Audio classification

Objective

- To classify give a audio whether it is emitted by nornal fan or by a abnormal fan.

Loading the data using curl wget extension

```
!wget --header="Host: zenodo.org" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
```

```
--2021-04-09 03:25:15--  https://zenodo.org/record/3384388/files/-6_dB_fan.zip?download=
Resolving zenodo.org (zenodo.org)... 137.138.76.77
Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10878096548 (10G) [application/octet-stream]
Saving to: '-6_dB_fan.zip'

-6_dB_fan.zip       100%[===================>]  10.13G  7.14MB/s    in 14m 17s

2021-04-09 03:39:33 (12.1 MB/s) - '-6_dB_fan.zip' saved [10878096548/10878096548]
```

```
#unzip the loaded data
!unzip -qn "/content/-6_dB_fan.zip"
```

Importing required libraries and packages

```
import numpy as np
import pandas as pd
import os
import datetime
import IPython.display as ipd
import librosa
import librosa.display
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
```

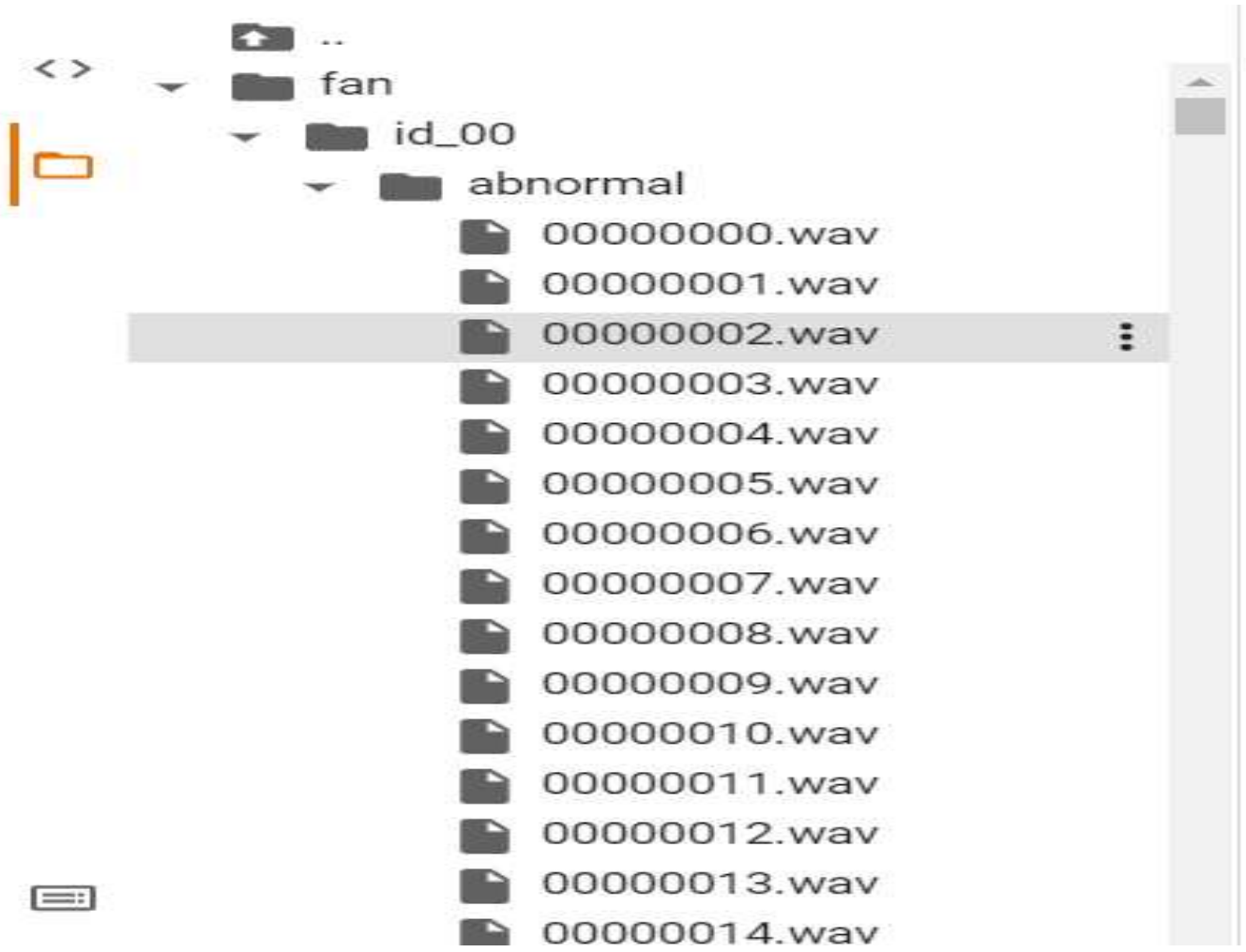
```
from tensorflow.keras import layers
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ModelCheckpoint ,TensorBoard,EarlyStopping,LearningRat
from keras.preprocessing import sequence
from tensorflow.keras.layers import concatenate
from tensorflow.keras.layers import Input, LSTM, Dense,GlobalAveragePooling1D,GlobalAveragePo
from tensorflow.keras.models import Model
import tensorflow as tf
```

Eploratory Data Analysis

- This is MIMII Dataset available online https://zenodo.org/record/3384388#. YG7E0egzZPZ
- Dataset consists of audio file of sound emiteted by fan and having two classes , normal fan audio file abnormal fan audio file
- Due to colab restiction I have taken only id_00 file data for the model which consists of 1418 samples.

```
from IPython.display import Image
Image('/content/Screenshot (115).png',width=800,height=500)
```

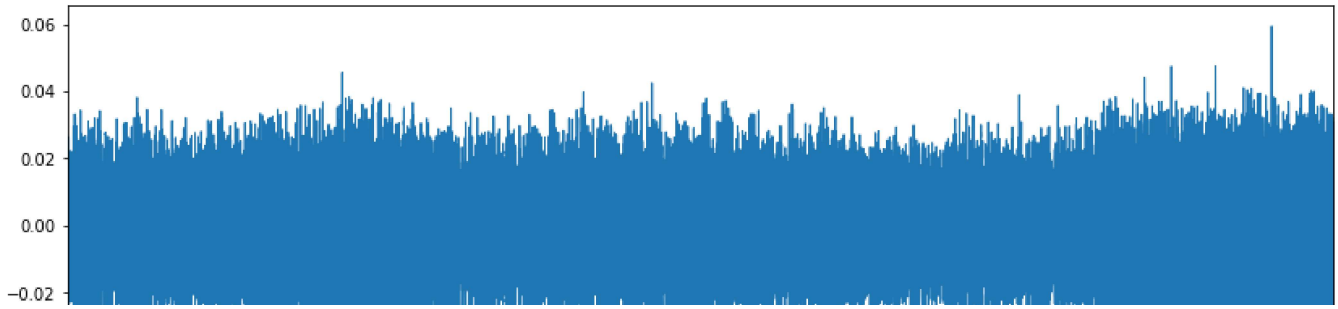
```
from IPython.display import Image
Image('/content/Screenshot (114).png',width=800,height=500)
```




```
###
```

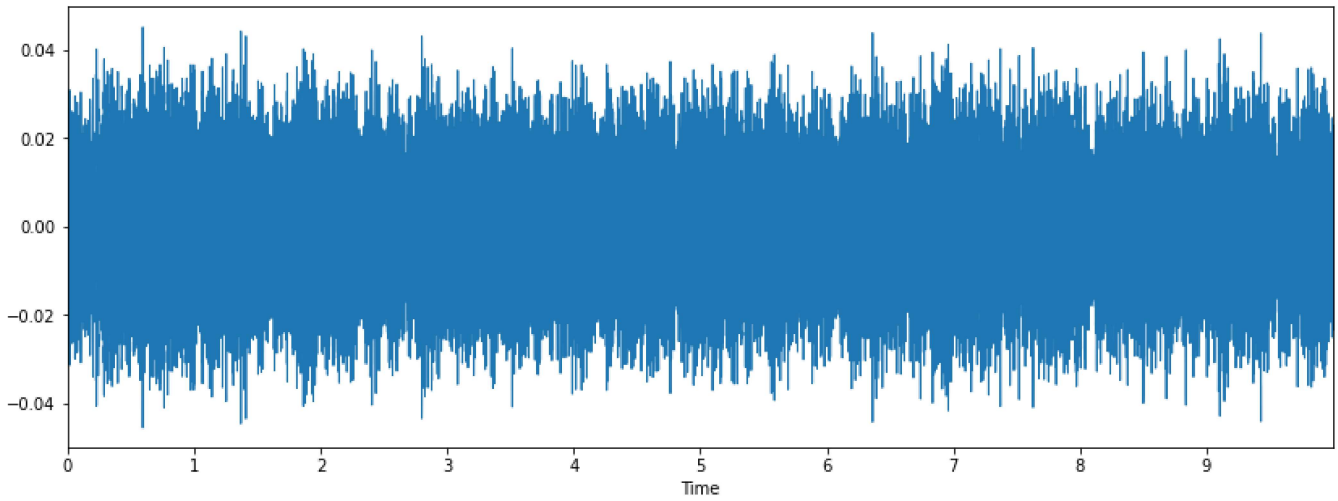
```
###abnormal Sound
plt.figure(figsize=(14,5))
data,sample_rate=librosa.load("/content/fan/id_00/abnormal/00000002.wav")
librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio("/content/fan/id_00/abnormal/00000002.wav")
```

0:00 / 0:10



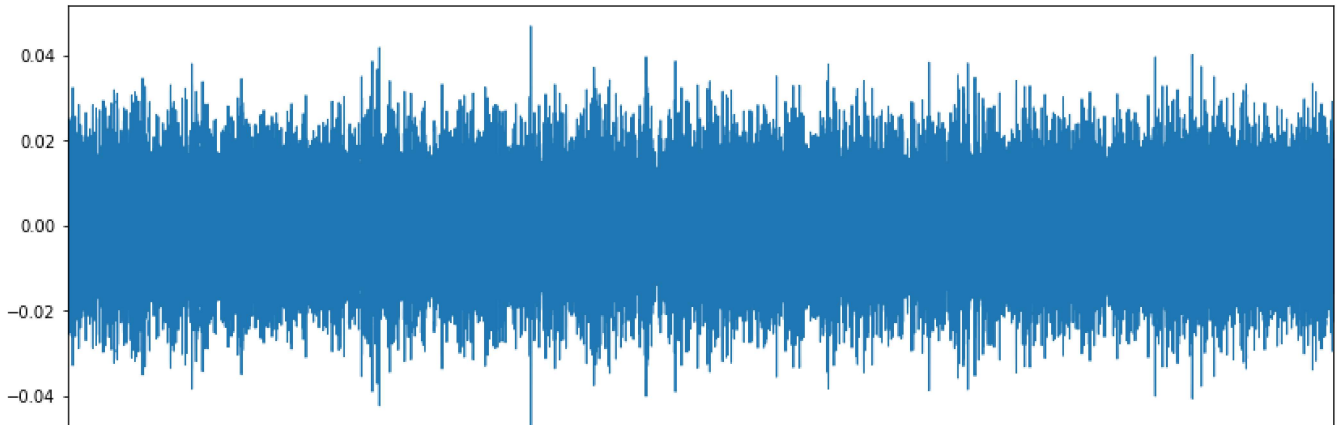

```
###abnormal Sound
plt.figure(figsize=(14,5))
data,sample_rate=librosa.load("/content/fan/id_00/abnormal/00000000.wav")
librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio("/content/fan/id_00/abnormal/00000000.wav")
```

0:00 / 0:10



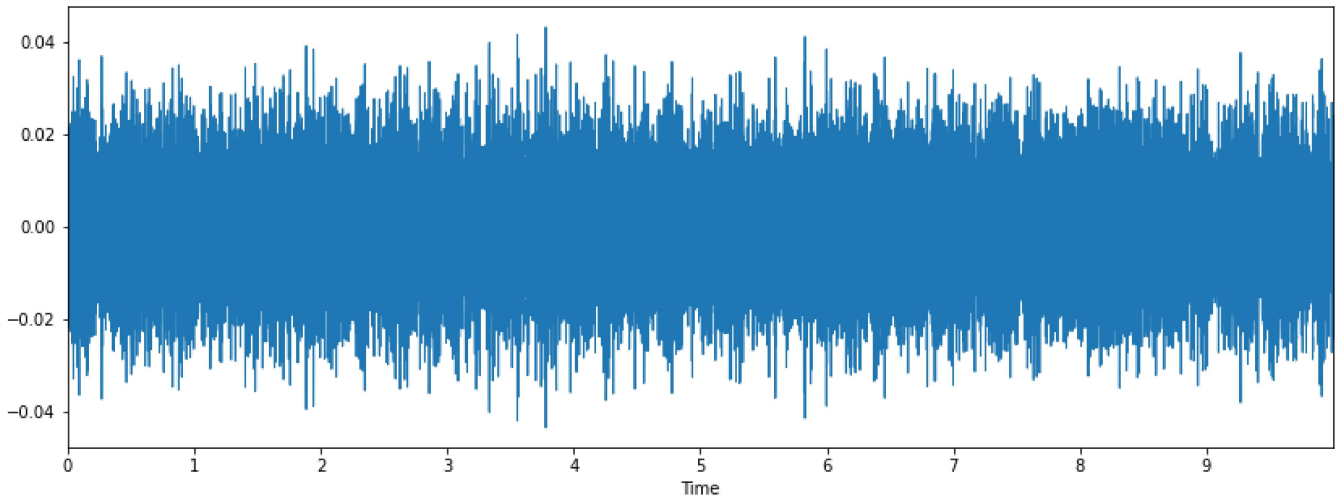

```
###normal Sound
plt.figure(figsize=(14,5))
data,sample_rate=librosa.load("/content/fan/id_00/normal/00000000.wav")
librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio("/content/fan/id_00/normal/00000000.wav")
```

0:00 / 0:10




```
###normal Sound
plt.figure(figsize=(14,5))
data,sample_rate=librosa.load("/content/fan/id_00/normal/00000002.wav")
librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio("/content/fan/id_00/normal/00000002.wav")
```

0:00 / 0:10



Reading files from the folder and labeling data

- abnormal as 0
- normal as 1

```
import os
files_abnormal=os.listdir('/content/fan/id_00/abnormal')
```

```
files_normal  = os.listdir('/content/fan/id_00/normal')
all_files=[]
all_leval =[]
for f in files_abnormal:
  name=str("/content/fan/id_00/abnormal/"+f)
  all_files.append(name)
  all_leval.append(0)
for f in files_normal:
  name=str("/content/fan/id_00/normal/"+f)
  all_files.append(name)
  all_leval.append(1)



all_files
```

```
['/content/fan/id_00/abnormal/00000382.wav',
 '/content/fan/id_00/abnormal/00000395.wav',
 '/content/fan/id_00/abnormal/00000125.wav',
 '/content/fan/id_00/abnormal/00000253.wav',
 '/content/fan/id_00/abnormal/00000005.wav',
 '/content/fan/id_00/abnormal/00000134.wav',
 '/content/fan/id_00/abnormal/00000310.wav',
 '/content/fan/id_00/abnormal/00000088.wav',
 '/content/fan/id_00/abnormal/00000258.wav',
 '/content/fan/id_00/abnormal/00000296.wav',
 '/content/fan/id_00/abnormal/00000299.wav',
 '/content/fan/id_00/abnormal/00000124.wav',
 '/content/fan/id_00/abnormal/00000146.wav',
 '/content/fan/id_00/abnormal/00000044.wav',
 '/content/fan/id_00/abnormal/00000179.wav',
 '/content/fan/id_00/abnormal/00000229.wav',
 '/content/fan/id_00/abnormal/00000351.wav',
 '/content/fan/id_00/abnormal/00000243.wav',
 '/content/fan/id_00/abnormal/00000079.wav',
 '/content/fan/id_00/abnormal/00000050.wav',
 '/content/fan/id_00/abnormal/00000155.wav',
 '/content/fan/id_00/abnormal/00000374.wav',
 '/content/fan/id_00/abnormal/00000370.wav',
 '/content/fan/id_00/abnormal/00000001.wav',
 '/content/fan/id_00/abnormal/00000341.wav',
 '/content/fan/id_00/abnormal/00000147.wav',
 '/content/fan/id_00/abnormal/00000169.wav',
 '/content/fan/id_00/abnormal/00000305.wav',
 '/content/fan/id_00/abnormal/00000013.wav',
 '/content/fan/id_00/abnormal/00000007.wav',
 '/content/fan/id_00/abnormal/00000020.wav',
 '/content/fan/id_00/abnormal/00000202.wav',
 '/content/fan/id_00/abnormal/00000111.wav',
 '/content/fan/id_00/abnormal/00000118.wav',
 '/content/fan/id_00/abnormal/00000136.wav',
 '/content/fan/id_00/abnormal/00000392.wav',
 '/content/fan/id_00/abnormal/00000110.wav',
 '/content/fan/id_00/abnormal/00000159.wav',
 '/content/fan/id_00/abnormal/00000183.wav',
 '/content/fan/id_00/abnormal/00000313.wav',
```

```
 '/content/fan/id_00/abnormal/00000161.wav',
 '/content/fan/id_00/abnormal/00000357.wav',
 '/content/fan/id_00/abnormal/00000163.wav',
 '/content/fan/id_00/abnormal/00000231.wav',
 '/content/fan/id_00/abnormal/00000166.wav',
 '/content/fan/id_00/abnormal/00000115.wav',
 '/content/fan/id_00/abnormal/00000074.wav',
 '/content/fan/id_00/abnormal/00000090.wav',
 '/content/fan/id_00/abnormal/00000277.wav',
 '/content/fan/id_00/abnormal/00000314.wav',
 '/content/fan/id_00/abnormal/00000170.wav',
 '/content/fan/id_00/abnormal/00000078.wav',
 '/content/fan/id_00/abnormal/00000138.wav',
 '/content/fan/id_00/abnormal/00000368.wav',
 '/content/fan/id_00/abnormal/00000057.wav',
 '/content/fan/id_00/abnormal/00000039.wav',
 '/content/fan/id_00/abnormal/00000280.wav',
 '/content/fan/id_00/abnormal/00000282.wav',
 '/content/fan/id_00/abnormal/00000190.wav',
```

```
len(all_files)
```

```
1418
```

Creating dataFrame

```
import pandas as pd
df_audio=pd.DataFrame()
df_audio['path']=all_files
df_audio['label']=all_leval
```

```
df_audio.label.value_counts()
```

```
1    1011
0     407
Name: label, dtype: int64
```

Sampling data

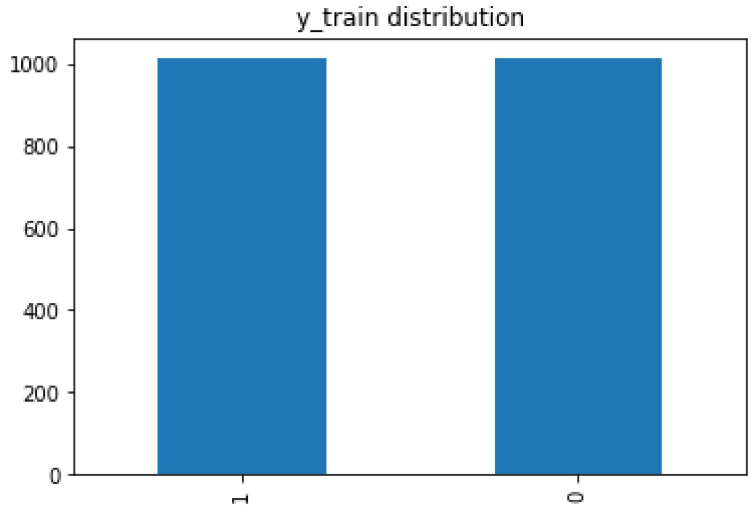
- dataset is imbalance for balancing it , we are doing up sampling

```
from sklearn.utils import resample
# Separate majority and minority classes
df_majority = df_audio[df_audio['label']==1]
df_minority = df_audio[df_audio['label']==0]

# Upsample minority class
df_minority_upsampled = resample(df_minority
```

```
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=1011,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df_audio_new = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
df_audio_new.label.value_counts()
```

```
1    1011
0    1011
Name: label, dtype: int64
```

```
import matplotlib.pyplot as plt
ax1 = df_audio_new['label'].value_counts().plot(kind='bar')
plt.title("y_train distribution")
plt.show()
```




```
#info
df_audio_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2022 entries, 407 to 292
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   path    2022 non-null   object
 1   label   2022 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 47.4+ KB
```

Shuffling Data

```
from sklearn.utils import shuffle
df_audio_new = shuffle(df_audio_new, 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 20%
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_audio_new['path'],df_audio_new['label'
```

```
y_test
```

```
599     1
1060    1
309     0
160     0
233     0
       ..
231     0
324     0
1409    1
957     1
832     1
Name: label, Length: 405, dtype: int64
```

Data Preprocessing

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

```
#https://librosa.org/doc/main/generated/librosa.load.html
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
```

```
sample_rate
```

```
22050
```

```
#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(X_train.values[i])[0])
  a2.append(load_wav(X_train.values[i])[1])

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

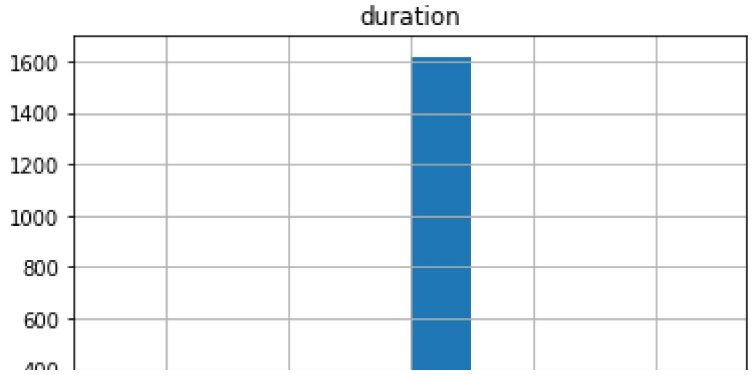
```
100%|██████████| 1617/1617 [10:17<00:00,  2.62it/s]
100%|██████████| 405/405 [02:33<00:00,  2.63it/s]
```

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

```
1617
1617
405
405
```

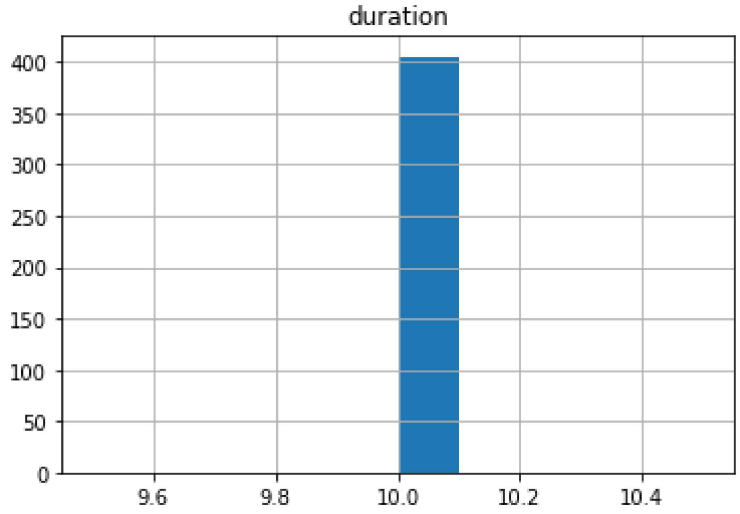
```
X_train_processed=pd.DataFrame()
X_test_processed=pd.DataFrame()
X_train_processed['raw_data']=a1
X_train_processed['duration']=a2
X_test_processed['raw_data']=b1
X_test_processed['duration']=b2


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

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f80dd3cb750>]],
      dtype=object)
```




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

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f80dd3866d0>]],
      dtype=object)
```



- As each sample is having 10 sec duration. While loading the audio files, we are using sampling rate of 22050 so one sec will give array of length 22050. so, our maximum length sequence should be 10*22050 = 220500
- But length of sequence will be too large that's why we are taking only 5*22050 = 110250

```
max_seq_length=22050*5
X_train_pad_seq=[]
X_train_mask= []
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%|██████████| 1617/1617 [00:45<00:00, 35.34it/s]
```

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

```
X_train_pad_seq.shape
```

```
(1617, 110250)
```

```
X_test_processed.shape
```

```
(405, 2)
```

```
max_seq_length=22050*5
X_test_pad_seq=[]
X_test_mask=[]
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%|██████████| 405/405 [00:11<00:00, 36.56it/s]
```

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

```
X_test_pad_seq[0]
```

```
array([0.00105738, 0.00105972, 0.00041713, ..., 0.00471232, 0.00636652,
       0.00749081])
```

```
X_test_mask[0]
```

```
X_test_mask[0]
```

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

1. Giving Raw data directly.

```
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., 1.], dtype=float32)
```

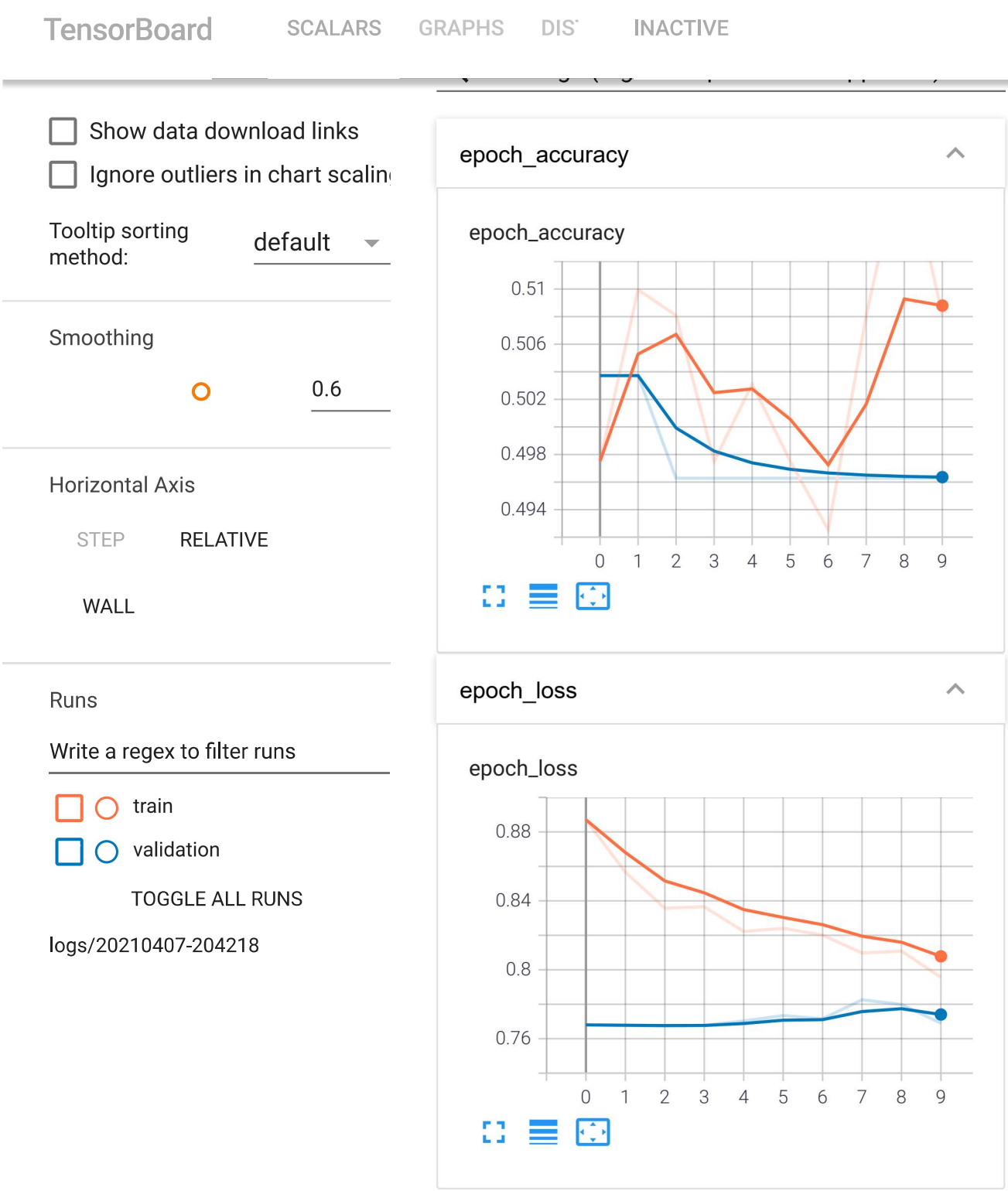
Model1

- Bulding Model With LSTM

```
#https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
input1 = Input(shape=(44100, 1,))
input_mask=Input(shape=(44100,),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=2,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_norm
model1= Model(inputs=(input1,input_mask),outputs=Out)
```

```
%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)
```

```
model1.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=["accuracy"])
```

```
model1.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 44100, 1)]   0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 44100)]      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, 2)            22          dense_3[0][0]
==================================================================================================
Total params: 11,780
Trainable params: 11,708
Non-trainable params: 72
__________________________________________________________________________________________________
```

```
#trainig 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
51/51 [==============================] - 111s 1s/step - loss: 0.9171 - accuracy: 0.4819
Epoch 2/10
51/51 [==============================] - 65s 1s/step - loss: 0.8692 - accuracy: 0.5135 -
Epoch 3/10
51/51 [==============================] - 65s 1s/step - loss: 0.8392 - accuracy: 0.4994 -
Epoch 4/10
51/51 [==============================] - 65s 1s/step - loss: 0.8392 - accuracy: 0.4940 -
```

```
Epoch 5/10
51/51 [==============================] - 66s 1s/step - loss: 0.8449 - accuracy: 0.4803 -
Epoch 6/10
51/51 [==============================] - 65s 1s/step - loss: 0.8229 - accuracy: 0.4977 -
Epoch 7/10
51/51 [==============================] - 65s 1s/step - loss: 0.8252 - accuracy: 0.4920 -
Epoch 8/10
51/51 [==============================] - 66s 1s/step - loss: 0.8198 - accuracy: 0.4991 -
Epoch 9/10
51/51 [==============================] - 66s 1s/step - loss: 0.8138 - accuracy: 0.5219 -
Epoch 10/10
51/51 [==============================] - 66s 1s/step - loss: 0.7980 - accuracy: 0.5105 -
<tensorflow.python.keras.callbacks.History at 0x7f4ea99699d0>
```

Converting into spectrogram and giving spectrogram data as input

- We can use librosa to convert raw data into spectrogram. A spectrogram shows the features in a two-dimensional representation with the intensity of a frequency at a point in time i.e we are converting Time domain to frequency domain

```
#http://man.hubwiz.com/docset/LibROSA.docset/Contents/Resources/Documents/generated/librosa.c
#https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
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%|██████████| 1617/1617 [00:28<00:00, 55.92it/s]
100%|██████████| 405/405 [00:07<00:00, 55.20it/s]
```

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



X_train_spectrogram.shape
```

```
(1617, 64, 216)
```

```
X_test_spectrogram.shape
```

```
(405, 64, 216)
```

Model2

- Bulding Model LSTM and, giving spectrogram data as input

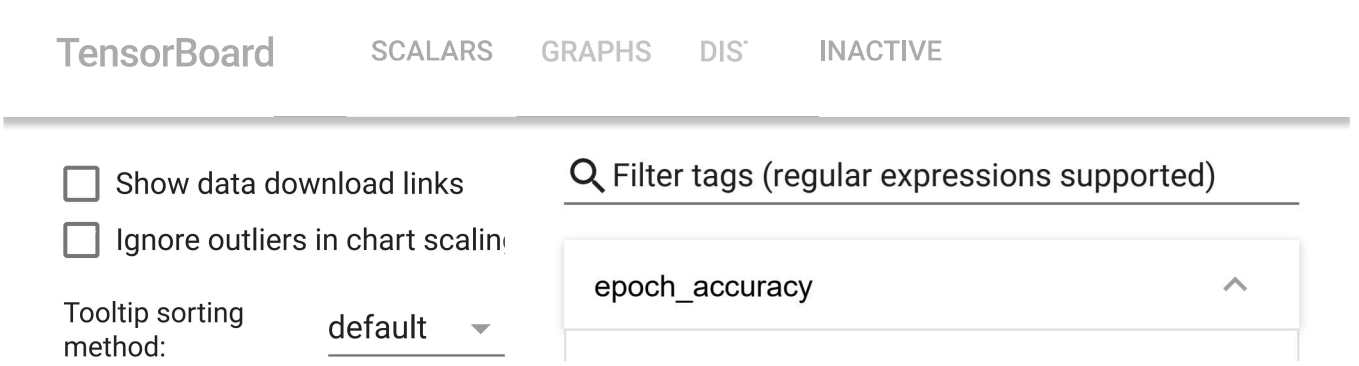
```
#https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
input1 = Input(shape=(64, 216,))
LSTM_layer= LSTM(256,dropout=0.01,return_sequences=True)(input1)
avg=GlobalAveragePooling1D(data_format='channels_last')(LSTM_layer)
x = Dense(256,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=31),k
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=2,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_norm
model2= Model(inputs=input1,outputs=Out)
```

```
model2.summary()
```

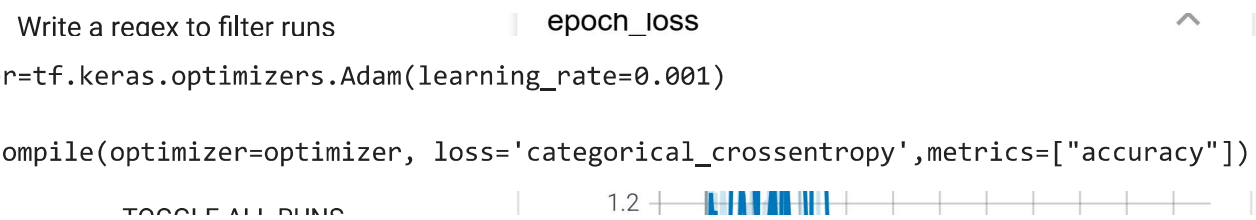
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, 216)]         0
_________________________________________________________________
lstm (LSTM)                  (None, 64, 256)           484352
_________________________________________________________________
global_average_pooling1d (Gl (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
```

```
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
Output (Dense)               (None, 2)                 66
=================================================================
Total params: 595,234
Trainable params: 594,338
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=["accuracy"])
```

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

    Epoch 1/200

    Epoch 00001: LearningRateScheduler reducing learning rate to 0.001.
    51/51 [==============================] - 6s 54ms/step - loss: 0.5201 - accuracy: 0.76
    Epoch 2/200

    Epoch 00002: LearningRateScheduler reducing learning rate to 0.001.
    51/51 [==============================] - 1s 25ms/step - loss: 0.5191 - accuracy: 0.77
    Epoch 3/200

    Epoch 00003: LearningRateScheduler reducing learning rate to 0.001.
    51/51 [==============================] - 1s 25ms/step - loss: 0.5410 - accuracy: 0.74
    Epoch 4/200

    Epoch 00004: LearningRateScheduler reducing learning rate to 0.001.
    51/51 [==============================] - 1s 25ms/step - loss: 0.5259 - accuracy: 0.75
    Epoch 5/200
```

```
Epoch 00005: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.5124 - accuracy: 0.77
Epoch 6/200

Epoch 00006: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.5440 - accuracy: 0.75
Epoch 7/200

Epoch 00007: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.5119 - accuracy: 0.77
Epoch 8/200

Epoch 00008: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.5151 - accuracy: 0.76
Epoch 9/200

Epoch 00009: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.5065 - accuracy: 0.76
Epoch 10/200

Epoch 00010: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.5116 - accuracy: 0.76
Epoch 11/200

Epoch 00011: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 26ms/step - loss: 0.4910 - accuracy: 0.79
Epoch 12/200

Epoch 00012: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.5064 - accuracy: 0.77
Epoch 13/200

Epoch 00013: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.4962 - accuracy: 0.78
Epoch 14/200

Epoch 00014: LearningRateScheduler reducing learning rate to 0.001.
51/51 [==============================] - 1s 25ms/step - loss: 0.4861 - accuracy: 0.79
Epoch 15/200
```

```
# save it as a h5 file

from tensorflow.keras.models import load_model

model2.save('audio_classification_model2.h5')


from tensorflow.keras.models import load_model


model=load_model('/content/audio_classification_model2.h5')


Pred_output=model.predict(X_test_spectrogram)
```

```
Pred_output
```

```
array([[0.2847415 , 0.7152585 ],
       [0.26866093, 0.73133904],
       [0.99409986, 0.00590016],
       [0.8632034 , 0.13679664],
       [0.2718735 , 0.7281265 ],
       [0.873952  , 0.126048  ],
       [0.26806155, 0.7319385 ],
       [0.27665323, 0.7233468 ],
       [0.26802203, 0.73197794],
       [0.62093765, 0.37906232],
       [0.71224624, 0.28775382],
       [0.35807887, 0.6419211 ],
       [0.26799732, 0.7320027 ],
       [0.7375338 , 0.26246622],
       [0.2828359 , 0.7171641 ],
       [0.26987356, 0.73012644],
       [0.29029065, 0.70970935],
       [0.29089537, 0.7091046 ],
       [0.3371343 , 0.6628657 ],
       [0.26799688, 0.73200315],
       [0.26813886, 0.7318611 ],
       [0.30143788, 0.69856215],
       [0.26798895, 0.7320111 ],
       [0.9217836 , 0.07821633],
       [0.27896973, 0.7210303 ],
       [0.70181596, 0.29818407],
       [0.74067056, 0.25932947],
       [0.26828298, 0.73171705],
       [0.26798978, 0.73201025],
       [0.31083733, 0.6891627 ],
       [0.2764801 , 0.72351986],
       [0.9657312 , 0.03426875],
       [0.2780277 , 0.72197235],
       [0.26905063, 0.7309494 ],
       [0.45222458, 0.5477754 ],
       [0.92493474, 0.07506519],
       [0.2784151 , 0.7215849 ],
       [0.2682315 , 0.7317685 ],
       [0.2757435 , 0.72425646],
       [0.8426281 , 0.15737194],
       [0.4879885 , 0.5120115 ],
       [0.27775165, 0.7222483 ],
       [0.26911992, 0.73088014],
       [0.28114358, 0.71885645],
       [0.26809087, 0.7319091 ],
       [0.8878844 , 0.11211561],
       [0.2711253 , 0.7288747 ],
       [0.60914385, 0.3908561 ],
       [0.8424859 , 0.15751404],
       [0.2921138 , 0.70788616],
       [0.2682202 , 0.73177975],
       [0.26831186, 0.7316881 ],
       [0.26859725, 0.7314028 ],
       [0.32305408, 0.676946  ],
```

```
            [0.2694722 , 0.73052776],
            [0.26898488, 0.7310151 ],
            [0.2682988 , 0.7317012 ],
            [0.27445832, 0.72554165],
            [0.26801807, 0.73198193],
```

```
preds_new = np.argmax(Pred_output, axis=1)
```

```
preds_new
```

```
array([1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 1, 1, 1, 1])
```

```
df_test=pd.DataFrame()
df_test['path']=X_test
df_test['true_label']=y_test
df_test['Pred_output'] =preds_new
```

```
df_test.head(20)
```

	path	true_label	Pred_output
599	/content/fan/id_00/normal/00000194.wav	1	1
1060	/content/fan/id_00/normal/00000755.wav	1	1
309	/content/fan/id_00/abnormal/00000129.wav	0	0
160	/content/fan/id_00/abnormal/00000362.wav	0	0
233	/content/fan/id_00/abnormal/00000227.wav	0	1
18	/content/fan/id_00/abnormal/00000079.wav	0	0
354	/content/fan/id_00/abnormal/00000245.wav	0	1
1000	/content/fan/id_00/normal/00000616.wav	1	1
721	/content/fan/id_00/normal/00000515.wav	1	1
152	/content/fan/id_00/abnormal/00000345.wav	0	0
143	/content/fan/id_00/abnormal/00000406.wav	0	0
588	/content/fan/id_00/normal/00000501.wav	1	1

Observation :

1) In this assignment we have performed audio classification ,will are classifing Normal fan and abnc
2)first of all we read the data and created a dataFrame of path(file path)
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.

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