/Copy of X_train_augmented (1).pkl", "rb")
X_test_augmented = open("/content/drive/MyDrive/Copy of X_test_augmented.pkl","rb")
y_train_augmented = open("/content/drive/MyDrive/Copy of y_train_augmented.pkl","rb")
y_test_augmented = open("/content/drive/MyDrive/Copy of y_test_augmented.pkl","rb")
X_train_augmented = pickle.load(X_train_augmented)
X_test_augmented = pickle.load(X_test_augmented)
y_train_augmented = pickle.load(y_train_augmented)
y_test_augmented = pickle.load(y_test_augmented)
```

```
max_length  = 17640
X_train_augmented_padded = pad_sequences(X_train_augmented, maxlen = max_length,dtype = 'float32', padding = 'post',truncating = 'post')
X_test_augmented_padded = pad_sequences(X_test_augmented, maxlen = max_length, dtype = 'float32', padding = 'post', truncating = 'post')

X_train_augmented_mask = X_train_augmented_padded != 0
X_test_augmented_mask = X_test_augmented_padded != 0
```

```
X_train_augmented_padded = np.expand_dims(X_train_augmented_padded, -1)
X_test_augmented_padded = np.expand_dims(X_test_augmented_padded, -1)
```

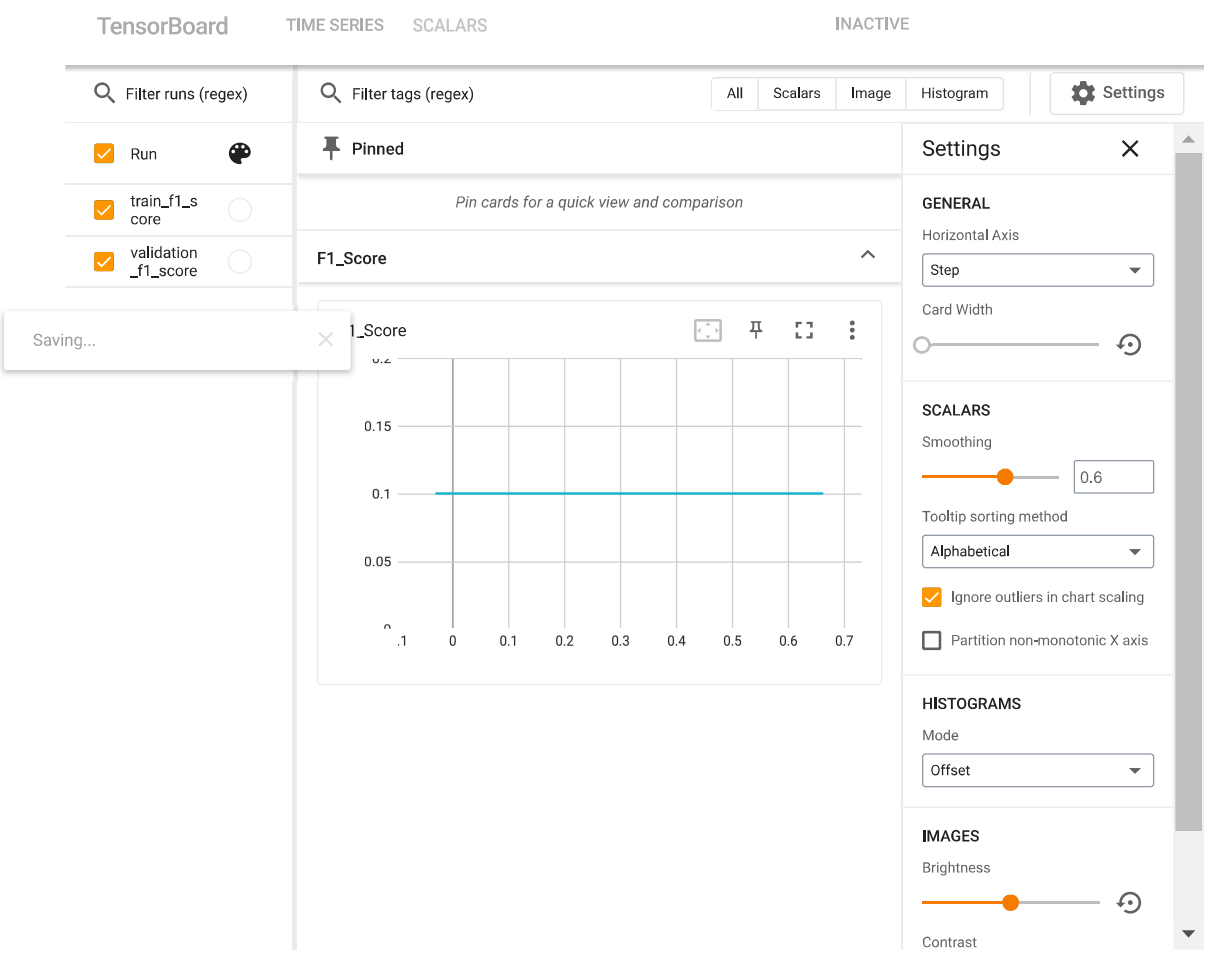
```
tensorboard = TensorBoard(log_dir = './logs')
checkpoint = ModelCheckpoint('best_model_3.h',save_best_only=True,mode='auto')
f1_score_cb = F1ScoreCB(([X_train_augmented_padded, X_train_augmented_mask], y_train_augmented), ([X_test_augmented_padded, X_test_augmented_mask], y_test
```

```
history = model_3.fit([X_train_augmented_padded, X_train_augmented_mask], y_train_augmented,
            validation_data = ([X_test_augmented_padded, X_test_augmented_mask], y_test_augmented),
            batch_size = 32 , epochs = 2 ,callbacks=[f1_score_cb,checkpoint,tensorboard],verbose=1)
```

```
Epoch 1/2
450/450 [==============================] - 125s 274ms/step
113/113 [==============================] - 33s 275ms/step
 - f1_score: 0.1 - val_f1_score: 0.1
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These func
450/450 [==============================] - 532s 1s/step - loss: 2.3026 - accuracy: 0.0973 - val_loss: 2.3026 - val_accuracy: 0.1000
```

```
Epoch 2/2
450/450 [==============================] - 123s 274ms/step
113/113 [==============================] - 31s 273ms/step
 - f1_score: 0.1 - val_f1_score: 0.1
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses while saving (showing 2 of 2). These func
450/450 [==============================] - 504s 1s/step - loss: 2.3026 - accuracy: 0.0970 - val_loss: 2.3026 - val_accuracy: 0.1000
```

```
%reload_ext tensorboard
%tensorboard --logdir $logdir
```



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_augmented_padded = np.squeeze(X_train_augmented_padded, axis = -1)
X_test_augmented_padded = np.squeeze(X_test_augmented_padded, axis = -1)
```

```
X_train_spectrogram_augmented = np.array([convert_to_spectrogram(seq) for seq in X_train_augmented_padded])
X_test_spectrogram_augmented = np.array([convert_to_spectrogram(seq) for seq in X_test_augmented_padded])
```

```
x= Input(shape = (64,35,)) # this will take inputs of 64 time steps, each 35 dimensional
lstm_1 = LSTM(units=256,return_sequences=True)(x)
lstm_1_averaged = tf.math.reduce_mean(lstm_1, axis = -1)
dense_1 = Dense(256,activation='relu',kernel_initializer = 'he_normal')(lstm_1_averaged)
dense_2 = Dense(100,activation='relu')(dense_1)
dense_3 = Dense(50,activation='relu',kernel_initializer = 'he_normal')(dense_2)
output = Dense(10, activation='softmax')(dense_3)
model_4 = Model(inputs=x, outputs=output)
opt = tf.keras.optimizers.Adam(learning_rate=0.00005)
model_4.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
model_4.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 64, 35)]          0

lstm_1 (LSTM)                (None, 64, 256)           299008

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

dense_4 (Dense)              (None, 256)               16640

dense_5 (Dense)              (None, 100)               25700
```

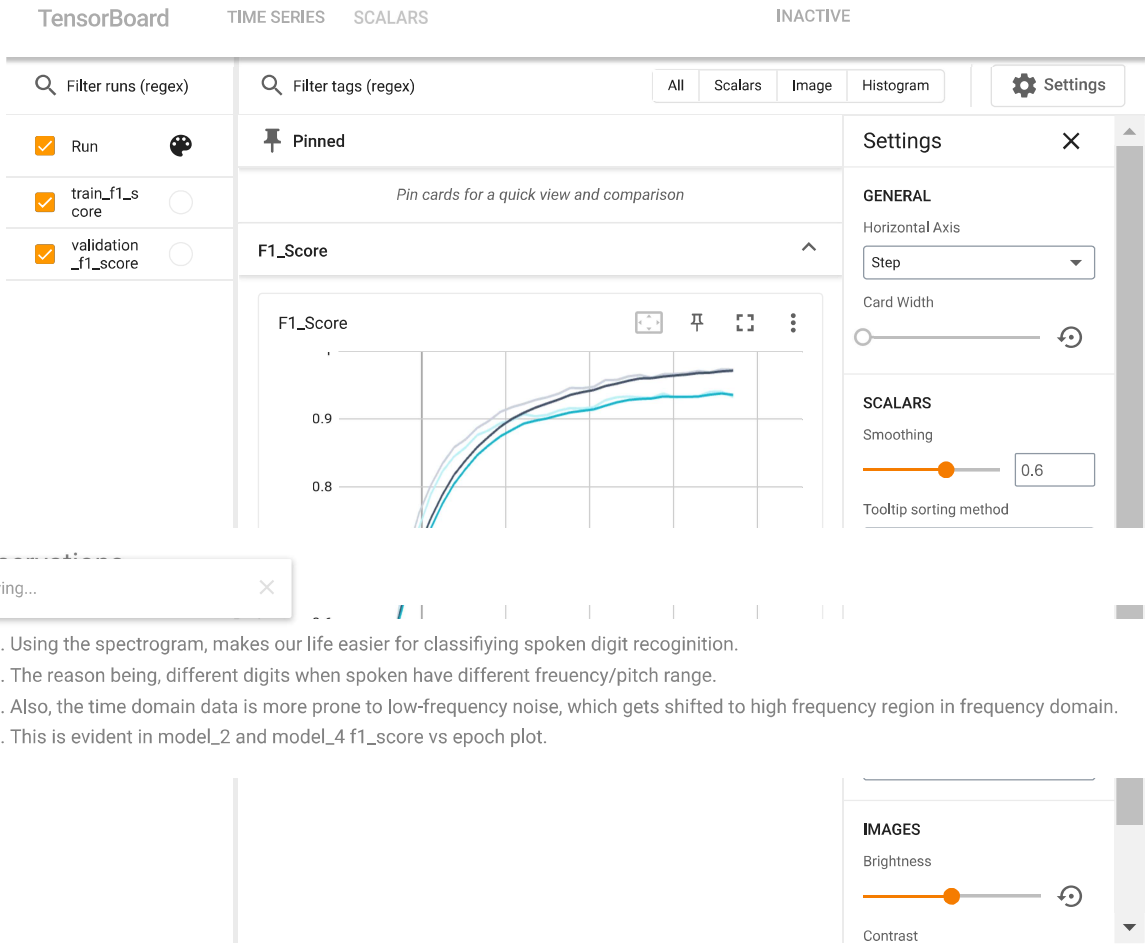
```
 dense_6 (Dense)             (None, 50)                5050

 dense_7 (Dense)             (None, 10)                510

=================================================================
Total params: 346,908
Trainable params: 346,908
Non-trainable params: 0
_________________________________________________________________
```

```
!rm -rf ./logs/
logdir = os.path.join('logs', datetime.datetime.now().strftime('%Y%m%d-%H%M%S'))
tensorboard = TensorBoard(log_dir = './logs')
checkpoint = ModelCheckpoint('best_model_4.h',save_best_only=True,mode='auto')
f1_score_cb = F1ScoreCB((X_train_spectrogram_augmented, y_train_augmented), (X_test_spectrogram_augmented, y_test_augmented))
```

```
history = model_4.fit(X_train_spectrogram_augmented, y_train_augmented,
            validation_data = (X_test_spectrogram_augmented, y_test_augmented),
            batch_size = 16 , epochs = 30,verbose=1,callbacks=[f1_score_cb,checkpoint,tensorboard])
```

```
Epoch 1/30
  1/900 [..............................] - ETA: 2:43:06 - loss: 2.3000 - accuracy: 0.0625WARNING:tensorflow:Callback method `on_train_batch_end` is slow compare
450/450 [==============================] - 3s 4ms/step
113/113 [==============================] - 1s 3ms/step
                                                 ore: 0.5844
                                                 ctions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
                                                 ======] - 29s 20ms/step - loss: 1.8313 - accuracy: 0.3875 - val_loss: 1.3196 - val_accuracy: 0.5844
Epoch 2/30
450/450 [==============================] - 1s 3ms/step
113/113 [==============================] - 0s 4ms/step
 - f1_score: 0.7015 - val_f1_score: 0.6883
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
900/900 [==============================] - 17s 19ms/step - loss: 1.0905 - accuracy: 0.6500 - val_loss: 0.9327 - val_accuracy: 0.6883
Epoch 3/30
450/450 [==============================] - 2s 4ms/step
113/113 [==============================] - 1s 5ms/step
 - f1_score: 0.7609 - val_f1_score: 0.7394
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
900/900 [==============================] - 19s 21ms/step - loss: 0.8248 - accuracy: 0.7261 - val_loss: 0.7493 - val_accuracy: 0.7394
Epoch 4/30
450/450 [==============================] - 1s 3ms/step
113/113 [==============================] - 1s 5ms/step
 - f1_score: 0.8014 - val_f1_score: 0.7881
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
900/900 [==============================] - 16s 18ms/step - loss: 0.6644 - accuracy: 0.7781 - val_loss: 0.6108 - val_accuracy: 0.7881
Epoch 5/30
450/450 [==============================] - 2s 3ms/step
113/113 [==============================] - 0s 3ms/step
 - f1_score: 0.8333 - val_f1_score: 0.8219
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
900/900 [==============================] - 17s 19ms/step - loss: 0.5545 - accuracy: 0.8126 - val_loss: 0.5278 - val_accuracy: 0.8219
Epoch 6/30
450/450 [==============================] - 1s 3ms/step
113/113 [==============================] - 0s 3ms/step
 - f1_score: 0.8574 - val_f1_score: 0.8439
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
900/900 [==============================] - 17s 19ms/step - loss: 0.4769 - accuracy: 0.8424 - val_loss: 0.4616 - val_accuracy: 0.8439
Epoch 7/30
450/450 [==============================] - 1s 3ms/step
113/113 [==============================] - 0s 3ms/step
 - f1_score: 0.8693 - val_f1_score: 0.8575
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
900/900 [==============================] - 16s 18ms/step - loss: 0.4195 - accuracy: 0.8613 - val_loss: 0.4120 - val_accuracy: 0.8575
Epoch 8/30
450/450 [==============================] - 1s 3ms/step
113/113 [==============================] - 0s 4ms/step
 - f1_score: 0.8862 - val_f1_score: 0.8756
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
900/900 [==============================] - 16s 18ms/step - loss: 0.3748 - accuracy: 0.8767 - val_loss: 0.3696 - val_accuracy: 0.8756
Epoch 9/30
450/450 [==============================] - 2s 5ms/step
113/113 [==============================] - 1s 4ms/step
 - f1_score: 0.8965 - val_f1_score: 0.8828
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses while saving (showing 2 of 2). The
900/900 [==============================] - 16s 17ms/step - loss: 0.3367 - accuracy: 0.8876 - val_loss: 0.3492 - val_accuracy: 0.8828
Epoch 10/30
450/450 [==============================] - 1s 3ms/step
```

```
%reload_ext tensorboard
%tensorboard --logdir $logdir
```



Saving...

1. Using the spectrogram, makes our life easier for classifiying spoken digit recoginition.
2. The reason being, different digits when spoken have different freuency/pitch range.
3. Also, the time domain data is more prone to low-frequency noise, which gets shifted to high frequency region in frequency domain.
4. This is evident in model_2 and model_4 f1_score vs epoch plot.

✓ 4s completed at 8:23 AM