SPEECH EMOTION ANALYZER

A PROJECT REPORT

Submitted by

SANKALP JAIN [RA1911026010119]
AMULYA [RA1911026010114]
ENZO DELOYE [VA2111001010001]

Under the guidance of

Dr. A. Revathi

(Assistant Professor, Department of Computational Intelligence)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTATIONAL INTELLIGENCE

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

S.R.M. Nagar, Kattankulathur, Kancheepuram District

APRIL 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**SPEECH EMOTION ANALYZER**" is the bonafide work of "**SANKALP JAIN [RA1911026010119]**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. A. Revathi
GUIDE
Assistant Professor
Dept. of Computational Intelligence

SIGNATURE

Dr. R. Annie Uthra, Professor
HEAD OF THE DEPARTMENT
Dept. of Computational Intelligence

Signature of the Internal Examiner

Signature of the External Examiner

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide, Dr. A. Revathi, her valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing the project. All through the work, in spite of her busy schedule, she has extended cheerful and cordial support to me for completing this project work.

SANKALP JAIN

ABSTRACT

The problem of speech emotion recognition can be solved by analyzing one or more of these features. Choosing to follow the lexical features would require a transcript of the speech which would further require an additional step of text extraction from speech if one wants to predict emotions from real-time audio. Similarly, going forward with analyzing visual features would require the excess to the video of the conversations which might not be feasible in every case while the analysis on the acoustic features can be done in real-time while the conversation is taking place as we'd just need the audio data for accomplishing our task.We checked the distribution of labels with respect to emotions and gender and found that while the data is balanced for six emotions viz***. neutral, happy, sad, angry, fear*** and ***disgust***, the number of labels was slightly less for ***surprise*** and negligible for ***boredom***. While the slightly fewer instances of surprise can be overlooked on account of it being a rarer emotion, the imbalance against boredom was rectified later by clubbing sadness and boredom together due to them being similar acoustically. It's also worth noting that boredom could have been combined with neutral emotion but since both ***sadness*** and ***boredom*** are negative emotions, it made more sense to combine them.The idea behind creating this project was to build a machine learning model that could detect emotions from the speech we have with each other all the time. Nowadays personalization is something that is needed in all the things we experience everyday. So why not have an emotion detector that will gauge your emotions and in the future recommend you different things based on your mood.

This can be used by multiple industries to offer different services like marketing companies suggesting you to buy products based on your emotions, the automotive industry can detect the person's emotions and adjust the speed of autonomous cars as required to avoid any collisions etc.

INTRODUCTION

1.1 OBJECTIVE

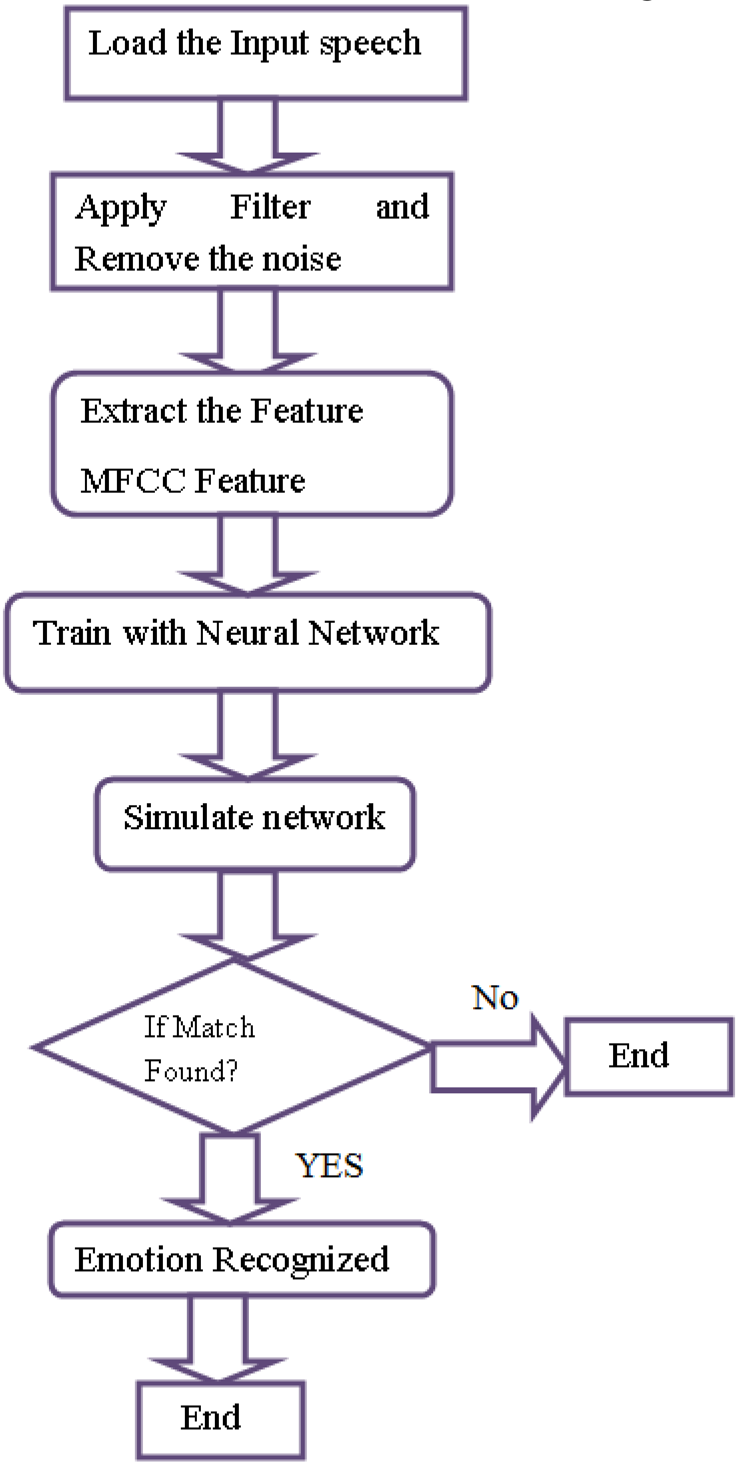
The project's goal is to detect emotions through speech using Natural Language Processing and Machine Learning algorithms .The idea behind creating this project was to build a machine learning model that could detect emotions from the speech we have with each other all the time. Nowadays personalization is something that is needed in all the things we experience everyday.

1.2 SCOPE

There is big scope for this project. It will help in determining certain emotions through voice which will help organizations to determine certainty of emotion more accurately with visual features .Speech emotion recognition has been formulated as a pattern recognition problem that mainly involves feature extraction and emotion classification. Speech emotion recognition has found increasing applications in practice, e.g., in security, medicine, entertainment, education.

ARCHITECTURE DIAGRAM

Figure 2.1



IMPLEMENTATION

Importing the required libraries

In [1]:
```
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from matplotlib.pyplot import specgram
import keras
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
from keras.models import Model
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import confusion_matrix
```

```
Using TensorFlow backend.
```

In [2]:
```
from keras import regularizers
```

In [3]:
```
import os
```

In [4]:
```
mylist= os.listdir('RawData/')
```

In [5]:
```
type(mylist)
```

Out[5]:
```
list
```

In [6]:
```
print(mylist[1800])
```

```
f01 (10).wav
```

In [7]:
```
print(mylist[400][6:-16])
```

```
03
```

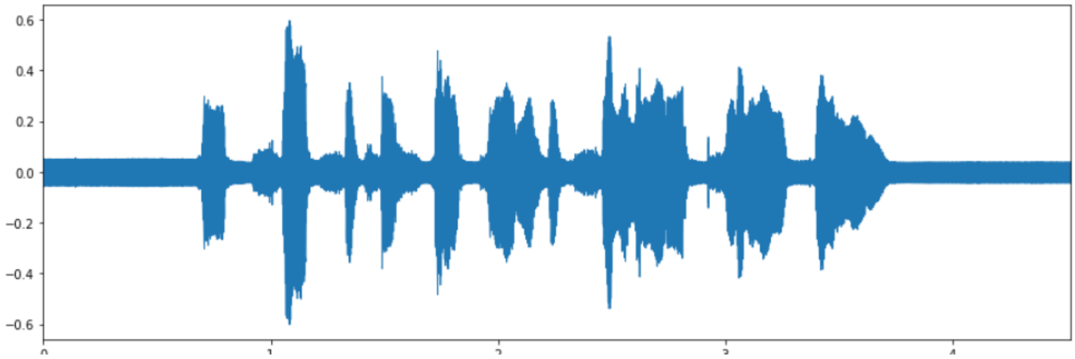
Plotting the audio file's waveform and its spectrogram

In [8]:
```
data, sampling_rate = librosa.load('RawData/f11 (2).wav')
```

In [9]:
```
% pylab inline
import os
import pandas as pd
import librosa
import glob

plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)
```

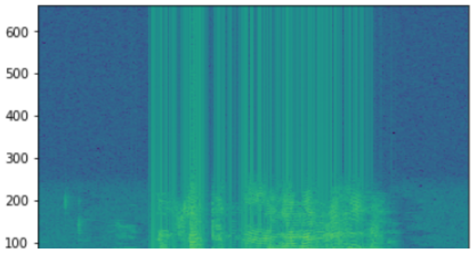
```
Populating the interactive namespace from numpy and matplotlib
```

Out[9]:
```
<matplotlib.collections.PolyCollection at 0x281d95e7358>
```



In [10]:

```
import matplotlib.pyplot as plt
import scipy.io.wavfile
import numpy as np
import sys


sr,x = scipy.io.wavfile.read('RawData/f10 (2).wav')

## Parameters: 10ms step, 30ms window
nstep = int(sr * 0.01)
nwin  = int(sr * 0.03)
nfft = nwin

window = np.hamming(nwin)

## will take windows x[n1:n2].  generate
## and loop over n2 such that all frames
## fit within the waveform
nn = range(nwin, len(x), nstep)

X = np.zeros( (len(nn), nfft//2) )

for i,n in enumerate(nn):
    xseg = x[n-nwin:n]
    z = np.fft.fft(window * xseg, nfft)
    X[i,:] = np.log(np.abs(z[:nfft//2]))

plt.imshow(X.T, interpolation='nearest',
    origin='lower',
    aspect='auto')

plt.show()
```



Setting the labels

In [11]:

```
feeling_list=[]
for item in mylist:
    if item[6:-16]=='02' and int(item[18:-4])%2==0:
        feeling_list.append('female_calm')
    elif item[6:-16]=='02' and int(item[18:-4])%2==1:
        feeling_list.append('male_calm')
    elif item[6:-16]=='03' and int(item[18:-4])%2==0:
        feeling_list.append('female_happy')
    elif item[6:-16]=='03' and int(item[18:-4])%2==1:
        feeling_list.append('male_happy')
    elif item[6:-16]=='04' and int(item[18:-4])%2==0:
        feeling_list.append('female_sad')
    elif item[6:-16]=='04' and int(item[18:-4])%2==1:
        feeling_list.append('male_sad')
    elif item[6:-16]=='05' and int(item[18:-4])%2==0:
        feeling_list.append('female_angry')
    elif item[6:-16]=='05' and int(item[18:-4])%2==1:
        feeling_list.append('male_angry')
    elif item[6:-16]=='06' and int(item[18:-4])%2==0:
        feeling_list.append('female_fearful')
    elif item[6:-16]=='06' and int(item[18:-4])%2==1:
        feeling_list.append('male_fearful')
    elif item[:1]=='a':
        feeling_list.append('male_angry')
    elif item[:1]=='f':
        feeling_list.append('male_fearful')
    elif item[:1]=='h':
        feeling_list.append('male_happy')
    #elif item[:1]=='n':
        #feeling_list.append('neutral')
    elif item[:2]=='sa':
        feeling_list.append('male_sad')
```

In [12]:

```
labels = pd.DataFrame(feeling_list)
```

In [13]:

```
labels[:10]
```

In [13]:

```
labels[:10]
```

Out[13]:

	0
0	male_calm
1	female_calm
2	male_calm
3	female_calm
4	male_calm
5	female_calm
6	male_calm
7	female_calm
8	male_calm
9	female_calm

Getting the features of audio files using librosa

In [14]:

```
df = pd.DataFrame(columns=['feature'])
bookmark=0
for index,y in enumerate(mylist):
    if mylist[index][6:-16]!='01' and mylist[index][6:-16]!='07' and mylist[index][6:-16]!='08' and mylist[index][:2]!='su' and mylist[index][:1]!='n'
        X, sample_rate = librosa.load('RawData/'+y, res_type='kaiser_fast',duration=2.5,sr=22050*2,offset=0.5)
        sample_rate = np.array(sample_rate)
        mfccs = np.mean(librosa.feature.mfcc(y=X,
                                            sr=sample_rate,
                                            n_mfcc=13),
                        axis=0)
        feature = mfccs
        #[float(i) for i in feature]
        #feature1=feature[:135]
        df.loc[bookmark] = [feature]
        bookmark=bookmark+1
```

Out[15]:

	feature
0	[-70.2677641611, -70.2677641611, -70.267764161...
1	[-65.7076524007, -65.7076524007, -63.114722422...
2	[-65.4824988827, -65.4824988827, -65.482498882...
3	[-64.5284491035, -64.5284491035, -64.528449103...
4	[-62.3643105275, -59.9347251381, -61.869599961...

In [16]:

```
df3 = pd.DataFrame(df['feature'].values.tolist())
```

df3[:5]

In [17]:

```
newdf = pd.concat([df3,labels], axis=1)
```

In [18]:

```
rnewdf = newdf.rename(index=str, columns={"0": "label"})
```

In [19]:

```
rnewdf[:5]
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9	...	207	208	209	210	21
0	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	...	-57.447461	-58.896493	-58.751002	-57.405669	-60.07847
1	-65.707652	-65.707652	-63.114722	-61.518999	-61.097138	-63.424602	-63.720067	-56.854608	-55.168972	-54.640002	...	-39.792147	-40.613166	-41.209201	-41.439204	-43.99428
2	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	...	-31.346553	-34.310774	-35.800705	-35.936112	-37.63184
3	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-65.928222	...	-48.674306	-48.596082	-47.602751	-43.049195	-42.65954
4	-62.364311	-59.934725	-61.869600	-67.495764	-71.071811	-65.679826	-63.394396	-65.503349	-61.856639	-60.005421	...	-39.071328	-41.897121	-40.865430	-38.290605	-36.37239

5 rows × 217 columns

In [20]:

```
from sklearn.utils import shuffle
rnewdf = shuffle(newdf)
rnewdf[:10]
```

Out[20]:

	0	1	2	3	4	5	6	7	8	9	...	207	208	209	210	
1130	-21.316034	-21.405165	-23.427492	-22.760799	-23.077980	-22.940379	-22.521370	-22.318232	-23.623762	-22.484483	...	-30.090434	-25.397213	-24.566368	-25.313775	-27.1
1611	-28.023489	-27.012539	-24.843077	-26.073845	-26.605960	-25.415706	-26.137463	-25.882579	-26.144014	-25.486000	...	-9.060266	-10.795528	-11.224794	-10.564187	-10.6
606	-47.580376	-47.580376	-47.580376	-47.580376	-47.580376	-47.580376	-47.580376	-47.580376	-47.580376	-47.580376	...	-17.349710	-17.208624	-17.796563	-17.936725	-17.3
1356	-24.687263	-23.900377	-22.685773	-23.947191	-25.353984	-25.198551	-26.250432	-26.599865	-27.327227	-26.959255	...	-2.508218	-2.625317	-3.431428	-5.119769	-7.3
535	-59.254190	-55.769097	-53.499152	-53.206541	-51.857334	-51.743066	-53.126957	-53.163642	-53.088024	-54.972413	...	-52.252787	-50.024027	-50.142246	-50.522777	-51.5
1397	-13.596653	-17.466871	-26.128334	-27.447346	-28.217343	-28.169207	-28.163326	-29.641523	-30.600591	-31.308342	...	-39.573127	-38.788175	-38.404735	-41.522524	
813	-57.705173	-57.728170	-56.672727	-56.152195	-57.413794	-58.473491	-58.040922	-57.550879	-58.224060	-56.276742	...	-52.148175	-52.575092	-52.733973	-53.780274	-54.4
1538	-29.934310	-31.293358	-32.880679	-34.448076	-35.698980	-38.608411	-40.439090	-40.267872	-41.617890	-44.013965	...	-43.273342	-37.321326	-30.551111	-27.834584	-26.9
111	-62.748082	-62.748239	-62.759160	-62.761158	-62.758553	-62.763573	-62.760713	-62.748239	-62.748239	-62.774872	...	-39.684156	-43.283529	-44.693196	-42.968453	-41.4
1042	-25.975306	-22.825102	-21.225165	-20.821093	-19.913892	-20.578139	-22.024481	-20.963704	-14.915607	-11.356564	...	NaN	NaN	NaN	NaN	

10 rows × 217 columns

In [21]:

```
rnewdf=rnewdf.fillna(0)
```

Dividing the data into test and train

In [22]:

```
newdf1 = np.random.rand(len(rnewdf)) < 0.8
train = rnewdf[newdf1]
test = rnewdf[~newdf1]
```

In [23]:

```
train[250:260]
```

Out[23]:

	0	1	2	3	4	5	6	7	8	9	...	207	208	209	210	
384	-66.348220	-66.348220	-66.348220	-66.348220	-66.348220	-62.797142	-65.721075	-66.348220	-66.348220	-66.348220	...	-52.630403	-49.383379	-49.429265	-52.419689	-53.6
595	-66.730016	-66.730016	-66.730016	-66.730016	-66.730016	-66.730016	-66.730016	-66.730016	-66.730016	-66.730016	...	-66.730016	-66.730016	-66.659748	-66.092053	-66.4
419	-53.769216	-55.061158	-53.305227	-51.120282	-52.921001	-52.180352	-53.490261	-54.648841	-52.985473	-51.930319	...	-48.560605	-47.684605	-46.392758	-47.169001	-48.0
1318	0.141154	-1.890112	-6.834041	-7.146915	-7.428035	-7.331070	-7.330199	-7.143712	-7.353791	-7.521930	...	-21.726595	-21.101817	-20.511847	-19.783785	-22.0
680	-43.826744	-44.310900	-44.328308	-44.203774	-44.118380	-44.075670	-43.239178	-43.239971	-43.038306	-42.939907	...	-20.941555	-25.617258	-26.698866	-25.630834	-28.0
1344	-28.520928	-26.120322	-26.568178	-26.995864	-26.450248	-26.345966	-26.882970	-25.830752	-26.015952	-27.689731	...	0.000000	0.000000	0.000000	0.000000	0.0
723	-43.612680	-43.612680	-43.612680	-43.612680	-43.612680	-43.612680	-43.612680	-43.612680	-43.612680	-43.612680	...	-42.085371	-42.376162	-43.340164	-42.947368	-43.2
1265	-39.991394	-35.741445	-28.598387	-26.013478	-26.699656	-23.846577	-22.048639	-22.696152	-24.532903	-30.425296	...	-42.750622	-42.401878	-42.229751	-41.913388	-40.7
607	-46.424902	-46.442376	-47.522544	-47.161764	-47.574394	-49.036715	-49.656684	-50.018894	-49.297414	-48.125211	...	-47.007094	-47.213200	-46.885091	-47.983125	-50.0
543	-59.961592	-57.833689	-59.973265	-61.480152	-61.480152	-61.480152	-61.480152	-61.480152	-60.477495	-57.440553	...	-42.841297	-46.036947	-47.290371	-48.049756	-52.7

10 rows × 217 columns

In [24]:

```
trainfeatures = train.iloc[:, :-1]
```

In [25]:

```
trainlabel = train.iloc[:, -1:]
```

In [26]:

```
testfeatures = test.iloc[:, :-1]
```

In [27]:

```
testlabel = test.iloc[:, -1:]
```

In [28]:

```
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder

X_train = np.array(trainfeatures)
```

In [28]:

```
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder

X_train = np.array(trainfeatures)
y_train = np.array(trainlabel)
X_test = np.array(testfeatures)
y_test = np.array(testlabel)

lb = LabelEncoder()

y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:111: DataConversionWarning: A column-vector y was passed when a 1d array was e
xpected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [29]:

```
y_train
```

Out[29]:

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

In [30]:

```
X_train.shape
```

Out[30]:

```
(1378, 216)
```

Changing dimension for CNN model

In [31]:

```
x_traincnn =np.expand_dims(X_train, axis=2)
x_testcnn= np.expand_dims(X_test, axis=2)
```

In [45]:

```
model = Sequential()

model.add(Conv1D(256, 5,padding='same',
                 input_shape=(216,1)))
model.add(Activation('relu'))
model.add(Conv1D(128, 5,padding='same'))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(128, 5,padding='same',))
model.add(Activation('relu'))
#model.add(Conv1D(128, 5,padding='same',))
#model.add(Activation('relu'))
#model.add(Conv1D(128, 5,padding='same',))
#model.add(Activation('relu'))
#model.add(Dropout(0.2))
model.add(Conv1D(128, 5,padding='same',))
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dense(10))
model.add(Activation('softmax'))
opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
```

In [46]:
```
model.summary()
```

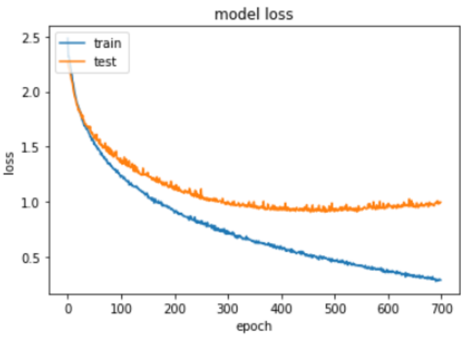
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_9 (Conv1D)            (None, 216, 256)          1536
_________________________________________________________________
activation_11 (Activation)   (None, 216, 256)          0
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 216, 128)          163968
_________________________________________________________________
activation_12 (Activation)   (None, 216, 128)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 216, 128)          0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 27, 128)           0
_________________________________________________________________
conv1d_11 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_13 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_14 (Activation)   (None, 27, 128)           0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3456)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                34570
_________________________________________________________________
activation_15 (Activation)   (None, 10)                0
=================================================================
Total params: 364,170
Trainable params: 364,170
Non-trainable params: 0
_________________________________________________________________
```

In [47]:
```
model.compile(loss='categorical_crossentropy', optimizer=opt,metrics=['accuracy'])
```

Removed the whole training part for avoiding unnecessary long epochs list

In [ ]:
```
cnnhistory=model.fit(x_traincnn, y_train, batch_size=16, epochs=700, validation_data=(x_testcnn, y_test))
```

In [37]:
```
plt.plot(cnnhistory.history['loss'])
plt.plot(cnnhistory.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Saving the model

In [112…
```
model_name = 'Emotion_Voice_Detection_Model.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')
# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
```

Loading the model

In [137…

```
# loading json and creating model
from keras.models import model_from_json
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("saved_models/Emotion_Voice_Detection_Model.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
score = loaded_model.evaluate(x_testcnn, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
Loaded model from disk
acc: 72.73%
```

Predicting emotions on the test data

In [138…

```
preds = loaded_model.predict(x_testcnn,
                         batch_size=32,
                         verbose=1)
```

```
319/319 [==============================] - 0s
```

In [139…

```
preds
```

Out[139…

```
array([[  3.49815641e-12,   1.18043589e-10,   1.13181663e-19, ...,
          1.80016723e-05,   7.36836637e-06,   1.14132257e-04],
       [  3.87504338e-16,   5.73074694e-23,   1.31673211e-14, ...,
          1.75147681e-04,   1.85760673e-05,   9.99805748e-01],
       [  8.39285008e-07,   3.43896300e-11,   5.38965035e-03, ...,
          9.93317604e-01,   1.99900052e-04,   1.05243188e-03],
       ...,
       [  3.49616457e-04,   1.94651744e-04,   6.65218568e-06, ...,
          1.67340226e-02,   6.44345134e-02,   9.10043001e-01],
       [  4.23396705e-06,   1.59581254e-11,   6.03030126e-12, ...,
          6.36715861e-03,   9.64888096e-01,   2.34207995e-02],
       [  3.69524572e-31,   0.00000000e+00,   0.00000000e+00, ...,
          5.24333927e-07,   9.99998808e-01,   6.36476927e-10]], dtype=float32)
```

In [115…

```
preds1=preds.argmax(axis=1)
```

In [116…

```
preds1
```

Out[116…

```
array([6, 9, 7, 1, 7, 8, 1, 2, 5, 8, 7, 1, 9, 9, 4, 0, 5, 0, 5, 8, 4, 1, 5,
       9, 7, 5, 7, 1, 7, 5, 8, 1, 8, 9, 2, 2, 1, 8, 6, 0, 5, 9, 1, 0, 7, 5,
       5, 7, 7, 0, 0, 7, 0, 0, 6, 5, 3, 7, 5, 8, 5, 4, 8, 8, 9, 7, 2, 8, 6,
       1, 5, 6, 8, 6, 5, 3, 4, 8, 8, 9, 9, 0, 8, 9, 4, 5, 0, 0, 5, 5, 7, 9,
       4, 7, 8, 6, 9, 5, 6, 8, 1, 7, 0, 8, 8, 7, 3, 2, 7, 8, 7, 9, 7, 9, 5,
       7, 8, 6, 0, 1, 6, 9, 1, 5, 8, 7, 1, 8, 6, 9, 3, 7, 7, 4, 6, 5, 8, 8,
       1, 0, 5, 0, 7, 6, 5, 7, 4, 9, 2, 5, 4, 7, 5, 6, 8, 5, 5, 4, 8, 8, 2,
       7, 5, 7, 6, 9, 8, 9, 9, 0, 2, 5, 7, 8, 3, 0, 4, 6, 6, 9, 9, 9, 3, 8,
       1, 7, 2, 5, 8, 1, 8, 4, 4, 8, 5, 2, 9, 9, 5, 6, 0, 5, 9, 0, 8, 7, 4,
       6, 6, 3, 9, 5, 9, 6, 1, 7, 5, 7, 8, 4, 9, 2, 5, 6, 6, 4, 2, 1, 8, 2,
       7, 5, 7, 3, 5, 6, 8, 7, 1, 6, 0, 0, 4, 7, 4, 4, 4, 4, 4, 0, 8, 1, 6,
       7, 4, 8, 7, 8, 2, 9, 6, 2, 7, 8, 3, 9, 9, 7, 2, 5, 7, 2, 9, 5, 5, 7,
       8, 5, 6, 8, 1, 2, 5, 9, 4, 5, 5, 6, 7, 8, 7, 0, 9, 5, 9, 5, 9, 7, 8,
       1, 2, 8, 7, 7, 5, 7, 8, 7, 7, 4, 5, 8, 5, 0, 0, 5, 9, 8, 8], dtype=int64)
```

In [117…

```
abc = preds1.astype(int).flatten()
```

In [118…

```
predictions = (lb.inverse_transform((abc)))
```

In [119...

```
preddf = pd.DataFrame({'predictedvalues': predictions})
preddf[:10]
```

Out[119...

	predictedvalues
0	male_calm
1	male_sad
2	male_fearful
3	female_calm
4	male_fearful
5	male_happy
6	female_calm
7	female_fearful
8	male_angry
9	male_happy

In [120...

```
actual=y_test.argmax(axis=1)
abc123 = actual.astype(int).flatten()
actualvalues = (lb.inverse_transform((abc123)))
```

In [121...

```
actualdf = pd.DataFrame({'actualvalues': actualvalues})
actualdf[:10]
```

Out[121...

	actualvalues
0	male_calm
1	male_sad
2	male_fearful
3	female_fearful
4	male_fearful
5	male_happy
6	female_calm
7	female_angry
8	male_angry
9	male_happy

In [122...

```
finaldf = actualdf.join(preddf)
```

Actual v/s Predicted emotions

In [128...

```
finaldf[170:180]
```

Out[128...

	actualvalues	predictedvalues
170	female_fearful	female_fearful
171	male_angry	male_angry
172	male_fearful	male_fearful
173	male_happy	male_happy
174	female_happy	female_happy
175	female_angry	female_angry
176	female_angry	female_sad
177	male_sad	male_calm
178	male_angry	male_calm
179	male_sad	male_sad

Recording live audio file -:

In [20]:

```
import pyaudio
import wave

CHUNK = 1024
FORMAT = pyaudio.paInt16 #paInt8
CHANNELS = 2
RATE = 44100 #sample rate
RECORD_SECONDS = 4
WAVE_OUTPUT_FILENAME = "output10.wav"

p = pyaudio.PyAudio()

stream = p.open(format=FORMAT,
                channels=CHANNELS,
                rate=RATE,
                input=True,
                frames_per_buffer=CHUNK) #buffer

print("* recording")

frames = []

for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
    data = stream.read(CHUNK)
    frames.append(data) # 2 bytes(16 bits) per channel

print("* done recording")

stream.stop_stream()
stream.close()
p.terminate()

wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
wf.setnchannels(CHANNELS)
wf.setsampwidth(p.get_sample_size(FORMAT))
wf.setframerate(RATE)
wf.writeframes(b''.join(frames))
wf.close()
```

```
* recording
* done recording
```

Predictions live demonstrations-

Live Demo

The file 'output10.wav' in the next cell is the file that was recorded live using the code in AudioRecoreder notebook found in the repository

In [485…

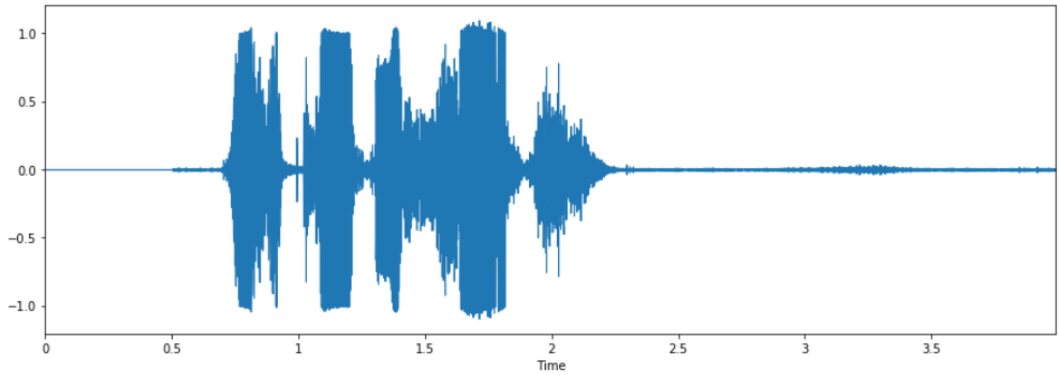
```
data, sampling_rate = librosa.load('output10.wav')
```

In [486…

```
% pylab inline
import os
import pandas as pd
import librosa
import glob

plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)
```

```
Populating the interactive namespace from numpy and matplotlib
<matplotlib.collections.PolyCollection at 0x23b43824048>
```

Out[486…



In [487…
```
#livedf= pd.DataFrame(columns=['feature'])
X, sample_rate = librosa.load('output10.wav', res_type='kaiser_fast',duration=2.5,sr=22050*2,offset=0.5)
sample_rate = np.array(sample_rate)
mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13),axis=0)
featurelive = mfccs
livedf2 = featurelive
```

In [488…
```
livedf2= pd.DataFrame(data=livedf2)
```

In [489…
```
livedf2 = livedf2.stack().to_frame().T
```

In [490…
```
livedf2
```

Out[490…

	0	1	2	3	4	5	6	7	8	9	...	206	207	208	209	210
	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
0	-18.203564	-21.471836	-22.52221	-21.712259	-22.264288	-20.707904	-21.726444	-21.76865	-24.302736	-22.250634	...	-24.273819	-24.639939	-24.929152	-24.43919	-25.210171

1 rows × 216 columns

In [491…
```
twodim= np.expand_dims(livedf2, axis=2)
```

In [492…
```
livepreds = loaded_model.predict(twodim,
                         batch_size=32,
                         verbose=1)
```
```
1/1 [==============================] - 0s
```

In [493…
```
livepreds
```

Out[493…
```
array([[  9.24052530e-22,   0.00000000e+00,   3.62402176e-26,
          1.30680162e-36,   4.47264152e-28,   1.00000000e+00,
          1.80208343e-30,   2.76873961e-27,   3.62227194e-23,
          1.67396652e-11]], dtype=float32)
```

In [494…
```
livepreds1=livepreds.argmax(axis=1)
```

In [495…
```
liveabc = livepreds1.astype(int).flatten()
```

In [496…
```
livepredictions = (lb.inverse_transform((liveabc)))
livepredictions
```

Out[496…
```
array(['male_angry'], dtype=object)
```

Predicted vs actual values-:

Actual v/s Predicted emotions

In [128…
```
finaldf[170:180]
```

Out[128…

	actualvalues	predictedvalues
170	female_fearful	female_fearful
171	male_angry	male_angry
172	male_fearful	male_fearful
173	male_happy	male_happy
174	female_happy	female_happy
175	female_angry	female_angry
176	female_angry	female_sad
177	male_sad	male_calm
178	male_angry	male_calm
179	male_sad	male_sad

Made use of two different datasets:
1. [RAVDESS](https://zenodo.org/record/1188976).
This dataset includes around 1500 audio file input from 24 different actors. 12 male and 12 female where these actors record short audios in 8 different emotions i.e 1 = neutral, 2 = calm, 3 = happy, 4 = sad, 5 = angry, 6 = fearful, 7 = disgust, 8 = surprised.
Each audio file is named in such a way that the 7th character is consistent with the different emotions that they represent.

2. [SAVEE](http://kahlan.eps.surrey.ac.uk/savee/Download.html).
This dataset contains around 500 audio files recorded by 4 different male actors. The first two characters of the file name correspond to the different emotions that they portray.

CONCLUSION
Building the model was a challenging task as it involved a lot of trial and error methods, tuning etc. The model is very well trained to distinguish between male and female voices and it distinguishes with 100% accuracy. The model was tuned to detect emotions with more than 70% accuracy. Accuracy can be increased by including more audio files for training.

References

1 https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8805181
2 https://www.frontiersin.org/articles/10.3389/fcomp.2020.00014/full
3 https://link.springer.com/content/pdf/bbm%3A978-3-319-49220-9%2F1.pdf
4 https://www.javatpoint.com/nlp
5 https://zenodo.org/record/1188976
6 http://kahlan.eps.surrey.ac.uk/savee/Download.html