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, i

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

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

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

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

Every Grader function has to return True.

1. Use direct audio files to train the LSTM network
2. Create spectogram (visual representation of audio files) and use that to train the LSTM network
3. Perform augumentation on audio files and the size of the files will be much larger than original audio files and train the LSTM network
4. Convert above augumented audio files to spectogram and train the LSTM network

```
import numpy as np
import pandas as pd
```

```
import librosa
import os
from sklearn.model_selection import train_test_split
##if you need any imports you can do that here.
```

We shared recordings.zip, please unzip those.

```
#read the all file names in the recordings folder given by us
#(if you get entire path, it is very useful in future)
!gdown --id 1ZQFXDEQ4RomprWQLAGJUwg0jne_0b6Jl
!pip install patool
import patoolib
patoolib.extract_archive("recordings.zip")
```

```
#https://thispointer.com/python-how-to-get-list-of-files-in-directory-and-sub-directories/
def getListOfFiles(dirName):
    # create a list of file and sub directories
    # names in the given directory
    listOfFile = os.listdir(dirName)
    allFiles = list()
    # Iterate over all the entries
    for entry in listOfFile:
        # Create full path
        fullPath = os.path.join(dirName, entry)
        # If entry is a directory then get the list of files in this directory
        if os.path.isdir(fullPath):
            allFiles = allFiles + getListOfFiles(fullPath)
        else:
            allFiles.append(fullPath)
    return allFiles
```

```
#save those files names as list in "all_files"
all_files = getListOfFiles('/content/recordings')
```

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

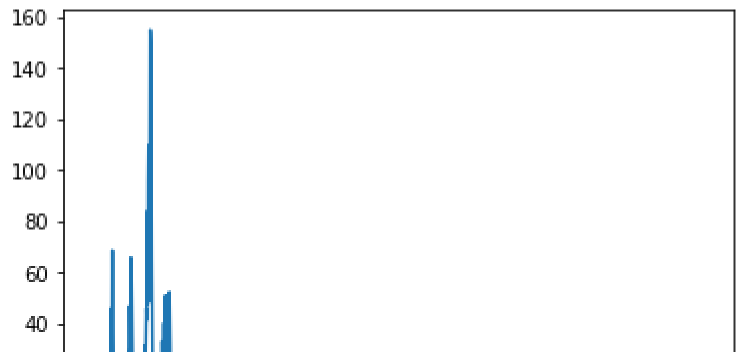
Exploring the sound dataset

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

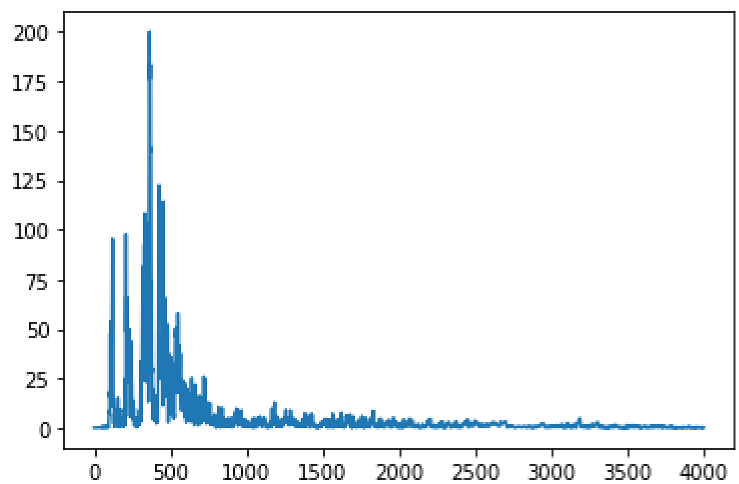
```
!git clone https://github.com/AllenDowney/ThinkDSP.git
```

```
fatal: destination path 'ThinkDSP' already exists and is not an empty directory.
```

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

def visualize_sound(file):
  # Read in audio file
  wave = thinkdsp.read_wave(file)

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

  # Play audio file
  wave.play()



visualize_sound('/content/recordings/0_yweweler_22.wav')
IPython.display.Audio('sound.wav')
```




```
visualize_sound('/content/recordings/0_jackson_14.wav')
IPython.display.Audio('sound.wav')
```

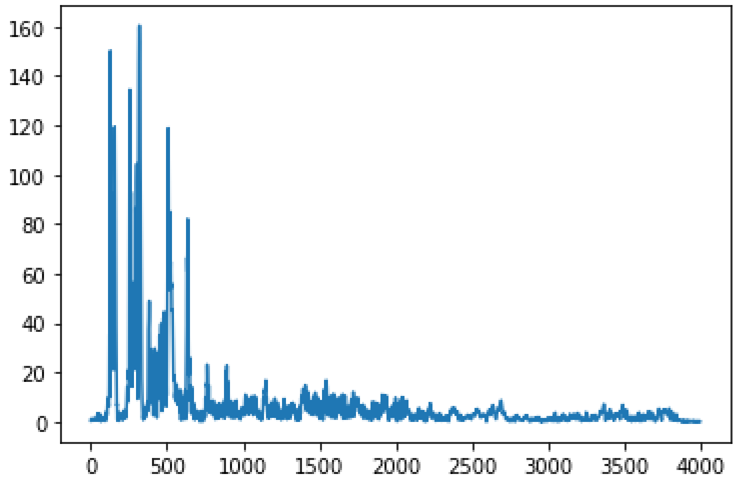



```
Writing sound.wav
```

0:00 / 0:00

```
visualize_sound('/content/recordings/0_nicolas_46.wav')
IPython.display.Audio('sound.wav')
```


```
visualize_sound('/content/recordings/0_theo_38.wav')
IPython.display.Audio('sound.wav')
```




```
Writing sound.wav
```

0:00 / 0:00

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
label = [i.split('/')[3].split('_')[0] for i in all_files]
df_audio = pd.DataFrame({'path':all_files,'label':label})
```

```
#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
```

```
from sklearn.utils import shuffle
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_train, X_test, y_train, y_test = train_test_split(df_audio['path'],df_audio['label'],test_s
```

Grader function 3

```
def grader_split():
    flag_len = (len(X_train)==1400) and (len(X_test)==600) and (len(y_train)==1400) and (len(
    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
```

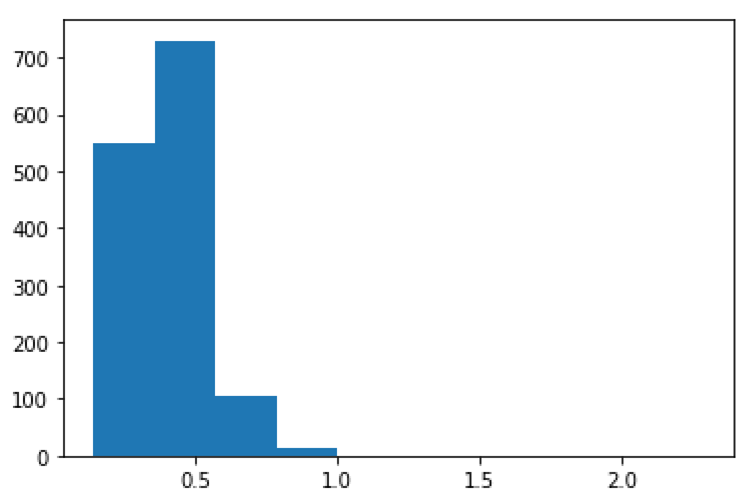
Preprocessing

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

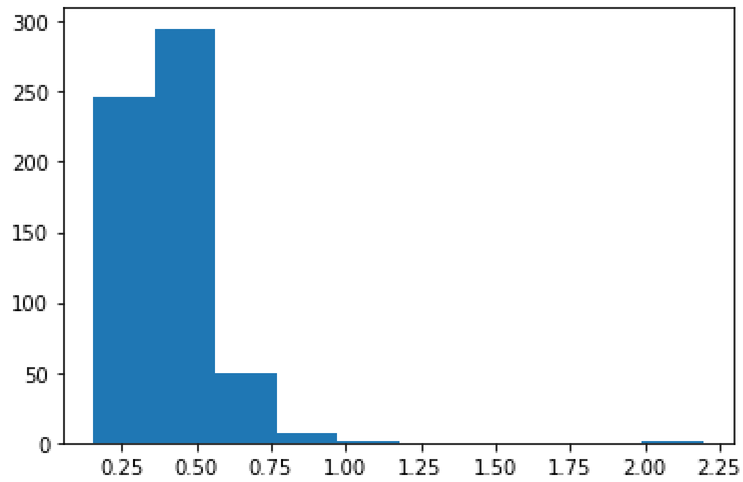
```
sample_rate = 22050
def load_wav(x, get_duration=True):
    '''This return the array values of audio with sampling rate of 22050 and Duration'''
    #loading the wav file with sampling rate of 22050
    samples, sample_rate = librosa.load(x, sr=22050)
    if get_duration:
        duration = librosa.get_duration(samples,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

raw_data_tr = []
duration_tr = []
raw_data_te = []
duration_te = []
for i in X_train:
  x,y = load_wav(i)
  raw_data_tr.append(x)
  duration_tr.append(y)
X_train_processed = pd.DataFrame({'raw_data':raw_data_tr,'duration':duration_tr})
for i in X_test:
  x,y = load_wav(i)
  raw_data_te.append(x)
  duration_te.append(y)
X_test_processed = pd.DataFrame({'raw_data':raw_data_te,'duration':duration_te})
```

```
#plot the histogram of the duration for trian
import matplotlib.pyplot as plt
plt.hist(X_train_processed.duration)
plt.show()
```



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




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

```
0th percentile is 0.1435374149659864
10th percentile is 0.2606938775510204
20th percentile is 0.30187755102040814
30th percentile is 0.3324489795918368
40th percentile is 0.35935600907029475
50th percentile is 0.3905215419501134
60th percentile is 0.41756009070294786
70th percentile is 0.44897052154195016
80th percentile is 0.48478911564625854
90th percentile is 0.5549160997732426
```

```
##print 90 to 100 percentile values with step size of 1.
for i in range(0,100,10):
  print(f"{i}th percentile is {np.percentile(X_test_processed['duration'].values,i)}")
```

```
0th percentile is 0.15741496598639457
10th percentile is 0.2595147392290249
20th percentile is 0.29777777777777775
30th percentile is 0.3309115646258503
40th percentile is 0.3582222222222222
50th percentile is 0.38834467120181404
60th percentile is 0.41619047619047617
70th percentile is 0.44286167800453513
80th percentile is 0.4827573696145125
90th percentile is 0.5569433106575965
```

Grader function 4

```
def grader_processed():
```

```
def grader_processed():
    flag_columns = (all(X_train_processed.columns==['raw_data', 'duration'])) and (all(X_test
    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
```

```
## as discussed above, Pad with Zero if length of sequence is less than 17640 else Truncate t
## 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 ma
from tensorflow.keras.preprocessing.sequence import pad_sequences
X_train_pad_seq = pad_sequences(X_train_processed['raw_data'],maxlen=max_length,dtype='float3
X_test_pad_seq = pad_sequences(X_test_processed['raw_data'],maxlen=max_length,dtype='float32'
X_train_mask = np.array([i==0 for i in X_train_pad_seq])
X_test_mask = np.array([i==0 for i in X_test_pad_seq])
```

Grader function 5

```
def grader_padoutput():
    flag_padshape = (X_train_pad_seq.shape==(1400, 17640)) and (X_test_pad_seq.shape==(600, 1
    flag_maskshape = (X_train_mask.shape==(1400, 17640)) and (X_test_mask.shape==(600, 17640)
    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

```
y_train = np.array([int(i) for i in y_train])
y_test = np.array([int(i) for i in y_test])
```

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

```
#https://imgur.com/8YULUcu
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={}):
        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)


## 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

input = Input(shape=(X_train_pad_seq.shape[1],1),name='Input Layer')
mask = Input(shape=(X_train_mask.shape[1]),dtype='bool',name='Mask')
lstm = LSTM(30)(input, mask = mask)
dense =  Dense(32)(lstm)
output = Dense(10,activation='softmax')(dense)


model = Model(inputs=[input, mask],outputs=output)
model.summary()
```

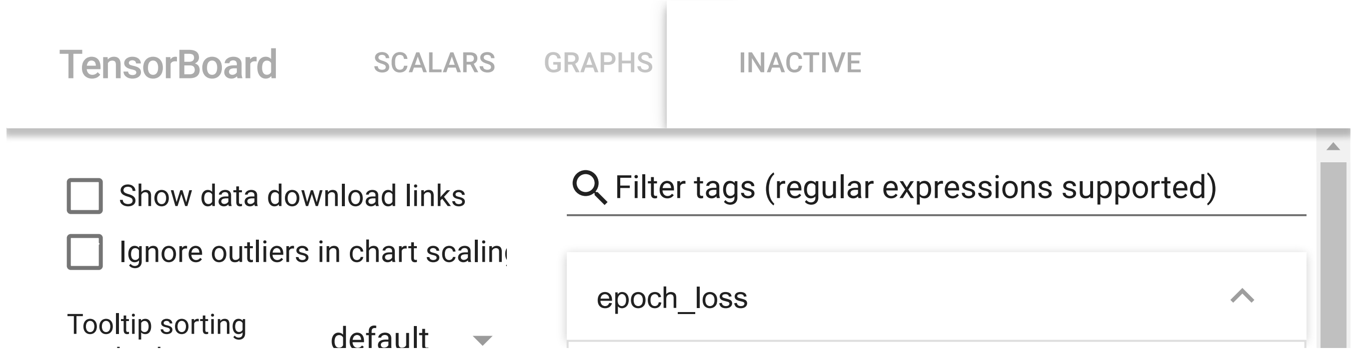
```
Model: "model"
_____________________________________________________________________________________________
 Layer (type)                  Output Shape         Param #     Connected to
=============================================================================================
 Input Layer (InputLayer)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

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

 lstm (LSTM)                   (None, 30)           3840        ['Input Layer[0][0]',
                                                                 'Mask[0][0]']

 dense (Dense)                 (None, 32)           992         ['lstm[0][0]']

 dense_1 (Dense)               (None, 10)           330         ['dense[0][0]']

=============================================================================================
Total params: 5,162
Trainable params: 5,162
Non-trainable params: 0
_____________________________________________________________________________________________
```

```
from tensorflow.keras.callbacks import TensorBoard

tensorboard_callback = TensorBoard(log_dir='logs1',write_graph=True,histogram_freq=1)
metric = Metrics(([X_test_pad_seq, X_test_mask], y_test))
```

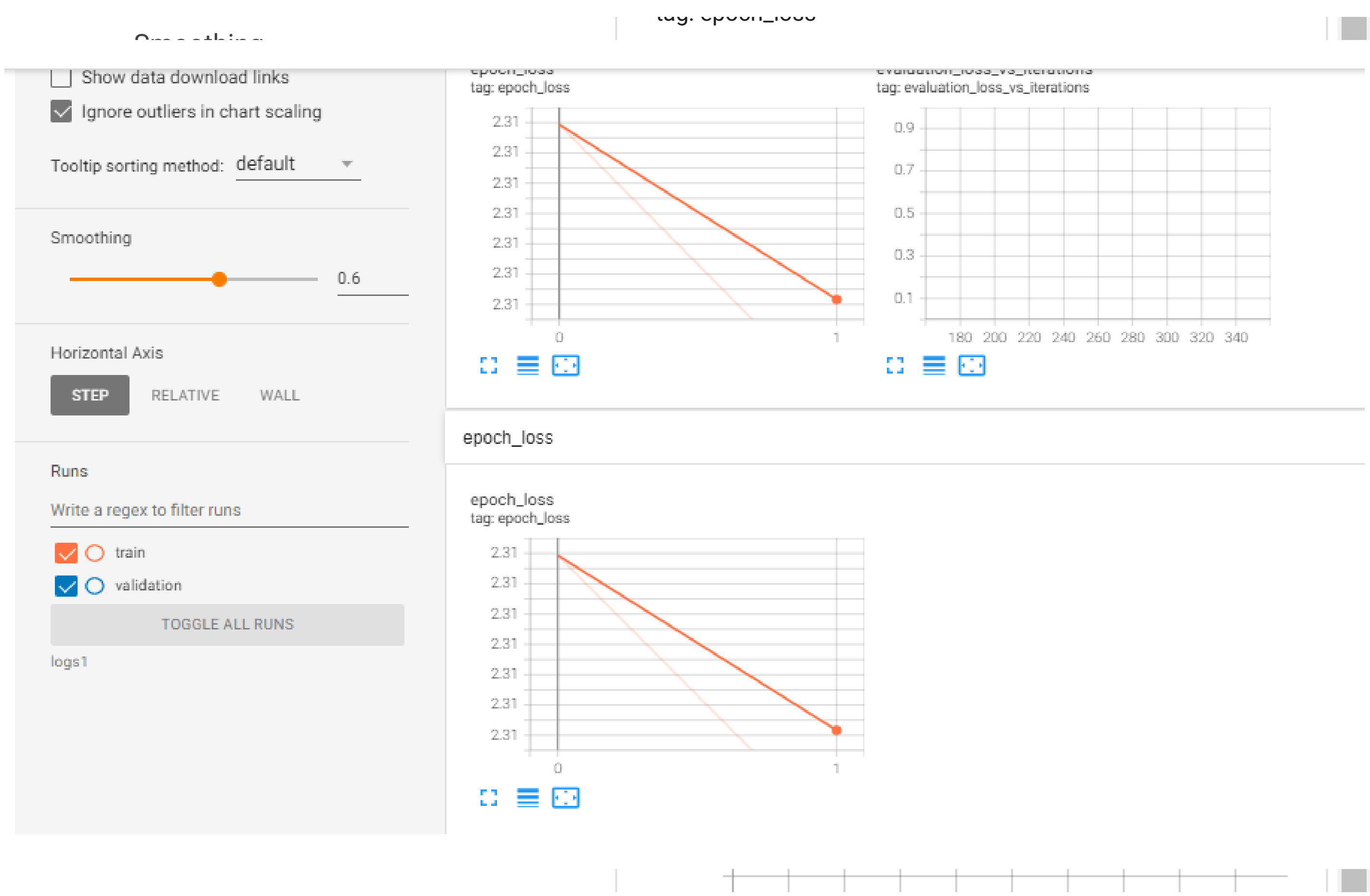
```
model.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(learn


#train your model
model.fit([X_train_pad_seq, X_train_mask],y_train,validation_data=([X_test_pad_seq, X_test_ma
```

```
Epoch 1/2
175/175 [==============================] - ETA: 0s - loss: 2.3126 val_f1_score 0.1000000
175/175 [==============================] - 3882s 22s/step - loss: 2.3126 - val_loss: 2.3
Epoch 2/2
175/175 [==============================] - ETA: 0s - loss: 2.3107 val_f1_score 0.1000000
175/175 [==============================] - 3790s 22s/step - loss: 2.3107 - val_loss: 2.3
<keras.callbacks.History at 0x7f6b5009acd0>
```

```
%reload_ext tensorboard
%tensorboard --logdir logs1
```



TENSORBOARD SCREENSHOT



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
```

```
##use convert_to_spectrogram and convert every raw sequence in X_train_pad_seq and X_test_pad
## save those all in the X_train_spectrogram and X_test_spectrogram ( These two arrays must b
X_train_spectrogram = []
X_test_spectrogram = []
for i in X_train_pad_seq:
  X_train_spectrogram.append(convert_to_spectrogram(i))
for i in X_test_pad_seq:
  X_test_spectrogram.append(convert_to_spectrogram(i))
```

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

```
X_train_spectrogram[0]
```

```
array([[-32.064407, -29.277729, -30.074486, ..., -80.       , -80.       ,
        -80.       ],
       [-24.482014, -18.270372, -16.526098, ..., -80.       , -80.       ,
        -80.       ],
       [-19.260283, -14.424799, -15.12985 , ..., -80.       , -80.       ,
        -80.       ],
       ...,
       [-80.       , -80.       , -80.       , ..., -80.       , -80.       ,
        -80.       ],
       [-80.       , -80.       , -80.       , ..., -80.       , -80.       ,
        -80.       ],
       [-80.       , -80.       , -80.       , ..., -80.       , -80.       ,
        -80.       ]], dtype=float32)
```

Grader function 6

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

```
X_train_spectrogram[0].shape
```

```
(64, 35)
```

```
#https://stackoverflow.com/questions/50309488/how-to-get-the-average-of-a-time-series-lstm-ke
from tensorflow.keras.layers import GlobalAveragePooling1D
tf.keras.backend.clear_session()

input = Input(shape=(64,35),name='Input Layer')
lstm = LSTM(200,return_sequences=True)(input)
avg = GlobalAveragePooling1D()(lstm)
dense =  Dense(200, activation = 'relu')(avg)
output = Dense(10,activation='softmax')(dense)
model2 = Model(inputs=input ,outputs=output)
model2.summary()
```

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

 lstm (LSTM)                 (None, 64, 200)           188800

 global_average_pooling1d (G  (None, 200)              0
 lobalAveragePooling1D)

 dense (Dense)               (None, 200)               40200

 dense_1 (Dense)             (None, 10)                2010

=================================================================
Total params: 231,010
Trainable params: 231,010
Non-trainable params: 0
_________________________________________________________________
```

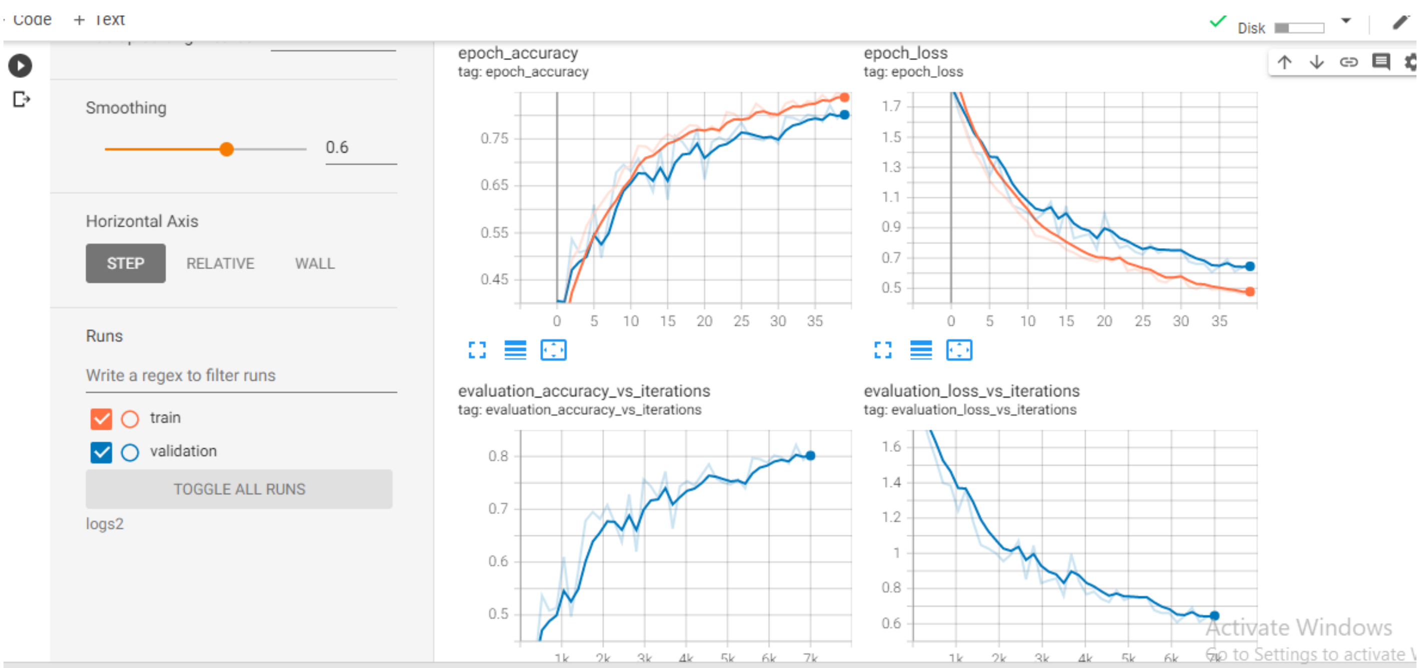
```
from tensorflow.keras.callbacks import TensorBoard

tensorboard_callback = TensorBoard(log_dir='logs2',write_graph=True,histogram_freq=1)
metric = Metrics((X_test_spectrogram, y_test))


model2.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(lear


model2.fit(X_train_spectrogram,y_train,validation_data=(X_test_spectrogram, y_test),batch_siz
```

```
Epoch 1/40
  1/175 [..............................] - ETA: 57:47 - loss: 2.3538 - accuracy: 0.00
170/175 [============================>.] - ETA: 0s - loss: 2.0437 - accuracy: 0.2676
175/175 [==============================] - 22s 12ms/step - loss: 2.0357 - accuracy: 0
Epoch 2/40
174/175 [============================>.] - ETA: 0s - loss: 1.7129 - accuracy: 0.3973
175/175 [==============================] - 1s 8ms/step - loss: 1.7104 - accuracy: 0.3
Epoch 3/40
175/175 [==============================] - ETA: 0s - loss: 1.5210 - accuracy: 0.4929
175/175 [==============================] - 1s 8ms/step - loss: 1.5210 - accuracy: 0.4
Epoch 4/40
175/175 [==============================] - ETA: 0s - loss: 1.4002 - accuracy: 0.5186
175/175 [==============================] - 1s 8ms/step - loss: 1.4002 - accuracy: 0.5
Epoch 5/40
172/175 [============================>.] - ETA: 0s - loss: 1.3168 - accuracy: 0.5676
175/175 [==============================] - 1s 8ms/step - loss: 1.3180 - accuracy: 0.5
Epoch 6/40
173/175 [============================>.] - ETA: 0s - loss: 1.2118 - accuracy: 0.5925
175/175 [==============================] - 1s 8ms/step - loss: 1.2135 - accuracy: 0.5
Epoch 7/40
168/175 [===========================>..] - ETA: 0s - loss: 1.1496 - accuracy: 0.6131
175/175 [==============================] - 1s 7ms/step - loss: 1.1503 - accuracy: 0.6
Epoch 8/40
172/175 [============================>.] - ETA: 0s - loss: 1.1062 - accuracy: 0.6359
175/175 [==============================] - 1s 8ms/step - loss: 1.1062 - accuracy: 0.6
Epoch 9/40
167/175 [===========================>..] - ETA: 0s - loss: 1.0569 - accuracy: 0.6437
175/175 [==============================] - 1s 7ms/step - loss: 1.0504 - accuracy: 0.6
Epoch 10/40
172/175 [============================>.] - ETA: 0s - loss: 0.9853 - accuracy: 0.6868
175/175 [==============================] - 1s 8ms/step - loss: 0.9855 - accuracy: 0.6
Epoch 11/40
173/175 [============================>.] - ETA: 0s - loss: 0.9396 - accuracy: 0.6908
175/175 [==============================] - 1s 8ms/step - loss: 0.9372 - accuracy: 0.6
Epoch 12/40
173/175 [============================>.] - ETA: 0s - loss: 0.8470 - accuracy: 0.7355
175/175 [==============================] - 1s 7ms/step - loss: 0.8482 - accuracy: 0.7
Epoch 13/40
175/175 [==============================] - ETA: 0s - loss: 0.8381 - accuracy: 0.7343
175/175 [==============================] - 1s 8ms/step - loss: 0.8381 - accuracy: 0.7
Epoch 14/40
168/175 [===========================>..] - ETA: 0s - loss: 0.8216 - accuracy: 0.7225
175/175 [==============================] - 1s 8ms/step - loss: 0.8165 - accuracy: 0.7
```

```
Epoch 15/40
168/175 [===========================>..] - ETA: 0s - loss: 0.7977 - accuracy: 0.7507
175/175 [==============================] - 1s 7ms/step - loss: 0.8001 - accuracy: 0.74
Epoch 16/40
171/175 [============================>.] - ETA: 0s - loss: 0.7630 - accuracy: 0.7566
175/175 [==============================] - 1s 8ms/step - loss: 0.7561 - accuracy: 0.7
Epoch 17/40
175/175 [==============================] - ETA: 0s - loss: 0.7356 - accuracy: 0.7529
175/175 [==============================] - 1s 8ms/step - loss: 0.7356 - accuracy: 0.7
Epoch 18/40
173/175 [============================>.] - ETA: 0s - loss: 0.7060 - accuracy: 0.7666
175/175 [==============================] - 1s 8ms/step - loss: 0.7073 - accuracy: 0.7
Epoch 19/40
175/175 [==============================] - ETA: 0s - loss: 0.6865 - accuracy: 0.7793
```

```
%reload_ext tensorboard
%tensorboard --logdir logs2
```

Tensorboard screenshot



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]:
```

```
        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_ste
            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
```

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_s
```

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.

```
train_aug = []
for i in X_train:
  train_aug.extend(generate_augmented_data(i))
test = []
for i in X_test:
  x,y = load_wav(i)
```

```
  test.append(x)
train_aug = np.array(train_aug)
test = np.array(test)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: VisibleDeprecationWarnin

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: VisibleDeprecationWarnin
  if __name__ == '__main__':
```

```
y_aug = []
for i in y_train:
  ele = [i] * 9
  y_aug.extend(ele)


X_train_pad_seq = pad_sequences(train_aug,maxlen=max_length,dtype='float32', padding='post',t
X_test_pad_seq = pad_sequences(test,maxlen=max_length,dtype='float32', padding='post',truncat
X_train_mask = np.array([i==0 for i in X_train_pad_seq])
X_test_mask = np.array([i==0 for i in X_test_pad_seq])


y_train = np.array([int(i) for i in y_aug])
y_test = np.array([int(i) for i in y_test])


X_train_pad_seq = np.array(X_train_pad_seq)
X_test_pad_seq = np.array(X_test_pad_seq)


input = Input(shape=(X_train_pad_seq.shape[1],1),name='Input Layer')
mask = Input(shape=(X_train_mask.shape[1]),dtype='bool',name='Mask')
lstm = LSTM(25)(input, mask = mask)
dense =  Dense(32)(lstm)
output = Dense(10,activation='softmax')(dense)


model3 = Model(inputs=[input, mask],outputs=output)
model3.summary()
```

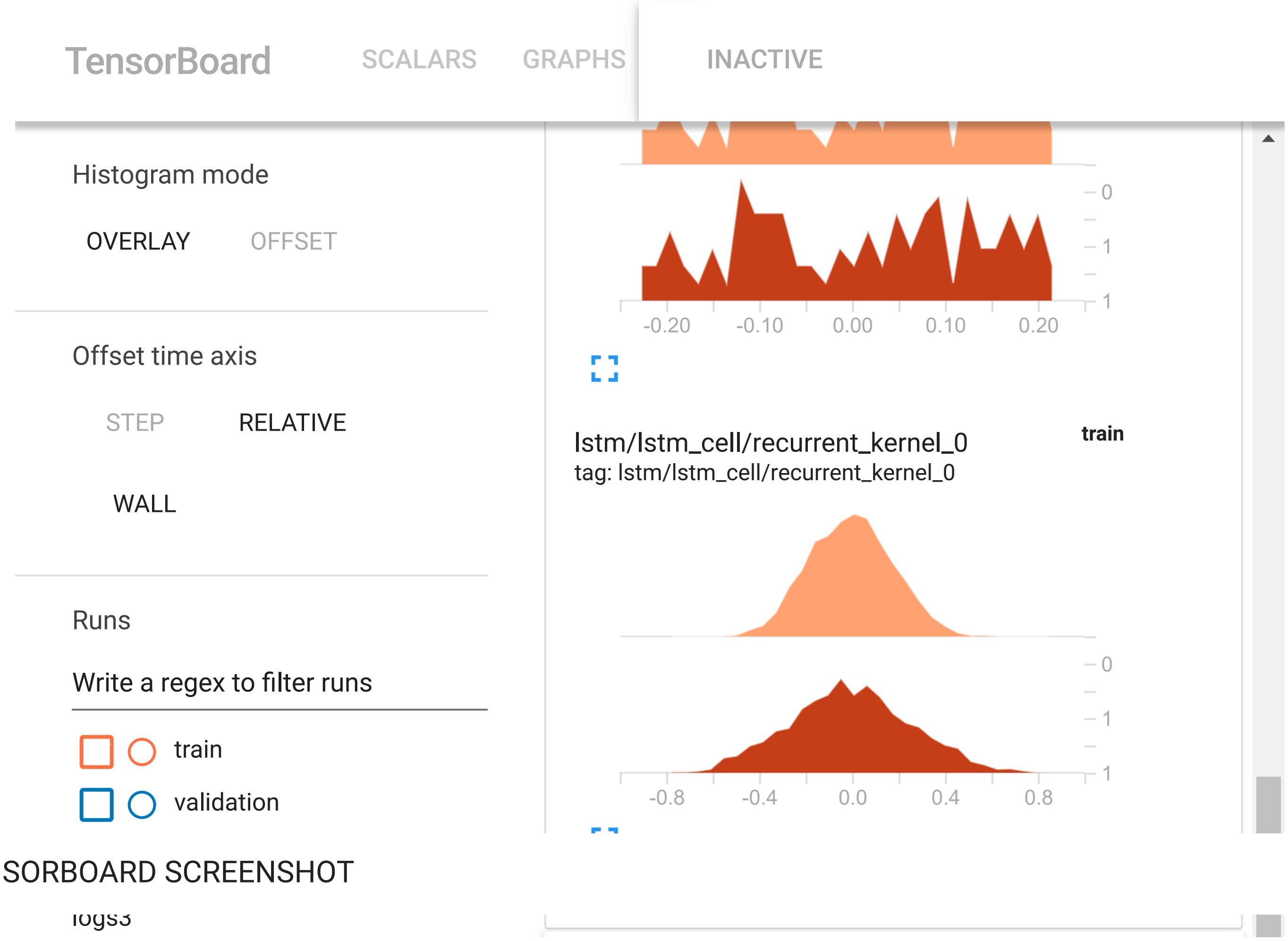
```
Model: "model"
_____________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=============================================================================================
 Input Layer (InputLayer)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

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

 lstm (LSTM)                    (None, 25)           2700        ['Input Layer[0][0]',
                                                                  'Mask[0][0]']

 dense (Dense)                  (None, 32)           832         ['lstm[0][0]']

 dense_1 (Dense)                (None, 10)           330         ['dense[0][0]']
```

```
==========================================================================================
Total params: 3,862
Trainable params: 3,862
Non-trainable params: 0
__________________________________________________________________________________________
```

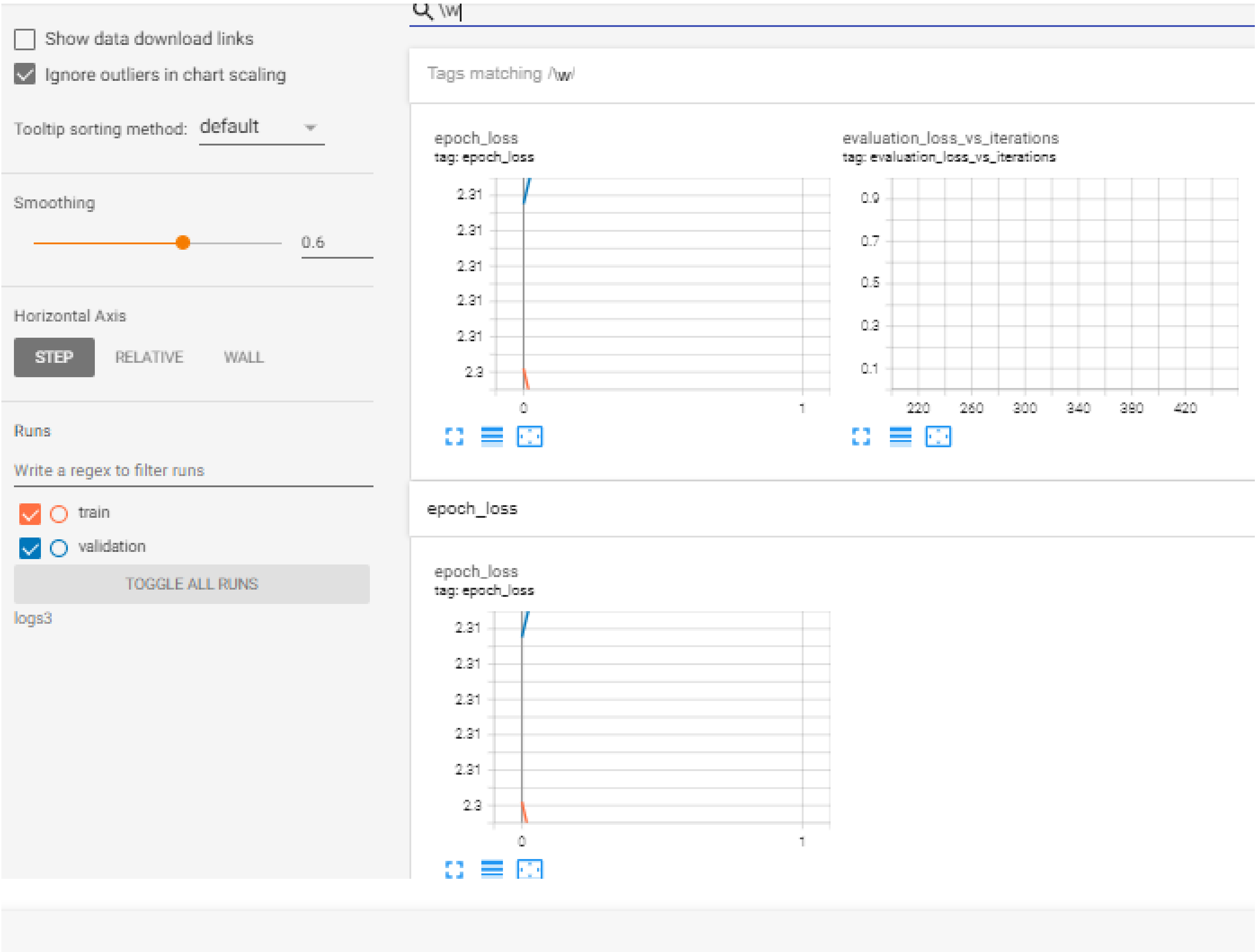
```
from tensorflow.keras.callbacks import TensorBoard

tensorboard_callback = TensorBoard(log_dir='logs3',write_graph=True,histogram_freq=1)
metric = Metrics(([X_test_pad_seq, X_test_mask], y_test))


model3.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(lear
model3.fit([X_train_pad_seq, X_train_mask],y_train,validation_data=([X_test_pad_seq, X_test_m
           batch_size=64,
           epochs=2, steps_per_epoch=X_train_pad_seq.shape[0]//64,
           callbacks=[tensorboard_callback,metric])
```

```
Epoch 1/2
225/225 [==============================] - ETA: 0s - loss: 2.3048   val_f1_score 0.1025
225/225 [==============================] - 5599s 25s/step - loss: 2.3048 - val_loss: 2.3
Epoch 2/2
225/225 [==============================] - ETA: 0s - loss: 2.2941   val_f1_score 0.1025
225/225 [==============================] - 5440s 24s/step - loss: 2.2941 - val_loss: 2.3
<keras.callbacks.History at 0x7fa50b090510>
```

```
%reload_ext tensorboard
%tensorboard --logdir logs3
```



TENSORBOARD SCREENSHOT



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 = []
X_test_spectrogram = []
for i in X_train_pad_seq:
  X_train_spectrogram.append(convert_to_spectrogram(i))
for i in X_test_pad_seq:
  X_test_spectrogram.append(convert_to_spectrogram(i))
X_train_spectrogram = np.array(X_train_spectrogram)
X_test_spectrogram = np.array(X_test_spectrogram)
```

```
#https://stackoverflow.com/questions/50309488/how-to-get-the-average-of-a-time-series-lstm-ke
from tensorflow.keras.layers import GlobalAveragePooling1D
tf.keras.backend.clear_session()

input = Input(shape=(64,35),name='Input Layer')
lstm = LSTM(200,return_sequences=True)(input)
avg = GlobalAveragePooling1D()(lstm)
dense =  Dense(200, activation = 'relu')(avg)
output = Dense(10,activation='softmax')(dense)
model4 = Model(inputs=input ,outputs=output)
model4.summary()
```

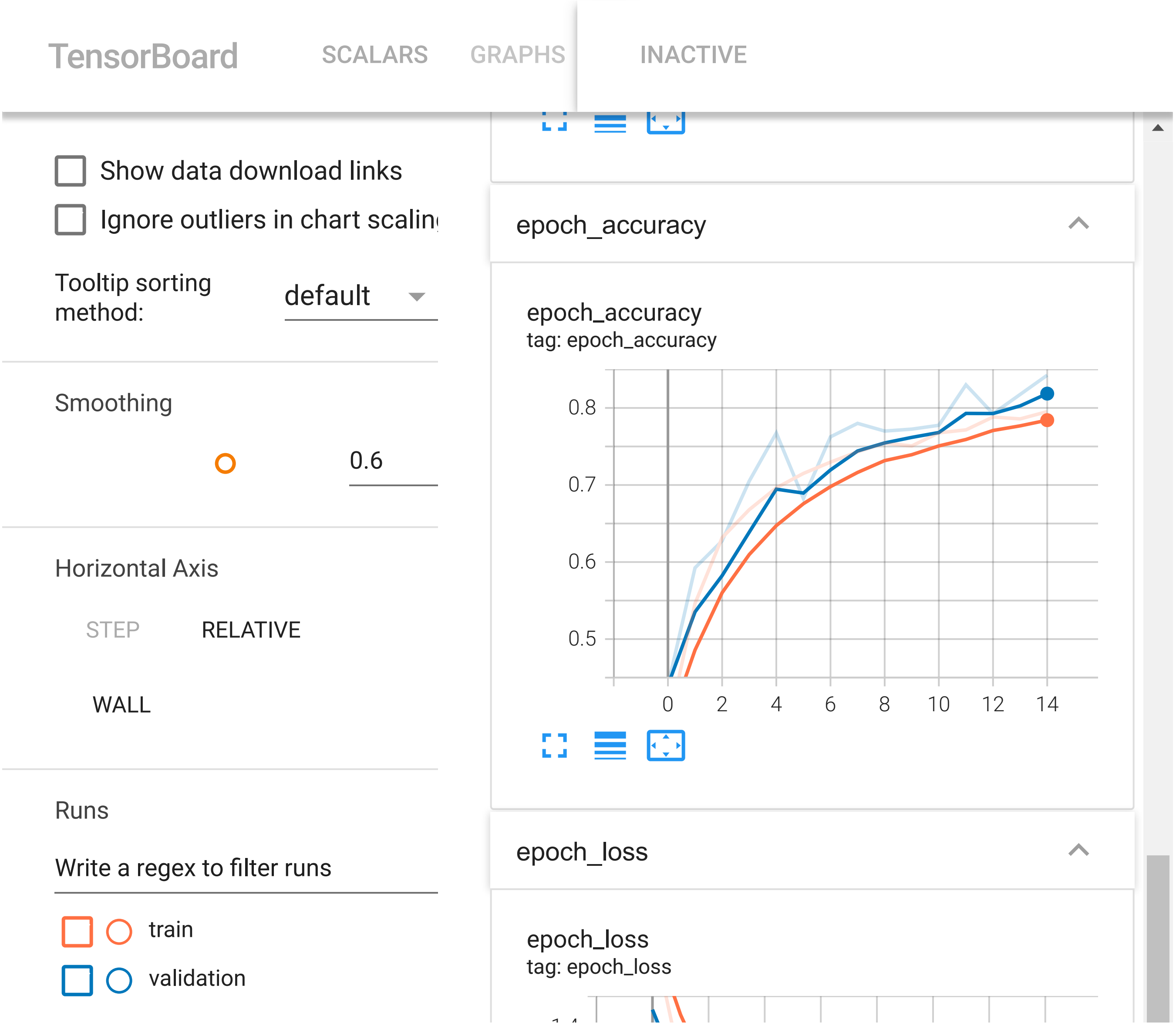
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input Layer (InputLayer)    [(None, 64, 35)]          0

 lstm (LSTM)                 (None, 64, 200)           188800

 global_average_pooling1d (G  (None, 200)              0
 lobalAveragePooling1D)

 dense (Dense)               (None, 200)               40200

 dense_1 (Dense)             (None, 10)                2010

=================================================================
Total params: 231,010
Trainable params: 231,010
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.callbacks import TensorBoard

tensorboard_callback = TensorBoard(log_dir='logs4',write_graph=True,histogram_freq=1)
metric = Metrics((X_test_spectrogram, y_test))


model4.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(lear
model4.fit(X_train_spectrogram,y_train,validation_data=(X_test_spectrogram, y_test),batch_siz
```

```
Epoch 1/15
   1/1800 [..............................] - ETA: 2:52:05 - loss: 2.4794 - accuracy: 0.0
1796/1800 [============================>.] - ETA: 0s - loss: 1.7045 - accuracy: 0.3848
1800/1800 [==============================] - 23s 10ms/step - loss: 1.7042 - accuracy: 0
Epoch 2/15
1793/1800 [============================>.] - ETA: 0s - loss: 1.2829 - accuracy: 0.5457
1800/1800 [==============================] - 12s 6ms/step - loss: 1.2821 - accuracy: 0.5
Epoch 3/15
1800/1800 [==============================] - ETA: 0s - loss: 1.0469 - accuracy: 0.6316
1800/1800 [==============================] - 10s 6ms/step - loss: 1.0469 - accuracy: 0.6
```

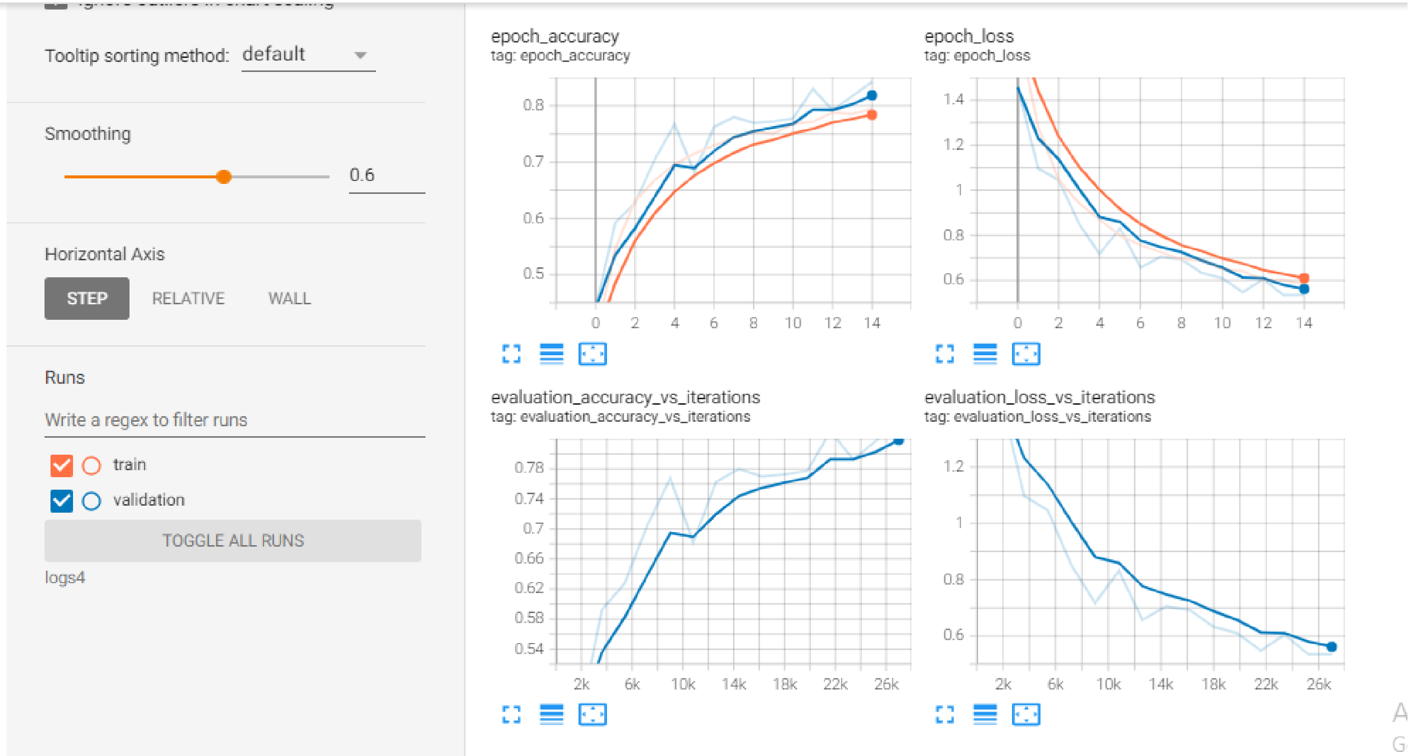
```
Epoch 4/15
1795/1800 [============================>.] - ETA: 0s - loss: 0.9430 - accuracy: 0.6680
1800/1800 [==============================] - 10s 6ms/step - loss: 0.9432 - accuracy: 0.6
Epoch 5/15
1793/1800 [============================>.] - ETA: 0s - loss: 0.8703 - accuracy: 0.6962
1800/1800 [==============================] - 10s 6ms/step - loss: 0.8702 - accuracy: 0.6
Epoch 6/15
1799/1800 [============================>.] - ETA: 0s - loss: 0.7988 - accuracy: 0.7148
1800/1800 [==============================] - 10s 6ms/step - loss: 0.7984 - accuracy: 0.7
Epoch 7/15
1793/1800 [============================>.] - ETA: 0s - loss: 0.7578 - accuracy: 0.7292
1800/1800 [==============================] - 10s 6ms/step - loss: 0.7566 - accuracy: 0.7
Epoch 8/15
1793/1800 [============================>.] - ETA: 0s - loss: 0.7250 - accuracy: 0.7432
1800/1800 [==============================] - 10s 6ms/step - loss: 0.7250 - accuracy: 0.7
Epoch 9/15
1794/1800 [============================>.] - ETA: 0s - loss: 0.6930 - accuracy: 0.7542
1800/1800 [==============================] - 10s 6ms/step - loss: 0.6925 - accuracy: 0.7
Epoch 10/15
1792/1800 [============================>.] - ETA: 0s - loss: 0.6900 - accuracy: 0.7508
1800/1800 [==============================] - 10s 6ms/step - loss: 0.6894 - accuracy: 0.7
Epoch 11/15
1798/1800 [============================>.] - ETA: 0s - loss: 0.6504 - accuracy: 0.7676
1800/1800 [==============================] - 10s 6ms/step - loss: 0.6502 - accuracy: 0.7
Epoch 12/15
1799/1800 [============================>.] - ETA: 0s - loss: 0.6401 - accuracy: 0.7715
1800/1800 [==============================] - 10s 6ms/step - loss: 0.6399 - accuracy: 0.7
Epoch 13/15
1800/1800 [==============================] - ETA: 0s - loss: 0.6016 - accuracy: 0.7883
1800/1800 [==============================] - 10s 6ms/step - loss: 0.6016 - accuracy: 0.7
Epoch 14/15
1794/1800 [============================>.] - ETA: 0s - loss: 0.6022 - accuracy: 0.7855
1800/1800 [==============================] - 10s 6ms/step - loss: 0.6017 - accuracy: 0.7
Epoch 15/15
1792/1800 [============================>.] - ETA: 0s - loss: 0.5840 - accuracy: 0.7951
1800/1800 [==============================] - 10s 6ms/step - loss: 0.5834 - accuracy: 0.7
<keras.callbacks.History at 0x7fa487570250>
```

```
%reload_ext tensorboard
%tensorboard --logdir logs4
```



Double-click (or enter) to edit

le + Text



1.The reason why accuracy and f1-score is almost the same because the formula to calculate the both is similar. In case multi-class classification where there is no true negative or false negative - it's just that a point is either classified correctly or not. Let a point being correctly classified be true positve and incorrectly classified as false positive, then

2. Accuracy = (tp+tn) / (tp+tn+fp+fn) = 2tp / (2tp + 2fp)
3. F1 score = 2tp / (2tp + fp + fn) = 2tp / (2tp + 2fp)

✓ 3s completed at 9:52 PM