Spoken Digit Recognition

In this notebook, You will do Spoken Digit Recognition.

Input - speech signal, output - digit number

It contains

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

Dataset Link

Instructions:

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

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

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.

Libraries and versions

```
pip install -U pip
pip install -U pandas==1.5.0
pip install -U numpy==1.23.4
pip install -U seaborn==0.12.0
pip install -U tqdm==4.64.1
pip install -U tensorflow==2.10.0
pip install -U scikit-learn==1.1.2
pip install -U librosa==0.9.2
pip install -U pydot==1.4.2
pip install -U prettytable==3.4.1
pip install -U plotly-express==0.4.1
```

In [1]:

```
import os
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from plotly.subplots import make_subplots

from tqdm import tqdm
from datetime import datetime
from prettytable import FRAME
from prettytable import PrettyTable
from prettytable import SINGLE_BORDER
from IPython.display import Audio
from IPython.display import YouTubeVideo

from sklearn.utils import shuffle
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split

import librosa
import librosa.display

import tensorflow as tf
```

```python
from tensorflow.keras.layers import LSTM
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import GlobalAveragePooling1D
from tensorflow.keras.utils import pad_sequences
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau

import warnings
warnings.filterwarnings('ignore')

plt.style.use('fivethirtyeight')
```

We shared recordings.zip, please unzip those.

In [2]:

```python
# 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"

DIR_NAME = 'recordings/'
all_files = [DIR_NAME + file for file in os.listdir(DIR_NAME)]
print(f'Number of files recorded : {len(all_files)}')
```

```
Number of files recorded : 2000
```

Grader function 1

In [3]:

```python
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[3]:

```
True
```

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

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

Eg: 0_jackson_0 --> 0

0_jackson_43 --> 0

Exploring the sound dataset

In [4]:

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

YT_videos = ['https://youtu.be/37zCgCdV468', 'https://youtu.be/m3XbqfIij_Y',
             'https://youtu.be/0a_d-zaUti8', 'https://youtu.be/ZqpSb5p1xQo']

print('\nLink to the reference videos listed bellow\n', '-' * 41)
for idx, video in enumerate(YT_videos):
    print(f'{idx+1}. {video}')
```

```
Link to the reference videos listed bellow
 -----------------------------------------
1. https://youtu.be/37zCgCdV468
2. https://youtu.be/m3XbqfIij_Y
3. https://youtu.be/0a_d-zaUti8
4. https://youtu.be/ZqpSb5p1xQo
```

1. 

2. 

3. 

4. 

In [5]:

```
# Lets listen few random audio files from the collection
# https://plotly.com/python/subplots/

random_list = np.random.choice(all_files, size = 5, replace = False)

fig = go.Figure()
fig = make_subplots(rows = 5)
for idx, audio in enumerate(random_list):

    print(audio.split('/')[-1])
    display(Audio(audio))
    raw_data, sr = librosa.load(audio)
    fig.add_scatter(y = raw_data, row = idx+1, col = 1, name = audio.split('/')[-1])

fig.update_layout(height = 900, width = 850, title_text = 'Wave form of sampled audio files')
fig.show()
```

8_nicolas_3.wav

00:00 00:00

6_jackson_7.wav

00:00 00:00

8_theo_2.wav

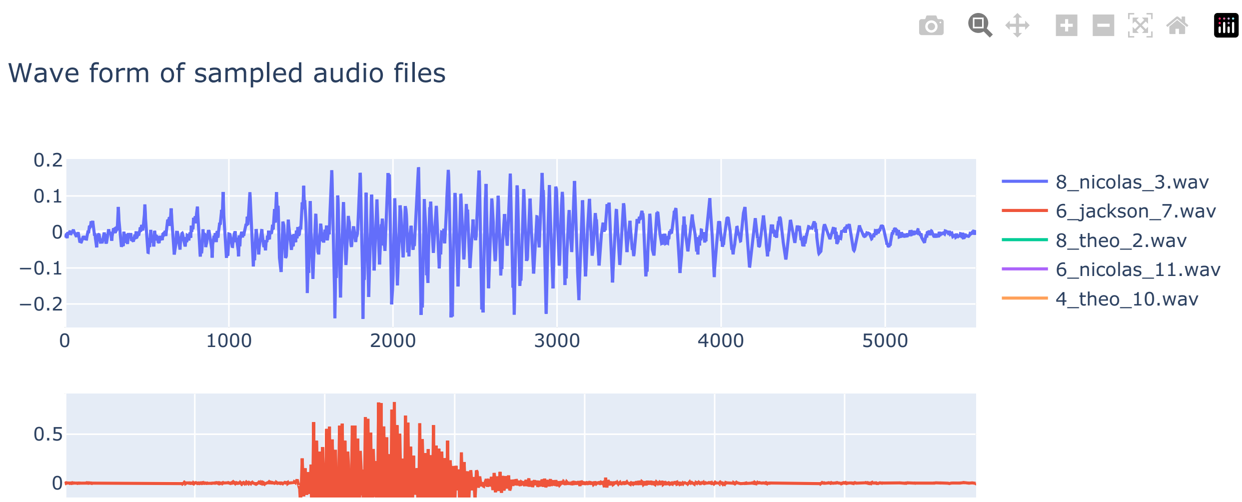
00:00 00:00

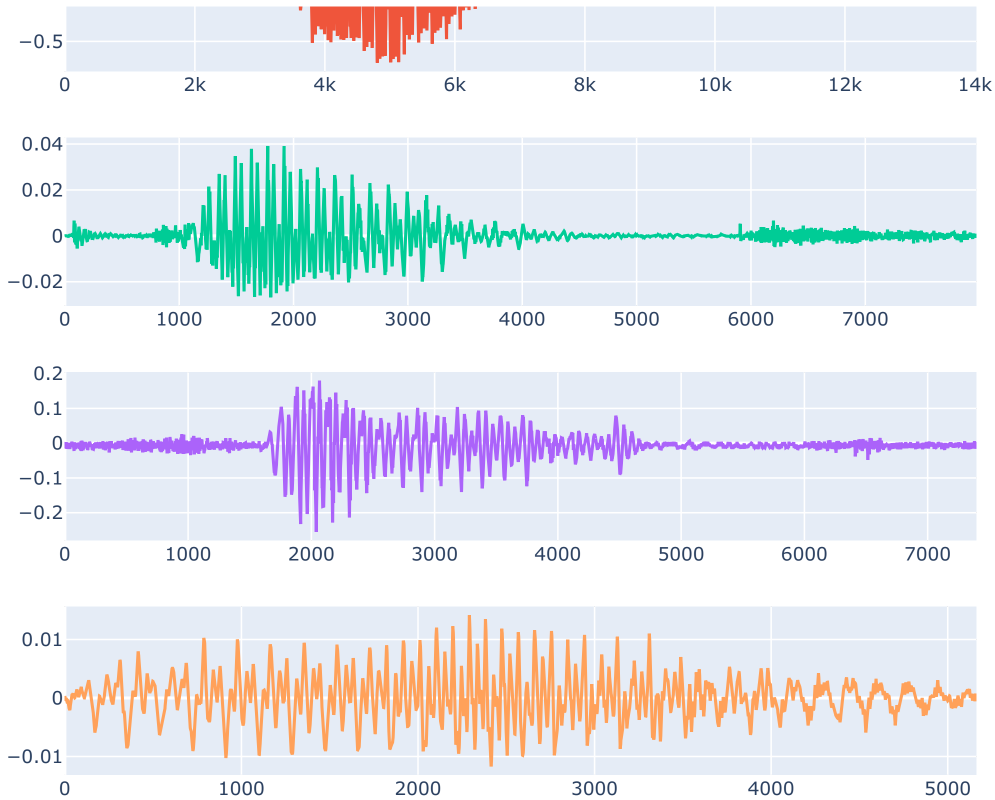
6_nicolas_11.wav

00:00 00:00

4_theo_10.wav

00:00 00:00





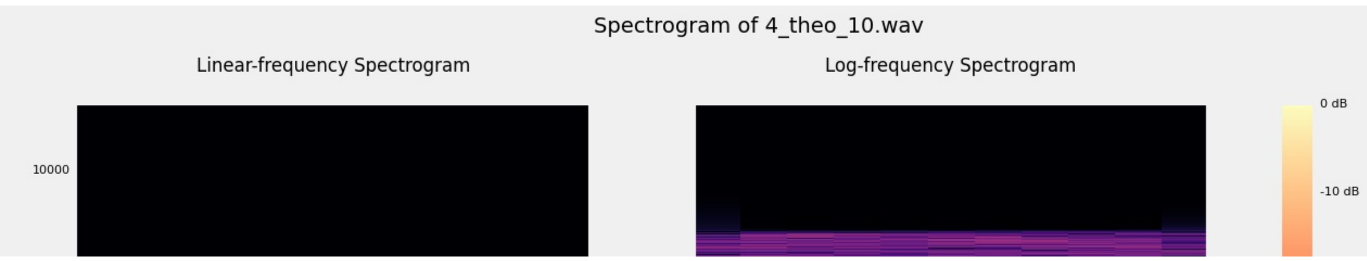
In [6]:

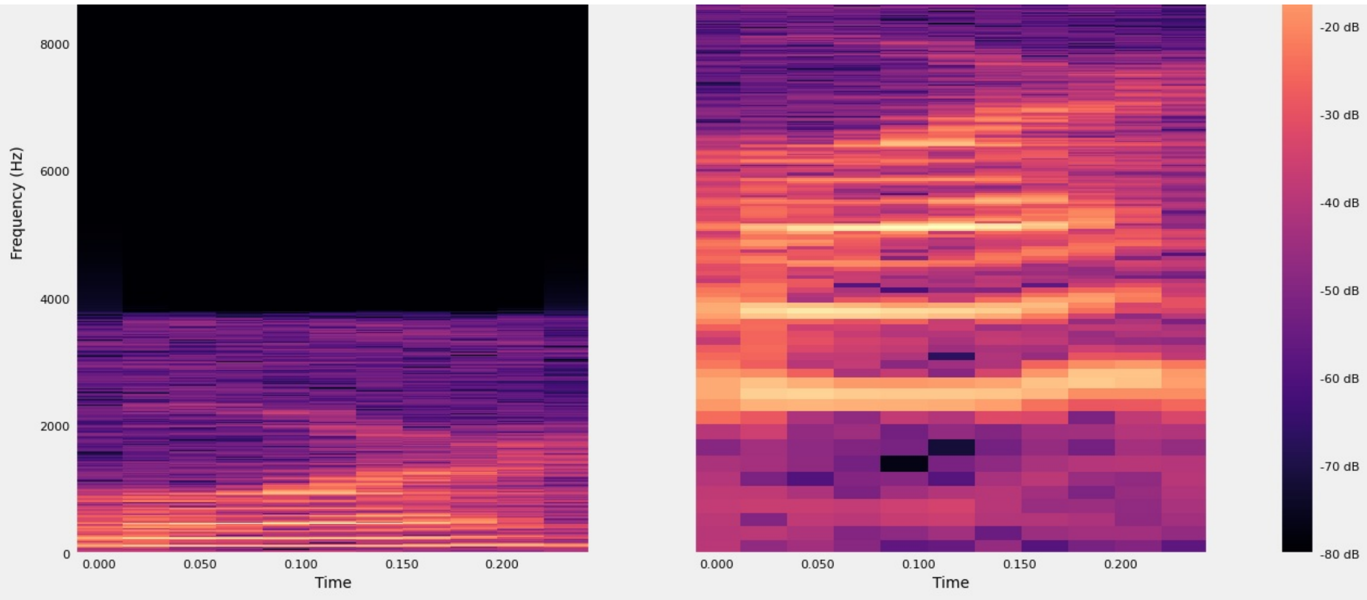
```
# Convet wave form to Frequency-Time format

D = librosa.stft(raw_data) # Short-time Fourier transform (STFT)
S_db = librosa.amplitude_to_db(np.abs(D), ref = np.max)
print(f"Shape of 'S_db' :: {S_db.shape}")
```

```
Shape of 'S_db' :: (1025, 11)
```

In [7]:

```
# https://stackoverflow.com/a/39566040
# https://librosa.org/doc/main/generated/librosa.display.specshow.html

plt.rc('xtick', labelsize = 8)
plt.rc('ytick', labelsize = 8)
plt.rc('axes', labelsize = 10)
plt.rc('axes', titlesize = 12)

fig, ax = plt.subplots(nrows = 1, ncols = 2, sharex = True, figsize = (15, 8))
fig.suptitle(f"Spectrogram of {audio.split('/')[-1]}\n", fontsize = 14)

img = librosa.display.specshow(S_db, y_axis = 'linear', x_axis = 'time', ax = ax[0])
ax[0].set_title('Linear-frequency Spectrogram\n')
ax[0].set_ylabel('Frequency (Hz)')

img = librosa.display.specshow(S_db, y_axis = 'log', x_axis = 'time', ax = ax[1])
ax[1].set_title('Log-frequency Spectrogram\n')
ax[1].set_ylabel('Frequency (Hz)')
ax[1].label_outer()

fig.colorbar(img, ax = ax, format = '%.f dB')

plt.show()
```





Creating dataframe

In [8]:

```
# 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

labels = [file.split('/')[-1][-0] for file in all_files]
df_audio = pd.DataFrame({'path' : all_files, 'label' : labels})

df_audio.head()
```

Out[8]:

	path	label
0	recordings/3_yweweler_46.wav	3
1	recordings/3_nicolas_28.wav	3
2	recordings/3_jackson_40.wav	3
3	recordings/3_yweweler_16.wav	3
4	recordings/4_jackson_42.wav	4

In [9]:

```
# info

df_audio.info()
```

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

Grader function 2

In [10]:

```
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

Out[10]: True

In [11]:
```
df_audio = shuffle(df_audio, random_state = 33) # don't change the random state
df_audio.head()
```

Out[11]:

	path	label
766	recordings/4_yweweler_19.wav	4
182	recordings/6_theo_2.wav	6
1763	recordings/1_theo_12.wav	1
1814	recordings/8_jackson_34.wav	8
596	recordings/1_yweweler_0.wav	1

Train and Validation split

In [12]:
```
# 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 [13]:
```
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[13]: True

Preprocessing

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

In [14]:
```
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:

        # From version 0.10 passing these as positional arguments will result in an error
        # librosa.get_duration(samples, sample_rate)
        duration = librosa.get_duration(y = samples, sr = sample_rate)
        return [samples, duration]

    else:
        return samples
```

In [15]:
```
# use load_wav function that was written above to get every wave
```

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

def process_wave_data(files, name):

    duration, sample = [], []

    for file in tqdm(files, name + ' '):
        sample_ , duration_ = load_wav(file)
        sample.append(sample_)
        duration.append(duration_)

    return pd.DataFrame({'raw_data' : sample, 'duration' : duration})

X_train_processed = process_wave_data(X_train, 'X_train')
X_test_processed = process_wave_data(X_test, 'X_test')
```

```
X_train : 100%|██████████| 1400/1400 [00:27<00:00, 51.11it/s]
X_test : 100%|██████████| 600/600 [00:05<00:00, 112.11it/s]
```

In [16]:

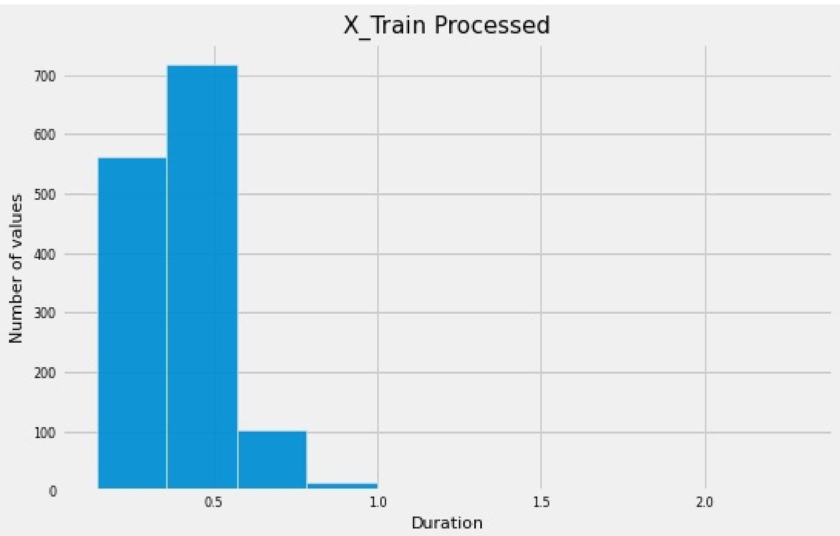
```python
X_train_processed.head()
```

Out[16]:

	raw_data	duration
0	[-0.000117121606, 0.00048175635, 0.0005162705,...	0.491020
1	[-0.013005042, -0.011989501, -0.010623834, -0....	0.323039
2	[0.009224007, 0.009306979, 0.0042455555, -0.00...	0.312381
3	[-0.000104876926, -9.907236e-05, -6.296669e-05...	0.303401
4	[-0.008764728, -0.010537301, -0.009207653, -0....	0.515510

In [17]:

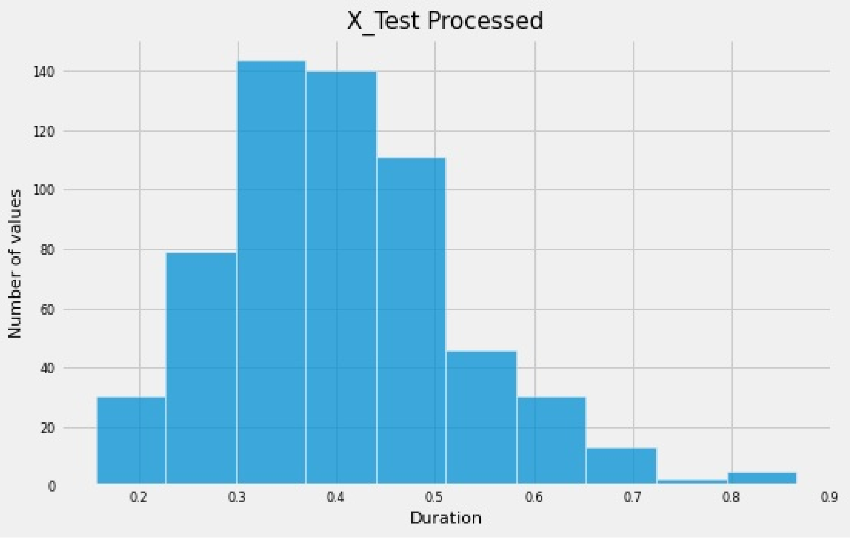
```python
# plot the histogram of the duration for trian

plt.rc('axes', titlesize = 15)
plt.rc('axes', labelsize = 11)

plt.rcParams['figure.figsize'] = [8, 5]

ax = sns.histplot(X_train_processed.duration, bins = 10)

ax.set_xlabel('Duration')
ax.set_ylabel('Number of values')
ax.set_title('X_Train Processed')
sns.histplot(X_train_processed.duration, bins = 10)
plt.show()
```



In [18]:

```python
# plot the histogram of the duration for test

ax = sns.histplot(X_test_processed.duration,bins = 10)

ax.set_xlabel('Duration')
ax.set_ylabel('Number of values')
```

```
ax.set_title('X_Test Processed')

plt.show()
```



In [19]:
```
# Percentile print function

def percentiles(df, lower, upper, incerment):
     for val in range(lower, upper + incerment, incerment):
            print(f'{val:3d}th percentile is {np.percentile(df.duration, val)}')
```

In [20]:
```
# print 0 to 100 percentile values with step size of 10 for train data duration.

percentiles(X_train_processed, 0, 100, 10)
```

```
  0th percentile is 0.1435374149659864
 10th percentile is 0.2599909297052154
 20th percentile is 0.29777777777777775
 30th percentile is 0.32875283446712017
 40th percentile is 0.35705215419501135
 50th percentile is 0.3868934240362812
 60th percentile is 0.4157823129251701
 70th percentile is 0.4435374149659864
 80th percentile is 0.4821587301587302
 90th percentile is 0.5533242630385488
100th percentile is 2.282766439909297
```

In [21]:
```
# print 90 to 100 percentile values with step size of 1.

percentiles(X_train_processed, 90, 100, 1)
```

```
 90th percentile is 0.5533242630385488
 91th percentile is 0.5659854875283448
 92th percentile is 0.5794503401360545
 93th percentile is 0.5995106575963719
 94th percentile is 0.6133696145124716
 95th percentile is 0.6227800453514739
 96th percentile is 0.6431455782312925
 97th percentile is 0.664770068027211
 98th percentile is 0.7138956916099773
 99th percentile is 0.7963900226757369
100th percentile is 2.282766439909297
```

Grader function 4

In [22]:
```
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[22]: True

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

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

Also create a masking vector for train and test.

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

In [23]:

```
max_length  = 17640
```

In [24]:

```
# 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.

# https://www.tensorflow.org/api_docs/python/tf/keras/utils/pad_sequences

X_train_pad_seq = pad_sequences(X_train_processed.raw_data, maxlen = max_length, padding = 'post',
                                                    value = 0, dtype = 'float', truncating = 'post')
X_test_pad_seq = pad_sequences(X_test_processed.raw_data, maxlen = max_length, padding = 'post',
                                                  value = 0, dtype = 'float', truncating = 'post')

print(f"Shape of 'X_train_pad_seq' :: {X_train_pad_seq.shape}")
print(f"Shape of 'X_test_pad_seq'  :: {X_test_pad_seq.shape}")

X_train_mask = X_train_pad_seq.astype('bool')
X_test_mask = X_test_pad_seq.astype('bool')
```

```
Shape of 'X_train_pad_seq' :: (1400, 17640)
Shape of 'X_test_pad_seq'  :: (600, 17640)
```

Grader function 5

In [25]:

```
def grader_padoutput():

    flag_padshape = (X_train_pad_seq.shape==(1400, 17640)) and (X_test_pad_seq.shape==(600, 17640)) and \
                    (y_train.shape==(1400,))

    flag_maskshape = (X_train_mask.shape==(1400, 17640)) and (X_test_mask.shape==(600, 17640)) and \
                    (y_test.shape==(600,))

    flag_dtype = (X_train_mask.dtype==bool) and (X_test_mask.dtype==bool)

    return flag_padshape and flag_maskshape and flag_dtype

grader_padoutput()
```

Out[25]: True

Acceptance Criteria

Model	Micro F1 score
Model 1 & Model 3	0.10
Model 2 & Model 4	0.80

1. Giving Raw data directly.

Now we have

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

We will create a LSTM model which takes this input.

Task:

1. Create an LSTM network which takes "X_train_pad_seq" as input, "X_train_mask" as mask input. You can use any number of LSTM cells. Please read LSTM documentation(https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM) in tensorflow to know more about mask and also https://www.tensorflow.org/guide/keras/masking_and_padding
2. Get the final output of the LSTM and give it to Dense layer of any size and then give it to Dense layer of size 10(because we have 10 outputs) and then compile with the sparse categorical cross entropy(because we are not converting it to one hot vectors). Also check the datatype of class labels(y_values) and make sure that you convert your class labels to integer datatype before fitting in the model.
3. While defining your model make sure that you pass both the input layer and mask input layer as input to lstm layer as follows

```
lstm_output = self.lstm(input_layer, mask=masking_input_layer)
```

4. Use tensorboard to plot the graphs of loss and metric(use custom micro F1 score as metric) and histograms of gradients. You can write your code for computing F1 score using this link

5. make sure that it won't overfit.

6. You are free to include any regularization

In [26]:
```
if not os.path.isdir('results'):
    os.mkdir('results')
```

In [27]:
```
y_train_int = y_train.astype('int')
y_test_int = y_test.astype('int')
```

In [28]:
```
# as discussed above, please write the architecture of the model.
# you will have two input layers in your model (data input layer and mask input layer)
# make sure that you have defined the data type of masking layer as bool

# https://keras.io/api/models/model/

def model_1_3(max_length, model_name):

    tf.keras.backend.clear_session()

    input_pad = Input(shape = (max_length, 1))
    input_mask = Input(shape = max_length, dtype = 'bool')
    x = LSTM(25)(input_pad, mask = input_mask)
    x = Dense(50, activation = 'relu')(x)
    x = Dropout(0.5)(x)
    x = BatchNormalization()(x)
    x = Dense(40, activation = 'relu')(x)
    x = Dropout(0.3)(x)
    output_ = Dense(10, activation = 'softmax')(x)

    return Model(inputs = [input_pad, input_mask], outputs = output_, name = model_name)
```

In [29]:
```
model_raw = model_1_3(max_length, 'RAW_Data_Alone')

model_raw.summary()
```

```
Model: "RAW_Data_Alone"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 17640, 1)]   0           []

 input_2 (InputLayer)           [(None, 17640)]      0           []

 lstm (LSTM)                    (None, 25)           2700        ['input_1[0][0]',
                                                                  'input_2[0][0]']

 dense (Dense)                  (None, 50)           1300        ['lstm[0][0]']

 dropout (Dropout)              (None, 50)           0           ['dense[0][0]']

 batch_normalization (BatchNorm  (None, 50)          200         ['dropout[0][0]']
 alization)

 dense_1 (Dense)                (None, 40)           2040        ['batch_normalization[0][0]']
```

```
 dropout_1 (Dropout)           (None, 40)          0          ['dense_1[0][0]']

 dense_2 (Dense)               (None, 10)          410        ['dropout_1[0][0]']

==================================================================================================
Total params: 6,650
Trainable params: 6,550
Non-trainable params: 100
__________________________________________________________________________________________________
```

In [30]:

```python
# https://stackoverflow.com/a/59564740

class AccThreshold(Callback):

    def __init__(self, thr_val, plus):
        self.thr_val = thr_val + plus

    def on_epoch_end(self, epoch, logs = {}):
        val_microF1 = logs.get('val_micro_f1')
        if (val_microF1 >= self.thr_val) and epoch >= 5:
            print(f'\n\n\tTerminating training at epoch {epoch+1} with a validation micro F1 score of {val_microF
            self.model.stop_training = True
```

In [31]:

```python
def call_back_list(thr_val, thr_plus, location):

    '''
    Generating keras Callbacks and return callback list

    Input
    -----
    1. Throshold value : Acceptance critiria
    2. Model stops at which accuracy
    3. Loaction to save ModelCheckPoint and Logs in model training
    '''

    cust_callback = AccThreshold(thr_val, thr_plus)

    model_pat  = f'results/Checkpoints/{location}/'
    filepath = model_pat + 'EPO_{epoch:02d}-F1_{val_micro_f1:.04}.h5'
    chek_pt = ModelCheckpoint(filepath, monitor = 'val_micro_f1', verbose = 0, save_best_only = True, )

    logDir = f'results/logs/{location}/' + datetime.now().strftime('%y_%h%d_%H%M')
    t_board = TensorBoard(log_dir = logDir, histogram_freq = 1)

    reduce_lr = ReduceLROnPlateau(monitor = 'val_micro_f1', factor = 0.2, verbose = 0, patience = 5)

    return [cust_callback, chek_pt, t_board, reduce_lr]
```

In [32]:

```python
def f1_micro(y_true,y_pred):
    return f1_score(y_true, y_pred, average = 'micro')

def micro_f1(y_true, y_proba):
    y_pred = tf.math.argmax(y_proba, axis = 1)
    return tf.py_function(f1_micro, (y_true, y_pred), tf.double)
```

In [33]:

```python
# https://keras.io/api/metrics/accuracy_metrics/#sparsecategoricalaccuracy-class
# https://keras.io/api/optimizers/adamax/

callBacks = call_back_list(0.1, 0, '1_Data_Raw')

model_raw.compile(optimizer = 'Adamax', loss = 'sparse_categorical_crossentropy', metrics = [micro_f1])

# train your model
# model1.fit([X_train_pad_seq,X_train_mask],y_train_int,.........)

EPOCH = 40
raw_LSTM = model_raw.fit(x = [X_train_pad_seq, X_train_mask], y = y_train_int, epochs = EPOCH,
           validation_data = ([X_test_pad_seq, X_test_mask], y_test_int), callbacks = callBacks)
```

```
Epoch 1/40
44/44 [==============================] - 28s 512ms/step - loss: 2.3091 - micro_f1: 0.0961 - val_loss: 2.3026 - va
l_micro_f1: 0.0998 - lr: 0.0010
Epoch 2/40
44/44 [==============================] - 19s 429ms/step - loss: 2.3067 - micro_f1: 0.0857 - val_loss: 2.3026 - va
l_micro_f1: 0.0998 - lr: 0.0010
Epoch 3/40
44/44 [==============================] - 21s 486ms/step - loss: 2.3016 - micro_f1: 0.0997 - val_loss: 2.3026 - va
l_micro_f1: 0.0998 - lr: 0.0010
Epoch 4/40
44/44 [==============================] - 21s 475ms/step - loss: 2.3031 - micro_f1: 0.1106 - val_loss: 2.3026 - va
```

```
l_micro_f1: 0.0998 - lr: 0.0010
Epoch 5/40
44/44 [==============================] - 21s 474ms/step - loss: 2.3032 - micro_f1: 0.0964 - val_loss: 2.3026 - va
l_micro_f1: 0.0998 - lr: 0.0010
Epoch 6/40
44/44 [==============================] - 19s 438ms/step - loss: 2.3044 - micro_f1: 0.0843 - val_loss: 2.3026 - va
l_micro_f1: 0.0998 - lr: 0.0010
Epoch 7/40
44/44 [==============================] - 19s 436ms/step - loss: 2.3026 - micro_f1: 0.0994 - val_loss: 2.3026 - va
l_micro_f1: 0.0998 - lr: 2.0000e-04
Epoch 8/40
44/44 [==============================] - ETA: 0s - loss: 2.3040 - micro_f1: 0.1020

        Terminating training at epoch 8 with a validation micro F1 score of 0.10307 %

44/44 [==============================] - 21s 481ms/step - loss: 2.3040 - micro_f1: 0.1020 - val_loss: 2.3026 - va
l_micro_f1: 0.1031 - lr: 2.0000e-04
```

In [34]:

```
def save_model_history(filename, base_model, model):

    '''
    Saves model and model history

    Input
    -----
    1. File name to store model history file
    2. Keras model name
    3. Saved keras model name
    '''

    with open('results/' + filename + '_history.pkl', 'wb') as file:
        pickle.dump(model.history, file)

    train_epo = len(model.epoch)
    final_f1 = model.history['val_micro_f1'][-1]
    base_model.save(f'results/{filename}_Epo_{train_epo}_F1_{final_f1:.4f}.h5')
```

In [35]:

```
save_model_history('1_data_raw', model_raw, raw_LSTM)
```

Observation

Using Raw data is not a good option because loss and micro-f1 is not imporving

2. Converting into spectrogram and giving spectrogram data as input

We can use librosa to convert raw data into spectrogram. A spectrogram shows the features in a two-dimensional representation with the intensity of a frequency at a point in time i.e we are converting Time domain to frequency domain. you can read more about this in https://pnsn.org/spectrograms/what-is-a-spectrogram

In [36]:

```
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
```

In [37]:

```
# use convert_to_spectrogram and convert every raw sequence in X_train_pad_seq and X_test_pad-seq.
# save those all in the X_train_spectrogram and X_test_spectrogram ( These two arrays must be numpy arrays)

X_train_spectrogram  = np.array([convert_to_spectrogram(data) for data in tqdm(X_train_pad_seq, \
                                                                              'X_train_pad_seq : ')])

X_test_spectrogram  = np.array([convert_to_spectrogram(data) for data in tqdm(X_test_pad_seq, \
                                                                             'X_test_pad_seq : ')])
```

```
X_train_pad_seq : 100%|██████████| 1400/1400 [00:07<00:00, 198.23it/s]
X_test_pad_seq : 100%|██████████| 600/600 [00:02<00:00, 203.13it/s]
```

In [38]:

```
print(f'X_train_spectrogram --> Shape :: {X_train_spectrogram.shape}, Type :: {type(X_train_spectrogram)}')
print(f'X_test_spectrogram  --> Shape :: {X_test_spectrogram.shape} , Type :: {type(X_test_spectrogram)}')
```

```
X_train_spectrogram --> Shape :: (1400, 64, 35), Type :: <class 'numpy.ndarray'>
```

```
X_test_spectrogram  --> Shape :: (600, 64, 35) , Type :: <class 'numpy.ndarray'>
```

Grader function 6

In [39]:
```
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[39]:
```
True
```

Now we have

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

We will create a LSTM model which takes this input.

Task:

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

In [40]:
```
# write the architecture of the model
# print model.summary and make sure that it is following point 2 mentioned above

def model_2_4(max_length, model_name):

    tf.keras.backend.clear_session()

    input_ = Input(shape = X_train_spectrogram.shape[1:])
    x = LSTM(64, return_sequences = True)(input_)
    x = LSTM(64, return_sequences = True)(x)
    x = GlobalAveragePooling1D()(x)
    x = Dense(50, activation = 'relu')(x)
    x = BatchNormalization()(x)
    x = Dense(20, activation = 'relu')(x)
    output_ = Dense(10, activation = 'softmax')(x)

    return Model(inputs = input_, outputs = output_, name = model_name)
```

In [41]:
```
model_spectrogram = model_2_4(max_length, 'Spectrogram_Data_Alone')
model_spectrogram.summary()
```

```
Model: "Spectrogram_Data_Alone"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64, 35)]          0

 lstm (LSTM)                 (None, 64, 64)            25600

 lstm_1 (LSTM)               (None, 64, 64)            33024

 global_average_pooling1d (G  (None, 64)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 50)                3250

 batch_normalization (BatchN  (None, 50)               200
 ormalization)

 dense_1 (Dense)             (None, 20)                1020

 dense_2 (Dense)             (None, 10)                210

=================================================================
```

```
Total params: 63,304
Trainable params: 63,204
Non-trainable params: 100
_________________________________________________________________
```

In [42]:

```
# By apply ReduceLROnPlateau it's not getting converged thus not applying it in this model

callBacks = call_back_list(0.8, 0.05, '2_Data_Spectrogram')[:-1]

model_spectrogram.compile(optimizer = 'Adamax', loss = 'sparse_categorical_crossentropy', metrics = [micro_f1])

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

EPOCH = 100

spectrogram_LSTM = model_spectrogram.fit(x = X_train_spectrogram, y = y_train_int, epochs = EPOCH,
                      validation_data = (X_test_spectrogram, y_test_int), callbacks = callBacks)
```

```
Epoch 1/100
 5/44 [==>...........................] - ETA: 0s - loss: 2.4119 - micro_f1: 0.1063
```

```
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0109s v
s `on_train_batch_end` time: 0.0169s). Check your callbacks.
```

```
44/44 [==============================] - 5s 40ms/step - loss: 2.2187 - micro_f1: 0.1626 - val_loss: 2.2781 - val_
micro_f1: 0.1064
Epoch 2/100
44/44 [==============================] - 1s 16ms/step - loss: 2.0018 - micro_f1: 0.3059 - val_loss: 2.2393 - val_
micro_f1: 0.1908
Epoch 3/100
44/44 [==============================] - 1s 25ms/step - loss: 1.8743 - micro_f1: 0.3925 - val_loss: 2.1881 - val_
micro_f1: 0.3037
Epoch 4/100
44/44 [==============================] - 1s 24ms/step - loss: 1.7679 - micro_f1: 0.4276 - val_loss: 2.1101 - val_
micro_f1: 0.3531
Epoch 5/100
44/44 [==============================] - 1s 16ms/step - loss: 1.6802 - micro_f1: 0.4500 - val_loss: 2.0471 - val_
micro_f1: 0.3388
Epoch 6/100
44/44 [==============================] - 1s 15ms/step - loss: 1.5651 - micro_f1: 0.4912 - val_loss: 1.9495 - val_
micro_f1: 0.3481
Epoch 7/100
44/44 [==============================] - 1s 15ms/step - loss: 1.4863 - micro_f1: 0.5137 - val_loss: 1.7802 - val_
micro_f1: 0.4413
Epoch 8/100
44/44 [==============================] - 1s 15ms/step - loss: 1.3978 - micro_f1: 0.5438 - val_loss: 1.6539 - val_
micro_f1: 0.4945
Epoch 9/100
44/44 [==============================] - 1s 15ms/step - loss: 1.3363 - micro_f1: 0.5469 - val_loss: 1.5623 - val_
micro_f1: 0.4518
Epoch 10/100
44/44 [==============================] - 1s 15ms/step - loss: 1.2517 - micro_f1: 0.6025 - val_loss: 1.3609 - val_
micro_f1: 0.5581
Epoch 11/100
44/44 [==============================] - 1s 15ms/step - loss: 1.1491 - micro_f1: 0.6342 - val_loss: 1.2603 - val_
micro_f1: 0.6025
Epoch 12/100
44/44 [==============================] - 1s 15ms/step - loss: 1.0778 - micro_f1: 0.6600 - val_loss: 1.2022 - val_
micro_f1: 0.5943
Epoch 13/100
44/44 [==============================] - 1s 14ms/step - loss: 1.0148 - micro_f1: 0.6816 - val_loss: 1.2161 - val_
micro_f1: 0.5981
Epoch 14/100
44/44 [==============================] - 1s 16ms/step - loss: 0.9657 - micro_f1: 0.6937 - val_loss: 1.1152 - val_
micro_f1: 0.6151
Epoch 15/100
44/44 [==============================] - 1s 16ms/step - loss: 0.8876 - micro_f1: 0.7161 - val_loss: 1.0006 - val_
micro_f1: 0.6859
Epoch 16/100
44/44 [==============================] - 1s 30ms/step - loss: 0.8240 - micro_f1: 0.7450 - val_loss: 0.9915 - val_
micro_f1: 0.6749
Epoch 17/100
44/44 [==============================] - 1s 30ms/step - loss: 0.7613 - micro_f1: 0.7616 - val_loss: 0.9054 - val_
micro_f1: 0.6831
Epoch 18/100
44/44 [==============================] - 1s 32ms/step - loss: 0.7558 - micro_f1: 0.7588 - val_loss: 0.8343 - val_
micro_f1: 0.7462
Epoch 19/100
44/44 [==============================] - 1s 33ms/step - loss: 0.6926 - micro_f1: 0.7808 - val_loss: 0.8812 - val_
micro_f1: 0.7045
Epoch 20/100
44/44 [==============================] - 1s 34ms/step - loss: 0.6395 - micro_f1: 0.8106 - val_loss: 0.7793 - val_
micro_f1: 0.7440
Epoch 21/100
```

```
44/44 [==============================] - 1s 30ms/step - loss: 0.5864 - micro_f1: 0.8248 - val_loss: 0.7667 - val_
micro_f1: 0.7226
Epoch 22/100
44/44 [==============================] - 1s 31ms/step - loss: 0.5638 - micro_f1: 0.8179 - val_loss: 0.7142 - val_
micro_f1: 0.7582
Epoch 23/100
44/44 [==============================] - 1s 27ms/step - loss: 0.5290 - micro_f1: 0.8478 - val_loss: 0.7546 - val_
micro_f1: 0.7632
Epoch 24/100
44/44 [==============================] - 1s 33ms/step - loss: 0.4990 - micro_f1: 0.8475 - val_loss: 0.7140 - val_
micro_f1: 0.7714
Epoch 25/100
44/44 [==============================] - 1s 27ms/step - loss: 0.4721 - micro_f1: 0.8613 - val_loss: 0.8107 - val_
micro_f1: 0.7264
Epoch 26/100
44/44 [==============================] - 1s 25ms/step - loss: 0.4610 - micro_f1: 0.8610 - val_loss: 0.6634 - val_
micro_f1: 0.8032
Epoch 27/100
44/44 [==============================] - 1s 16ms/step - loss: 0.4484 - micro_f1: 0.8549 - val_loss: 0.8946 - val_
micro_f1: 0.6859
Epoch 28/100
44/44 [==============================] - 1s 16ms/step - loss: 0.4381 - micro_f1: 0.8615 - val_loss: 0.7122 - val_
micro_f1: 0.7867
Epoch 29/100
44/44 [==============================] - 1s 15ms/step - loss: 0.4078 - micro_f1: 0.8646 - val_loss: 0.6947 - val_
micro_f1: 0.7796
Epoch 30/100
44/44 [==============================] - 1s 16ms/step - loss: 0.3592 - micro_f1: 0.8968 - val_loss: 0.6189 - val_
micro_f1: 0.8120
Epoch 31/100
44/44 [==============================] - 1s 16ms/step - loss: 0.3865 - micro_f1: 0.8672 - val_loss: 0.5680 - val_
micro_f1: 0.8169
Epoch 32/100
44/44 [==============================] - 1s 16ms/step - loss: 0.3567 - micro_f1: 0.8885 - val_loss: 0.6008 - val_
micro_f1: 0.8032
Epoch 33/100
44/44 [==============================] - 1s 15ms/step - loss: 0.3130 - micro_f1: 0.8935 - val_loss: 0.5900 - val_
micro_f1: 0.8174
Epoch 34/100
44/44 [==============================] - 1s 15ms/step - loss: 0.3123 - micro_f1: 0.9098 - val_loss: 0.5611 - val_
micro_f1: 0.8289
Epoch 35/100
44/44 [==============================] - 1s 15ms/step - loss: 0.2894 - micro_f1: 0.9141 - val_loss: 0.5689 - val_
micro_f1: 0.8289
Epoch 36/100
44/44 [==============================] - 1s 15ms/step - loss: 0.2782 - micro_f1: 0.9148 - val_loss: 0.5382 - val_
micro_f1: 0.8322
Epoch 37/100
39/44 [=========================>....] - ETA: 0s - loss: 0.2443 - micro_f1: 0.9319

        Terminating training at epoch 37 with a validation micro F1 score of 0.85033 %

44/44 [==============================] - 1s 14ms/step - loss: 0.2559 - micro_f1: 0.9242 - val_loss: 0.4836 - val_
micro_f1: 0.8503
```

In [43]:

```
save_model_history('2_data_spectrogram', model_spectrogram, spectrogram_LSTM)
```

Observation

Using Raw data is not a good option because loss and micro-f1 is not imporving

3. Data augmentation with raw features

Till now we have done with 2000 samples only. It is very less data. We are giving the process of generating augmented data below.

There are two types of augmentation:

1. time stretching - Time stretching either increases or decreases the length of the file. For time stretching we move the file 30% faster or slower
2. pitch shifting - pitch shifting moves the frequencies higher or lower. For pitch shifting we shift up or down one half-step.

In [44]:

```
# 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 [45]:

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

In [46]:

```
len(aug_temp)
```

Out[46]: 9

Follow the steps

1. Split data 'df_audio' into train and test (80-20 split)

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

In [47]:

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

1. Do augmentation only on X_train,pass each point of X_train to generate_augmented_data function.After augmentation we will get 14400 train points. Make sure that you are augmenting the corresponding class labels (y_train) also.
2. Preprocess your X_test using load_wav function.
3. Convert the augmented_train_data and test_data to numpy arrays.
4. Perform padding and masking on augmented_train_data and test_data.
5. After padding define the model similar to model 1 and fit the data

In [48]:

```
# https://stackoverflow.com/a/41175538

def get_augmented_n_idx_data(df, name_):

    aug_data = []
    idx_values = []

    for idx, data in tqdm(zip(df.index, df), total = len(df), desc = name_):
        augment_values = generate_augmented_data(data)
        for aug in augment_values:
            aug_data.append(aug)
            idx_values.append(idx)

    return aug_data, idx_values

def get_augmented_data_for_y(df, idx_list, name_):

    aug_labels = []

    for idx, label in tqdm(zip(df.index, df), total = len(df), desc = name_):
        for idx_val in idx_list:
            if idx == idx_val:
                aug_labels.append(label)

    return aug_labels
```

In [49]:

```
augmented_train_data, idx_values_train = get_augmented_n_idx_data(X_train, 'X_train Aug. ')
augment_y_train = get_augmented_data_for_y(y_train, idx_values_train, 'y_train Aug. ')
```

```
X_train Aug. : 100%|██████████| 1600/1600 [04:18<00:00,  6.20it/s]
y_train Aug. : 100%|██████████| 1600/1600 [00:00<00:00, 2349.95it/s]
```

In [50]:

```
print(f"Length of 'augmented_train_data' :: {len(augmented_train_data)}")
print(f"Length of 'augment_y_train' :: {len(augment_y_train)}")
```

```
Length of 'augmented_train_data' :: 14400
Length of 'augment_y_train' :: 14400
```

In [51]:
```python
# Using only raw_data NOT duration from the process_wave_data() output

test_data = process_wave_data(X_test, 'X_test').raw_data
```

```
X_test : 100%|██████████| 400/400 [00:03<00:00, 111.02it/s]
```

In [52]:
```python
augmented_train_data = np.array(augmented_train_data)
print(f"Type of 'augmented_train_data' :: {type(augmented_train_data)}")

test_data = np.array(test_data)
print(f"Type of 'test_data_processed'  :: {type(test_data)}")

print(f"Type of 'augment_y_train'  :: {type(augment_y_train)}")
print("\nConverting 'augment_y_train' from 'Python list' to 'Numpy Array'")

augment_y_train = np.array(augment_y_train)

print(f"\nType of 'augment_y_train'  :: {type(augment_y_train)}")

aug_y_train_int = augment_y_train.astype('int')
y_test_int = y_test.astype('int')
```

```
Type of 'augmented_train_data' :: <class 'numpy.ndarray'>
Type of 'test_data_processed'  :: <class 'numpy.ndarray'>
Type of 'augment_y_train'  :: <class 'list'>

Converting 'augment_y_train' from 'Python list' to 'Numpy Array'

Type of 'augment_y_train'  :: <class 'numpy.ndarray'>
```

In [53]:
```python
# https://stackoverflow.com/a/63853924

max_length  = 17640

augmented_train_data_seq = pad_sequences(augmented_train_data, maxlen = max_length, padding = 'post',
                                                     value = 0, dtype = 'float', truncating = 'post')

test_data_seq = pad_sequences(test_data, maxlen = max_length, padding = 'post', value = 0,
                                                      dtype = 'float', truncating = 'post')

print(f"Shape of 'augmented_train_data_seq' :: {augmented_train_data_seq.shape}")
print(f"Shape of 'test_data_seq' :: {test_data_seq.shape}")

print(f"\nShape of 'aug_y_train_int' :: {aug_y_train_int.shape}")
print(f"Shape of 'y_test_int' :: {y_test_int.shape}")

train_augmented_seq_mask = augmented_train_data_seq.astype('bool')
test_data_seq_mask = test_data_seq.astype('bool')
```

```
Shape of 'augmented_train_data_seq' :: (14400, 17640)
Shape of 'test_data_seq' :: (400, 17640)

Shape of 'aug_y_train_int' :: (14400,)
Shape of 'y_test_int' :: (400,)
```

In [54]:
```python
model_augmented = model_1_3(max_length, 'Data_Augmented')

model_augmented.summary()
```

```
Model: "Data_Augmented"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 17640, 1)]   0           []

 input_2 (InputLayer)           [(None, 17640)]      0           []

 lstm (LSTM)                    (None, 25)           2700        ['input_1[0][0]',
                                                                  'input_2[0][0]']

 dense (Dense)                  (None, 50)           1300        ['lstm[0][0]']

 dropout (Dropout)              (None, 50)           0           ['dense[0][0]']

 batch_normalization (BatchNorm  (None, 50)          200         ['dropout[0][0]']
 alization)

 dense_1 (Dense)                (None, 40)           2040        ['batch_normalization[0][0]']
```

```
 dropout_1 (Dropout)           (None, 40)          0          ['dense_1[0][0]']

 dense_2 (Dense)               (None, 10)          410        ['dropout_1[0][0]']

==================================================================================================
Total params: 6,650
Trainable params: 6,550
Non-trainable params: 100
__________________________________________________________________________________________________
```

Note - While fitting your model on the augmented data for model 3 you might face Resource exhaust error. One simple hack to avoid that is save the augmented_train_data,augment_y_train,test_data and y_test to Drive or into your local system. Then restart the runtime so that now you can train your model with full RAM capacity. Upload these files again in the new runtime session perform padding and masking and then fit your model.

In [55]:

```
callBacks = call_back_list(0.1, 0.003, '3_Data_Augmented')

model_augmented.compile(optimizer = 'Adamax', loss = 'sparse_categorical_crossentropy', metrics = [micro_f1])

EPOCH = 40

augmented_LSTM = model_augmented.fit(x = [augmented_train_data_seq, train_augmented_seq_mask],
                                     validation_data = ([test_data_seq, test_data_seq_mask], y_test_int),
                                     y = aug_y_train_int, epochs = EPOCH, callbacks = callBacks)
```

```
Epoch 1/40
450/450 [==============================] - 181s 395ms/step - loss: 2.3045 - micro_f1: 0.0979 - val_loss: 2.3028 -
val_micro_f1: 0.1034 - lr: 0.0010
Epoch 2/40
450/450 [==============================] - 177s 393ms/step - loss: 2.3030 - micro_f1: 0.0983 - val_loss: 2.3028 -
val_micro_f1: 0.0938 - lr: 0.0010
Epoch 3/40
450/450 [==============================] - 177s 394ms/step - loss: 2.3029 - micro_f1: 0.1028 - val_loss: 2.3026 -
val_micro_f1: 0.1010 - lr: 0.0010
Epoch 4/40
450/450 [==============================] - 178s 395ms/step - loss: 2.3027 - micro_f1: 0.0956 - val_loss: 2.3025 -
val_micro_f1: 0.0986 - lr: 0.0010
Epoch 5/40
450/450 [==============================] - 178s 395ms/step - loss: 2.3028 - micro_f1: 0.0925 - val_loss: 2.3026 -
val_micro_f1: 0.1130 - lr: 0.0010
Epoch 6/40
450/450 [==============================] - 179s 397ms/step - loss: 2.3029 - micro_f1: 0.0991 - val_loss: 2.3026 -
val_micro_f1: 0.0986 - lr: 0.0010
Epoch 7/40
450/450 [==============================] - 180s 399ms/step - loss: 2.3027 - micro_f1: 0.0985 - val_loss: 2.3026 -
val_micro_f1: 0.0938 - lr: 0.0010
Epoch 8/40
450/450 [==============================] - 179s 398ms/step - loss: 2.3025 - micro_f1: 0.1007 - val_loss: 2.3026 -
val_micro_f1: 0.0962 - lr: 2.0000e-04
Epoch 9/40
450/450 [==============================] - ETA: 0s - loss: 2.3024 - micro_f1: 0.0983

        Terminating training at epoch 9 with a validation micro F1 score of 0.12500 %

450/450 [==============================] - 178s 397ms/step - loss: 2.3024 - micro_f1: 0.0983 - val_loss: 2.3026 -
val_micro_f1: 0.1250 - lr: 2.0000e-04
```

In [56]:

```
save_model_history('3_data_augmented', model_augmented, augmented_LSTM)
```

4. Data augmentation with spectogram data

1. use convert_to_spectrogram and convert the padded data from train and test data to spectogram data.
2. The shape of train data will be 14400 x 64 x 35 and shape of test_data will be 400 x 64 x35
3. Define the model similar to model 2 and fit the data

In [57]:

```
# Using augmented_train_data_seq, y_train_int, test_data_seq and y_test_int from model_3

augmented_train_data_spectrogram  = [convert_to_spectrogram(data) for data in tqdm(augmented_train_data_seq,
                                                                                    'X_train_pad_seq : ')]
test_data_spectrogram  = [convert_to_spectrogram(data) for data in tqdm(test_data_seq, 'X_test_pad_seq : ')]
```

```
augmented_train_data_spectrogram = np.array(augmented_train_data_spectrogram)
test_data_spectrogram = np.array(test_data_spectrogram)
```

```
X_train_pad_seq : 100%|██████████| 14400/14400 [01:39<00:00, 145.37it/s]
X_test_pad_seq : 100%|██████████| 400/400 [00:02<00:00, 147.29it/s]
```

In [58]:
```
print(f"Shape of 'augmented_train_data_spectrogram' :: {augmented_train_data_spectrogram.shape}")
print(f"Shape of 'test_data_spectrogram' :: {test_data_spectrogram.shape}")

print(f"\nShape of 'aug_y_train_int' :: {aug_y_train_int.shape}")
print(f"Shape of 'y_test_int' :: {y_test_int.shape}")
```

```
Shape of 'augmented_train_data_spectrogram' :: (14400, 64, 35)
Shape of 'test_data_spectrogram' :: (400, 64, 35)

Shape of 'aug_y_train_int' :: (14400,)
Shape of 'y_test_int' :: (400,)
```

In [59]:
```
model_aug_spectro = model_2_4(max_length, 'Data_Augment_nd_Spectro')

model_aug_spectro.summary()
```

```
Model: "Data_Augment_nd_Spectro"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 64, 35)]          0         
                                                                 
 lstm (LSTM)                 (None, 64, 64)            25600     
                                                                 
 lstm_1 (LSTM)               (None, 64, 64)            33024     
                                                                 
 global_average_pooling1d (G  (None, 64)               0         
 lobalAveragePooling1D)                                          
                                                                 
 dense (Dense)               (None, 50)                3250      
                                                                 
 batch_normalization (BatchN  (None, 50)               200       
 ormalization)                                                   
                                                                 
 dense_1 (Dense)             (None, 20)                1020      
                                                                 
 dense_2 (Dense)             (None, 10)                210       
                                                                 
=================================================================
Total params: 63,304
Trainable params: 63,204
Non-trainable params: 100
_________________________________________________________________
```

In [60]:
```
callBacks = call_back_list(0.8, 0.05, '4_Data_Augment_Spectro') #[:-1] # Not applying ReduceLROnPlateau callback

model_aug_spectro.compile(optimizer = 'Adamax', loss = 'sparse_categorical_crossentropy', metrics = [micro_f1])

EPOCH = 100

aug_spectro_LSTM = model_aug_spectro.fit(x = augmented_train_data_spectrogram, y = aug_y_train_int,
                                         validation_data = (test_data_spectrogram, y_test_int),
                                         epochs = EPOCH,  callbacks = callBacks)
```

```
Epoch 1/100
  6/450 [..............................] - ETA: 5s - loss: 2.4196 - micro_f1: 0.0990

WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0088s v
s `on_train_batch_end` time: 0.0144s). Check your callbacks.
450/450 [==============================] - 8s 11ms/step - loss: 1.9535 - micro_f1: 0.3001 - val_loss: 1.7482 - va
l_micro_f1: 0.3894 - lr: 0.0010
Epoch 2/100
450/450 [==============================] - 4s 9ms/step - loss: 1.4458 - micro_f1: 0.4909 - val_loss: 1.1964 - val
_micro_f1: 0.6082 - lr: 0.0010
Epoch 3/100
450/450 [==============================] - 4s 9ms/step - loss: 1.1049 - micro_f1: 0.6174 - val_loss: 0.9548 - val
_micro_f1: 0.6923 - lr: 0.0010
Epoch 4/100
450/450 [==============================] - 5s 11ms/step - loss: 0.8611 - micro_f1: 0.6970 - val_loss: 0.8578 - va
l_micro_f1: 0.7139 - lr: 0.0010
```

```
Epoch 5/100
450/450 [==============================] - 4s 9ms/step - loss: 0.7256 - micro_f1: 0.7447 - val_loss: 0.6956 - val
_micro_f1: 0.7620 - lr: 0.0010
Epoch 6/100
450/450 [==============================] - 4s 9ms/step - loss: 0.6030 - micro_f1: 0.7918 - val_loss: 0.5067 - val
_micro_f1: 0.8269 - lr: 0.0010
Epoch 7/100
448/450 [============================>.] - ETA: 0s - loss: 0.4800 - micro_f1: 0.8381

        Terminating training at epoch 7 with a validation micro F1 score of 0.86058 %

450/450 [==============================] - 4s 9ms/step - loss: 0.4798 - micro_f1: 0.8381 - val_loss: 0.4304 - val
_micro_f1: 0.8606 - lr: 2.0000e-04
```

In [61]:

```
save_model_history('4_data_aug_spectro', model_aug_spectro, aug_spectro_LSTM)
```

In [62]:

```
model_list = [('RAW_Data_Alone', raw_LSTM), ('Spectrogram_Data_Alone', spectrogram_LSTM),
              ('Data_Augmented', augmented_LSTM), ('Data_Augment_nd_Spectro', aug_spectro_LSTM)]

table = PrettyTable(['Sl.No', 'Model Name', 'Epochs Ran', 'Micro-F1', 'Val. Micro-F1',
                     'Change in Loss (initial-final)'], hrules = True)

table.set_style(SINGLE_BORDER)

for idx, (name, model)  in enumerate(model_list):

    model = model.history
    epochs = len(model['val_loss'])
    micro_f1 = round(model['micro_f1'][-1], 5)
    val_micro_f1 = round(model['val_micro_f1'][-1], 5)
    loss = round(model['loss'][0] - model['loss'][-1], 5)

    table.add_row([idx+1, name, epochs, micro_f1, val_micro_f1, loss])

print(table)
```

Sl.No	Model Name	Epochs Ran	Micro-F1	Val. Micro-F1	Change in Loss (initial-final)
1	RAW_Data_Alone	8	0.10204	0.10307	0.00509
2	Spectrogram_Data_Alone	37	0.92424	0.85033	1.96278
3	Data_Augmented	9	0.09833	0.125	0.00206
4	Data_Augment_nd_Spectro	7	0.83812	0.86058	1.47374