Speech_detection

August 9, 2020

[3]:
```
import numpy as np
import pandas as pd
import librosa
import os

from pathlib import Path
```

We shared recordings.zip, please unzip those.

[5]:
```
RECORDINGS_DIR = Path('/content/recordings/')
all_files = os.listdir(RECORDINGS_DIR)
```

Grader function 1

[ ]:
```
def grader_files():
    temp = len(all_files) == 2000
    temp1 = all([x[-3:] == "wav" for x in all_files])
    temp = temp and temp1
    return temp
grader_files()
```

[ ]:
```
True
```

Create a dataframe(name=df_audio) with two columns(path, label).
You can get the label from the first letter of name.
Eg: 0_jackson_0 –> 0
0_jackson_43 –> 0

[6]:
```
#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
def create_dataframe(file_names):
  path, label = [], []
  for file_name in file_names:
    path.append(RECORDINGS_DIR/file_name)
    label.append(file_name[0])
```

```
  df = pd.DataFrame({
      'path': path,
      'label': label
  })

  return df

df_audio  = create_dataframe(all_files)
```

[ ]:
```
#info
df_audio.info()
```

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

Grader function 2

[ ]:
```
def grader_df():
    flag_shape = df_audio.shape==(2000,2)
    flag_columns = all(df_audio.columns==['path', 'label'])
    list_values = list(df_audio.label.value_counts())
    flag_label = len(list_values)==10
    flag_label2 = all([i==200 for i in list_values])
    final_flag = flag_shape and flag_columns and flag_label and flag_label2
    return final_flag
grader_df()
```

[ ]: True

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

[7]:
```
#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%

from sklearn.model_selection import train_test_split

X, y = df_audio['path'], df_audio['label']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
↪random_state=45, test_size=0.3)
```

Grader function 3

[ ]:
```
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()
```

[ ]: True

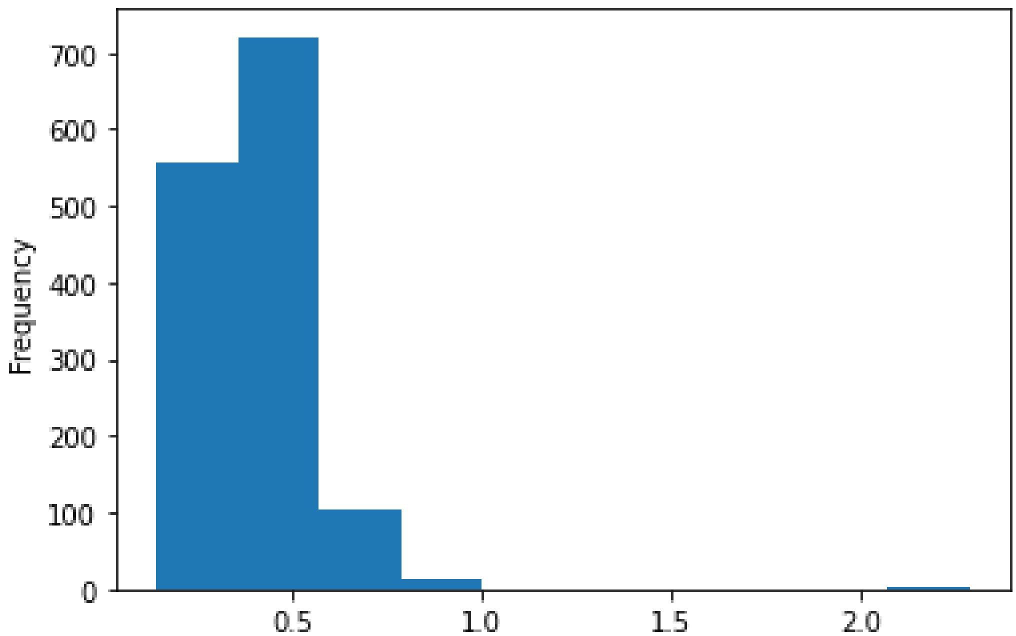
[8]:
```
sample_rate = 22050
def load_wav(x, get_duration=True):
    '''This return the array values of audio with sampling rate of 22050 and␣
↪Duration'''
    #loading the wav file with sampling rate of 22050
    samples, sample_rate = librosa.load(x, sr=22050)
    if get_duration:
        duration = librosa.get_duration(samples, sample_rate)
        return [samples, duration]
    else:
        return samples
```

[10]:
```
%%time
#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(data):
  raw_data, duration = [], []
  for wav in data:
    wav_data = load_wav(wav)
    raw_data.append(wav_data[0])
    duration.append(wav_data[1])

  return pd.DataFrame({
      'raw_data': raw_data,
```

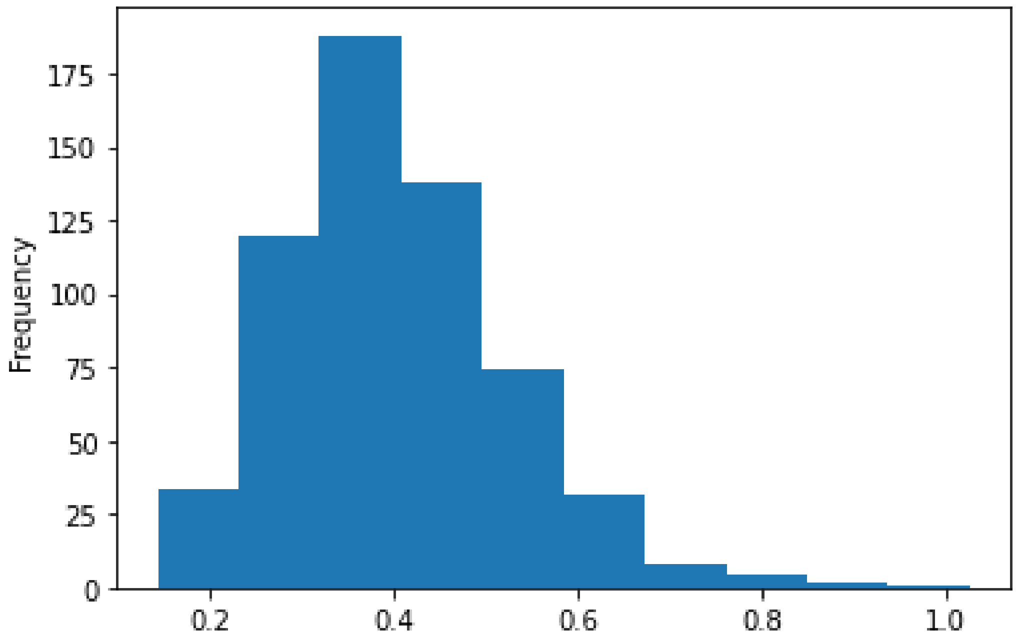
```
        'duration': duration,
    })


X_train_processed = process(X_train)
X_test_processed = process(X_test)
```

```
CPU times: user 32.8 s, sys: 39.2 s, total: 1min 12s
Wall time: 3min 54s
```

[ ]:
```
#plot the histogram of the duration for trian
X_train_processed['duration'].plot.hist();
```



[ ]:
```
#plot the histogram of the duration for trian
X_test_processed['duration'].plot.hist();
```



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

cal_percentiles(X_train_processed['duration'].values, 0, 101, 10)
```

```
0 th percentile is 0.1435374149659864
10 th percentile is 0.25839909297052155
20 th percentile is 0.30187755102040814
30 th percentile is 0.3346485260770975
40 th percentile is 0.36226757369614515
50 th percentile is 0.3925396825396825
60 th percentile is 0.41944671201814054
70 th percentile is 0.4493378684807256
80 th percentile is 0.48478911564625854
90 th percentile is 0.5557097505668935
100 th percentile is 2.282766439909297
```

[ ]:
```
##print 90 to 100 percentile values with step size of 1.
cal_percentiles(X_train_processed['duration'].values, 90, 101, 1)
```

```
90 th percentile is 0.5549160997732426
91 th percentile is 0.5659854875283448
92 th percentile is 0.581082993197279
93 th percentile is 0.601049433106576
```

```
94 th percentile is 0.6133478458049886
95 th percentile is 0.6230385487528345
96 th percentile is 0.6400816326530612
97 th percentile is 0.6635741496598639
98 th percentile is 0.6897750566893422
99 th percentile is 0.7963859410430838
100 th percentile is 2.282766439909297
```

Grader function 4

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

[ ]:
```
True
```

[11]:
```
## 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.

from tensorflow.keras.preprocessing.sequence import pad_sequences

MAX_SEQUENCE_LENGTH  = 17640

X_train_pad_seq = pad_sequences(X_train_processed['raw_data'], MAX_SEQUENCE_LENGTH, 'float32')
X_test_pad_seq = pad_sequences(X_test_processed['raw_data'], MAX_SEQUENCE_LENGTH, 'float32')

X_train_mask = (X_train_pad_seq != 0.0)
X_test_mask = (X_test_pad_seq != 0.0)
```

Grader function 5

[ ]:
```
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()
```

[ ]: True

0.0.1 1. Giving Raw data directly.

[12]:
```
from tensorflow.keras.layers import Input, LSTM, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
```

[ ]:
```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

[ ]:
```
X_train_lstm_seq_inp = np.expand_dims(X_train_pad_seq, 2)
X_test_lstm_seq_inp = np.expand_dims(X_test_pad_seq, 2)

X_train_lstm_inp = [X_train_lstm_seq_inp, X_train_mask]
X_test_lstm_inp = [X_test_lstm_seq_inp, X_test_mask]
```

1 Model 1

[13]:
```
## as discussed above, please write the LSTM

class SpeechDetectionModel(Model):

  def __init__(self, units):
    super(SpeechDetectionModel, self).__init__()
    self.lstm = LSTM(units)
    self.dense = Dense(1, activation='relu')
    self.output_layer = Dense(10, activation='softmax')

  def call(self, inputs):
    sequences, mask = inputs
    x = self.lstm(sequences, mask=mask)
    x = self.dense(x)
    x = self.output_layer(x)

    return x
```

[14]:
```
from sklearn.metrics import f1_score
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard,␣
↪ReduceLROnPlateau, Callback

class F1Callback(Callback):
```

```
  def __init__(self, validation_data):
    super(F1Callback, self).__init__()
    self.validation_data = validation_data

  def on_epoch_end(self, epoch, logs={}):
    probs = self.model.predict(self.validation_data[0])
    preds = np.argmax(probs, axis=1)
    y_true = np.argmax(self.validation_data[1], axis=1)
    score = f1_score(y_true, preds, average='micro')
    logs['micro_f1_score'] = score
```

[ ]:
```
f1_callback = F1Callback(validation_data=(X_test_lstm_inp, y_test))
tensorboard = TensorBoard(write_images=True)

callbacks = [
             f1_callback,
             tensorboard,
]

model = SpeechDetectionModel(1)
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

[ ]:
```
#train your model
!rm -rf ./logs/*

model.fit(X_train_lstm_inp, y_train, epochs=3,
          validation_data=(X_test_lstm_inp, y_test),
          callbacks=callbacks)
```

```
Epoch 1/3
44/44 [==============================] - 1744s 40s/step - loss: 2.3032 -
val_loss: 2.3026 - micro_f1_score: 0.1000
Epoch 2/3
44/44 [==============================] - 1686s 38s/step - loss: 2.3029 -
val_loss: 2.3026 - micro_f1_score: 0.1133
Epoch 3/3
44/44 [==============================] - 1677s 38s/step - loss: 2.3029 -
val_loss: 2.3026 - micro_f1_score: 0.1000
```

1.0.1 2. Converting into spectrogram and giving spectrogram data as input

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

```
    return logmel_spectrum
```

[ ]:
```
def make_spectrograms(data):
  spectograms = []
  for i in data:
    spectograms.append(convert_to_spectrogram(i))

  return np.array(spectograms)
```

[ ]:
```
##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 = make_spectrograms(X_train_pad_seq)
X_test_spectrogram = make_spectrograms(X_test_pad_seq)
```

Grader function 6

[ ]:
```
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()
```

[ ]: True

2 Model 2

[ ]:
```
from tensorflow.keras import backend

class SpectrogramModel(Model):

  def __init__(self, units):
    super(SpectrogramModel, self).__init__()
    self.lstm_1 = LSTM(units, return_sequences=True)
    self.lstm_2 = LSTM(units, return_sequences=True)
    self.lstm_3 = LSTM(units, return_sequences=True)

    self.dense = Dense(64, activation='relu')
    self.output_layer = Dense(10, activation='softmax')

  def feature_average(self, sequences):
    return backend.mean(sequences, axis=2)

  def call(self, inputs):
    x = self.lstm_1(inputs)
```

```
    x = self.lstm_2(x)
    x = self.lstm_3(x)
    x = self.feature_average(x)
    x = self.dense(x)
    x = self.output_layer(x)

    return x
```

[ ]:
```
f1_callback = F1Callback(validation_data=(X_test_spectrogram, y_test))
early_stopping = EarlyStopping(patience=5)
tensorboard = TensorBoard(write_images=True)
reduce_lr = ReduceLROnPlateau(patience=3)

callbacks = [
             early_stopping,
             f1_callback,
             tensorboard,
             reduce_lr,
]

model = SpectrogramModel(32)
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

[ ]:
```
#train your model
!rm -rf ./logs/*

model.fit(X_train_spectrogram, y_train, epochs=100,
          validation_data=(X_test_spectrogram, y_test),
          callbacks=callbacks)
```

```
Epoch 1/100
 2/44 [>...] - ETA: 10s - loss:
2.2998WARNING:tensorflow:Method (on_train_batch_end) is slow compared to the
batch update (0.211291). Check your callbacks.
44/44 [==============================] - 6s 139ms/step - loss: 2.2654 -
val_loss: 2.1142 - micro_f1_score: 0.2100 - lr: 0.0010
Epoch 2/100
44/44 [==============================] - 4s 86ms/step - loss: 1.9536 - val_loss:
1.8749 - micro_f1_score: 0.2983 - lr: 0.0010
Epoch 3/100
44/44 [==============================] - 4s 86ms/step - loss: 1.7555 - val_loss:
1.7060 - micro_f1_score: 0.3517 - lr: 0.0010
Epoch 4/100
44/44 [==============================] - 4s 86ms/step - loss: 1.6187 - val_loss:
1.5659 - micro_f1_score: 0.4283 - lr: 0.0010
Epoch 5/100
44/44 [==============================] - 4s 87ms/step - loss: 1.4973 - val_loss:
```

```
1.4748 - micro_f1_score: 0.4317 - lr: 0.0010
Epoch 6/100
44/44 [==============================] - 4s 89ms/step - loss: 1.3772 - val_loss:
1.4971 - micro_f1_score: 0.4383 - lr: 0.0010
Epoch 7/100
44/44 [==============================] - 4s 88ms/step - loss: 1.2738 - val_loss:
1.2320 - micro_f1_score: 0.5567 - lr: 0.0010
Epoch 8/100
44/44 [==============================] - 4s 88ms/step - loss: 1.1329 - val_loss:
1.1170 - micro_f1_score: 0.5817 - lr: 0.0010
Epoch 9/100
44/44 [==============================] - 4s 89ms/step - loss: 1.0594 - val_loss:
1.1674 - micro_f1_score: 0.5467 - lr: 0.0010
Epoch 10/100
44/44 [==============================] - 4s 88ms/step - loss: 0.9983 - val_loss:
1.0755 - micro_f1_score: 0.6300 - lr: 0.0010
Epoch 11/100
44/44 [==============================] - 4s 89ms/step - loss: 0.9359 - val_loss:
0.9762 - micro_f1_score: 0.6400 - lr: 0.0010
Epoch 12/100
44/44 [==============================] - 4s 91ms/step - loss: 0.9504 - val_loss:
1.0116 - micro_f1_score: 0.6117 - lr: 0.0010
Epoch 13/100
44/44 [==============================] - 4s 92ms/step - loss: 0.8913 - val_loss:
0.9672 - micro_f1_score: 0.6433 - lr: 0.0010
Epoch 14/100
44/44 [==============================] - 4s 90ms/step - loss: 0.8367 - val_loss:
0.9108 - micro_f1_score: 0.6667 - lr: 0.0010
Epoch 15/100
44/44 [==============================] - 4s 89ms/step - loss: 0.7653 - val_loss:
0.8710 - micro_f1_score: 0.6683 - lr: 0.0010
Epoch 16/100
44/44 [==============================] - 4s 88ms/step - loss: 0.7633 - val_loss:
0.8163 - micro_f1_score: 0.7217 - lr: 0.0010
Epoch 17/100
44/44 [==============================] - 4s 88ms/step - loss: 0.7788 - val_loss:
0.8225 - micro_f1_score: 0.7017 - lr: 0.0010
Epoch 18/100
44/44 [==============================] - 4s 87ms/step - loss: 0.7442 - val_loss:
0.8036 - micro_f1_score: 0.7133 - lr: 0.0010
Epoch 19/100
44/44 [==============================] - 4s 89ms/step - loss: 0.6879 - val_loss:
0.8068 - micro_f1_score: 0.6967 - lr: 0.0010
Epoch 20/100
44/44 [==============================] - 4s 88ms/step - loss: 0.6590 - val_loss:
0.7383 - micro_f1_score: 0.7367 - lr: 0.0010
Epoch 21/100
44/44 [==============================] - 4s 89ms/step - loss: 0.6262 - val_loss:
```

```
0.7602 - micro_f1_score: 0.7400 - lr: 0.0010
Epoch 22/100
44/44 [==============================] - 4s 88ms/step - loss: 0.6299 - val_loss:
0.7003 - micro_f1_score: 0.7517 - lr: 0.0010
Epoch 23/100
44/44 [==============================] - 4s 87ms/step - loss: 0.6063 - val_loss:
0.7776 - micro_f1_score: 0.7150 - lr: 0.0010
Epoch 24/100
44/44 [==============================] - 4s 88ms/step - loss: 0.7053 - val_loss:
0.7194 - micro_f1_score: 0.7367 - lr: 0.0010
Epoch 25/100
44/44 [==============================] - 4s 88ms/step - loss: 0.5717 - val_loss:
0.6845 - micro_f1_score: 0.7733 - lr: 0.0010
Epoch 26/100
44/44 [==============================] - 4s 88ms/step - loss: 0.5584 - val_loss:
0.7880 - micro_f1_score: 0.7283 - lr: 0.0010
Epoch 27/100
44/44 [==============================] - 4s 88ms/step - loss: 0.5961 - val_loss:
0.7566 - micro_f1_score: 0.7483 - lr: 0.0010
Epoch 28/100
44/44 [==============================] - 4s 88ms/step - loss: 0.5531 - val_loss:
0.6883 - micro_f1_score: 0.7733 - lr: 0.0010
Epoch 29/100
44/44 [==============================] - 4s 87ms/step - loss: 0.4548 - val_loss:
0.6182 - micro_f1_score: 0.7950 - lr: 1.0000e-04
Epoch 30/100
44/44 [==============================] - 4s 88ms/step - loss: 0.4080 - val_loss:
0.5989 - micro_f1_score: 0.7900 - lr: 1.0000e-04
Epoch 31/100
44/44 [==============================] - 4s 88ms/step - loss: 0.3968 - val_loss:
0.5927 - micro_f1_score: 0.7983 - lr: 1.0000e-04
Epoch 32/100
44/44 [==============================] - 4s 88ms/step - loss: 0.3894 - val_loss:
0.5918 - micro_f1_score: 0.8000 - lr: 1.0000e-04
Epoch 33/100
44/44 [==============================] - 4s 88ms/step - loss: 0.3855 - val_loss:
0.5904 - micro_f1_score: 0.8033 - lr: 1.0000e-04
Epoch 34/100
44/44 [==============================] - 4s 88ms/step - loss: 0.3772 - val_loss:
0.5933 - micro_f1_score: 0.8033 - lr: 1.0000e-04
Epoch 35/100
44/44 [==============================] - 4s 88ms/step - loss: 0.3716 - val_loss:
0.5934 - micro_f1_score: 0.8050 - lr: 1.0000e-04
Epoch 36/100
44/44 [==============================] - 4s 90ms/step - loss: 0.3662 - val_loss:
0.5885 - micro_f1_score: 0.8133 - lr: 1.0000e-04
Epoch 37/100
44/44 [==============================] - 4s 90ms/step - loss: 0.3640 - val_loss:
```

```
0.5986 - micro_f1_score: 0.8000 - lr: 1.0000e-04
Epoch 38/100
44/44 [==============================] - 4s 89ms/step - loss: 0.3637 - val_loss:
0.5894 - micro_f1_score: 0.8100 - lr: 1.0000e-04
Epoch 39/100
44/44 [==============================] - 4s 88ms/step - loss: 0.3558 - val_loss:
0.5860 - micro_f1_score: 0.8083 - lr: 1.0000e-04
Epoch 40/100
44/44 [==============================] - 4s 88ms/step - loss: 0.3540 - val_loss:
0.5866 - micro_f1_score: 0.8067 - lr: 1.0000e-04
Epoch 41/100
44/44 [==============================] - 4s 89ms/step - loss: 0.3481 - val_loss:
0.5864 - micro_f1_score: 0.8117 - lr: 1.0000e-04
Epoch 42/100
44/44 [==============================] - 4s 87ms/step - loss: 0.3456 - val_loss:
0.5928 - micro_f1_score: 0.8117 - lr: 1.0000e-04
Epoch 43/100
44/44 [==============================] - 4s 88ms/step - loss: 0.3370 - val_loss:
0.5899 - micro_f1_score: 0.8117 - lr: 1.0000e-05
Epoch 44/100
44/44 [==============================] - 4s 89ms/step - loss: 0.3328 - val_loss:
0.5873 - micro_f1_score: 0.8100 - lr: 1.0000e-05
```

[ ]: <tensorflow.python.keras.callbacks.History at 0x7f0c90c6dcf8>

2.0.1 3. data augmentation

[15]:
```
## 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
```

[16]:
```
temp_path = df_audio.iloc[0].path
aug_temp = generate_augmented_data(temp_path)
```

[17]:
```
len(aug_temp)
```

[17]: 9

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

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

[18]:
```
def augment_data(data):
  augmented_data, labels = [], []
  for i in range(len(data)):
    row = data.iloc[i]
    ret = generate_augmented_data(row['path'])
    augmented_data += [i for i in ret]
    labels += [row['label'] for _ in ret]

  return pd.DataFrame({
      'augmented_data': augmented_data,
      'labels': labels,
  })
```

[19]:
```
%%time

augmented_df = augment_data(df_audio)
```

```
CPU times: user 6min 14s, sys: 1min 24s, total: 7min 38s
Wall time: 11min 9s
```

[20]:
```
X, y = augmented_df['augmented_data'], augmented_df['labels']

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
↪random_state=45, test_size=0.3)
```

[ ]:
```
cal_percentiles(X_train.map(len), 90, 101, 1)
```

```
90 th percentile is 14336.0
91 th percentile is 15360.0
92 th percentile is 15360.0
93 th percentile is 15872.0
94 th percentile is 15872.0
95 th percentile is 16384.0
96 th percentile is 17408.0
97 th percentile is 17920.0
98 th percentile is 19456.0
99 th percentile is 21504.0
100 th percentile is 72192.0
```

Over 98 percentile of data has len 20000 or below

[21]:
```
MAX_SEQUENCE_LENGTH  = 20000
```

```
X_train_pad_seq = pad_sequences(X_train, MAX_SEQUENCE_LENGTH, 'float32')
X_test_pad_seq = pad_sequences(X_test, MAX_SEQUENCE_LENGTH, 'float32')

X_train_mask = (X_train_pad_seq != 0.0)
X_test_mask = (X_test_pad_seq != 0.0)
```

[22]:
```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

[23]:
```
X_train_lstm_seq_inp = np.expand_dims(X_train_pad_seq, 2)
X_test_lstm_seq_inp = np.expand_dims(X_test_pad_seq, 2)

X_train_lstm_inp = [X_train_lstm_seq_inp, X_train_mask]
X_test_lstm_inp = [X_test_lstm_seq_inp, X_test_mask]
```

3 Model 3

[24]:
```
f1_callback = F1Callback(validation_data=(X_test_lstm_inp, y_test))
early_stopping = EarlyStopping(patience=5)
tensorboard = TensorBoard(write_images=True)
reduce_lr = ReduceLROnPlateau(patience=3)

callbacks = [
             f1_callback,
             tensorboard,
]

model = SpeechDetectionModel(1)
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

[25]:
```
!rm -rf ./logs/*

model.fit(X_train_lstm_inp, y_train, epochs=2,
          validation_data=(X_test_lstm_inp, y_test),
          callbacks=callbacks)
```

```
Epoch 1/2
  1/394 [...] - ETA: 0s - loss:
2.3026WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after
2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
394/394 [==============================] - 6689s 17s/step - loss: 2.3028 -
val_loss: 2.3026
```

```
Epoch 2/2
394/394 [==============================] - 6699s 17s/step - loss: 2.3028 -
val_loss: 2.3026
```

[25]:
```
<tensorflow.python.keras.callbacks.History at 0x7fe90d9b9e80>
```

[ ]:
```python
X_train_spectrogram = make_spectrograms(X_train_pad_seq)
X_test_spectrogram = make_spectrograms(X_test_pad_seq)
```

4 Model 4

[ ]:
```python
f1_callback = F1Callback(validation_data=(X_test_spectrogram, y_test))
early_stopping = EarlyStopping(patience=3)
tensorboard = TensorBoard(write_images=True)
reduce_lr = ReduceLROnPlateau()

callbacks = [
             early_stopping,
             f1_callback,
             tensorboard,
             reduce_lr,
]

model = SpectrogramModel(32)
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

[ ]:
```python
#train your model
!rm -rf ./logs/*

model.fit(X_train_spectrogram, y_train, epochs=100,
          validation_data=(X_test_spectrogram, y_test),
          callbacks=callbacks)
```

```
Epoch 1/100
  2/394 [...] - ETA: 1:59 - loss:
2.3031WARNING:tensorflow:Method (on_train_batch_end) is slow compared to the
batch update (0.264320). Check your callbacks.
394/394 [==============================] - 38s 95ms/step - loss: 1.6598 -
val_loss: 1.2676 - micro_f1_score: 0.4985 - lr: 0.0010
Epoch 2/100
394/394 [==============================] - 36s 92ms/step - loss: 0.9634 -
val_loss: 0.7757 - micro_f1_score: 0.7128 - lr: 0.0010
Epoch 3/100
394/394 [==============================] - 36s 92ms/step - loss: 0.7105 -
val_loss: 0.6901 - micro_f1_score: 0.7474 - lr: 0.0010
Epoch 4/100
394/394 [==============================] - 36s 92ms/step - loss: 0.6013 -
```

```
val_loss: 0.6187 - micro_f1_score: 0.7748 - lr: 0.0010
Epoch 5/100
394/394 [==============================] - 36s 92ms/step - loss: 0.5451 -
val_loss: 0.4959 - micro_f1_score: 0.8248 - lr: 0.0010
Epoch 6/100
394/394 [==============================] - 36s 92ms/step - loss: 0.4756 -
val_loss: 0.5461 - micro_f1_score: 0.8039 - lr: 0.0010
Epoch 7/100
394/394 [==============================] - 36s 92ms/step - loss: 0.4161 -
val_loss: 0.3773 - micro_f1_score: 0.8631 - lr: 0.0010
Epoch 8/100
394/394 [==============================] - 37s 93ms/step - loss: 0.3940 -
val_loss: 0.3569 - micro_f1_score: 0.8750 - lr: 0.0010
Epoch 9/100
394/394 [==============================] - 36s 93ms/step - loss: 0.3529 -
val_loss: 0.3422 - micro_f1_score: 0.8761 - lr: 0.0010
Epoch 10/100
394/394 [==============================] - 38s 97ms/step - loss: 0.3356 -
val_loss: 0.3032 - micro_f1_score: 0.8963 - lr: 0.0010
Epoch 11/100
394/394 [==============================] - 37s 93ms/step - loss: 0.3190 -
val_loss: 0.2934 - micro_f1_score: 0.8933 - lr: 0.0010
Epoch 12/100
394/394 [==============================] - 37s 93ms/step - loss: 0.2960 -
val_loss: 0.3084 - micro_f1_score: 0.8881 - lr: 0.0010
Epoch 13/100
394/394 [==============================] - 36s 93ms/step - loss: 0.2638 -
val_loss: 0.3160 - micro_f1_score: 0.8833 - lr: 0.0010
Epoch 14/100
394/394 [==============================] - 36s 93ms/step - loss: 0.2635 -
val_loss: 0.3575 - micro_f1_score: 0.8711 - lr: 0.0010
```

```
[ ]: <tensorflow.python.keras.callbacks.History at 0x7f0c87e56128>
```