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 ce
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 pla
d.

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 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.

In [1]:
```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Mounted at /gdrive
/gdrive
```

In [2]:
```
import numpy as np
import pandas as pd
import librosa
import os
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import tensorflow as tf

##if you need any imports you can do that here.
```

We shared recordings.zip, please unzip those.

In [3]:
```
#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"

#!unzip  "/gdrive/My Drive/Spoken_digit/recordings.zip" -d "/gdrive/My Drive/"

all_files_name =  os.listdir('/gdrive/My Drive/recordings')

all_files  = []
labels = []
# get path of all_files
for i in tqdm(all_files_name):
  file_path = '/gdrive/My Drive/recordings/'+str(i)
  all_files.append(file_path)
  split =  int(i.split('_')[0])
  labels.append(split)
```

```
100%|██████████| 2000/2000 [00:00<00:00, 436611.04it/s]
```

In [4]:
```
all_files[:5]
```

Out[4]:
```
['/gdrive/My Drive/recordings/7_theo_33.wav',
 '/gdrive/My Drive/recordings/5_jackson_35.wav',
 '/gdrive/My Drive/recordings/3_jackson_19.wav',
 '/gdrive/My Drive/recordings/9_yweweler_20.wav',
 '/gdrive/My Drive/recordings/0_theo_28.wav']
```

In [5]:
```
labels[:5]
```

Out[5]:
```
[7, 5, 3, 9, 0]
```

Grader function 1

In [6]:
```
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()
```

Out[6]:
```
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

In [7]:
```
#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
df_audio = pd.DataFrame({'path' :all_files, 'label':labels })
```

In [8]:
```
#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

In [9]:
```
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()
```

Out[9]: True

In [10]:
```
from sklearn.utils import shuffle
df_audio = shuffle(df_audio, random_state=33)#don't change the random state
```

Train and Validation split

In [11]:
```
#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,stratify = y,random_state=45)
```

Grader function 3

In [12]:
```
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()
```

Out[12]:
```
True
```

Preprocessing

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

In [13]:
```
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
```

In [14]:
```
#use load_wav function that was written above to get every wave.
#save it in X_train_processed and X_test_processed
# X_train_processed/X_test_processed should be dataframes with two columns(raw_data, duration) with same index of X_train/y_train
X_train_processed = []
X_test_processed = []

# X_train
for i in tqdm(range(len(X_train))):
  processor = load_wav(X_train.iloc[i])
  X_train_processed.append(processor)

############################################################
#X_test
for i in tqdm(range(len(X_test))):
  processor = load_wav(X_test.iloc[i])
  X_test_processed.append(processor)
```

```
100%|██████████| 1400/1400 [08:01<00:00,  2.91it/s]
100%|██████████| 600/600 [03:29<00:00,  2.86it/s]
```

In [15]:
```
train_1 = X_train_processed.copy()
test_1 = X_test_processed.copy()
```

In [16]:
```
X_train_processed[0]
```

Out[16]:
```
[array([-0.00060508, -0.00061847, -0.00026167, ..., -0.00063194,
        -0.00041468,  0.        ], dtype=float32), 0.3652607709750567]
```

In [18]:
```
new_X_train =  pd.DataFrame(data={'raw_data':[X_train_processed[i][0] for i in range(len(X_train_processed))], 'duration':[X_train_proce
new_X_test =   pd.DataFrame(data={'raw_data':[X_test_processed[i][0] for i in range(len(X_test_processed))], 'duration':[X_test_processe
```

In [19]:
```
new_X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   raw_data  600 non-null    object 
 1   duration  600 non-null    float64
dtypes: float64(1), object(1)
memory usage: 9.5+ KB
```

In [20]:
```
new_X_train.head(2)
```

Out[20]:

	raw_data	duration
0	[-0.0006050776, -0.0006184709, -0.00026166602,...	0.365261
1	[0.00019366974, -0.0023843, -0.0058723832, -0....	0.299773

In [21]:
```
new_X_test['raw_data'][0]
```

Out[21]:
```
array([-4.9881153e-05,  6.9615439e-06,  5.9910067e-06, ...,
       -3.3086265e-04, -2.4591145e-04,  0.0000000e+00], dtype=float32)
```

In [22]:
```
new_X_train.head(5)
```

Out[22]:

	raw_data	duration
0	[-0.0006050776, -0.0006184709, -0.00026166602,...	0.365261
1	[0.00019366974, -0.0023843, -0.0058723832, -0....	0.299773
2	[-0.008321674, -0.013692323, -0.01608319, -0.0...	0.246032
3	[0.0005516098, 0.00037881808, 0.00012904029, -...	0.277415
4	[-0.00054838305, -0.000720711, -0.00075189554,...	0.472381

In [23]:
```
new_X_test.head(5)
```

Out[23]:

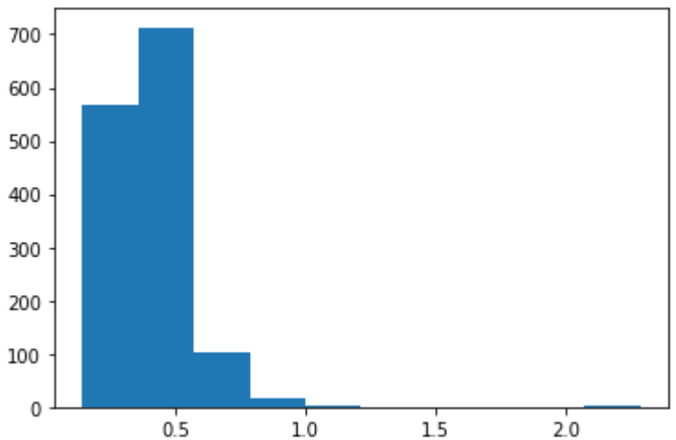
	raw_data	duration
0	[-4.9881153e-05, 6.961544e-06, 5.9910067e-06, ...	0.400907
1	[0.0013453948, 0.0012860884, 0.00067011634, -0...	0.477143
2	[-0.00095882645, -0.019986238, -0.02934129, -0...	0.297778
3	[-9.786627e-06, -0.00016342092, -0.0004535091,...	0.447755
4	[0.011016414, 0.013351645, 0.013998778, 0.0138...	0.537506

In [24]:
```
#new_X_train.to_csv('/gdrive/My Drive/Spoken_digit/X_train.csv',index=False)
#new_X_test.to_csv('/gdrive/My Drive/Spoken_digit/X_test.csv',index=False)
```

In [25]:
```
#new_X_train = pd.read_csv('/gdrive/My Drive/Spoken_digit/X_train.csv')
#new_X_test  = pd.read_csv('/gdrive/My Drive/Spoken_digit/X_test.csv')
```

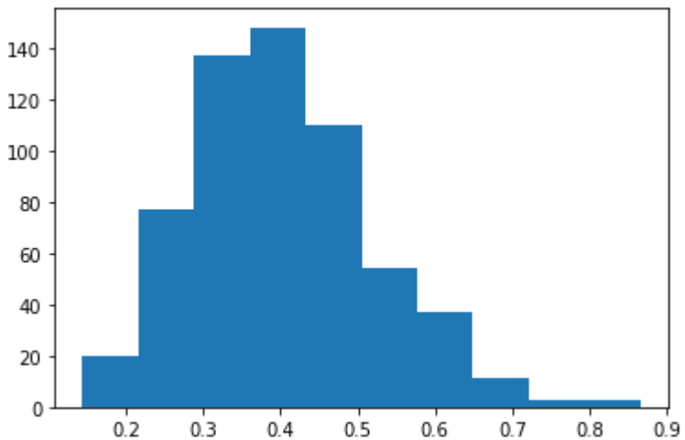
In [26]:
```
#plot the histogram of the duration for trian
plt.hist(new_X_train['duration'])
```

Out[26]:
```
(array([566., 713., 102.,  15.,   2.,   0.,   0.,   0.,   0.,   2.]),
 array([0.14353741, 0.35746032, 0.57138322, 0.78530612, 0.99922902,
        1.21315193, 1.42707483, 1.64099773, 1.85492063, 2.06884354,
        2.28276644]),
 <a list of 10 Patch objects>)
```



In [27]:
```
#Plot for test data
plt.hist(new_X_test['duration'])
```

Out[27]:
```
(array([ 20.,  77., 137., 148., 110.,  54.,  37.,  11.,   3.,   3.]),
 array([0.14362812, 0.2158322 , 0.28803628, 0.36024036, 0.43244444,
        0.50464853, 0.57685261, 0.64905669, 0.72126077, 0.79346485,
        0.86566893]),
 <a list of 10 Patch objects>)
```



In [28]:
```
# refer - https://www.geeksforgeeks.org/numpy-percentile-in-python/
for i in range (0,101,10):
    p = np.percentile(new_X_train['duration'], i)
    print(str(i)+" Percentile: "+ str(p))
```

```
0 Percentile: 0.1435374149659864
10 Percentile: 0.2608934240362812
20 Percentile: 0.2977233560090703
30 Percentile: 0.3297777777777778
40 Percentile: 0.35663492063492064
50 Percentile: 0.389750566893424
60 Percentile: 0.41427664399092967
70 Percentile: 0.44360544217687076
80 Percentile: 0.4822312925170068
90 Percentile: 0.5535283446712018
100 Percentile: 2.282766439909297
```

In [29]:
```
# refer - https://www.geeksforgeeks.org/numpy-percentile-in-python/
for i in range (90,101,1):
    p = np.percentile(new_X_train['duration'], i)
    print(str(i)+" Percentile: "+ str(p))
```

```
90 Percentile: 0.5535283446712018
91 Percentile: 0.5659854875283448
92 Percentile: 0.5794503401360545
93 Percentile: 0.5938775510204082
94 Percentile: 0.6082149659863945
95 Percentile: 0.622421768707483
96 Percentile: 0.6424979591836734
97 Percentile: 0.6729219954648525
98 Percentile: 0.7120553287981859
99 Percentile: 0.8072766439909297
100 Percentile: 2.282766439909297
```

Grader function 4

In [30]:
```
X_train_processed = new_X_train
X_test_processed  = new_X_test
```

In [31]:
```
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()
```

Out[31]:
```
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_proc
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 lengt
*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.

In [32]:
```
max_length  = 17640
```

In [33]:
```
## as discussed above, Pad with Zero if length of sequence is less than 17640 else Truncate the number.
## save in the X_train_pad_seq, X_test_pad_seq
## also Create masking vector X_train_mask, X_test_mask

## all the X_train_pad_seq, X_test_pad_seq, X_train_mask, X_test_mask will be numpy arrays mask vector dtype must be bool.
#X_train padding Sequences
X_train_pad_seq  = []
for i in tqdm(range(len(new_X_train))):
  sequences  = [new_X_train['raw_data'][i]]
  aaa  = tf.keras.preprocessing.sequence.pad_sequences(
      sequences, maxlen=max_length, dtype='float', padding='post', truncating='post',
      value=100)
  X_train_pad_seq.extend(aaa)

#################################################################################################################

#X_test padding Sequences
X_test_pad_seq  = []
for i in tqdm(range(len(new_X_test))):
  sequences  = [new_X_test['raw_data'][i]]
  aaa  = tf.keras.preprocessing.sequence.pad_sequences(
      sequences, maxlen=max_length, dtype='float', padding='post', truncating='post',
      value=100)
  X_test_pad_seq.extend(aaa)
```

```
100%|██████████| 1400/1400 [00:00<00:00, 8257.82it/s]
100%|██████████| 600/600 [00:00<00:00, 8749.22it/s]
```

In [34]:
```
X_train_pad_seq = np.array(X_train_pad_seq)
X_test_pad_seq = np.array(X_test_pad_seq)
print('shape of X_train padding', X_train_pad_seq.shape)
print('shape of X_test padding', X_test_pad_seq.shape)
```

```
shape of X_train padding (1400, 17640)
shape of X_test padding (600, 17640)
```

In [35]:
```
# X_train_mask
X_train_mask  = []
for i in tqdm(range(len(new_X_train))):
  X_train_mask_replace = np.where(X_train_pad_seq[i]!=100.0, 1, X_train_pad_seq[i])
  X_train_mask_replace = np.where(X_train_mask_replace==100.0, 0, X_train_mask_replace)
  X_train_mask.append(X_train_mask_replace)

#########################################################################################

X_test_mask  = []
for i in tqdm(range(len(new_X_test))):
  X_test_mask_replace = np.where(X_test_pad_seq[i]!=100.0, 1, X_train_pad_seq[i])
  X_test_mask_replace = np.where(X_test_mask_replace==100.0, 0, X_test_mask_replace)
  X_test_mask.append(X_test_mask_replace)
```

```
100%|██████████| 1400/1400 [00:00<00:00, 11055.14it/s]
100%|██████████| 600/600 [00:00<00:00, 11506.64it/s]
```

In [36]:
```
X_train_mask[:2]
```

Out[36]:
```
[array([1., 1., 1., ..., 0., 0., 0.]), array([1., 1., 1., ..., 0., 0., 0.])]
```

In [37]:
```
X_train_mask = np.array(X_train_mask).astype('bool')
X_test_mask  = np.array(X_test_mask).astype('bool')
```

In [38]:
```
X_train_mask[:2]
```

Out[38]:
```
array([[ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False]])
```

Grader function 5

In [39]:
```
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,))
    #print(flag_padshape)
    flag_maskshape = (X_train_mask.shape==(1400, 17640)) and (X_test_mask.shape==(600, 17640)) and (y_test.shape==(600,))
    #print(flag_maskshape)
    flag_dtype = (X_train_mask.dtype==bool) and (X_test_mask.dtype==bool)
    #print(flag_dtype)
    return flag_padshape and flag_maskshape and flag_dtype
grader_padoutput()
```

Out[39]: 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
read LSTM documentation(https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM) in tensorflow to know more about mask and a
s://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 hav
puts) 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

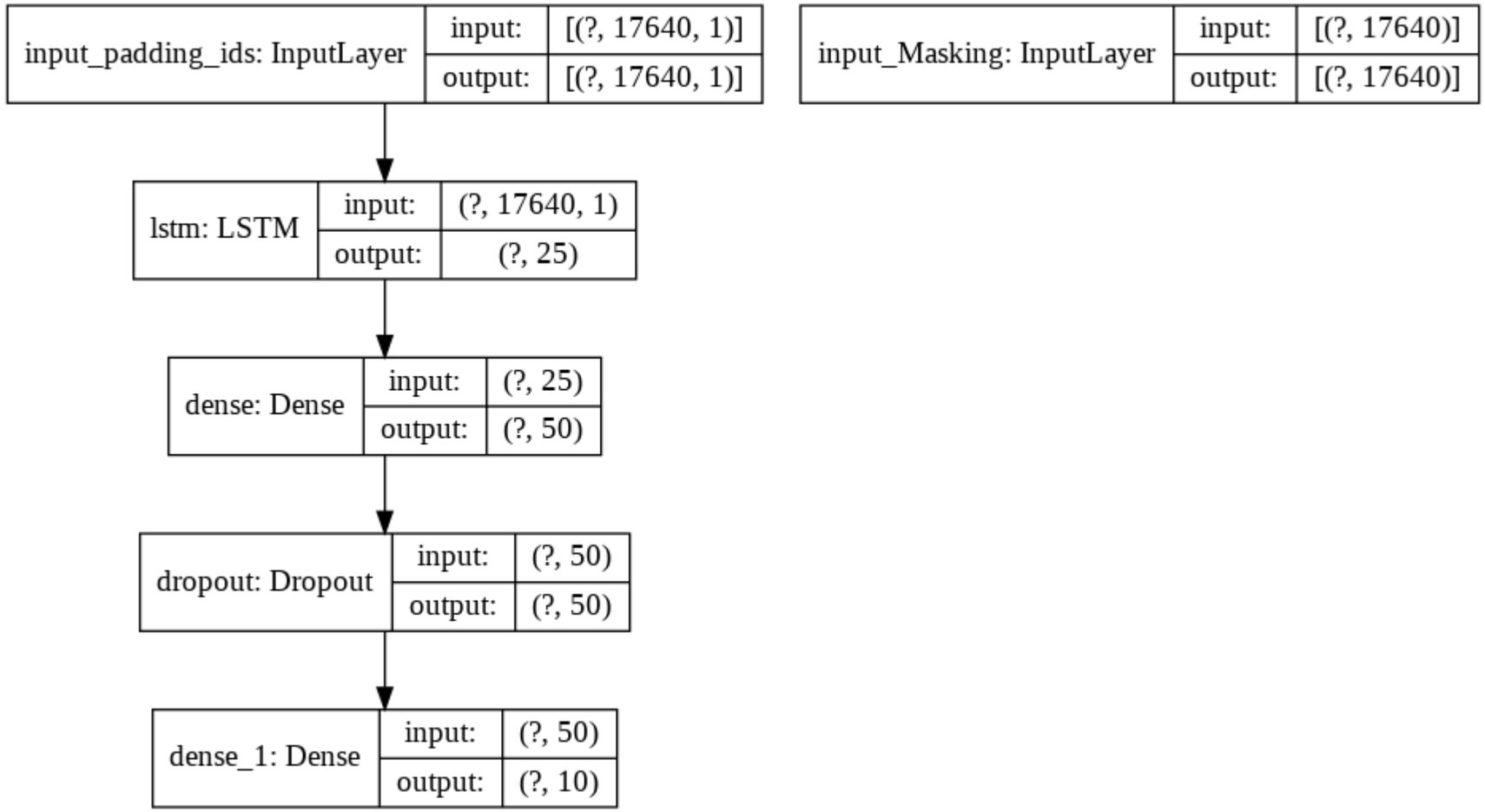
In [40]:
```
from tensorflow.keras.layers import Input, LSTM, Dense,Masking,Dropout
from tensorflow.keras.models import Model
import tensorflow as tf
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
from keras.regularizers import l2,l1_l2,l1
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau, TensorBoard,LearningRat
import datetime
from tensorflow.keras.layers import  AveragePooling1D,GlobalAveragePooling1D,MaxPooling1D,TimeDistributed
import random as rn
```

In [ ]:
```
## as discussed above, please write the LSTM
time_steps = 17640
n_features = 1
#this is input words. Sequence of words represented as integers
input_padding = tf.keras.layers.Input(shape=(time_steps,n_features), name="input_padding_ids")

#mask vector if you are padding anything
input_mask = tf.keras.layers.Input(shape=(time_steps), name="input_Masking")


lstm  = LSTM(25)(input_padding)
x = Dense(50, activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(lstm)
x = Dropout(0.2)(x)
output = Dense(10, activation = 'softmax')(x)

model = Model(inputs=[input_mask,input_padding],outputs=output)
model.summary()
```

```
Model: "functional_13"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_padding_ids (InputLayer)  [(None, 17640, 1)]   0
_________________________________________________________________________________________________
lstm_11 (LSTM)               (None, 25)                2700        input_padding_ids[0][0]
_________________________________________________________________________________________________
dense_14 (Dense)             (None, 50)                1300        lstm_11[0][0]
_________________________________________________________________________________________________
dropout_7 (Dropout)          (None, 50)                0           dense_14[0][0]
_________________________________________________________________________________________________
input_Masking (InputLayer)   [(None, 17640)]           0
_________________________________________________________________________________________________
dense_15 (Dense)             (None, 10)                510         dropout_7[0][0]
=================================================================================================
Total params: 4,510
Trainable params: 4,510
Non-trainable params: 0
_________________________________________________________________________________________________
```

In [ ]:
```
dot_img_file = '/tmp/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```

Out[28]:



In [ ]:

```
ACCURACY_THRESHOLD = 0.1
class Metrics(tf.keras.callbacks.Callback):
  def __init__(self, validation):
    super(Metrics, self).__init__()
    self.validation_data = validation

  def on_train_begin(self, logs={}):
    self.val_f1s = []


  def on_epoch_end(self, epoch, logs={}):

    val_predict = self.model.predict(self.validation_data[0])
    val_predict = np.argmax(val_predict, axis=1)

    val_targ = self.validation_data[1]

    _val_f1 = f1_score(val_targ, val_predict,average='micro')
    self.val_f1s.append(_val_f1)

    print(' — val_f1: %f ' %(_val_f1))

    if (_val_f1 > ACCURACY_THRESHOLD):
      print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD*100))
      self.model.stop_training = True


metrics = Metrics((X_train_pad_seq, y_train))
```

In [ ]:
```
# Call back
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=10, verbose=1)
filepath="/gdrive/My Drive/model_save/best_model-{epoch:02d}.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1, save_best_only=True, mode='auto')
# TensorBoard Creation
%load_ext tensorboard
folder_name = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Create log folder - TensorBoard
log_dir="/gdrive/My Drive/logs/fit/" + folder_name
tensorboard_callback =TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True)
```

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

In [ ]:
```
folder_name
```

Out[103]:
```
'20201112-144741'
```

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

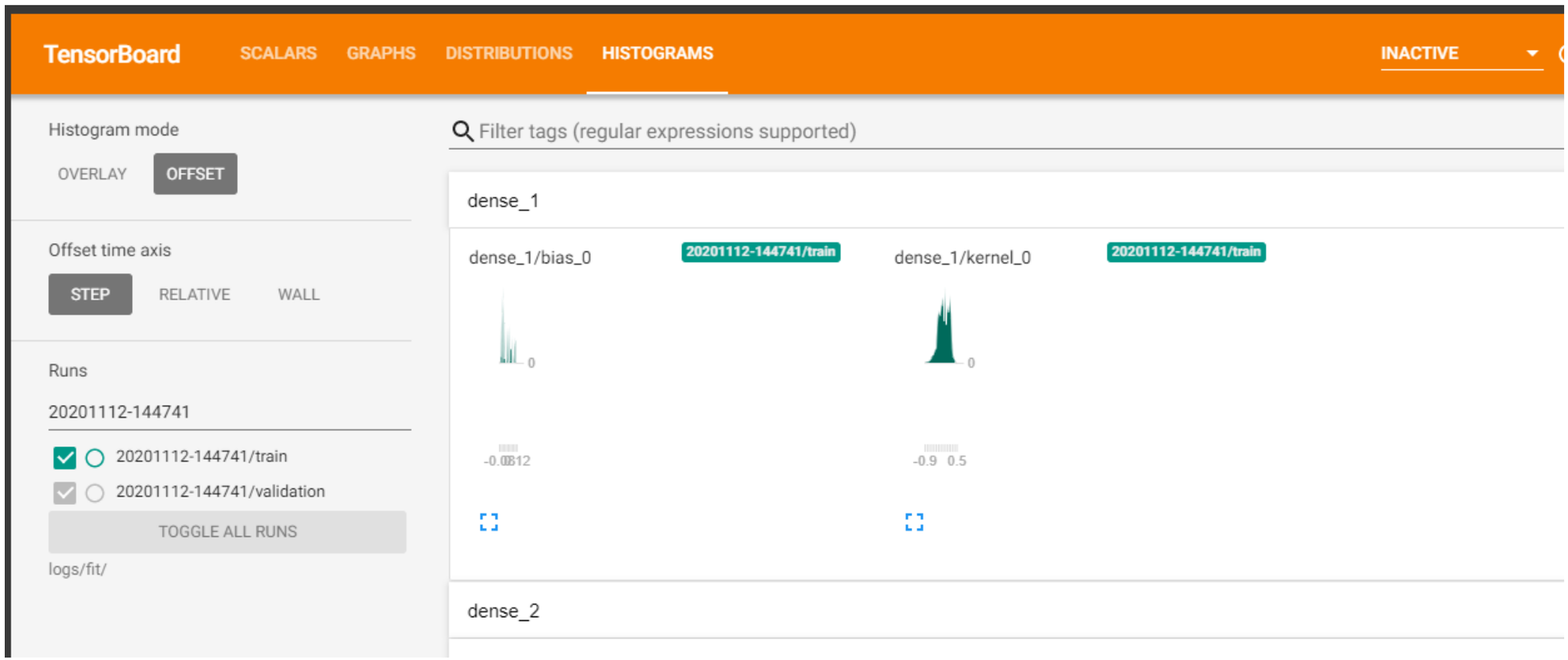
In [ ]:
```
model.compile(loss='sparse_categorical_crossentropy', optimizer= 'adam', metrics=['accuracy'])
model.fit(x = X_train_pad_seq,  y=y_train, epochs=40,verbose=1,batch_size=64, validation_data=(X_test_pad_seq, y_test),
          callbacks =[checkpoint,tensorboard_callback,earlystop,metrics])
```

```
Epoch 1/40
 2/22 [=>............................] - ETA: 41s - loss: 2.2067 - accuracy: 0.1797WARNING:tensorflow:Callbacks method `on_train_batch_e
compared to the batch time (batch time: 0.9592s vs `on_train_batch_end` time: 3.1906s). Check your callbacks.
22/22 [==============================] - ETA: 0s - loss: 2.1795 - accuracy: 0.1607
Epoch 00001: val_accuracy improved from 0.16000 to 0.17286, saving model to /gdrive/My Drive/model_save/best_model-01.h5
 — val_f1: 0.172857

Reached 10.00% accuracy, so stopping training!!
22/22 [==============================] - 43s 2s/step - loss: 2.1795 - accuracy: 0.1607 - val_loss: 2.1571 - val_accuracy: 0.1729
```

Out[99]:
```
<tensorflow.python.keras.callbacks.History at 0x7f6e58218fd0>
```

In [ ]:
```
#Model 1 - results
os.chdir('/gdrive/My Drive/')
%tensorboard --logdir logs/fit/
```

Output hidden; open in https://colab.research.google.com (https://colab.research.google.com) to view.



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 t
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 h
nsn.org/spectrograms/what-is-a-spectrogram

In [41]:
```
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)
    #print(logmel_spectrum.shape)
    return logmel_spectrum
```

In [42]:
```
##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)
##Train data: X_train_pad_seq, X_train_mask and y_train
#Test data: X_test_pad_seq, X_test_mask and y_test


X_train_spectrogram  = []
for i in tqdm(range(len(X_train_pad_seq))):
  aaa = convert_to_spectrogram(X_train_pad_seq[i])
  X_train_spectrogram.append(aaa)


##########################################################################################################

#X_test padding Sequences
X_test_spectrogram  = []
for i in tqdm(range(len(X_test_pad_seq))):
  bbb = convert_to_spectrogram(X_test_pad_seq[i])
  X_test_spectrogram.append(bbb)
```

```
100%|██████████| 1400/1400 [00:08<00:00, 168.63it/s]
100%|██████████| 600/600 [00:03<00:00, 170.37it/s]
```

In [43]:
```
X_train_spectrogram = np.array(X_train_spectrogram)
X_test_spectrogram = np.array(X_test_spectrogram)
print('shape of X_train spectrogram', X_train_spectrogram.shape)
print('shape of X_test spectrogram', X_test_spectrogram.shape)
```

```
shape of X_train spectrogram (1400, 64, 35)
shape of X_test spectrogram (600, 64, 35)
```

Grader function 6

In [44]:
```
def grader_spectrogram():
    flag_shape = (X_train_spectrogram.shape==(1400,64, 35)) and (X_test_spectrogram.shape == (600, 64, 35))
    return flag_shape
grader_spectrogram()
```

Out[44]: 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 (#., time_steps, features) average the output of every time step i.e, you should get (#.,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 micro F1 score as metric) and histograms of gradients.
5. make sure that it won't overfit.
6. You are free to include any regularization

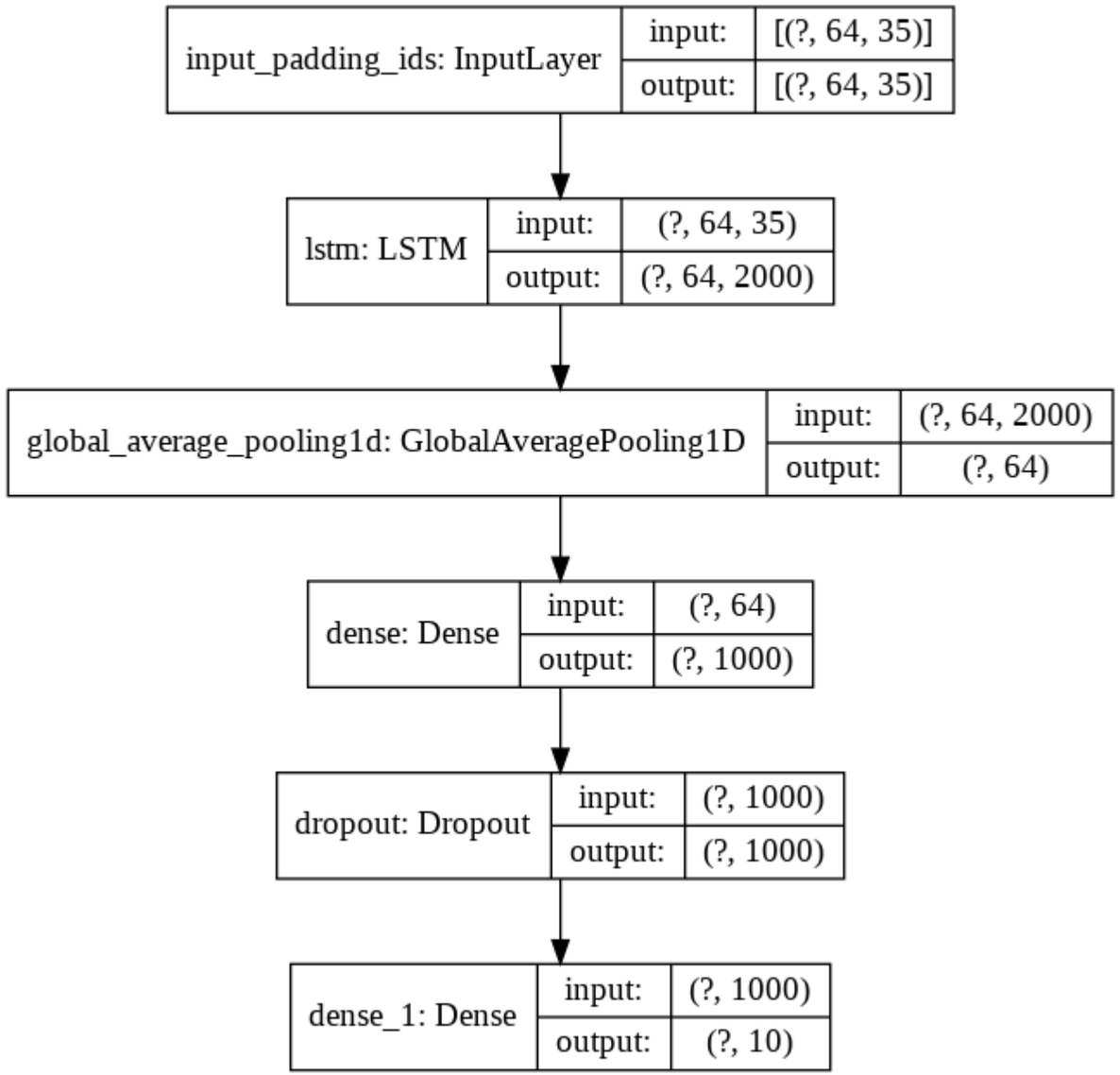
In [51]:
```
tf.keras.backend.clear_session()

## Set the random seed values to regenerate the model.
np.random.seed(0)
rn.seed(0)


## as discussed above, please write the LSTM
time_steps = 64
n_features = 35
#this is input words. Sequence of words represented as integers
input_padding = tf.keras.layers.Input(shape=(time_steps,n_features), name="input_padding_ids")
lstm  = LSTM(2000,return_sequences=True)(input_padding)
global_average  = GlobalAveragePooling1D(data_format='channels_first')(lstm)
#res = tf.reduce_mean(global_average , axis = 1, keepdims = True)
x = Dense(1000, activation = 'relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),activity_regularizer=l2(0.000000001))(g
x = Dropout(rate=0.5)(x)
output = Dense(10, activation = 'softmax')(x)

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

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_padding_ids (InputLaye [(None, 64, 35)]          0
_________________________________________________________________
lstm (LSTM)                  (None, 64, 2000)          16288000
_________________________________________________________________
global_average_pooling1d (Gl (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 1000)              65000
_________________________________________________________________
dropout (Dropout)            (None, 1000)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10010
=================================================================
Total params: 16,363,010
Trainable params: 16,363,010
```

```
Non-trainable params: 0
_________________________________________________________________
```

In [52]:
```
dot_img_file = '/tmp/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```

Out[52]:



In [53]:
```
ACCURACY_THRESHOLD = 0.8
class Metrics(tf.keras.callbacks.Callback):
  def __init__(self, validation):
    super(Metrics, self).__init__()
    self.validation_data = validation

  def on_train_begin(self, logs={}):
    self.val_f1s_score = []
    self.f1_score_best = 0
    self.epoch_value  = 1


  def on_epoch_end(self, epoch, logs={}):



    val_predict = self.model.predict(self.validation_data[0])
    val_predict = np.argmax(val_predict, axis=1)

    val_targ = self.validation_data[1]

    _val_f1 = f1_score(val_targ, val_predict,average='micro')
    self.val_f1s_score.append(_val_f1)

    print(' — val_f1: %f ' %(_val_f1))


    if  _val_f1 > self.f1_score_best:
      print('F1_score improved from '+str(self.f1_score_best ) + ' to '+str(_val_f1) +' Epoch value '+str(self.epoch_value))
      self.f1_score_best = _val_f1
    else:
      print('Model not improved, still best f1_score reamins  '+str(self.f1_score_best ))

    self.epoch_value = self.epoch_value +1
    if (_val_f1 >= ACCURACY_THRESHOLD):
      print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD*100))
      self.model.stop_training = True
      return

metrics = Metrics((X_test_spectrogram, y_test))
```

In [54]:
```
# Call back

# Learning rate scheduler
def my_learning_rate(epoch, lrate):
  if epoch <=500:
    print('Learning rate changed to 0.0001')
    lrate = 0.0001

  else:
    print('Learning rate changed to 0.00001')
    lrate = 0.00001

  return lrate

lrs = LearningRateScheduler(my_learning_rate)
red_learn = tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_accuracy",
    factor=0.1,
    patience=15,
    verbose=0,
    mode="auto",
    min_delta=0.0001,
    cooldown=0,
    min_lr=0
)
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=180, verbose=1)
filepath="/gdrive/My Drive/model_save/best_model2-{epoch:02d}.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1, save_best_only=True, mode='max')
# TensorBoard Creation
%load_ext tensorboard
folder_name = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Create log folder - TensorBoard
log_dir="/gdrive/My Drive/logs/fit/" + folder_name
tensorboard_callback =TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True)
```

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

In [55]:
```
folder_name
```

Out[55]:
```
'20201122-030151'
```

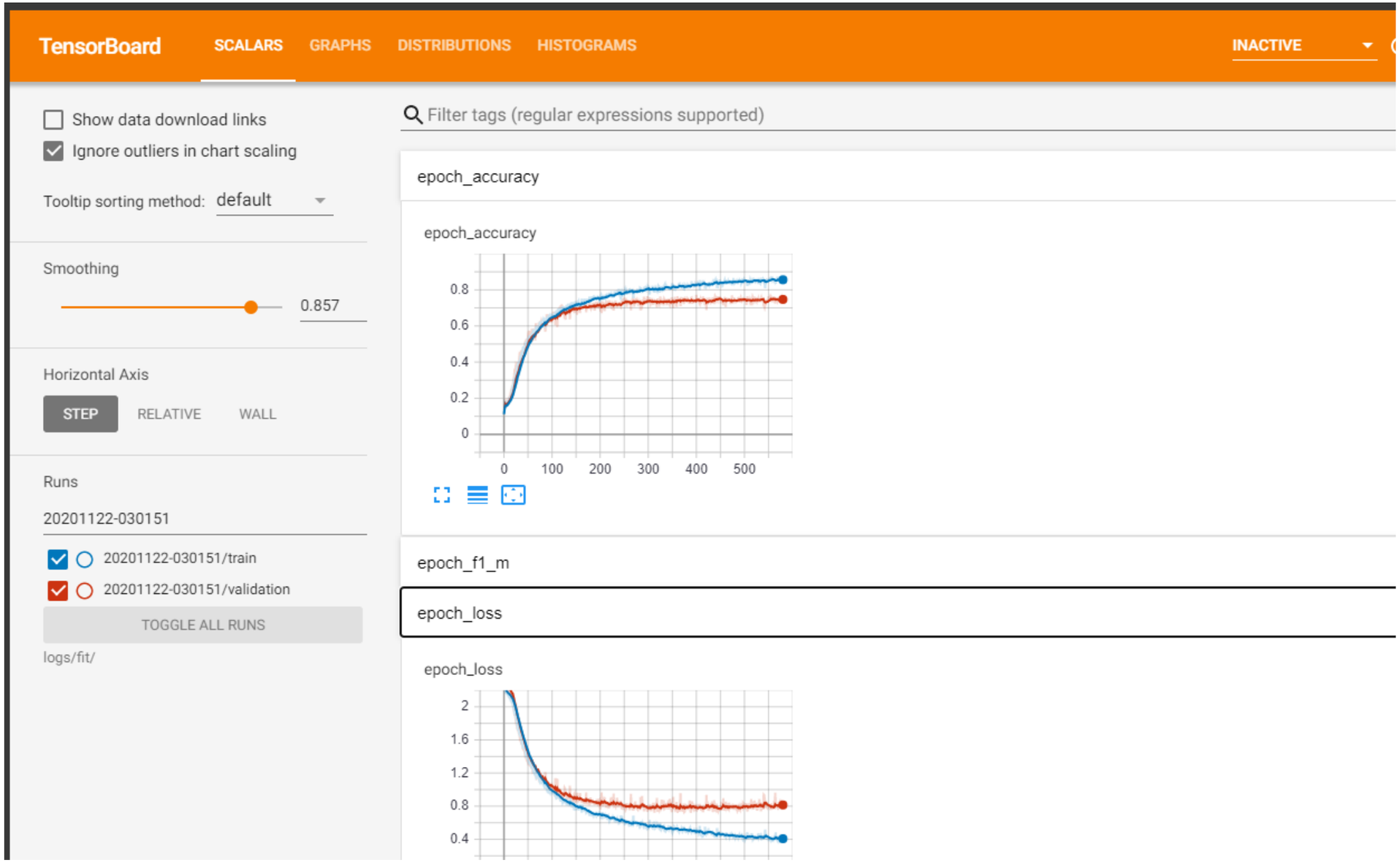
In [56]:
```
optim =  tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(loss='sparse_categorical_crossentropy', optimizer= optim,metrics=['accuracy'])
model.fit(x = X_train_spectrogram,  y=y_train, epochs=1000,verbose=1, validation_data=(X_test_spectrogram, y_test),
          callbacks =[earlystop,tensorboard_callback,metrics])
```

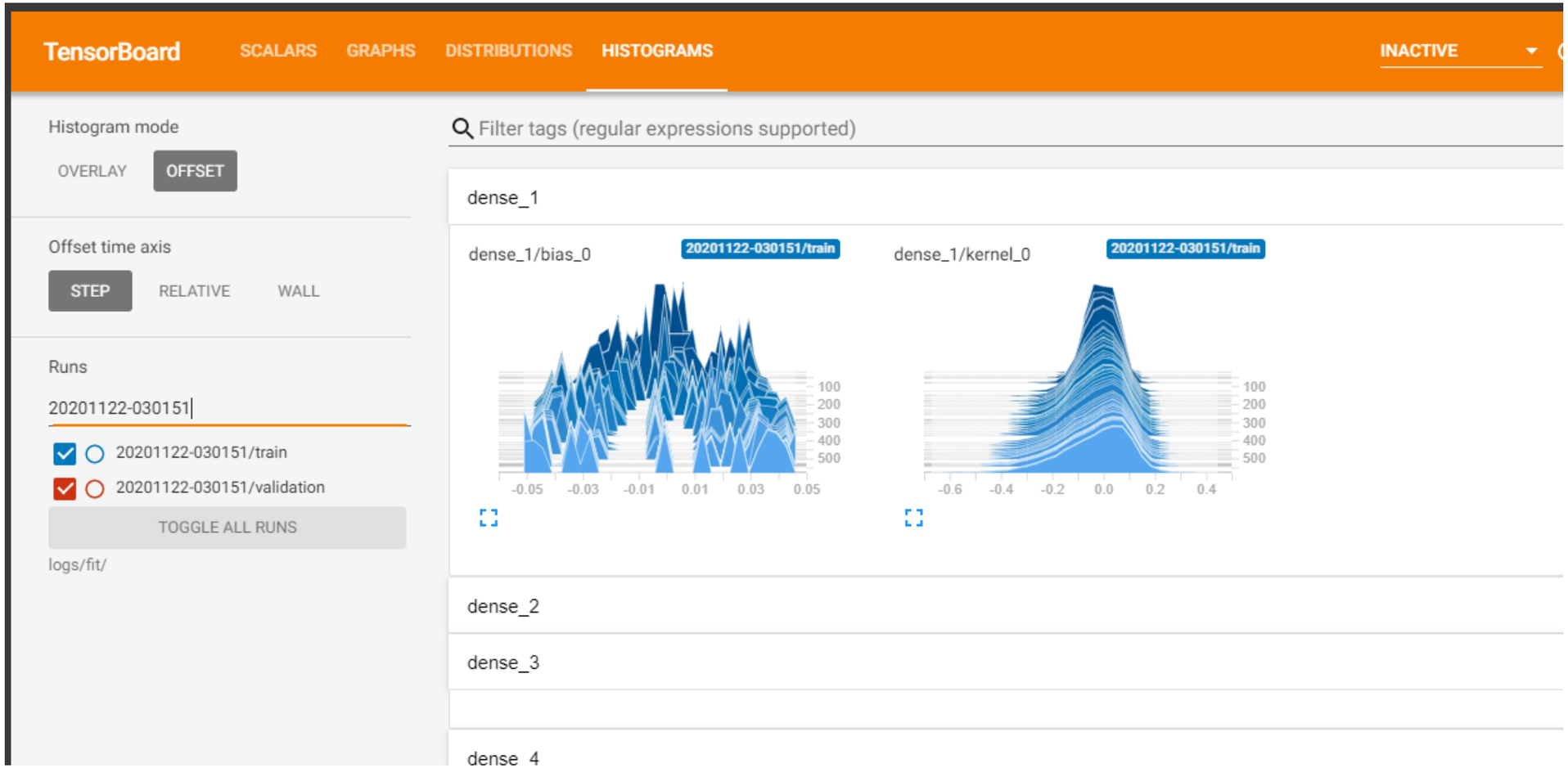
```
Epoch 578/1000
44/44 [==============================] - ETA: 0s - loss: 0.3927 - accuracy: 0.8564 — val_f1: 0.753333

Model not improved, still best f1_score reamins  0.7716666666666666
44/44 [==============================] - 6s 144ms/step - loss: 0.3927 - accuracy: 0.8564 - val_loss: 0.8013 - val_accuracy: 0.7533
Epoch 579/1000
44/44 [==============================] - ETA: 0s - loss: 0.4035 - accuracy: 0.8564 — val_f1: 0.738333
Model not improved, still best f1_score reamins  0.7716666666666666
44/44 [==============================] - 6s 144ms/step - loss: 0.4035 - accuracy: 0.8564 - val_loss: 0.7908 - val_accuracy: 0.7383
Epoch 580/1000
44/44 [==============================] - ETA: 0s - loss: 0.4312 - accuracy: 0.8514 — val_f1: 0.758333
Model not improved, still best f1_score reamins  0.7716666666666666
44/44 [==============================] - 6s 143ms/step - loss: 0.4312 - accuracy: 0.8514 - val_loss: 0.8307 - val_accuracy: 0.7583
Epoch 581/1000
44/44 [==============================] - ETA: 0s - loss: 0.3885 - accuracy: 0.8614 — val_f1: 0.751667
Model not improved, still best f1_score reamins  0.7716666666666666
44/44 [==============================] - 6s 143ms/step - loss: 0.3885 - accuracy: 0.8614 - val_loss: 0.7752 - val_accuracy: 0.7517
Epoch 00581: early stopping
```

Out[56]:
```
<tensorflow.python.keras.callbacks.History at 0x7fc32f392668>
```

In [58]:
```
#Model 1 - results
os.chdir('/gdrive/My Drive/')
%tensorboard --logdir logs/fit/
```

```
Output hidden; open in https://colab.research.google.com (https://colab.research.google.com) to view.
```


TensorBoard
SCALARS
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.857
Horizontal Axis
STEP
RELATIVE
WALL
Runs
20201122-030151
20201122-030151/train
20201122-030151/validation
TOGGLE ALL RUNS
logs/fit/
Filter tags (regular expressions supported)
epoch_accuracy
epoch_accuracy
0.8
0.6
0.4
0.2
0
0
100
200
300
400
500
epoch_f1_m
epoch_loss
epoch_loss
2
1.6
1.2
0.8
0.4



3. data augmentation

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% r slower
2. pitch shifting - pitch shifting moves the frequencies higher or lower. For pitch shifting we shift up or down one half-step.

In [59]:
```
## 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
```

In [60]:
```
temp_path = df_audio.iloc[0].path
aug_temp = generate_augmented_data(temp_path)
```

In [61]:
```
aug_temp[8]
```

Out[61]:
```
array([-0.00034318, -0.00020249, -0.00010748, ...,  0.0001497 ,
        0.0002406 ,  0.00029225], dtype=float32)
```

In [62]:
```
len(aug_temp)
```

Out[62]:
```
9
```

In [63]:
```
df_audio.iloc[0].label
```

Out[63]:
```
7
```

In [63]:
```

```

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

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

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

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

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

In [64]:
```
X = df_audio['path']
y = df_audio['label']
X_train , X_test , y_train , y_test  = train_test_split(X,y,test_size=0.2,stratify = y,random_state=45)
```

In [65]:
```
print(X_train.iloc[0])
print(y_train.iloc[0])
```

```
/gdrive/My Drive/recordings/8_jackson_35.wav
8
```

In [66]:
```
X_train.head()
```

Out[66]:
```
597      /gdrive/My Drive/recordings/8_jackson_35.wav
126      /gdrive/My Drive/recordings/2_nicolas_13.wav
1763    /gdrive/My Drive/recordings/7_yweweler_40.wav
302          /gdrive/My Drive/recordings/3_theo_2.wav
1493    /gdrive/My Drive/recordings/9_yweweler_27.wav
Name: path, dtype: object
```

In [67]:
```
X_train_spectrogram_augument  = []
y_train_augument = []
y1 = 0
for i in tqdm(range(len(X_train))):
  aaa = generate_augmented_data(X_train.iloc[i])
  for k in range(len(aaa)):
    value =  aaa[k]
    label = y_train.iloc[y1]
    X_train_spectrogram_augument.append(value)
    y_train_augument.append(label)
  y1+=1

# Augumentation only to train data
X_test_spectrogram_augument  = []
y_test_augument = []
for i in tqdm(range(len(X_test))):
  processor = load_wav(X_test.iloc[i], get_duration=False)
  X_test_spectrogram_augument.append(processor)
  y_test_augument.append(y_test.iloc[i])
```

```
100%|██████████| 1600/1600 [13:46<00:00,  1.94it/s]
100%|██████████| 400/400 [02:15<00:00,  2.95it/s]
```

In [68]:
```
new_X_train_augument = pd.DataFrame({'raw_input':X_train_spectrogram_augument , 'label':y_train_augument })
new_X_test_augument  = pd.DataFrame({'raw_input':X_test_spectrogram_augument , 'label':y_test_augument })
```

In [69]:
```
new_X_train_augument.head(2)
```

Out[69]:

	raw_input	label
0	[0.005341887, 0.0069970735, 0.007084101, 0.007...	8
1	[0.006056736, 0.0074520707, 0.007774388, 0.007...	8

In [70]:
```
new_X_train_augument.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14400 entries, 0 to 14399
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   raw_input  14400 non-null  object
 1   label      14400 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 225.1+ KB
```

In [71]:
```
new_X_test_augument.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   raw_input  400 non-null    object
 1   label      400 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 6.4+ KB
```

In [72]:
```
X_train_processed = new_X_train_augument
X_test_processed  = new_X_test_augument
```

In [73]:
```
max_length  = 17640

## as discussed above, Pad with Zero if length of sequence is less than 17640 else Truncate the number.
## save in the X_train_pad_seq, X_test_pad_seq
## also Create masking vector X_train_mask, X_test_mask

## all the X_train_pad_seq, X_test_pad_seq, X_train_mask, X_test_mask will be numpy arrays mask vector dtype must be bool.
#X_train padding Sequences
X_train_pad_seq  = []
for i in tqdm(range(len(new_X_train_augument))):
  sequences  = [new_X_train_augument['raw_input'][i]]
  aaa  = tf.keras.preprocessing.sequence.pad_sequences(
      sequences, maxlen=max_length, dtype='float', padding='post', truncating='post',
      value=100)
  X_train_pad_seq.extend(aaa)

#########################################################################################################

#X_test padding Sequences
X_test_pad_seq  = []
for i in tqdm(range(len(new_X_test_augument))):
  sequences  = [new_X_test_augument['raw_input'][i]]
  aaa  = tf.keras.preprocessing.sequence.pad_sequences(
      sequences, maxlen=max_length, dtype='float', padding='post', truncating='post',
      value=100)
  X_test_pad_seq.extend(aaa)
```

```
100%|██████████| 14400/14400 [00:01<00:00, 8781.28it/s]
100%|██████████| 400/400 [00:00<00:00, 15583.23it/s]
```

In [74]:
```
X_train_pad_seq = np.array(X_train_pad_seq)
X_test_pad_seq = np.array(X_test_pad_seq)
print('shape of X_train padding', X_train_pad_seq.shape)
print('shape of X_test padding', X_test_pad_seq.shape)
```

```
shape of X_train padding (14400, 17640)
shape of X_test padding (400, 17640)
```

In [75]:
```
# X_train_mask
X_train_mask  = []
for i in tqdm(range(len(new_X_train_augument))):
  X_train_mask_replace = np.where(X_train_pad_seq[i]!=100.0, 1, X_train_pad_seq[i])
  X_train_mask_replace = np.where(X_train_mask_replace==100.0, 0, X_train_mask_replace)
  X_train_mask.append(X_train_mask_replace)

##########################################################################################

X_test_mask  = []
for i in tqdm(range(len(new_X_test_augument))):
  X_test_mask_replace = np.where(X_test_pad_seq[i]!=100.0, 1, X_train_pad_seq[i])
  X_test_mask_replace = np.where(X_test_mask_replace==100.0, 0, X_test_mask_replace)
  X_test_mask.append(X_test_mask_replace)
```

```
100%|██████████| 14400/14400 [00:01<00:00, 9347.69it/s]
100%|██████████| 400/400 [00:00<00:00, 10266.38it/s]
```

In [76]:
```
X_train_mask[:2]
```

Out[76]:
```
[array([1., 1., 1., ..., 0., 0., 0.]), array([1., 1., 1., ..., 0., 0., 0.])]
```

In [77]:
```
X_train_mask = np.array(X_train_mask).astype('bool')
X_test_mask  = np.array(X_test_mask).astype('bool')
```

In [78]:
```
X_train_mask[:2]
```

Out[78]:
```
array([[ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False]])
```

In [79]:
```
from tensorflow.keras.layers import Input, LSTM, Dense,Masking,Dropout
from tensorflow.keras.models import Model
import tensorflow as tf
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
from keras.regularizers import l2,l1_l2,l1
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau, TensorBoard,LearningRat
import datetime
from tensorflow.keras.layers import  AveragePooling1D,GlobalAveragePooling1D,MaxPooling1D,TimeDistributed
import random as rn
```

In [ ]:
```
## as discussed above, please write the LSTM
time_steps = 17640
n_features = 1
#this is input words. Sequence of words represented as integers
input_padding = tf.keras.layers.Input(shape=(time_steps,n_features), name="input_padding_ids")

#mask vector if you are padding anything
input_mask = tf.keras.layers.Input(shape=(time_steps), name="input_Masking")


lstm  = LSTM(128)(input_padding)
x = Dense(64, activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),kernel_regularizer=l2(0.0001))(lstm)
x = Dropout(0.2)(x)
output = Dense(10, activation = 'softmax')(x)

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

```
Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_padding_ids (InputLaye [(None, 17640, 1)]        0
_________________________________________________________________
lstm_5 (LSTM)                (None, 128)               66560
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 75,466
Trainable params: 75,466
Non-trainable params: 0
_________________________________________________________________
```

In [85]:
```
y_train_fit = new_X_train_augument['label']
y_test_fit = new_X_test_augument['label']
```

In [ ]:

```
ACCURACY_THRESHOLD = 0.1
class Metrics(tf.keras.callbacks.Callback):
  def __init__(self, validation):
    super(Metrics, self).__init__()
    self.validation_data = validation

  def on_train_begin(self, logs={}):
    self.val_f1s = []


  def on_epoch_end(self, epoch, logs={}):

    val_predict = self.model.predict(self.validation_data[0])
    val_predict = np.argmax(val_predict, axis=1)

    val_targ = self.validation_data[1]

    _val_f1 = f1_score(val_targ, val_predict,average='micro')
    self.val_f1s.append(_val_f1)

    print(' — val_f1: %f ' %(_val_f1))

    if (_val_f1 > ACCURACY_THRESHOLD):
      print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD*100))
      self.model.stop_training = True


metrics = Metrics((X_test_pad_seq, y_test_fit))
```

In [ ]:
```
# Call back
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=10, verbose=1)
filepath="/gdrive/My Drive/model_save/best_model-{epoch:02d}.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1, save_best_only=True, mode='auto')
# TensorBoard Creation
%load_ext tensorboard
folder_name = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Create log folder - TensorBoard
log_dir="/gdrive/My Drive/logs/fit/" + folder_name
tensorboard_callback =TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True)
```

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

In [ ]:
```
folder_name
```

Out[63]:
```
'20201120-160801'
```

In [ ]:
```

```

In [ ]:
```
model.compile(loss='sparse_categorical_crossentropy', optimizer= 'adam', metrics=['accuracy'])
model.fit(x = X_train_pad_seq,  y=y_train_fit, epochs=40,verbose=1, validation_data=(X_test_pad_seq,y_test_fit ),
          callbacks =[checkpoint,tensorboard_callback,earlystop,metrics])
```

```
Epoch 1/40
  1/450 [..............................] - ETA: 0s - loss: 2.4686 - accuracy: 0.0938WARNING:tensorflow:From /usr/local/lib/python3.6/dis
ensorflow/python/ops/summary_ops_v2.py:1277: stop (from tensorflow.python.eager.profiler) is deprecated and will be removed after 2020-(
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
  2/450 [..............................] - ETA: 14:12 - loss: 2.4677 - accuracy: 0.0625WARNING:tensorflow:Callbacks method `on_train_bat
low compared to the batch time (batch time: 1.1473s vs `on_train_batch_end` time: 2.6599s). Check your callbacks.
450/450 [==============================] - ETA: 0s - loss: 2.3075 - accuracy: 0.1081
Epoch 00001: val_accuracy improved from -inf to 0.10000, saving model to /gdrive/My Drive/model_save/best_model-01.h5
 — val_f1: 0.100000

Reached 10.00% accuracy, so stopping training!!
450/450 [==============================] - 337s 750ms/step - loss: 2.3075 - accuracy: 0.1081 - val_loss: 2.3090 - val_accuracy: 0.1000
```

Out[64]:
```
<tensorflow.python.keras.callbacks.History at 0x7f788a72da90>
```

MODEL - 4

In [80]:
```
##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)
##Train data: X_train_pad_seq, X_train_mask and y_train
#Test data: X_test_pad_seq, X_test_mask and y_test


X_train_spectrogram  = []
for i in tqdm(range(len(X_train_pad_seq))):
  aaa = convert_to_spectrogram(X_train_pad_seq[i])
  X_train_spectrogram.append(aaa)


#########################################################################################################

#X_test padding Sequences
X_test_spectrogram  = []
for i in tqdm(range(len(X_test_pad_seq))):
  bbb = convert_to_spectrogram(X_test_pad_seq[i])
  X_test_spectrogram.append(bbb)
```

```
100%|██████████| 14400/14400 [01:29<00:00, 160.27it/s]
100%|██████████| 400/400 [00:03<00:00, 127.82it/s]
```

In [81]:
```
X_train_spectrogram = np.array(X_train_spectrogram)
X_test_spectrogram = np.array(X_test_spectrogram)
print('shape of X_train spectrogram', X_train_spectrogram.shape)
print('shape of X_test spectrogram', X_test_spectrogram.shape)
```

```
shape of X_train spectrogram (14400, 64, 35)
shape of X_test spectrogram (400, 64, 35)
```

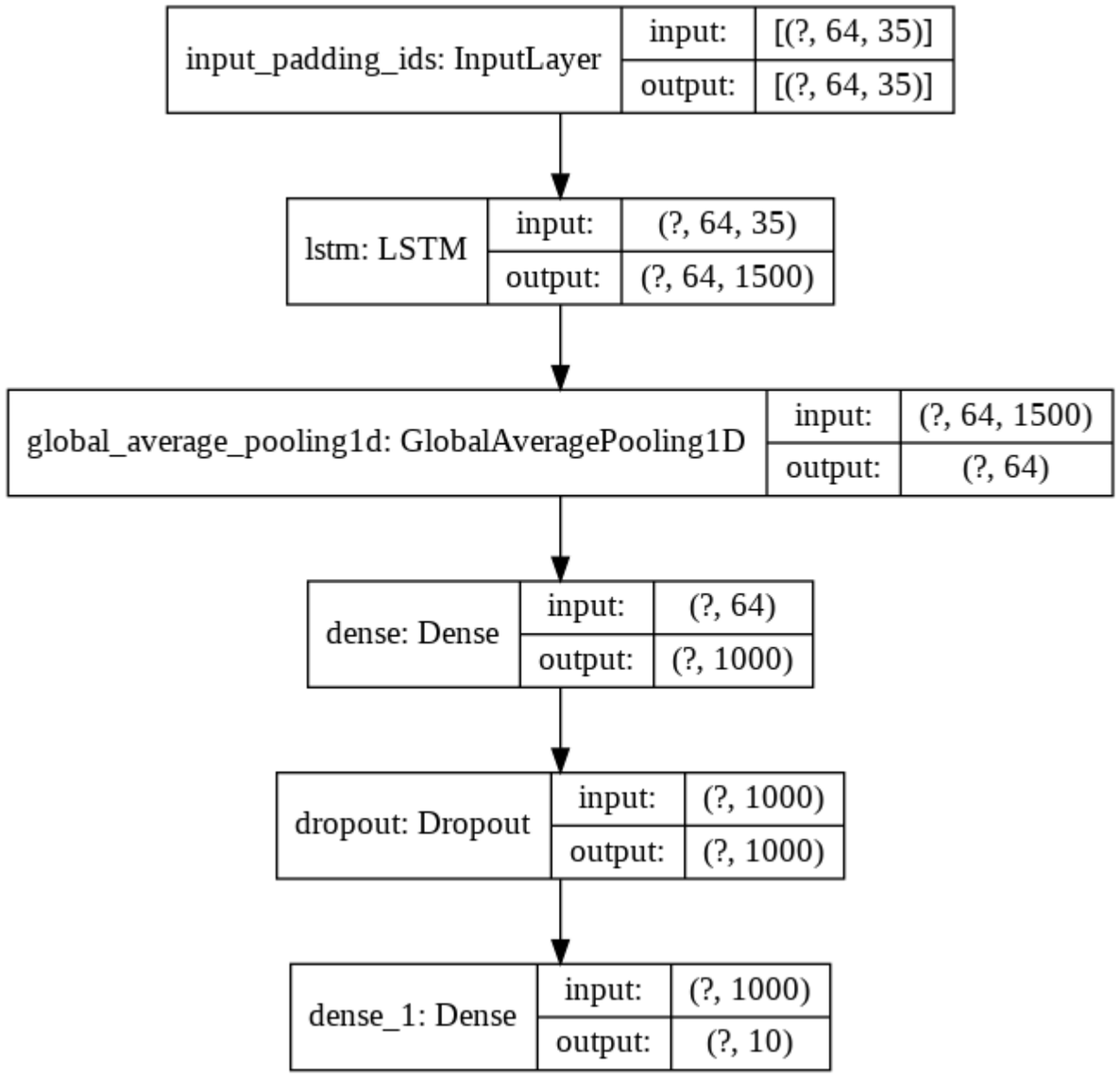
In [82]:
```
tf.keras.backend.clear_session()

## Set the random seed values to regenerate the model.
np.random.seed(0)
rn.seed(0)


## as discussed above, please write the LSTM
time_steps = 64
n_features = 35
#this is input words. Sequence of words represented as integers
input_padding = tf.keras.layers.Input(shape=(time_steps,n_features), name="input_padding_ids")
lstm  = LSTM(1500,return_sequences=True)(input_padding)
global_average  = GlobalAveragePooling1D(data_format='channels_first')(lstm)
#res = tf.reduce_mean(global_average , axis = 1, keepdims = True)
x = Dense(1000, activation = 'relu',kernel_initializer=tf.keras.initializers.he_normal(seed=45),activity_regularizer=l2(0.00001))(globa
x = Dropout(rate=0.4)(x)
output = Dense(10, activation = 'softmax')(x)

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

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_padding_ids (InputLaye [(None, 64, 35)]          0
_________________________________________________________________
lstm (LSTM)                  (None, 64, 1500)          9216000
_________________________________________________________________
global_average_pooling1d (Gl (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 1000)              65000
_________________________________________________________________
dropout (Dropout)            (None, 1000)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10010
=================================================================
Total params: 9,291,010
Trainable params: 9,291,010
```

```
Non-trainable params: 0
_________________________________________________________________
```

In [83]:
```
dot_img_file = '/tmp/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```

Out[83]:



In [86]:
```
ACCURACY_THRESHOLD = 0.8
class Metrics(tf.keras.callbacks.Callback):
  def __init__(self, validation):
    super(Metrics, self).__init__()
    self.validation_data = validation

  def on_train_begin(self, logs={}):
    self.val_f1s_score = []
    self.f1_score_best = 0
    self.epoch_value  = 1


  def on_epoch_end(self, epoch, logs={}):



    val_predict = self.model.predict(self.validation_data[0])
    val_predict = np.argmax(val_predict, axis=1)

    val_targ = self.validation_data[1]

    _val_f1 = f1_score(val_targ, val_predict,average='micro')
    self.val_f1s_score.append(_val_f1)

    print(' — val_f1: %f ' %(_val_f1))


    if  _val_f1 > self.f1_score_best:
      print('F1_score improved from '+str(self.f1_score_best ) + ' to '+str(_val_f1) +' Epoch value '+str(self.epoch_value))
      self.f1_score_best = _val_f1
    else:
      print('Model not improved, still best f1_score reamins  '+str(self.f1_score_best ))

    self.epoch_value = self.epoch_value +1
    if (_val_f1 >= ACCURACY_THRESHOLD):
      print("\nReached %2.2f%% accuracy, so stopping training!!" %(ACCURACY_THRESHOLD*100))
      self.model.stop_training = True
      return

metrics = Metrics((X_test_spectrogram, y_test_fit))
```

In [87]:
```
# Call back

# Learning rate scheduler
def my_learning_rate(epoch, lrate):
  if epoch <=500:
    print('Learning rate changed to 0.0001')
    lrate = 0.0001

  else:
    print('Learning rate changed to 0.00001')
    lrate = 0.00001

  return lrate

lrs = LearningRateScheduler(my_learning_rate)
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=150, verbose=1)
filepath="/gdrive/My Drive/model_save/best_model2-{epoch:02d}.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1, save_best_only=True, mode='max')
# TensorBoard Creation
%load_ext tensorboard
folder_name = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# Create log folder - TensorBoard
log_dir="/gdrive/My Drive/logs/fit/" + folder_name
tensorboard_callback =TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True)
```

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

In [88]:
```
folder_name
```

Out[88]:
```
'20201122-044011'
```

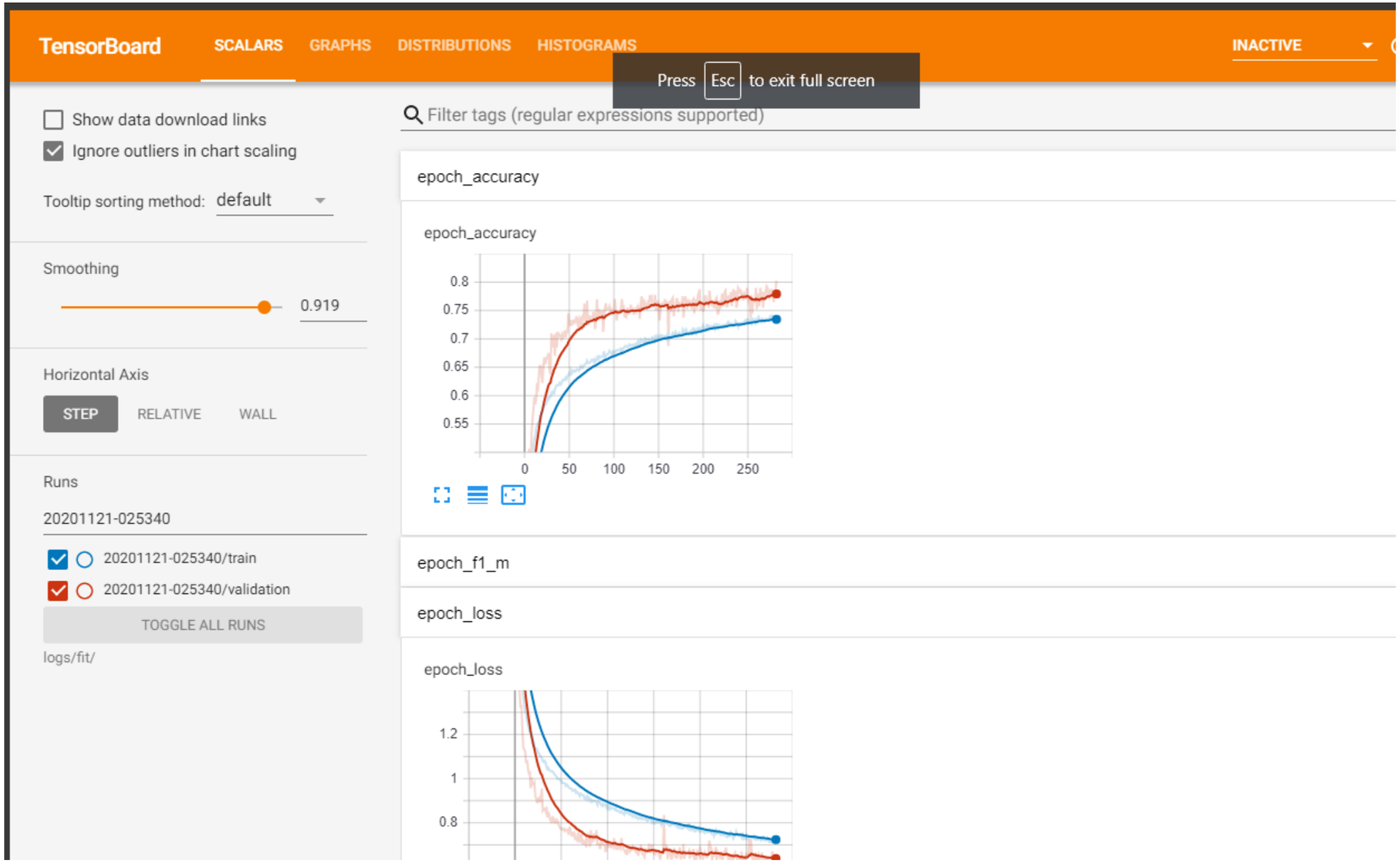
In [ ]:
```
optim =  tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(loss='sparse_categorical_crossentropy', optimizer= optim,metrics=['accuracy'])
model.fit(x = X_train_spectrogram,  y=y_train_fit, epochs=600,verbose=1, validation_data=(X_test_spectrogram, y_test_fit),
          callbacks =[earlystop,tensorboard_callback,metrics])
```

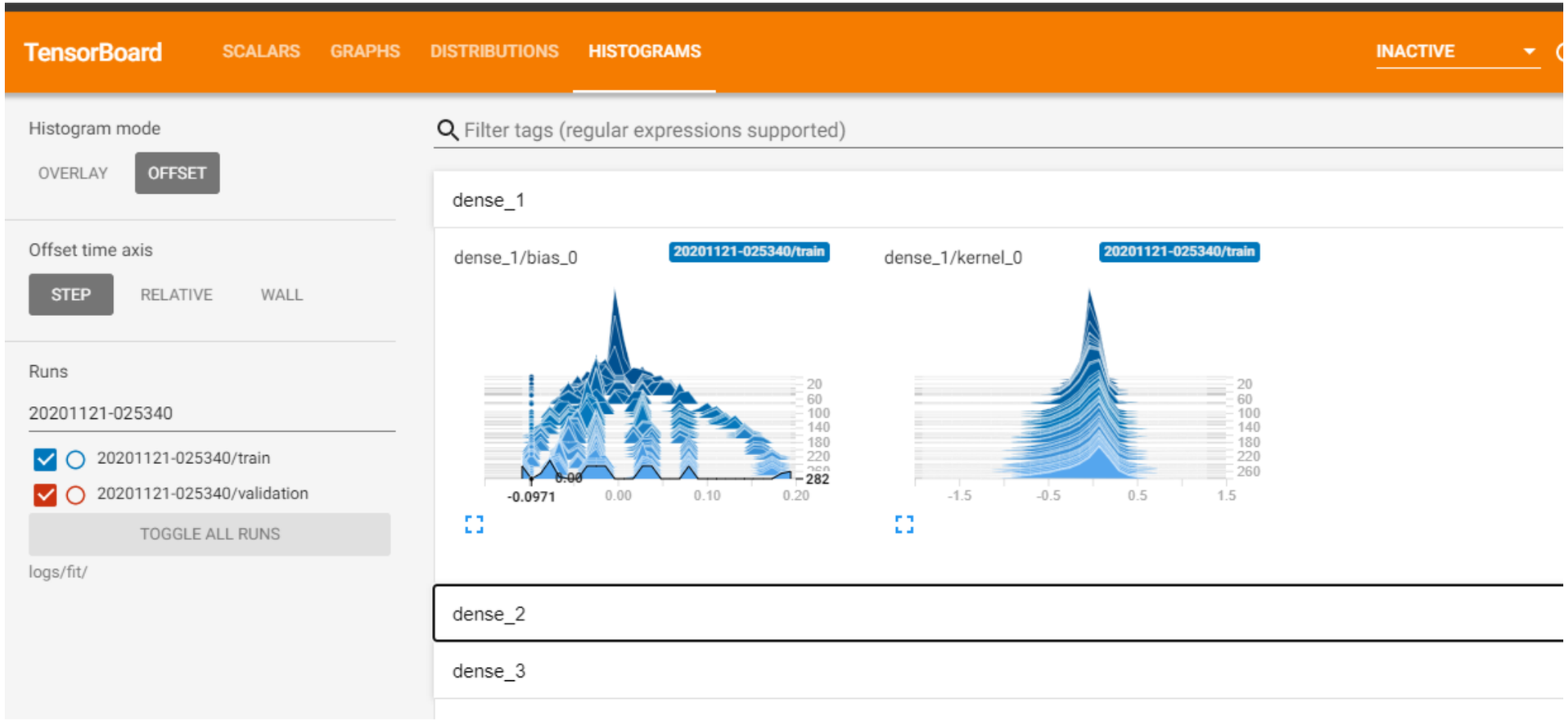
```
Epoch 280/600
449/450 [============================>.] - ETA: 0s - loss: 0.7209 - accuracy: 0.7364 — val_f1: 0.765000
Model not improved, still best f1_score reamins  0.7925
450/450 [==============================] - 15s 34ms/step - loss: 0.7209 - accuracy: 0.7362 - val_loss: 0.6548 - val_accuracy: 0.7650
Epoch 281/600
449/450 [============================>.] - ETA: 0s - loss: 0.7297 - accuracy: 0.7296 — val_f1: 0.782500
Model not improved, still best f1_score reamins  0.7925
450/450 [==============================] - 15s 34ms/step - loss: 0.7300 - accuracy: 0.7295 - val_loss: 0.6225 - val_accuracy: 0.7825
Epoch 282/600
449/450 [============================>.] - ETA: 0s - loss: 0.7163 - accuracy: 0.7385 — val_f1: 0.785000
Model not improved, still best f1_score reamins  0.7925
450/450 [==============================] - 15s 34ms/step - loss: 0.7160 - accuracy: 0.7386 - val_loss: 0.6189 - val_accuracy: 0.7850
Epoch 283/600
449/450 [============================>.] - ETA: 0s - loss: 0.7232 - accuracy: 0.7327 — val_f1: 0.802500
F1_score improved from 0.7925 to 0.8025 Epoch value 283

Reached 80.00% accuracy, so stopping training!!
450/450 [==============================] - 15s 34ms/step - loss: 0.7227 - accuracy: 0.7330 - val_loss: 0.6045 - val_accuracy: 0.8025
```

Out[47]:
```
<tensorflow.python.keras.callbacks.History at 0x7f2ce05d45c0>
```

In [ ]:
```
#Model 1 - results
os.chdir('/gdrive/My Drive/')
%tensorboard --logdir logs/fit/
```

```
Output hidden; open in https://colab.research.google.com (https://colab.research.google.com) to view.
```


TensorBoard
SCALARS
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
INACTIVE
Press Esc to exit full screen
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.919
Horizontal Axis
STEP
RELATIVE
WALL
Runs
20201121-025340
20201121-025340/train
20201121-025340/validation
TOGGLE ALL RUNS
logs/fit/
Filter tags (regular expressions supported)
epoch_accuracy
epoch_accuracy
0.8
0.75
0.7
0.65
0.6
0.55
0
50
100
150
200
250
epoch_f1_m
epoch_loss
epoch_loss
1.2
1
0.8


TensorBoard
SCALARS
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
INACTIVE
Histogram mode
OVERLAY
OFFSET
Offset time axis
STEP
RELATIVE
WALL
Runs
20201121-025340
20201121-025340/train
20201121-025340/validation
TOGGLE ALL RUNS
logs/fit/
Filter tags (regular expressions supported)
dense_1
dense_1/bias_0
20201121-025340/train
dense_1/kernel_0
20201121-025340/train
-0.0971
0.00
0.10
0.20
-1.5
-0.5
0.5
1.5
dense_2
dense_3

In [ ]: