STEVENS INSTITUTE OF TECHNOLOGY

Audio Language Detection

1st Shrutik Pawale - *MS in Data Science*

Roles: Data Pre-processing, EDA , Report

spawale@stevens.edu

2nd Daksh Shah - *ME in Applied Artificial Intelligence*

Roles: Feature Extraction, Model Training, Evaluating Results

dshah94@stevens.edu

May 9, 2023

Contents

List of Figures

1. INTRODUCTION

Audio language Detection is the method of identifying language from an audio clip by an unidentified speaker, regardless of gender, speaking style, and distinct age speaker. Finding the characteristics that may clearly and effectively differentiate various languages is a challenging task.

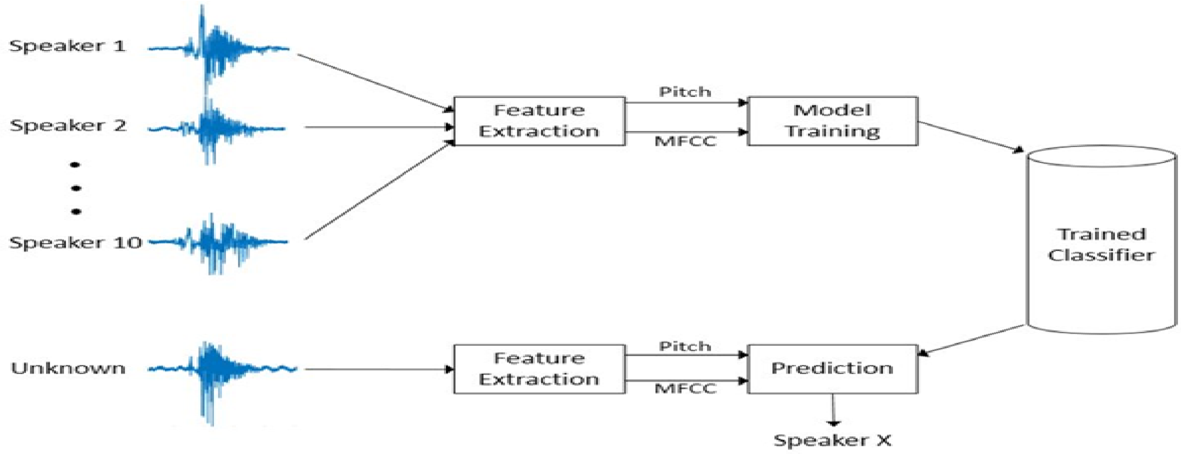

Figure 1.1: Project Outline

The above figure describes the goals of the project. The first step is to read the audio using the Soundfile package. It reads the audio and outputs the signal and sample rate. The next step is to extract features such as MFCC and pitch using the MFCC algorithm. Finally, a convolutional neural network model is trained and the model is evaluated.

2. Dataset and Feature extraction

2.1. Data Description

The given dataset contains 10 seconds of speech recorded in English, German, and Spanish languages. Samples are equally balanced between languages, genders, and speakers. The core of the train set is based on 73080 samples after applying several audio transformations (pitch, speed, and noise). No data augmentation has been applied. The number of unique speakers was increased by adjusting pitch (8 different levels) and speed (8 different levels). LibriVox recordings were used to prepare the dataset and particular attention was paid to a big variety of unique speakers since big variance forces the model to concentrate more on language properties than a specific voice.

The spectrogram implies how much the frequency of a signal changes over time. If the frequency matches at that second, then it will show less intensity. That is the reason, why there is a strip of dark line above 4096 Hz. This helps in identifying properties of nonlinear signals and that's why it is helpful in analyzing real-world data with a lot of frequency components and noise.

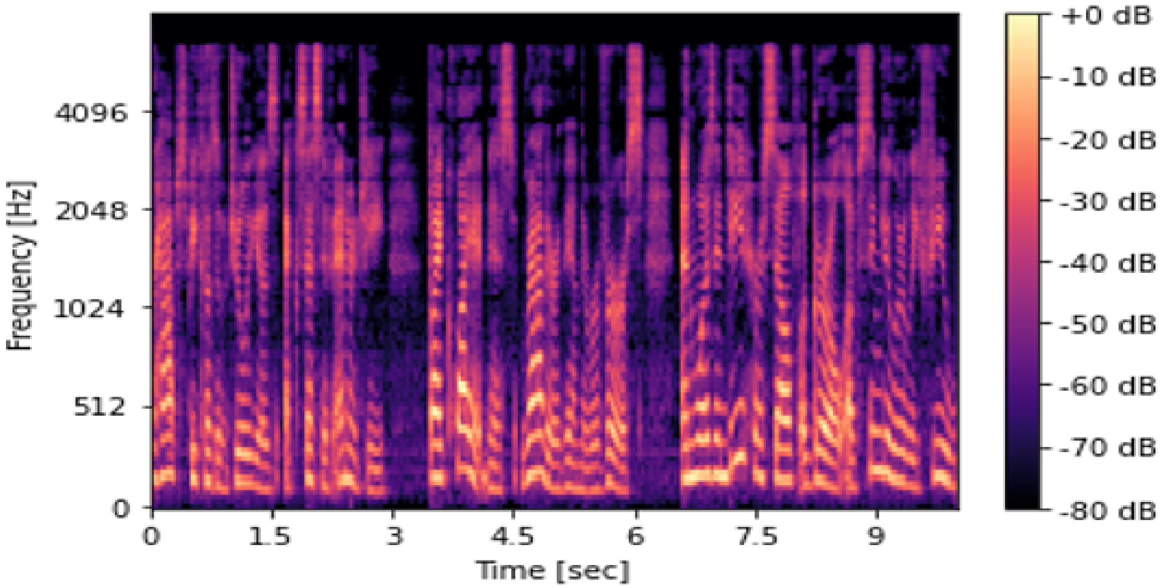

Figure 2.1: Spectrogram of a Sample

2.2. Data Processing

We can extract the following features from the Audio File Name Format:

```
"(language)_(gender)_(recording_ID).fragment(index)[.(transformation)(index)]"
```

filename	lang	gender	user_id	fragment	edit
train\es_f_1d27c6d589eeff17973ffd0b7a77a70a.fr...	es	f	1d27c6d589eeff17973ffd0b7a77a70a	fragment5	speed5
train\es_f_53b555eab2b3baada380f7d3ede20b20.fr...	es	f	53b555eab2b3baada380f7d3ede20b20	fragment14	pitch4
train\de_f_d94712992f41e3d8d21f22274b3d8fd9.fr...	de	f	d94712992f41e3d8d21f22274b3d8fd9	fragment24	noise6
train\en_f_10134f409d9b7b0b95fed6e025febcad.fr...	en	f	10134f409d9b7b0b95fed6e025febcad	fragment25	noise7
train\es_m_b8e0e6f56f02e6f8f79cc360958e5982.fr...	es	m	b8e0e6f56f02e6f8f79cc360958e5982	fragment8	noise4
...	...	...	...	...	...
train\es_m_d5b91a4ffb1ead826b7968ec19cbfa1c.fr...	es	m	d5b91a4ffb1ead826b7968ec19cbfa1c	fragment3	noise10
train\de_m_fc6bd6bb9d66a89bb8d8a8a7efa23e6b.fr...	de	m	fc6bd6bb9d66a89bb8d8a8a7efa23e6b	fragment4	noise6
train\de_m_d22535879801cc9c4452d9ed9de5bf61.fr...	de	m	d22535879801cc9c4452d9ed9de5bf61	fragment20	speed4
train\de_f_2825fa225d6ca4800f0cf0504b76ca65.fr...	de	f	2825fa225d6ca4800f0cf0504b76ca65	fragment11	pitch6
train\es_f_bf4285930fa46f2052e5bdbc37a8a4df.fr...	es	f	bf4285930fa46f2052e5bdbc37a8a4df	fragment21	speed2

Figure 2.2: DataFrame of Audio Samples Description

After reading the audio file using the sound file python package, we get the sample rate and an array of 220500 numbers. The resulting array contains the audio data as a sequence of samples, where each sample represents the amplitude of the audio signal at a specific point in time.

	0	1	2	3	4	5	6	7	8	9	...	220497	220498	220499	sample_rate
0	-0.020905	-0.031860	-0.028931	-0.020203	-0.002075	0.010193	0.013611	0.004120	-0.009247	-0.012665	...	0.000000	0.000000	0.000000	22050
1	-0.007965	-0.007202	0.003601	0.010895	0.014069	0.005890	-0.005829	-0.019653	-0.028564	-0.035706	...	0.018311	0.018646	0.011688	22050
2	-0.070190	-0.066132	-0.063599	-0.062103	-0.059601	-0.055115	-0.050598	-0.046295	-0.043060	-0.038849	...	0.015778	0.015717	0.017517	22050
3	-0.005951	-0.011993	-0.009888	-0.012848	-0.014374	-0.015961	-0.013062	-0.013824	-0.015961	-0.020050	...	-0.001495	-0.006683	-0.006561	22050
4	0.001556	0.001404	0.001617	0.002106	0.002625	0.002869	0.001709	0.000946	0.001740	0.002380	...	-0.038422	-0.038666	-0.038940	22050
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
495	-0.016266	-0.013611	0.000397	0.020569	0.032318	0.035248	0.035309	0.038361	0.045197	0.046844	...	-0.001801	0.019928	0.016479	22050
496	0.000122	0.001282	0.002045	0.002472	0.002960	0.003479	0.003418	0.002960	0.002747	0.002960	...	0.015900	0.013397	0.016937	22050
497	0.012054	0.015717	0.015717	0.018982	0.018433	0.018677	0.016052	0.018707	0.022522	0.032166	...	-0.009430	-0.018799	-0.018768	22050
498	-0.002380	0.003052	0.003204	-0.004486	-0.004395	0.001587	0.006195	-0.012115	-0.008118	-0.005341	...	0.000000	0.000000	0.000000	22050
499	0.006256	-0.006653	-0.027435	-0.017883	0.003571	0.021820	0.022003	0.009186	-0.001740	-0.008087	...	-0.007996	-0.004730	-0.002106	22050

Figure 2.3: DataFrame of Audio Samples Signal and Sample

2.3. Feature Extraction - MFCC

MFCC (Mel-Frequency Cepstral Coefficients) is a technique used in signal processing to analyze and represent the sound of a human voice or other sound signals. The sound is first broken down into many tiny pieces called frames, and then for each frame, the MFCC algorithm measures the power of different frequency bands within that frame. The frequency bands are spaced out in a way that is more like how the human ear perceives sound, which is why it's called Mel-frequency. Next, the algorithm applies some math operations to these frequency band measurements to reduce the dimensionality and capture the most important features of the sound. The resulting features are called cepstral coefficients and they can be used as inputs to machine learning models for tasks like speech recognition or music genre classification.

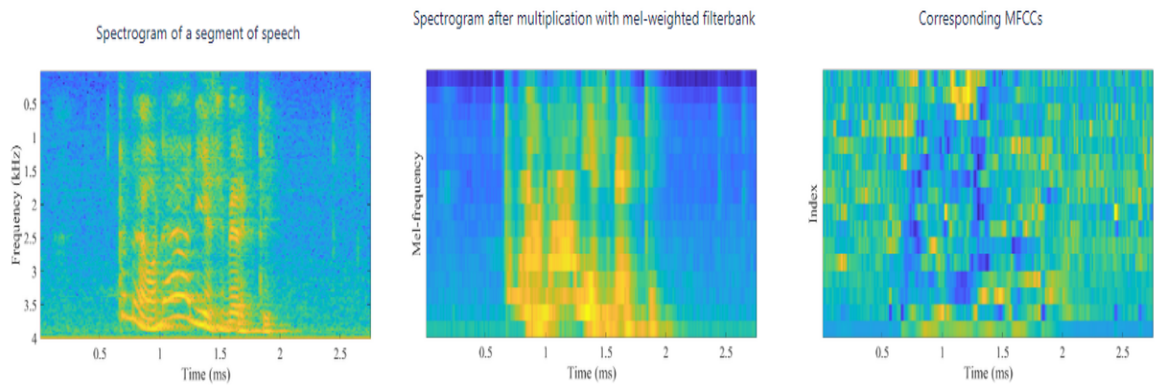

Figure 2.4: MFCC Visualization with Spectogram

Here is a step-wise illustration of how MFCC works for audio detection:

- Select an Audio Sample and apply Pre-emphasis, which amplifies higher frequencies to balance the audio spectrum.
- Perform Frame blocking and windowing to divide the audio into small frames and apply a window function to each frame.
- Apply Fast Fourier Transformation (FFT) to convert the audio from the time domain to the frequency domain.
- Construct the Mel-Scale Filter Bank, which is a set of triangular filters spaced according to the Mel scale, to capture relevant frequency bands.
- Compute the log and modulus of the filtered signals to enhance the representation of the audio's spectral characteristics.
- Apply Discrete Cosine Transformation (DCT) to the logarithmic filter bank energies to obtain the MFCC coefficients.
- Obtain the MFCC Output, which is a feature vector representing the audio sample's spectral properties.

By following these steps, the MFCC algorithm transforms the audio signal into a compact representation suitable for further analysis and classification in audio detection tasks.

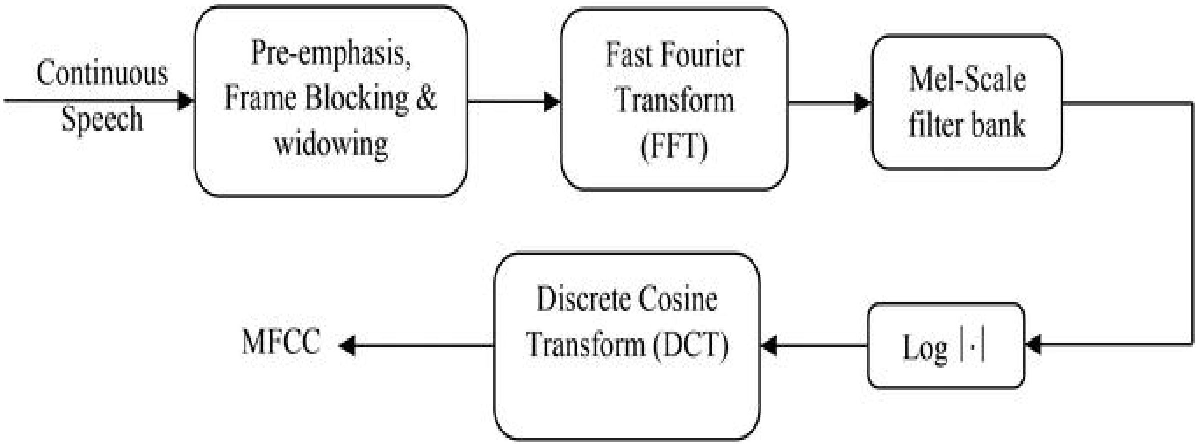

Figure 2.5: MFCC flowchart

3. MODEL IMPLEMENTATION

Since the output from MFCC algorithm is in a 2D array, we need to train it on a compatible model like Convolutional Neural Networks (CNNs). CNNs are particularly effective in capturing spatial dependencies and extracting relevant features from structured data, making them suitable for tasks such as image recognition, object detection, and audio analysis.

Why we used CNN for our audio language detection model:

- Spatial Relationships: CNNs excel at capturing spatial relationships in data. In audio language detection, the arrangement of audio frames or spectrogram slices is crucial for accurate classification. CNNs leverage convolutional layers, which apply filters to small regions of the input data, allowing them to detect local patterns and capture spatial dependencies within audio signals.

- Feature Extraction: CNNs are capable of automatically learning hierarchical representations of data. In the context of audio language detection, CNNs can automatically extract discriminative features from audio spectrograms or other time-frequency representations. This ability to learn and extract relevant features makes CNNs well-suited for distinguishing language-specific patterns and acoustic cues.

- Invariance to Local Variations: CNNs exhibit a degree of translation invariance, meaning they can recognize patterns regardless of their position in the input data. This property is advantageous for audio language detection, as the specific location of language-related features within an audio sample may vary. CNNs can effectively capture invariant features, enabling robust classification even with slight variations in temporal alignment.

- Parameter Sharing and Efficiency: CNNs employ parameter sharing, where the same set of weights is used across different regions of the input data. This sharing reduces the number of learnable parameters in the network, making CNNs computationally efficient and less prone to overfitting, especially in cases with limited training data.

- Previous Success in Audio Analysis: CNNs have demonstrated remarkable success in various audio-related tasks, such as speech recognition, music classification, and sound event detection. The effectiveness of CNNs in these areas highlights their potential for audio language detection, which also relies on analyzing audio signals for language identification.

Considering these advantages, we selected CNNs as the go-to model for our audio language detection project. The model's structure, as depicted in the figure below, consists of 5 convolutional and max pooling layers, followed by 2 dense layers with batch normalization applied at each step.

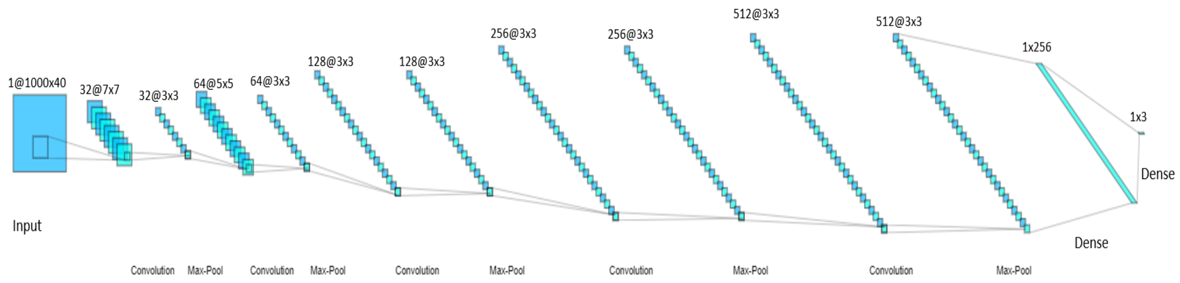

Figure 3.1: CNN Model Structure

The design of the CNN architecture is driven by specific considerations. Firstly, the number of kernels (parameters) doubles in size after each convolutional layer. This progressive incrementation allows deeper layers to capture more complex hidden features such as syllables or even words, while the base layers focus on extracting fundamental features like pitch and frequency. Additionally, the kernel size decreases after each layer, enabling the network to effectively learn and capture finer details in the audio data.

By leveraging this CNN structure, our model can effectively extract and utilize hierarchical representations of audio features, enhancing its ability to classify languages accurately. The combination of convolutional and pooling layers, along with dense layers and batch normalization, enables the model to capture spatial relationships, extract discriminative features, and improve generalization.

4. Results

The trained model utilized 13,200 audio samples for training, achieving an impressive training accuracy of nearly 100%. To evaluate the model's performance, a separate test set consisting of 3,200 samples was used. The evaluation resulted in an accuracy exceeding 99

The effectiveness of the model is further supported by the analysis of the confusion matrix for both the training and test datasets. It reveals minimal mispredictions, with the number of incorrect classifications in the single-digit range and deemed negligible. This indicates the model's ability to accurately recognize and classify audio samples in English, German, and Spanish languages.

Train Set

```
In [72]:  1 confusion_matrix(y_train1,y_pred1)
Out[72]: array([[4262,    4,    8],
                [   0, 4256,    0],
                [   3,    0, 4267]], dtype=int64)

In [73]:  1 print(classification_report(y_train1,y_pred1))
              precision    recall  f1-score   support

          de       1.00      1.00      1.00      4274
          en       1.00      1.00      1.00      4256
          es       1.00      1.00      1.00      4270

    accuracy                           1.00     12800
   macro avg       1.00      1.00      1.00     12800
weighted avg       1.00      1.00      1.00     12800
```

Test Set

```
In [75]:  1 confusion_matrix(y_test1,y_pred2)
Out[75]: array([[1054,    5,    9],
                [   5, 1055,    4],
                [   2,    1, 1065]], dtype=int64)

In [76]:  1 print(classification_report(y_test1,y_pred2))
              precision    recall  f1-score   support

          de       0.99      0.99      0.99      1068
          en       0.99      0.99      0.99      1064
          es       0.99      1.00      0.99      1068

    accuracy                           0.99      3200
   macro avg       0.99      0.99      0.99      3200
weighted avg       0.99      0.99      0.99      3200
```

Figure 4.1: Model Evaluation

5. Conclusion

Based on these results, we can confidently conclude that the developed model has effectively learned the distinguishing features of different languages, demonstrating its capability to identify and classify audio samples accurately. The model's high training accuracy, coupled with its exceptional performance on the test set, signifies its proficiency in language recognition. This model holds significant potential in various practical applications, such as speech processing, voice assistants, and language identification systems. Its ability to recognize languages with a high degree of accuracy can facilitate multilingual support in various domains, contributing to enhanced user experiences and improved language-based services.

Further research and development may involve expanding the model to handle additional languages or exploring techniques to handle variations in dialects or accents within a specific language. Additionally, fine-tuning the model on larger and more diverse datasets could potentially enhance its performance and robustness. In summary, the developed model has proven its effectiveness in audio language detection, achieving high accuracy and showcasing its proficiency in recognizing English, German, and Spanish languages.

References

[1] Singh, Gundeep and Sharma, Sahil and Chahar, Vijay and Kaur, Manjit and Baz, Mohammed and Masud, Mehedi, *Spoken Language Identification Using Deep Learning*, (2021)

[2] G. Montavon, *Deep learning for spoken language identification*, (2009)

[3] P. Kumar, A. Biswas, A. N. Mishra, and M. Chandra, *Spoken language identification using hybrid feature extraction methods*, (2010)

A. Python Code

```
import os
import time
import glob
import pandas as pd
import soundfile as sf
import scipy.signal as signal
import matplotlib.pyplot as plt
import gc
import IPython.display as ipd
import pickle
from sklearn.preprocessing import StandardScaler
import numpy as np
import scipy.signal
import warnings
warnings.filterwarnings('ignore')

from sklearn import preprocessing
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import train_test_split

from keras.models import Model, load_model, Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, Dense,
    Flatten
from keras.layers import Dropout, Input, Activation
from keras.optimizers import Nadam, SGD, Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import np_utils
from keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint
from keras.models import load_model
from keras.layers import BatchNormalization
from keras import regularizers
import math
from keras.callbacks import LearningRateScheduler

def generate_fb_and_mfcc(signal, sample_rate):

    # Pre-Emphasis
    pre_emphasis = 0.97
    emphasized_signal = np.append(
        signal[0],
        signal[1:] - pre_emphasis * signal[:-1])

    # Framing
    frame_size = 0.025
    frame_stride = 0.01

    # Convert from seconds to samples
    frame_length, frame_step = (
        frame_size * sample_rate,
        frame_stride * sample_rate)
```

```
    signal_length = len(emphasized_signal)
    frame_length = int(round(frame_length))
    frame_step = int(round(frame_step))

    # Make sure that we have at least 1 frame
    num_frames = int(
        np.ceil(float(np.abs(signal_length - frame_length)) /
    frame_step))

    pad_signal_length = num_frames * frame_step + frame_length
    z = np.zeros((pad_signal_length - signal_length))

    # Pad Signal to make sure that all frames have equal
    # number of samples without truncating any samples
    # from the original signal
    pad_signal = np.append(emphasized_signal, z)

    indices = (
        np.tile(np.arange(0, frame_length), (num_frames, 1)) +
        np.tile(
            np.arange(0, num_frames * frame_step, frame_step),
            (frame_length, 1)
        ).T
    )
    frames = pad_signal[indices.astype(np.int32, copy=False)]

    # Window
    frames *= np.hamming(frame_length)

    # Fourier-Transform and Power Spectrum
    NFFT = 512

    # Magnitude of the FFT
    mag_frames = np.absolute(np.fft.rfft(frames, NFFT))

    # Power Spectrum
    pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2))

    # Filter Banks
    nfilt = 40

    low_freq_mel = 0

    # Convert Hz to Mel
    high_freq_mel = (2595 * np.log10(1 + (sample_rate / 2) / 700))

    # Equally spaced in Mel scale
    mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)

    # Convert Mel to Hz
    hz_points = (700 * (10**(mel_points / 2595) - 1))
    bin = np.floor((NFFT + 1) * hz_points / sample_rate)
```

```

    fbank = np.zeros((nfilt, int(np.floor(NFFT / 2 + 1))))
    for m in range(1, nfilt + 1):
        f_m_minus = int(bin[m - 1])   # left
        f_m = int(bin[m])             # center
        f_m_plus = int(bin[m + 1])    # right

        for k in range(f_m_minus, f_m):
            fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m - 1])
        for k in range(f_m, f_m_plus):
            fbank[m - 1, k] = (bin[m + 1] - k) / (bin[m + 1] - bin[m])
    filter_banks = np.dot(pow_frames, fbank.T)

    # DCT
    filter_banks = np.where(
        filter_banks == 0,
        np.finfo(float).eps,
        filter_banks)

    # dB
    filter_banks = 20 * np.log10(filter_banks)

    return filter_banks

list1=list(pd.DataFrame(os.listdir('train'))[0].apply(lambda x: "train
   \\"+x).values)
df1=pd.DataFrame(list1)
df1['lang']=df1[0].apply(lambda x: x.split('_')[0][-2:])
df1['gender']=df1[0].apply(lambda x: x.split('_')[1])
df1['user_id']=df1[0].apply(lambda x: x.split('_')[-1].split('.')[0])
df1['fragment']=df1[0].apply(lambda x: x.split('_')[-1].split('.')[1])
df1['edit']=df1[0].apply(lambda x: x.split('_')[-1].split('.')[2])

german_m = []
german_f = []
spanish_m = []
spanish_f = []
english_m = []
english_f = []
for i in range(len(df1)):
    if df1['lang'][i]=='de' and df1['gender'][i]=='f':
        german_f.append(df1[0][i])
    if df1['lang'][i]=='de' and df1['gender'][i]=='m':
        german_m.append(df1[0][i])
    if df1['lang'][i]=='en' and df1['gender'][i]=='f':
        english_f.append(df1[0][i])
    if df1['lang'][i]=='en' and df1['gender'][i]=='m':
        english_m.append(df1[0][i])
    if df1['lang'][i]=='es' and df1['gender'][i]=='f':
        spanish_f.append(df1[0][i])
    if df1['lang'][i]=='es' and df1['gender'][i]=='m':
        spanish_m.append(df1[0][i])
```

```python

sig_spanish_m, sr_spanish_m = sf.read(file+spanish_m[0])
sig_spanish_f, sr_spanish_f = sf.read(file+spanish_f[0])
sig_english_m, sr_english_m = sf.read(file+english_m[0])
sig_english_f, sr_english_f = sf.read(file+english_f[0])
sig_german_m, sr_german_m = sf.read(file+german_m[0])
sig_german_f, sr_german_f = sf.read(file+german_f[0])

(f, S)= scipy.signal.welch(sig_german_m, sr_german_m, nperseg=1024)

plt.semilogy(f, S)
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD [V**2/Hz]')
plt.show()

# using padding
plt.subplots_adjust(left=0,
                    bottom=0,
                    right=2.0,
                    top=2.0,
                    wspace=0.4,
                    hspace=1)
#plt.show()
plt.subplot(6,2,1)
plt.title('Spanish Male')
Pxx, freqs, bins, im = plt.specgram(sig_spanish_m, Fs=sr_spanish_m)

# add axis labels
plt.ylabel('Freq')
plt.xlabel('Time in samples')
plt.subplot(6,2,2)
plt.title('Spanish Female')
Pxx, freqs, bins, im = plt.specgram(sig_spanish_f ,Fs=sr_spanish_f)
plt.ylabel('Freq')
plt.xlabel('Time in samples')

plt.subplot(6,2,3)
plt.plot(sig_spanish_m)
plt.ylabel('Amplitude')
plt.xlabel('sample')

plt.subplot(6,2,4)
plt.plot(sig_spanish_f)
plt.ylabel('Amplitude')
plt.xlabel('sample')

# add axis labels
# plt.ylabel('Freq')
# plt.xlabel('Time in samples')
plt.subplot(6,2,5)
plt.title('English Male')
Pxx, freqs, bins, im = plt.specgram(sig_english_m, Fs=sr_english_m)
```

```
# add axis labels
plt.ylabel('Freq')
plt.xlabel('Time in samples')
plt.subplot(6,2,6)
plt.title('English Female')
Pxx, freqs, bins, im = plt.specgram(sig_english_f ,Fs=sr_english_f)
plt.ylabel('Freq')
plt.xlabel('Time in samples')

plt.subplot(6,2,7)
plt.plot(sig_english_m)
plt.ylabel('Amplitude')
plt.xlabel('sample')

plt.subplot(6,2,8)
plt.plot(sig_english_f)
plt.ylabel('Amplitude')
plt.xlabel('sample')

# add axis labels
# plt.ylabel('Freq')
# plt.xlabel('Time in samples')
plt.subplot(6,2,9)
plt.title('German Male')
Pxx, freqs, bins, im = plt.specgram(sig_german_m ,Fs=sr_german_m)
# add axis labels
plt.ylabel('Freq')
plt.xlabel('Time in samples')
plt.subplot(6,2,10)
plt.ylabel('Freq')
plt.xlabel('Time in samples')
plt.title('German Female')
Pxx, freqs, bins, im = plt.specgram(sig_german_f ,Fs=sr_german_f)

plt.subplot(6,2,11)
plt.plot(sig_german_m)
plt.ylabel('Amplitude')
plt.xlabel('sample')

plt.subplot(6,2,12)
plt.plot(sig_german_f)
plt.ylabel('Amplitude')
plt.xlabel('sample')

X_train,X_test,y_train,y_test = train_test_split(df1,df1['lang'],
    stratify = df1[['lang','gender','user_id','edit']],test_size =
    0.78,random_state = 420)

list1=X_train[0].values

for i in range(round(len(list1)/500)):
    sigdf=pd.DataFrame()
```

```
    start = time.time()
    for j in range(500):
        signal, sample_rate = sf.read(list1[i*500+j])
        df=pd.DataFrame(signal.reshape(-1, len(signal)))
        df['sample_rate']=sample_rate
        df['filename']=list1[i*500+j]
        sigdf=pd.concat([sigdf,df])
    sigdf=sigdf.reset_index(drop=True)
    sigdf['lang']=sigdf['filename'].apply(lambda x: x.split('_')[0][-2:])
    sigdf['gender']=sigdf['filename'].apply(lambda x: x.split('_')[1])
    sigdf['user_id']=sigdf['filename'].apply(lambda x: x.split('_')[-1].split('.')[0])
    sigdf['fragment']=sigdf['filename'].apply(lambda x: x.split('_')[-1].split('.')[1])
    sigdf['edit']=sigdf['filename'].apply(lambda x: x.split('_')[-1].split('.')[2])
    sigdf.drop([220500],axis=1)
    filehandler = open(r'ex_data\singal_df_'+str(i*500+1)+'_'+str(i*500+j+1)+'.pkl',"wb")
    pickle.dump(sigdf,filehandler)
    filehandler.close()
    hours, rem = divmod(time.time()-start, 3600)
    minutes, seconds = divmod(rem, 60)
    print('ex_data\singal_df_'+str(i*500+1)+'_'+str(i*500+j+1)+'.csv')
    print("{:0>2}:{:0>2}:{:05.2f}".format(int(hours),int(minutes),seconds))

language_dummies=pd.DataFrame()
MFCC_array = []
sc = StandardScaler()
start=time.time()
for i in range(round(len(list1)/500)):
    file = open(r'ex_data\singal_df_'+str(i*500+1)+'_'+str(i*500+500)+'.pkl','rb')
    sigdf = pickle.load(file)
    singal_values=np.array(sigdf.iloc[:,:220500])
    language_dummies = pd.concat([language_dummies,pd.get_dummies(sigdf['filename'].apply(lambda x: x.split('_')[0][-2:]))])
    for i in range(0,len(singal_values)):
        MFCC = generate_fb_and_mfcc(singal_values[i], sigdf['sample_rate'][i])
        MFCC_sc = sc.fit_transform(MFCC)
        MFCC_array.append(MFCC_sc)

MFCC_array = np.array(MFCC_array)
filehandler = open(r'ex_data\MFCC_arrays.pkl',"wb")
pickle.dump(MFCC_array,filehandler)
filehandler.close()

filehandler = open(r'ex_data\language_dummies.pkl',"wb")
pickle.dump(language_dummies,filehandler)
```

```python
filehandler.close()

hours, rem = divmod(time.time()-start, 3600)
minutes, seconds = divmod(rem, 60)

print("{:0>2}:{:0>2}:{:05.2f}".format(int(hours),int(minutes),seconds))

start=time.time()
input_shape = (1000,40,1)

model = Sequential()
model.add(Conv2D(32,(7, 7), activation='relu', padding='valid', input_shape=input_shape))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3,3), strides=2, padding='same'))
model.add(Conv2D(64,(5,5), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3,3), strides=2, padding='same'))
model.add(Conv2D(128,(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3,3), strides=2, padding='same'))
model.add(Conv2D(256,(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3,3), strides=2, padding='same'))
model.add(Conv2D(512,(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3,3), strides=2, padding='same'))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))


adam = Adam()
def step_decay(epoch):
    initial_lrate = 0.00158
    drop = 0.9
    epochs_drop = 1
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
    return lrate


model.compile(loss='categorical_crossentropy',optimizer=adam,metrics=['accuracy'])
checkpoint = ModelCheckpoint(
                'model.h5',
                monitor='val_acc',
                verbose=0,
                save_best_only=True,
```

```
                mode='max'
                )

lrate = LearningRateScheduler(step_decay)
model.fit(X_train_MFCC,
          y_train_MFCC,
          epochs=9,
          callbacks=[checkpoint, lrate],
          verbose=2,
          validation_data=(X_test_MFCC, y_test_MFCC),
          batch_size=32)

hours, rem = divmod(time.time()-start, 3600)
minutes, seconds = divmod(rem, 60)

print("{:0>2}:{:0>2}:{:05.2f}".format(int(hours),int(minutes),seconds))

filehandler = open(r'ex_data\model.pkl',"wb")
pickle.dump(model,filehandler)
filehandler.close()

y_pred = model.predict(X_train_MFCC)
y_train1 = []
label={0:'de',1:'en',2:'es'}
for i in range(0,len(y_train_MFCC)):
    argmax = label[np.argmax(y_train_MFCC.iloc[i,:])]
    y_train1.append(argmax)
y_pred1 = []
for i in range(0,len(y_train_MFCC)):
    argmax = label[np.argmax(y_pred[i,:])]
    y_pred1.append(argmax)

print(classification_report(y_train1,y_pred1))
print(classification_report(y_train1,y_pred1))

y_pred_test = model.predict(X_test_MFCC)
y_test1 = []
label={0:'de',1:'en',2:'es'}
for i in range(0,len(y_test_MFCC)):
    argmax = label[np.argmax(y_test_MFCC.iloc[i,:])]
    y_test1.append(argmax)
y_pred2 = []
for i in range(0,len(y_test_MFCC)):
    argmax = label[np.argmax(y_pred_test[i,:])]
    y_pred2.append(argmax)

confusion_matrix(y_test1,y_pred2)
print(classification_report(y_test1,y_pred2))
```