Spoken Digit Recognition

In this notebook, You will do Spoken Digit Recognition.

Input - speech signal, output - digit number

It contains

1. Reading the dataset. and Preprocess the data set. Detailed instrctions are given below.
2. 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 man
2. Please read the instructions on the code cells and markdown cells. We will explain w
3. please return outputs in the same format what we asked. Eg. Don't return List of we
4. Please read the external links that we are given so that you will learn the concept
5. We are giving instructions at each section if necessary, please follow them.

Every Grader function has to return True.

```
!wget --header="Host: doc-0c-7s-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0

--2020-10-08 04:17:57--  https://doc-0c-7s-docs.googleusercontent.com/docs/securesc/rhul
Resolving doc-0c-7s-docs.googleusercontent.com (doc-0c-7s-docs.googleusercontent.com)..
Connecting to doc-0c-7s-docs.googleusercontent.com (doc-0c-7s-docs.googleusercontent.cor
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/x-zip-compressed]
Saving to: ‘recordings.zip’
```

```
recordings.zip          [  <=>               ]   8.85M  14.5MB/s    in 0.6s

2020-10-08 04:17:58 (14.5 MB/s) - ‘recordings.zip’ saved [9282934]
```

```
!unzip -qn "/content/drive/MyDrive/recordings.zip"
```

```
import numpy as np
import pandas as pd
import librosa
import os
##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)
#save those files names as list in "all_files"
```

```
import os
files=os.listdir('/content/recordings')
all_files=[]
for f in files:
  name=str(f)
  all_files.append(name)
```

```
len(all_files)
```

```
2000
```

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

```
label=[]
for i in range(2000):
  label.append(int(all_files[i].split('_')[0]))


#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



import pandas as pd
df_audio=pd.DataFrame()
df_audio['path']=all_files
df_audio['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   int64 
dtypes: int64(1), object(1)
memory usage: 31.4+ KB
```

```
#info
df_audio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   path    2000 non-null   object
 1   label   2000 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 31.4+ KB
```

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%
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_audio['path'],df_audio['label'], test_
```

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
```

```
import librosa
```

Double-click (or enter) to edit

```
#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
from tqdm import tqdm
a1=[]
a2=[]
b1=[]
b2=[]

for i in tqdm(range(X_train.shape[0])):
  a1.append(load_wav('/content/recordings/'+X_train.values[i])[0])
  a2.append(load_wav('/content/recordings/'+X_train.values[i])[1])

for i in tqdm(range(X_test.shape[0])):
  b1.append(load_wav('/content/recordings/'+X_test.values[i])[0])
  b2.append(load_wav('/content/recordings/'+X_test.values[i])[1])
```

```
100%|██████████| 1400/1400 [05:30<00:00,  4.24it/s]
100%|██████████| 600/600 [02:24<00:00,  4.16it/s]
```

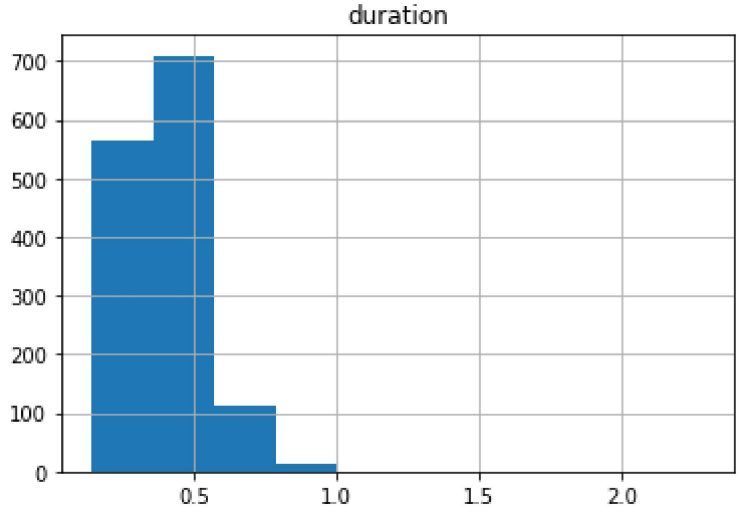
```
print(len(a1))
print(len(a2))
print(len(b1))
print(len(b2))
```

```
1400
1400
600
600
```

```
X_train_processed=pd.DataFrame()
X_test_processed=pd.DataFrame()
X_train_processed['raw_data']=a1
X_train_processed['duration']=a2
X_test_processed['raw_data']=b1
X_test_processed['duration']=b2
```

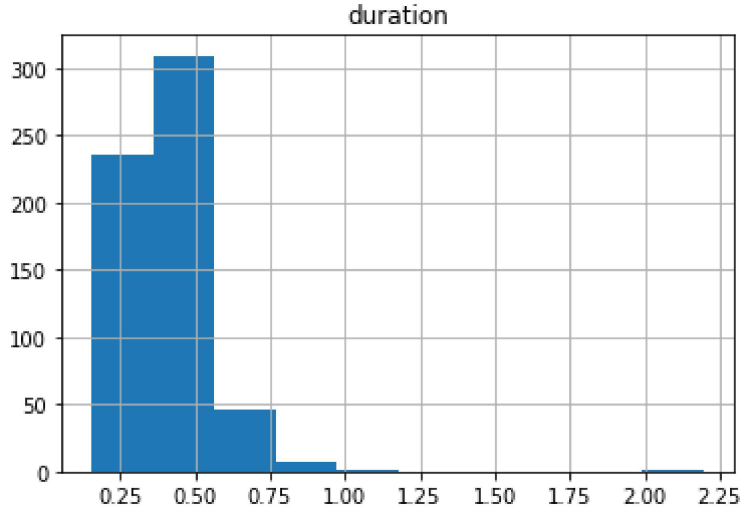
```
#plot the histogram of the duration for trian
X_train_processed.hist(column='duration')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcbbf6ec400>]],
      dtype=object)
```




```
#plot the histogram of the duration for trian
X_test_processed.hist(column='duration')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcbbf5b77b8>]],
      dtype=object)
```




```
#print 0 to 100 percentile values with step size of 10 for train data duration.
import numpy as np
for i in range(0,101,10):
```

```
if i%10==0:
  print(i,' th percentile is : ',np.percentile(X_train_processed['duration'], i))
```

```
0  th percentile is :  0.1435374149659864
10  th percentile is :  0.2571065759637188
20  th percentile is :  0.29795918367346935
30  th percentile is :  0.32884807256235826
40  th percentile is :  0.3564716553287982
50  th percentile is :  0.38752834467120184
60  th percentile is :  0.41583673469387755
70  th percentile is :  0.44297052154195016
80  th percentile is :  0.4833197278911565
90  th percentile is :  0.5565034013605445
100  th percentile is :  2.282766439909297
```

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

```
90  th percentile is :  0.5565034013605445
91  th percentile is :  0.5710530612244898
92  th percentile is :  0.5812916099773243
93  th percentile is :  0.5985732426303856
94  th percentile is :  0.6121541950113378
95  th percentile is :  0.6269614512471655
96  th percentile is :  0.6389206349206348
97  th percentile is :  0.6582807256235828
98  th percentile is :  0.689717006802721
99  th percentile is :  0.790681179138322
100  th percentile is :  2.282766439909297
```

Grader function 4

```
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 le

While loading the audio files, we are using sampling rate of 22050 so one sec will give arr

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 typ

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Input,Activation,BatchNormalization,Dropout,Embeddi
from tensorflow.keras.models import Model
import random as rn
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from tensorflow.keras import layers
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ModelCheckpoint ,TensorBoard,EarlyStopping,LearningRat
#from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from tensorflow.keras.layers import concatenate
```

```
max_seq_length=17640
X_train_pad_seq=[]     #np.zeros((1400,17640))
X_train_mask= []        #np.zeros((1400,17640))
for i in tqdm(range(X_train_processed.shape[0])):
  # pad if len less than max_seq_length
  if len(X_train_processed['raw_data'].values[i]) < max_seq_length:
        seq=X_train_processed['raw_data'].values[i].tolist()+ \
                                        [0 for item in range(max_seq_length-len(X_train_proc
  else:
        # Truncate if len greater then max_seq_length
        seq = X_train_processed['raw_data'].values[i].tolist()[:max_seq_length]
  #mask=[1 if x!=0 else 0 for x in tokens]
  mask=[1 if x!=0 else 0 for x in seq]
  mask=[bool(x) for x in mask]
  X_train_pad_seq.append(seq)
  X_train_mask.append(mask)
```

```
100%|██████████| 1400/1400 [00:05<00:00, 248.45it/s]
```

```
X_train_pad_seq=np.array(X_train_pad_seq)
X_train_mask=np.array(X_train_mask)
```

```
X_test_processed.shape
```

```
(600, 2)
```

```
max_seq_length=17640
X_test_pad_seq=[]      #np.zeros((600,17640))
X_test_mask=[]            #np.zeros((600,17640))
for i in tqdm(range(X_test_processed.shape[0])):
  # pad if len less than
  if len(X_test_processed['raw_data'].values[i]) < max_seq_length:
        # Add the ['PAD'] token
        seq=X_test_processed['raw_data'].values[i].tolist()+[0 for item in range(max_seq_leng
  else:
        # Truncate the tokens at maxLen - 1 and add a '[SEP]' tag.
        seq = X_test_processed['raw_data'].values[i].tolist()[:max_seq_length]
  #mask=[1 if x!=0 else 0 for x in tokens]
  mask=[1 if x!=0 else 0 for x in seq]
  mask=[bool(x) for x in mask]
  X_test_pad_seq.append(seq)
  X_test_mask.append(mask)
```

```
100%|██████████| 600/600 [00:02<00:00, 251.64it/s]
```

```
X_test_pad_seq=np.array(X_test_pad_seq)
X_test_mask=np.array(X_test_mask)
```

Double-click (or enter) to edit

```
X_test_mask[i]
```

```
array([ True,  True,  True, ..., False, False, False])
```

```
## 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
```

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.

```
`# This is formatted as code`
```

- bold textitalicized* text

```
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).
3. Use tensorboard to plot the graphs of loss and metric(use micro F1 score as metric) and histograms of gradients.
4. make sure that it won't overfit.
5. You are free to include any regularization

```
from tensorflow.keras.layers import Input, LSTM, Dense,GlobalAveragePooling1D,GlobalAveragePo
from tensorflow.keras.models import Model
import tensorflow as tf
```

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(y_train)
y_train_encoded = encoder.transform(y_train)
y_test_encoded = encoder.transform(y_test)
y_train_ohe = tf.keras.utils.to_categorical(y_train_encoded)
y_test_ohe = tf.keras.utils.to_categorical(y_test_encoded)
```

```
y_test_ohe[0]
```

```
     array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
## as discussed above, please write the LSTM
input1 = Input(shape=(17640, 1,))
input_mask=Input(shape=(17640,),dtype=bool)
LSTM_layer= LSTM(50,dropout=0.2)(inputs=input1,mask=input_mask)
x = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30),ke
x  =  Dropout(0.25)(x)
x   = BatchNormalization()(x)
x = Dense(12,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30),ke
x  =  Dropout(0.35)(x)
x   = BatchNormalization()(x)
x = Dense(8,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30),ker
x   = BatchNormalization()(x)
dense = Dense(10,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
Out = Dense(units=10,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_nor
model1= Model(inputs=(input1,input_mask),outputs=Out)
```

```
import tensorflow as tf
import keras.backend as K
import os
import datetime

#https://www.kaggle.com/c/liverpool-ion-switching/discussion/132646

def f1(y_true, y_pred):
    y_pred = K.round(y_pred)
    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    # tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.math.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)


from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
import tensorflow_addons as tfa


#https://stackoverflow.com/questions/43263111/defining-an-auc-metric-for-keras-to-support-eva
def auc( y_true, y_pred ) :
    score = tf.py_function( lambda y_true, y_pred : f1_score( y_true, y_pred, average='micro'
```

```
    return score
```

```
%load_ext tensorboard
```

```
#tensorbord for model1
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```

TensorBoard SCALARS GRAPHS DIS INACTIVE

```
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001)
model1.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=[tfa.metrics.F1Sc
```

Show data download links

```
model1.summary()
```

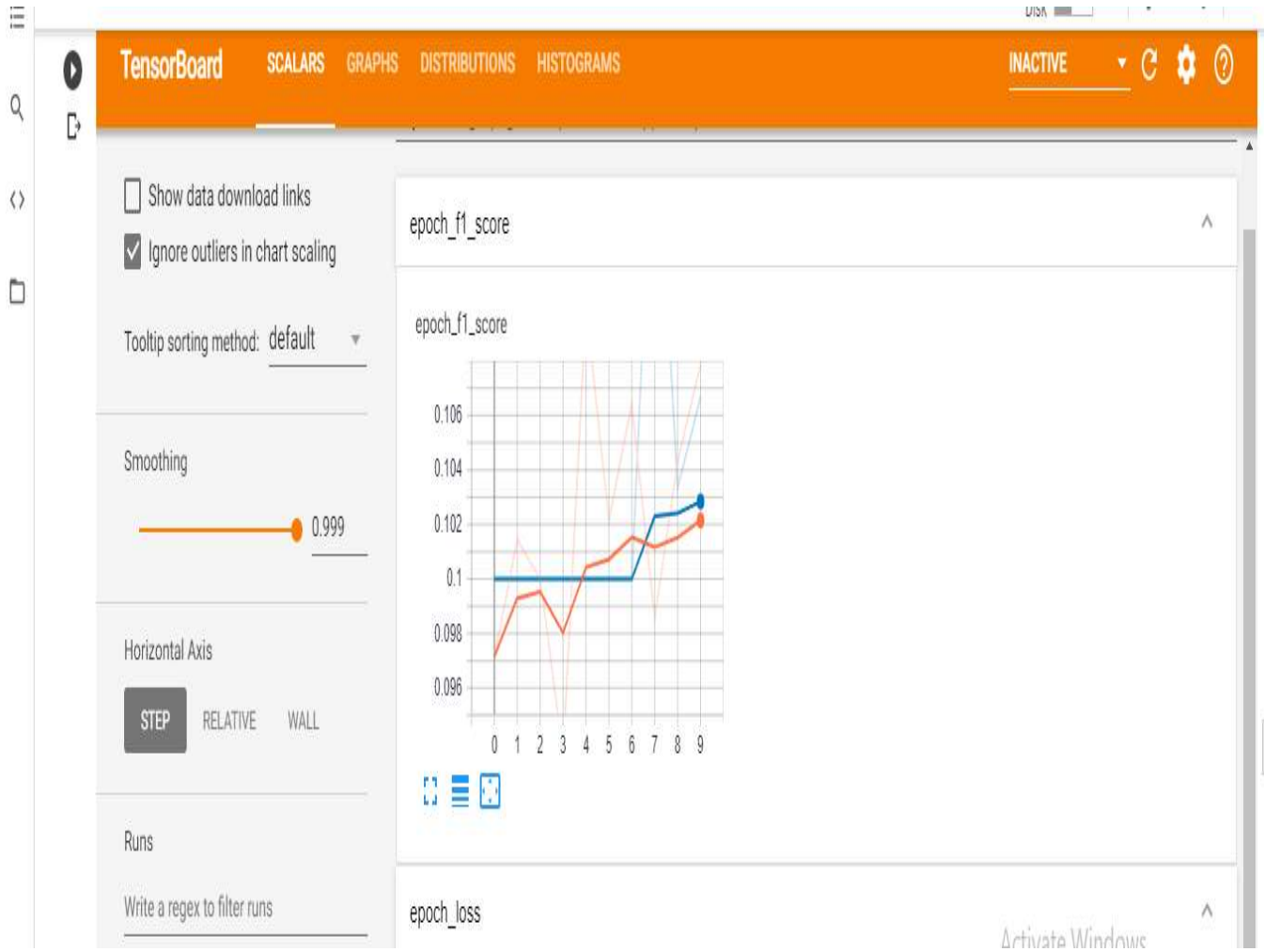
```
Model: "functional_1"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_1 (InputLayer)            [(None, 17640, 1)]   0
__________________________________________________________________________________________
input_2 (InputLayer)            [(None, 17640)]      0
__________________________________________________________________________________________
lstm (LSTM)                     (None, 50)           10400       input_1[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________
dense (Dense)                   (None, 16)           816         lstm[0][0]
__________________________________________________________________________________________
dropout (Dropout)               (None, 16)           0           dense[0][0]
__________________________________________________________________________________________
batch_normalization (BatchNorma (None, 16)           64          dropout[0][0]
__________________________________________________________________________________________
dense_1 (Dense)                 (None, 12)           204         batch_normalization[0]
__________________________________________________________________________________________
dropout_1 (Dropout)             (None, 12)           0           dense_1[0][0]
__________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 12)           48          dropout_1[0][0]
__________________________________________________________________________________________
dense_2 (Dense)                 (None, 8)            104         batch_normalization_1[0
__________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 8)            32          dense_2[0][0]
__________________________________________________________________________________________
dense_3 (Dense)                 (None, 10)           90          batch_normalization_2[0
__________________________________________________________________________________________
Output (Dense)                  (None, 10)           110         dense_3[0][0]
==========================================================================================
Total params: 11,868
Trainable params: 11,796
Non-trainable params: 72
__________________________________________________________________________________________
```

```
#train your model


model1.fit([X_train_pad_seq,X_train_mask],y_train_ohe,epochs=10, validation_data=([X_test_pad
          callbacks=[tensorboard_callback])
```

```
Epoch 1/10
 1/44 [..............................] - ETA: 0s - loss: 3.0455 - f1_score: 0.0625WARNIN
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
```

```
 2/44 [>.............................] - ETA: 1:22 - loss: 2.9234 - f1_score: 0.0625WARN
44/44 [==============================] - 48s 1s/step - loss: 2.6427 - f1_score: 0.0857 -
Epoch 2/10
44/44 [==============================] - 43s 978ms/step - loss: 2.6144 - f1_score: 0.089
Epoch 3/10
44/44 [==============================] - 42s 965ms/step - loss: 2.5719 - f1_score: 0.101
Epoch 4/10
44/44 [==============================] - 44s 992ms/step - loss: 2.5936 - f1_score: 0.085
Epoch 5/10
44/44 [==============================] - 44s 993ms/step - loss: 2.5798 - f1_score: 0.094
Epoch 6/10
44/44 [==============================] - 43s 976ms/step - loss: 2.5606 - f1_score: 0.098
Epoch 7/10
44/44 [==============================] - 43s 984ms/step - loss: 2.5468 - f1_score: 0.104
Epoch 8/10
44/44 [==============================] - 44s 993ms/step - loss: 2.5569 - f1_score: 0.115
Epoch 9/10
44/44 [==============================] - 43s 987ms/step - loss: 2.5617 - f1_score: 0.093
Epoch 10/10
44/44 [==============================] - 44s 995ms/step - loss: 2.5001 - f1_score: 0.111
<tensorflow.python.keras.callbacks.History at 0x7fe5c25c3780>
```

```
from IPython.display import Image
Image('/content/Screenshot (225).png',width=800,height=500)
```



```
from IPython.display import Image
Image('/content/Screenshot (226).png',width=800,height=500)
```

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 i intensity of a frequency at a point in time i.e we are converting Time domain to frequency

```
##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
```

```
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_spectrogram=[]
X_test_spectrogram=[]
for i in tqdm(range(X_train_processed.shape[0])):
  X_train_spectrogram.append(convert_to_spectrogram(X_train_pad_seq[i]))

for i in tqdm(range(X_test_processed.shape[0])):
  X_test_spectrogram.append(convert_to_spectrogram(X_test_pad_seq[i]))
```

```
100%|██████████| 1400/1400 [00:09<00:00, 149.83it/s]
100%|██████████| 600/600 [00:04<00:00, 132.12it/s]
```

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

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 outp
2. Average the output of every time step and give this to the Dense layer of any size.
3. give the above output to Dense layer of size 10(output layer) and train the network wit
4. Use tensorboard to plot the graphs of loss and metric(use micro F1 score as metric) and
5. make sure that it won't overfit.
6. You are free to include any regularization

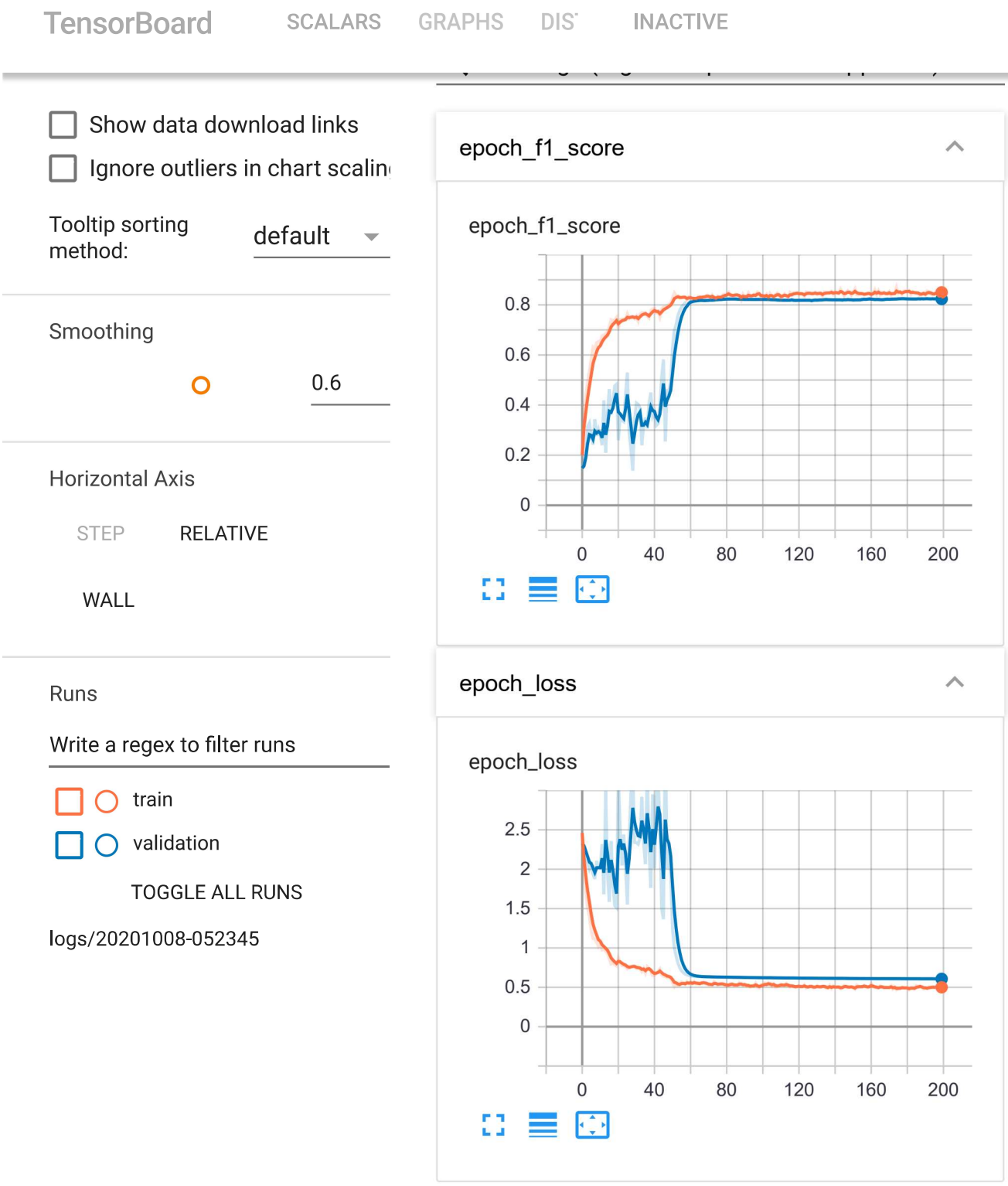
```
## as discussed above, please write the LSTM
input1 = Input(shape=(64, 35,))
```

```
LSTM_layer= LSTM(256,dropout=0.01,return_sequences=True)(input1)
#tr = tf.transpose(LSTM_layer,perm=[0, 2, 1])
avg=GlobalAveragePooling1D(data_format='channels_last')(LSTM_layer)
x = Dense(256,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=31),k
x  =  Dropout(0.01)(x)
x   = BatchNormalization()(x)
x = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=43),k
x  =  Dropout(0.01)(x)
x   = BatchNormalization()(x)
x = Dense(64,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=54),ke
x   = BatchNormalization()(x)
dense = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=40
Out = Dense(units=10,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_nor
model2= Model(inputs=input1,outputs=Out)


model2.summary()
```

```
Model: "functional_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 64, 35)]          0
_________________________________________________________________
lstm_10 (LSTM)               (None, 64, 256)           299008
_________________________________________________________________
global_average_pooling1d_10  (None, 256)               0
_________________________________________________________________
dense_40 (Dense)             (None, 256)               65792
_________________________________________________________________
dropout_20 (Dropout)         (None, 256)               0
_________________________________________________________________
batch_normalization_30 (Batc (None, 256)               1024
_________________________________________________________________
dense_41 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_21 (Dropout)         (None, 128)               0
_________________________________________________________________
batch_normalization_31 (Batc (None, 128)               512
_________________________________________________________________
dense_42 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_32 (Batc (None, 64)                256
_________________________________________________________________
dense_43 (Dense)             (None, 32)                2080
_________________________________________________________________
Output (Dense)               (None, 10)                330
=================================================================
Total params: 410,154
Trainable params: 409,258
Non-trainable params: 896
_________________________________________________________________
```

```
#tensorbord for model1
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
```

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```




```
def changeLearningRate(epochs,learning_rate):
  if epochs<50:
    learning_rate=0.001
    return learning_rate
```

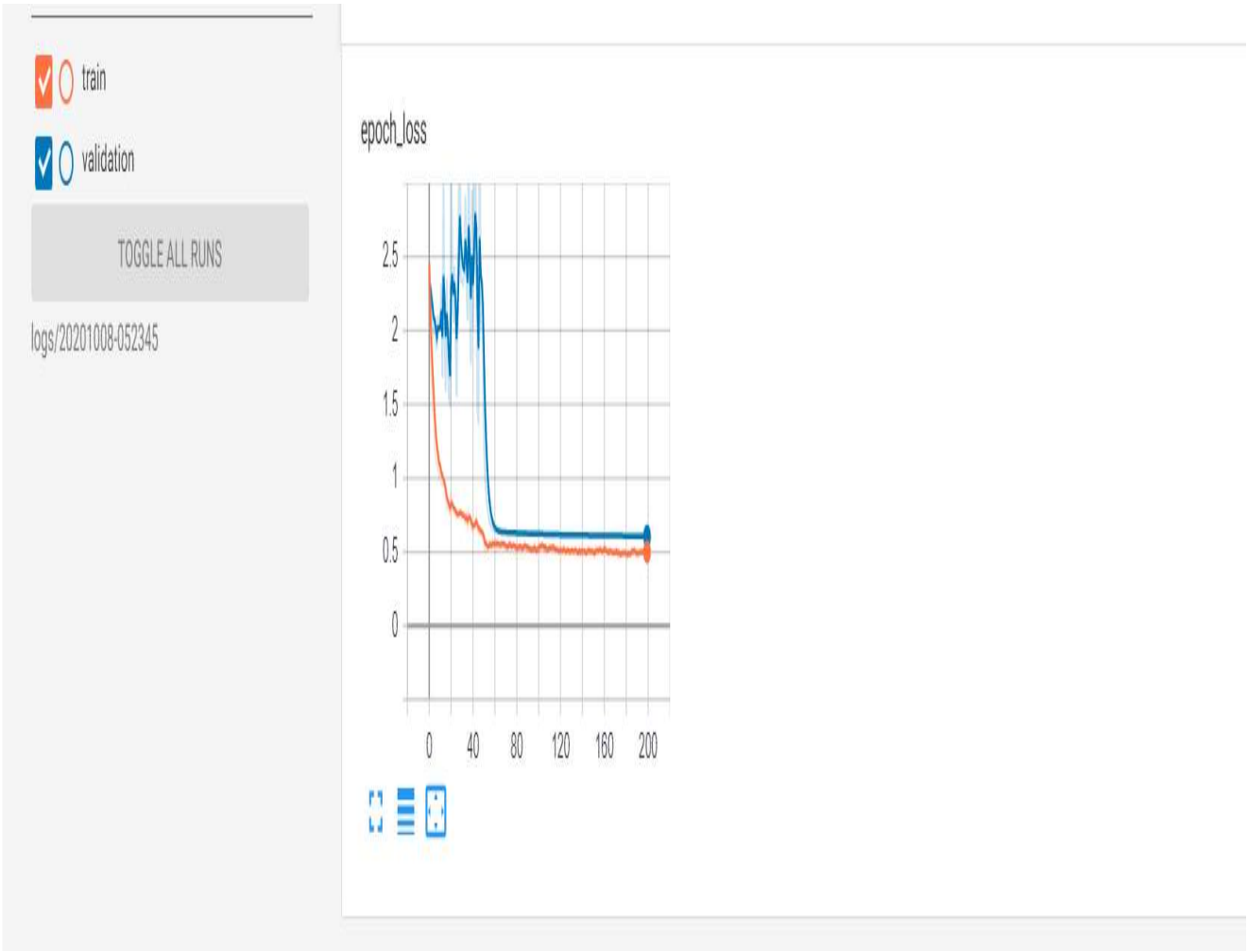
```
  elif epochs<75 and epochs>125:
    learning_rate=0.0001
    return learning_rate
  else:
    learning_rate=0.000001
    return learning_rate



lrschedule = LearningRateScheduler(changeLearningRate,verbose=1)


optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)

model2.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=[tfa.metrics.F1Sc


model2.fit(X_train_spectrogram,y_train_ohe,epochs=200, validation_data=(X_test_spectrogram,y_
          callbacks=[tensorboard_callback,lrschedule])
```

```
Epoch 00187: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 187/200
28/28 [==============================] - 0s 17ms/step - loss: 0.5092 - f1_score: 0.84

Epoch 00188: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 188/200
28/28 [==============================] - 0s 15ms/step - loss: 0.5211 - f1_score: 0.83

Epoch 00189: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 189/200
28/28 [==============================] - 0s 16ms/step - loss: 0.4956 - f1_score: 0.84

Epoch 00190: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 190/200
28/28 [==============================] - 0s 15ms/step - loss: 0.4803 - f1_score: 0.84

Epoch 00191: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 191/200
28/28 [==============================] - 0s 16ms/step - loss: 0.4917 - f1_score: 0.83

Epoch 00192: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 192/200
28/28 [==============================] - 0s 16ms/step - loss: 0.4827 - f1_score: 0.85

Epoch 00193: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 193/200
28/28 [==============================] - 0s 16ms/step - loss: 0.5037 - f1_score: 0.85

Epoch 00194: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 194/200
28/28 [==============================] - 0s 16ms/step - loss: 0.5019 - f1_score: 0.84

Epoch 00195: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 195/200
```

```
28/28 [==============================] - 0s 17ms/step - loss: 0.5058 - f1_score: 0.84

Epoch 00196: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 196/200
28/28 [==============================] - 0s 16ms/step - loss: 0.4983 - f1_score: 0.84

Epoch 00197: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 197/200
28/28 [==============================] - 0s 16ms/step - loss: 0.5073 - f1_score: 0.85

Epoch 00198: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 198/200
28/28 [==============================] - 0s 16ms/step - loss: 0.5107 - f1_score: 0.84

Epoch 00199: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 199/200
28/28 [==============================] - 0s 16ms/step - loss: 0.4962 - f1_score: 0.85

Epoch 00200: LearningRateScheduler reducing learning rate to 1e-06.
Epoch 200/200
28/28 [==============================] - 0s 16ms/step - loss: 0.4983 - f1_score: 0.85
<tensorflow.python.keras.callbacks.History at 0x7f932e171550>
```

```
from IPython.display import Image
Image('/content/Screenshot (72).png',width=1000,height=500)
```

```
from IPython.display import Image
Image('/content/Screenshot (73).png',width=1000,height=500)
```



3. data augmentation

Till now we have done with 2000 samples only. It is very less data. We are giving the proce

There are two types of augmentation:
1. time stretching - Time stretching either increases or decreases the length of the file.
2. pitch shifting - pitch shifting moves the frequencies higher or lower. For pitch shiftin

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


len(aug_temp)
```

```
9
```

As discussed above, for one data point, we will get 9 augmented data points.
We have 2000 data points(train plus test) so, after augmentation we will get 18000 (train - 12600, test - 5400).

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

```
from tqdm import tqdm


augmented_x_train=[]
augmented_y_train=[]
for i in tqdm(range(0,df_audio.shape[0])):
  aug_temp =  generate_augmented_data('/content/recordings/'+df_audio['path'].iloc[i])
  for j in range(len(aug_temp)):
    augmented_x_train.append(aug_temp[j])
    augmented_y_train.append(df_audio['label'].iloc[i])
```

```
100%|██████████| 2000/2000 [10:34<00:00,  3.15it/s]
```

```
#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%
# train test split
from sklearn.model_selection import train_test_split
X_train_aug, X_test_aug, y_train_aug, y_test_aug = train_test_split(augmented_x_train,augment


X_train_aug = np.array(X_train_aug)


X_test_aug = np.array(X_test_aug)
y_train_aug = np.array(y_train_aug)
y_test_aug = np.array(y_test_aug)
```

```
X_test_aug[0]
```

```
array([ 5.6166551e-04,  3.3622846e-04, -1.3086706e-04, ...,
       -8.1110295e-05, -5.5565975e-05, -4.7811420e-05], dtype=float32)
```

```
y_test_aug[0]
```

```
3
```

```
from tqdm import tqdm


max_seq_length=17640
X_train_pad_seq=[]     #np.zeros((1400,17640))
X_train_mask= []        #np.zeros((1400,17640))
for i in tqdm(range(X_train_aug.shape[0])):
  # pad if len less than max_seq_length
  if len(X_train_aug[i]) < max_seq_length:
        seq=X_train_aug[i].tolist()+ \
                                          [0 for item in range(max_seq_length-len(X_train_aug[
  else:
        # Truncate if len greater then max_seq_length
        seq = X_train_aug[i].tolist()[:max_seq_length]
  #mask=[1 if x!=0 else 0 for x in tokens]
  mask=[1 if x!=0 else 0 for x in seq]
  mask=[bool(x) for x in mask]
  X_train_pad_seq.append(seq)
  X_train_mask.append(mask)
```

```
100%|██████████| 12600/12600 [00:47<00:00, 265.21it/s]
```

```
max_seq_length=17640
X_test_pad_seq=[]      #np.zeros((600,17640))
X_test_mask=[]            #np.zeros((600,17640))
for i in tqdm(range(X_test_aug.shape[0])):
  # pad if len less than
  if len(X_test_aug[i]) < max_seq_length:
        # Add the ['PAD'] token
        seq=X_test_aug[i].tolist()+[0 for item in range(max_seq_length-len(X_test_aug[i]))]
  else:
        # Truncate the tokens at maxLen - 1 and add a '[SEP]' tag.
        seq = X_test_aug[i].tolist()[:max_seq_length]
  #mask=[1 if x!=0 else 0 for x in tokens]
  mask=[1 if x!=0 else 0 for x in seq]
  mask=[bool(x) for x in mask]
  X_test_pad_seq.append(seq)
```

```
    X_test_pad_seq.append(seq)
    X_test_mask.append(mask)
```

```
100%|██████████| 5400/5400 [00:23<00:00, 229.96it/s]
```

```
X_train_pad_seq=np.array(X_train_pad_seq)
X_train_mask=np.array(X_train_mask)
X_test_pad_seq=np.array(X_test_pad_seq)
X_test_mask=np.array(X_test_mask)
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Input,Activation,BatchNormalization,Dropout,Embeddi
from tensorflow.keras.models import Model
import random as rn
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from tensorflow.keras import layers
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ModelCheckpoint ,TensorBoard,EarlyStopping,LearningRat
#from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from tensorflow.keras.layers import concatenate
```

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(y_train_aug)
y_train_encoded = encoder.transform(y_train_aug)
y_test_encoded = encoder.transform(y_test_aug)
y_train_ohe = tf.keras.utils.to_categorical(y_train_encoded)
y_test_ohe = tf.keras.utils.to_categorical(y_test_encoded)
```

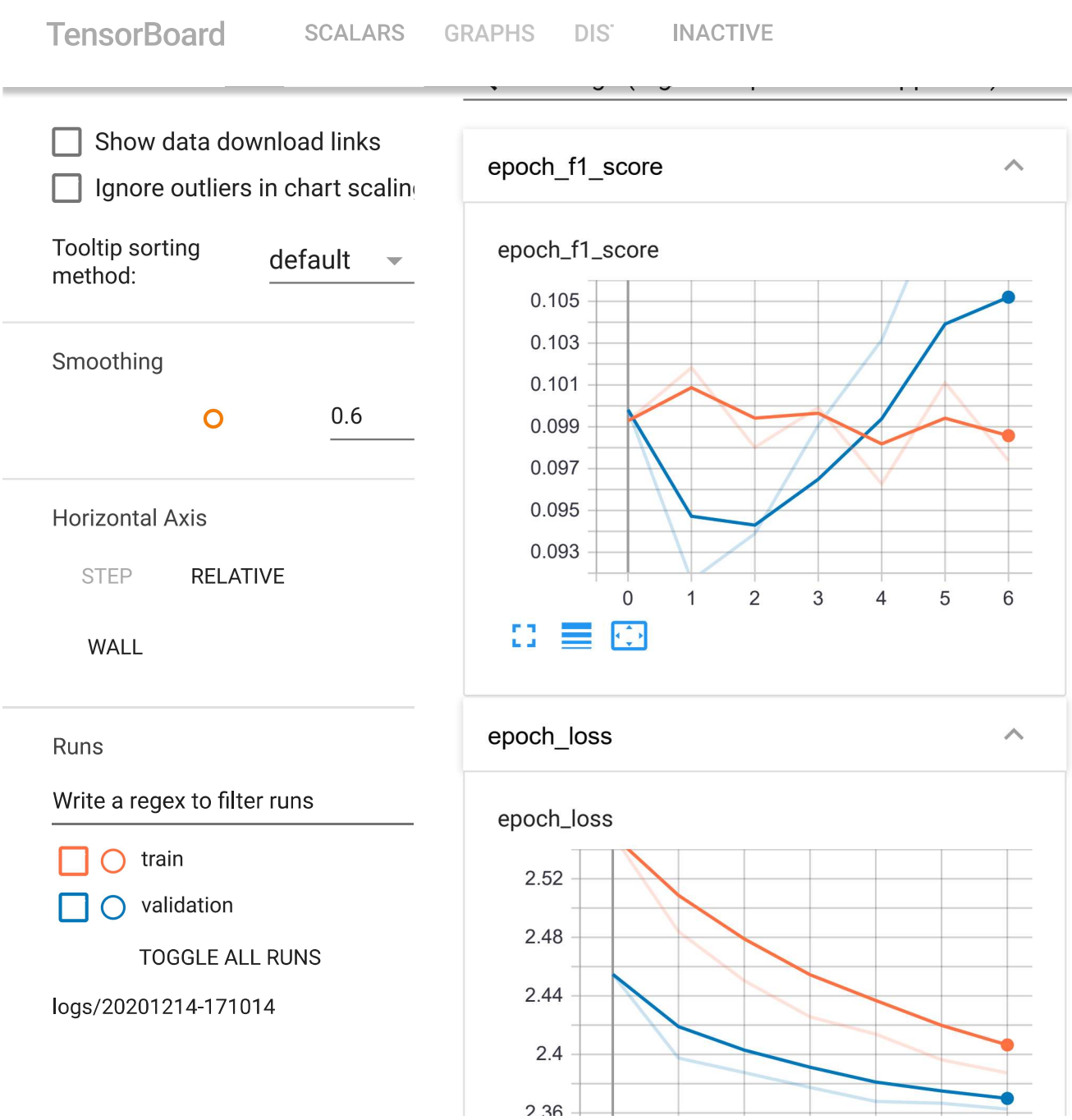
```
from tensorflow.keras.layers import Input, LSTM, Dense,GlobalAveragePooling1D,GlobalAveragePo
from tensorflow.keras.models import Model
import tensorflow as tf
```

Model3

```
## as discussed above, please write the LSTM
input1 = Input(shape=(17640, 1,))
input_mask=Input(shape=(17640,),dtype=bool)
LSTM_layer= LSTM(50,dropout=0.2)(inputs=input1,mask=input_mask)
x = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30),ke
x  =  Dropout(0.25)(x)
x   = BatchNormalization()(x)
```

```
x = Dense(12,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30),ke
x  =  Dropout(0.35)(x)
x   = BatchNormalization()(x)
x = Dense(8,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30),ker
x   = BatchNormalization()(x)
dense = Dense(10,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
Out = Dense(units=10,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_nor
model3= Model(inputs=(input1,input_mask),outputs=Out)
```

```
import tensorflow as tf
import keras.backend as K
import os
import datetime

#https://www.kaggle.com/c/liverpool-ion-switching/discussion/132646

def f1(y_true, y_pred):
    y_pred = K.round(y_pred)
    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    # tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.math.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)


from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
import tensorflow_addons as tfa


%load_ext tensorboard


#tensorbord for model1
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```




```
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001)
model3.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=[tfa.metrics.F1Sc
```

```
model3.summary()
```

```
Model: "functional_1"
```

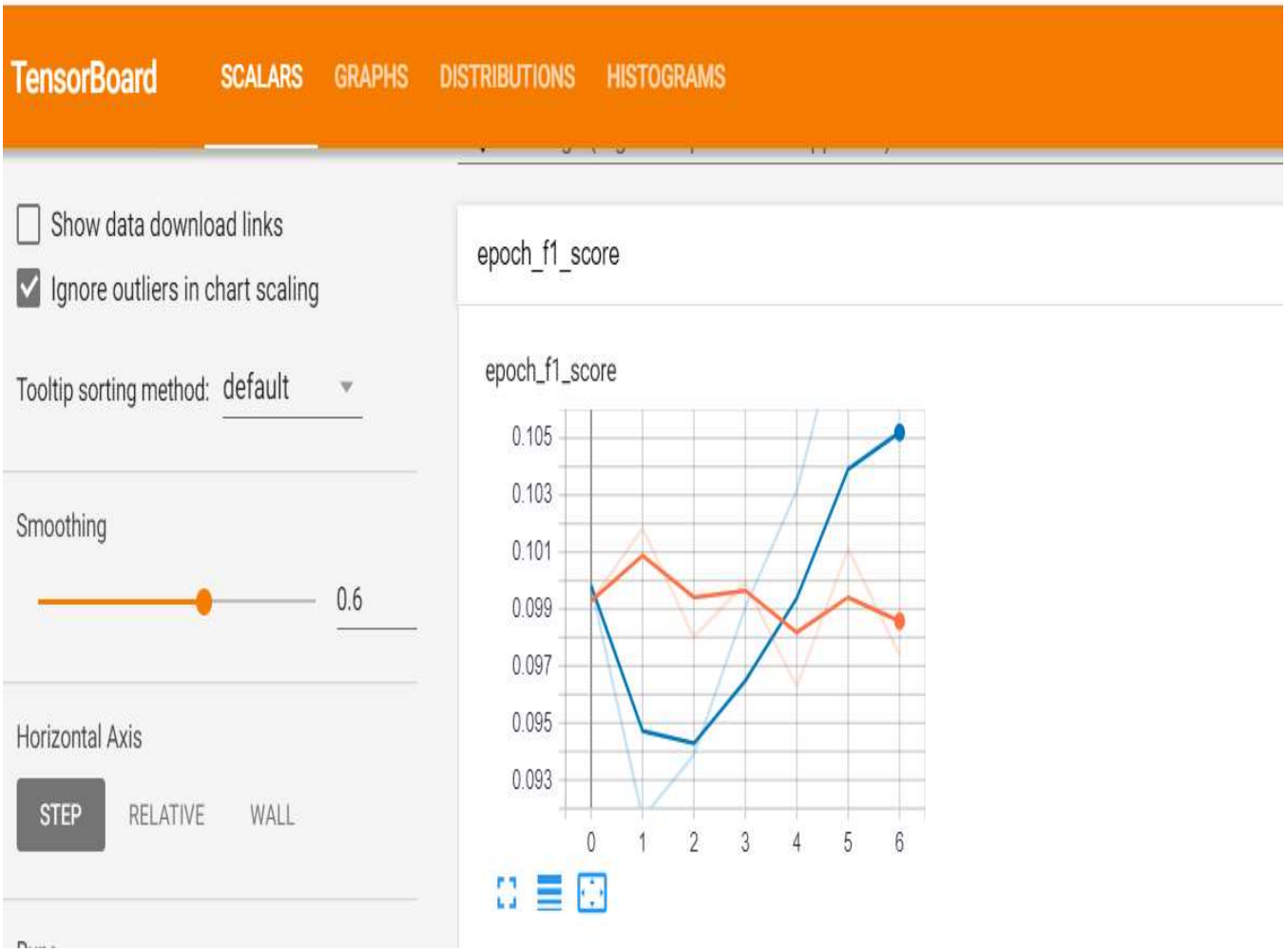
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 17640, 1)]	0	
input_2 (InputLayer)	[(None, 17640)]	0	
lstm (LSTM)	(None, 50)	10400	input_1[0][0]

```
                                                                 input_2[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 16)           816         lstm[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 16)           0           dense[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 16)           64          dropout[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 12)           204         batch_normalization[0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 12)           0           dense_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 12)           48          dropout_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 8)            104         batch_normalization_1[0
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 8)            32          dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 10)           90          batch_normalization_2[0
__________________________________________________________________________________________________
Output (Dense)                  (None, 10)           110         dense_3[0][0]
==================================================================================================
Total params: 11,868
Trainable params: 11,796
Non-trainable params: 72
__________________________________________________________________________________________________
```

```
model3.fit([X_train_pad_seq,X_train_mask],y_train_ohe,epochs=7, validation_data=([X_test_pad_
          callbacks=[tensorboard_callback])
```

```
Epoch 1/7
  1/394 [..............................] - ETA: 0s - loss: 2.7479 - f1_score: 0.0625WARN
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
  2/394 [..............................] - ETA: 12:57 - loss: 2.6330 - f1_score: 0.1094W
394/394 [==============================] - 312s 792ms/step - loss: 2.5499 - f1_score: 0
Epoch 2/7
394/394 [==============================] - 309s 783ms/step - loss: 2.4839 - f1_score: 0
Epoch 3/7
394/394 [==============================] - 307s 780ms/step - loss: 2.4504 - f1_score: 0
Epoch 4/7
394/394 [==============================] - 309s 783ms/step - loss: 2.4257 - f1_score: 0
Epoch 5/7
394/394 [==============================] - 308s 783ms/step - loss: 2.4138 - f1_score: 0
Epoch 6/7
394/394 [==============================] - 308s 781ms/step - loss: 2.3963 - f1_score: 0
Epoch 7/7
394/394 [==============================] - 310s 786ms/step - loss: 2.3873 - f1_score: 0
<tensorflow.python.keras.callbacks.History at 0x7f1a73d4b908>
```

```
from IPython.display import Image
Image('/content/Screenshot (66).png',width=1000,height=500)
```




```
from IPython.display import Image
Image('/content/Screenshot (67).png',width=1000,height=500)
```

Converting into spectrogram and giving spectrogram data as input

```
def convert_to_spectrogram(raw_data):
    '''converting to spectrogram'''
    spectrum = librosa.feature.melspectrogram(y=raw_data, sr=sample_rate, n_mels=64)
    logmel_spectrum = librosa.power_to_db(S=spectrum, ref=np.max)
    return logmel_spectrum
```

```
X_train_spectrogram=[]
X_test_spectrogram=[]
for i in tqdm(range(X_train_pad_seq.shape[0])):
  X_train_spectrogram.append(convert_to_spectrogram(X_train_pad_seq[i]))

for i in tqdm(range(X_test_pad_seq.shape[0])):
  X_test_spectrogram.append(convert_to_spectrogram(X_test_pad_seq[i]))
```

```
    100%|██████████| 12600/12600 [02:29<00:00, 84.15it/s]
    100%|██████████| 5400/5400 [02:25<00:00, 37.09it/s]
```

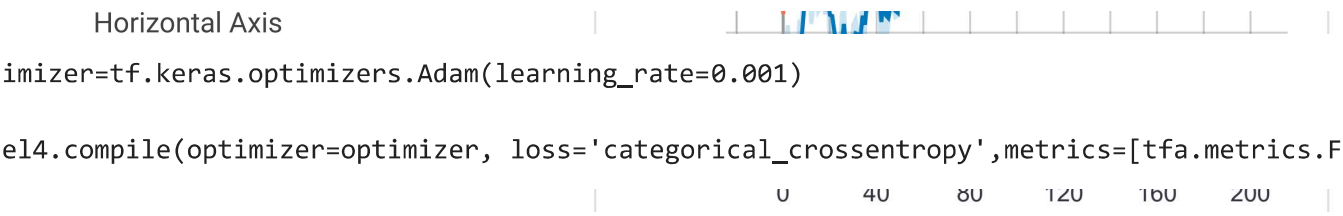
```
X_train_spectrogram = np.array(X_train_spectrogram)
X_test_spectrogram = np.array(X_test_spectrogram)
```

Model4

```
## as discussed above, please write the LSTM
input1 = Input(shape=(64, 35,))
LSTM_layer= LSTM(256,dropout=0.01,return_sequences=True)(input1)
#tr = tf.transpose(LSTM_layer,perm=[0, 2, 1])
avg=GlobalAveragePooling1D(data_format='channels_last')(LSTM_layer)
x = Dense(256,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=31),k
x  =  Dropout(0.01)(x)
x   = BatchNormalization()(x)
x = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=43),k
x  =  Dropout(0.01)(x)
x   = BatchNormalization()(x)
x = Dense(64,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=54),ke
x   = BatchNormalization()(x)
```

```
dense = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=40
Out = Dense(units=10,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_nor
model4= Model(inputs=input1,outputs=Out)


#tensorbord for model1
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```

```
def changeLearningRate(epochs,learning_rate):
  if epochs<50:
    learning_rate=0.001
    return learning_rate

  elif epochs<75 and epochs>125:
    learning_rate=0.0001
    return learning_rate
  else:
    learning_rate=0.000001
    return learning_rate



lrschedule = LearningRateScheduler(changeLearningRate,verbose=1)
```

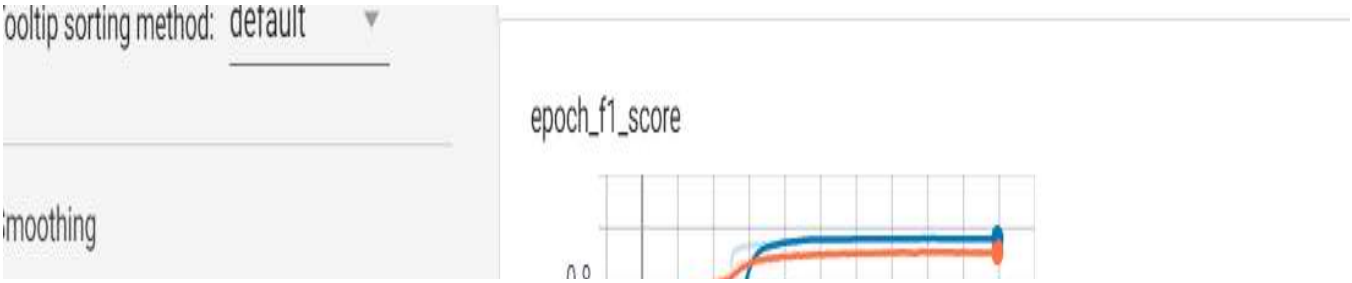



```
optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)

model4.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=[tfa.metrics.F1Sc

model4.fit(X_train_spectrogram,y_train_ohe,epochs=200, validation_data=(X_test_spectrogram,y_
          callbacks=[tensorboard_callback,lrschedule])
```

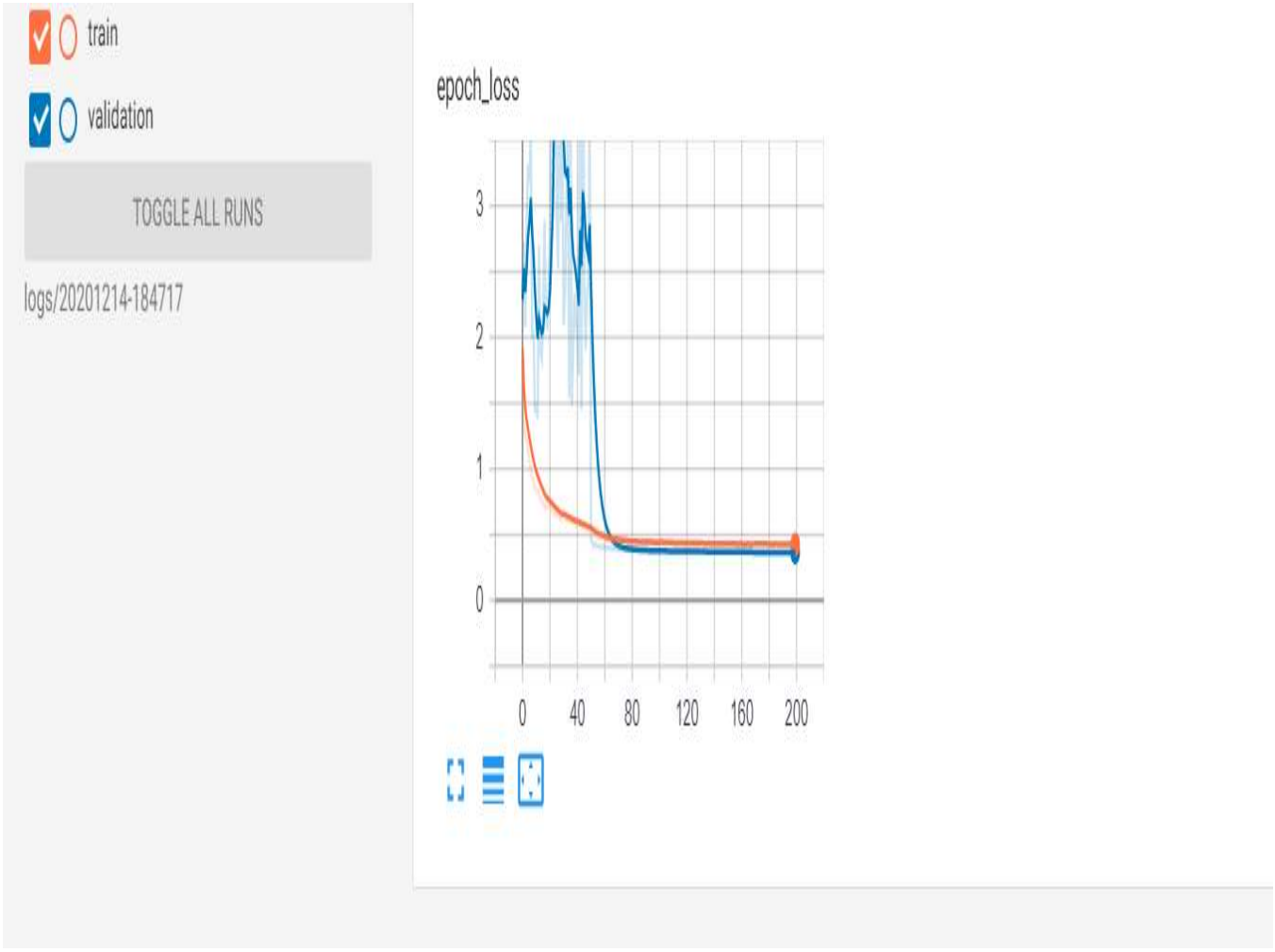
```
Epoch 6/200
252/252 [==============================] - 3s 13ms/step - loss: 1.0613 - f1_score: 0.

Epoch 00007: LearningRateScheduler reducing learning rate to 0.001.
Epoch 7/200
252/252 [==============================] - 3s 13ms/step - loss: 1.0044 - f1_score: 0.

Epoch 00008: LearningRateScheduler reducing learning rate to 0.001.
Epoch 8/200
252/252 [==============================] - 3s 13ms/step - loss: 0.9318 - f1_score: 0.

Epoch 00009: LearningRateScheduler reducing learning rate to 0.001.
Epoch 9/200
252/252 [==============================] - 3s 13ms/step - loss: 0.9159 - f1_score: 0.

Epoch 00010: LearningRateScheduler reducing learning rate to 0.001.
Epoch 10/200
252/252 [==============================] - 3s 13ms/step - loss: 0.8553 - f1_score: 0.

Epoch 00011: LearningRateScheduler reducing learning rate to 0.001.
Epoch 11/200
252/252 [==============================] - 3s 13ms/step - loss: 0.8343 - f1_score: 0.

Epoch 00012: LearningRateScheduler reducing learning rate to 0.001.
Epoch 12/200
252/252 [==============================] - 3s 13ms/step - loss: 0.8347 - f1_score: 0.

Epoch 00013: LearningRateScheduler reducing learning rate to 0.001.
Epoch 13/200
```

```
252/252 [==============================] - 3s 13ms/step - loss: 0.8212 - f1_score: 0.

Epoch 00014: LearningRateScheduler reducing learning rate to 0.001.
Epoch 14/200
252/252 [==============================] - 3s 13ms/step - loss: 0.7758 - f1_score: 0.

Epoch 00015: LearningRateScheduler reducing learning rate to 0.001.
Epoch 15/200
252/252 [==============================] - 3s 13ms/step - loss: 0.7610 - f1_score: 0.

Epoch 00016: LearningRateScheduler reducing learning rate to 0.001.
Epoch 16/200
252/252 [==============================] - 3s 13ms/step - loss: 0.7446 - f1_score: 0.

Epoch 00017: LearningRateScheduler reducing learning rate to 0.001.
Epoch 17/200
252/252 [==============================] - 3s 13ms/step - loss: 0.7161 - f1_score: 0.

Epoch 00018: LearningRateScheduler reducing learning rate to 0.001.
Epoch 18/200
252/252 [==============================] - 3s 13ms/step - loss: 0.7058 - f1_score: 0.

Epoch 00019: LearningRateScheduler reducing learning rate to 0.001.
Epoch 19/200
252/252 [==============================] - 3s 13ms/step - loss: 0.7532 - f1_score: 0.

Epoch 00020: LearningRateScheduler reducing learning rate to 0.001.
Epoch 20/200
252/252 [==============================] - 3s 13ms/step - loss: 0.7190 - f1_score: 0.
```

```
from IPython.display import Image
Image('/content/Screenshot (68).png',width=1000,height=500)
```




```
from IPython.display import Image
Image('/content/Screenshot (69).png',width=1000,height=500)
```



Observation :

1) In this assignment we have performed spoken digit Recognition ,the input will be speech and the ou
2)first of all we read the data and created a dataFrame of path(file path) and level(digit number)
3)Then split the data into train and test
4From the path we loaded the data as the it is in wav file we have used librosa library
5)then we have done masking the train and test , and used level encoder for y.
6)then created the first model.

Observation in tensorboard

Because this model is trained only on raw tada that’s why f1 score is very less around 0.1

7) Then we have converted row data into spectrogram
8) created Model2 and trained

Observation in tensorboard

with this data we got f1 score 0.82 without overfitting the model

9)Now we have used data augmentation technique and created 18000 data from 2000 data
10)Then trained model3

Observation in tensorboard

Because this model is trained only on raw tada that’s why f1 score is very less around 0.1

11) After that we have converted augmented data into spectrogram and trained model4

Observation in tensorboard

with this data we got f1 score 0.85 without overfitting the model

+ Code + Text