



Machine Learning Project

Speech Emotion Recognition

Made by:

- *EL Bouga Latia*
- *Soulala Achraf*

Supervised by:

Mousannif Hajar

Professor, IT Department FSSM Marrakech

Academic year: 2021 / 2022

Contents Table

Speech Emotion Recognition:	3
Where can we use it :	3
Speech Emotion Recognition Process:	4
Step 1 – Libraries:	4
Step 2 – DataSet:	5
RAVDESS dataset:	5
A little background information about the data:	5
Step 3 – Feature Extraction:	7
Step 4 – Loading and preparing the data:	10
Step 5 – modeling:	10
Loading and Splitting the data:	10
Defining the model:	11
Compile the model:	15
Fit the model:	16
Model Prediction Accuracy Score:	16
step 5: Final Testing & deployment	16
Future perspectives:	19

Speech Emotion Recognition:

As evident from the title, Speech Emotion Recognition (SER) is a system that can identify the emotion of different audio samples.

There are three classes of features in a speech namely, the lexical features (the vocabulary used), the visual features (the expressions the speaker makes) and the acoustic features (sound properties like pitch, tone, jitter, etc.).

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 access 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 need just the audio data for accomplishing our task. Hence, we choose to analyze the acoustic features in this work.

Where can we use it :

Even though it isn't that popular, SER has entered so many areas these years, including:

- **The medical field:** In the world of telemedicine where patients are evaluated over mobile platforms, the ability for a medical professional to discern what the patient is actually feeling can be useful in the healing process.
- **Customer service:** In call center conversation may be used to analyze the behavioral study of call attendants with the customers which helps to improve the quality of service.

- **Recommender systems:** Can be useful to recommend products to customers based on their emotions towards that product.

Speech Emotion Recognition Process:

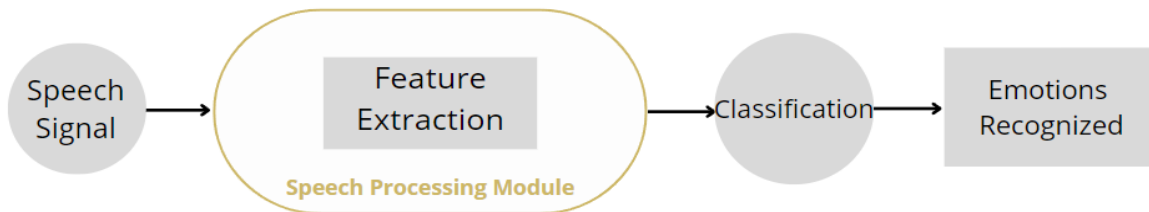


Figure 1: Emotion Recognition Speech

The process of speech emotion recognition is, we have a speech signal. We applied some modifications on this speech and we extracted some features from it for the final classification, that results in emotions recognized.

The steps we followed to carry out this project :

Step 1 – Libraries:

the libraries used in our project:

- **Librosa:** is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.
- **pydub:** Manipulate audio with a simple and easy high level interface.
- **Keras:** is an open-source software library that provides a Python interface for artificial neural networks.
- **Tensorflow:** TensorFlow is an open source Machine Learning library, created by Google, for developing and running Machine Learning and Deep Learning applications

- And other basic libraries like numpy, matplotlib, ...

Step 2 – DataSet:

RAVDESS dataset:

Our DataSet is RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song). It is a large dataset with an audio and video database. The original size of this data is around 24Gb. But we will use a smaller portion of it and not the whole dataset. This will help us to stay focused, train our model faster and to keep things simple. The small portion of the dataset can be found [here](#) on Kaggle.

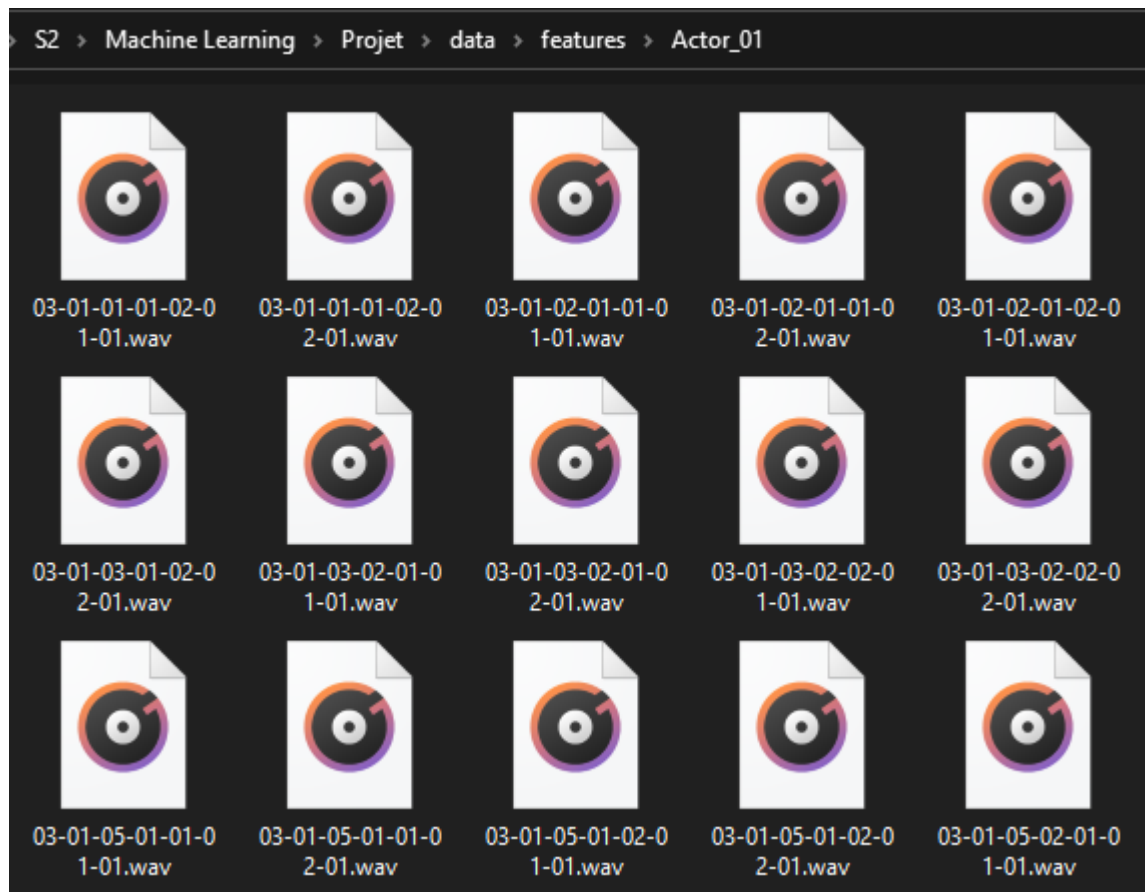
A little background information about the data:

This portion of the RAVDESS contains 5252 files rated by 24 professional actors: 12 female and 12 male. Speech emotions include 8 emotions: neutral, calm, happy, sad, angry, fearful, surprise and disgust expressions.

The file names are renamed following a particular pattern. This pattern consists of 7 parts. And these parts are divided as following:

- *Modality*: (01 = Audio + Video ‘files .mp4’, 02 = video-only ‘videos without sound’, 03 = audio-only ‘files .wav’).
- *Vocal channel*: (01 = speech, 02 = song).
- *Emotion*: (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
- *Emotional intensity*: (01 = normal, 02 = strong).
- *Statement*: (01 = “Kids are talking by the door”, 02 = “Dogs are sitting by the door”).
- *Repetition*: (01 = 1st repetition, 02 = 2nd repetition).
- *Actor*: (01 to 24).

Here is a screenshot of the Actor_1 folder within the dataset:



For our, we have used 5252 samples from the RAVDESS dataset (we eliminated videos without sound). So the sample comes from:

- Audio-only files (.wav).
- Video + Audio (.mp4); in this case, we have extracted audio from each file mp4 using the script below that allow us to convert mp4 to wav files:

```
import os
import subprocess

path = 'C:\Users\Visitor\Documents\FSSM\MASTER DS\S2\Machine Learning\Projet\data\features'

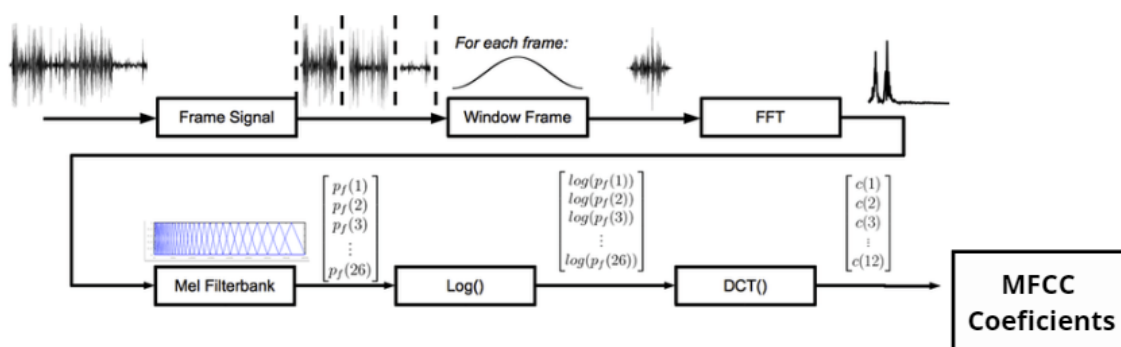
for filename in os.listdir(path):
    if(filename.endswith(".mp4")):
        command = "ffmpeg -i" + path + "/" + filename + " " + "-ab 160k -ac 2 -ar 44100 -vn " + filename[:-3] + ".wav"
        try:
            subprocess.call(command, shell=True)
        except ValueError:
            continue
        # Skip the file in case of error
```

Here are the labels of the emotion category. We are going to create this dictionary to use when training the machine learning model:

```
label_conversion = {'0': 'neutral',
                    '1': 'calm',
                    '2': 'happy',
                    '3': 'sad',
                    '4': 'angry',
                    '5': 'fearful',
                    '6': 'disgust',
                    '7': 'surprised'}
```

Step 3 – Feature Extraction:

Generally, there are many features of audio files, and most SER studies use spectral features as data extracted from the vocal tract, such as Linear Predictive Cepstral Coefficients (LPCC), Mel-Frequency Cepstral Coefficients (MFCC), and Formants. In our project, we extracted the features using the mel-frequency cepstrum coefficients (MFCC) method.



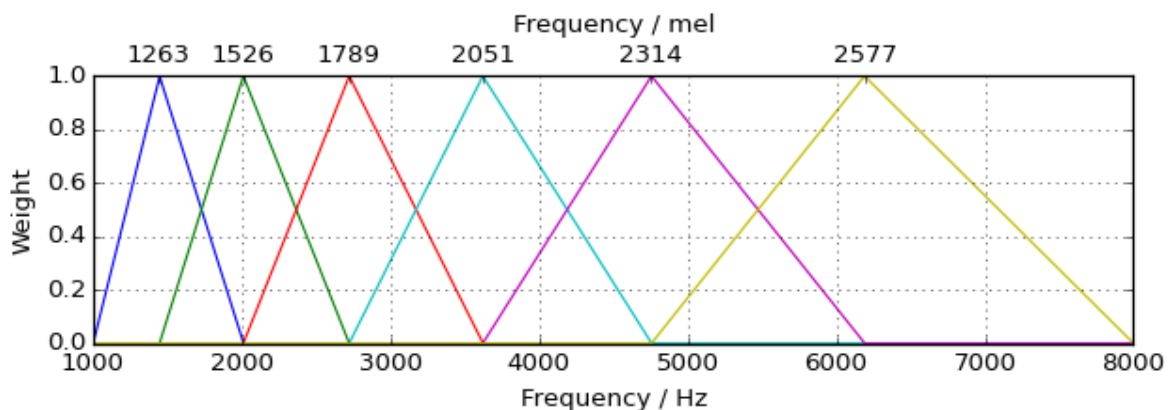
The mel frequency cepstral coefficients (MFCCS) of a signal are a small set of features which concisely describe the overall shape of a spectral envelope.

Steps to extract MFCCS:

- ➔ *Step 1*: Frame the signal into short frames, because an audio signal is constantly changing. This is why we frame the signal into 20-40 ms frames. If the frame is

much shorter we don't have enough samples to get a reliable spectral estimate, if it is longer, the signal changes too much throughout the frame.

- *Step 2*: For each frame, we calculate the power spectrum of each frame. This is motivated by the human cochlea (an organ in the ear) which vibrates at different spots depending on the frequency of the incoming sounds. Depending on the location in the cochlea that vibrates, different nerves fire informing the brain that certain frequencies are present. Our periodogram estimate performs a similar job for us, identifying which frequencies are present in the frame.
- *Step 3*: The periodogram spectral estimate still contains a lot of information not required for Automatic Speech Recognition (ASR). In particular the cochlea can not discern the difference between two closely spaced frequencies. This effect becomes more pronounced as the frequencies increase. For this reason, we take clumps of periodogram bins and sum them up to get an idea of how much energy exists in various frequency regions. This is performed by our Mel filterbank:



The first filter is very narrow and gives an indication of how much energy exists near 0 Hertz. As the frequencies get higher our filters get wider as we become less concerned about variations. We are only interested in roughly how much energy occurs at each spot. The mel scale tells us exactly how to space our filterbanks and how wide to make them.

→ *Step 4:* Once we have the filterbank energies, we take the logarithm of them. This is also motivated by human hearing: we don't hear loudness on a linear scale.

→ *Step 5:* The final step is to compute the DCT(discrete cosine transform) of the log filterbank energies. Because our filterbanks are all overlapping, the filterbank energies are quite correlated with each other. The DCT decorrelates the energies which means diagonal covariance matrices can be used to model the features in.

And the resulting features are called Mel Frequency Cepstral Coefficients.

To implement MFCC, we call librosa library:

```
path = '/content/drive/MyDrive/Emotion/features'
lst = []

for subdir, dirs, files in os.walk(path):
    for file in files:
        try:
            #Load librosa array, obtain mfccs, store the file and the mfccs information in a new array
            X, sample_rate = librosa.load(os.path.join(subdir,file), res_type='kaiser_fast')
            mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)
            # The instruction below converts the labels (from 1 to 8) to a series from 0 to 7
            # This is because our predictor needs to start from 0 otherwise it will try to predict also 0.

            #print(len(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40)))
            #print(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40))
            file = int(file[7:8]) - 1
            arr = mfccs, file
            lst.append(arr)
        # If the file is not valid, skip it
    except ValueError:
        continue
}
```

For each file .wav in our dataset (5252 file), we call the function (librosa.feature.mfcc) that returns 40 features for each frame, and we calculate the mean of those 40 MFCCS.

We apply these operations on the dataset files and the result would be a vector of 40 MFCCS for each audio file.

Then we extract the emotion from the name of the file, and we add it to our dataset as a label. So our data have a shape of 5252 rows and 40 columns without counting the label.

Step 4 – Loading and preparing the data:

In this step, we are done with feature extraction, so we are assigning the features with label emotions. and we take features as our input (X) and the labeled emotion as an output (y).

```
[ ] # Creating X and y: zip makes a list of all the first elements, and a list of all the second elements.
X, y = zip(*lst)

[ ] X = np.asarray(X)
y = np.asarray(y)

X.shape, y.shape

((5252, 40), (5252,))

[ ] # Saving joblib files to not load them again with the loop above

import joblib

X_name = 'X.joblib'
y_name = 'y.joblib'
save_dir = '/content/drive/MyDrive/Emotion'

savedX = joblib.dump(X, os.path.join(save_dir, X_name))
savedy = joblib.dump(y, os.path.join(save_dir, y_name))

[ ] # Loading saved models
import joblib

X = joblib.load('/content/drive/MyDrive/Emotion/X.joblib')
y = joblib.load('/content/drive/MyDrive/Emotion/y.joblib')
```

Step 5 – modeling:

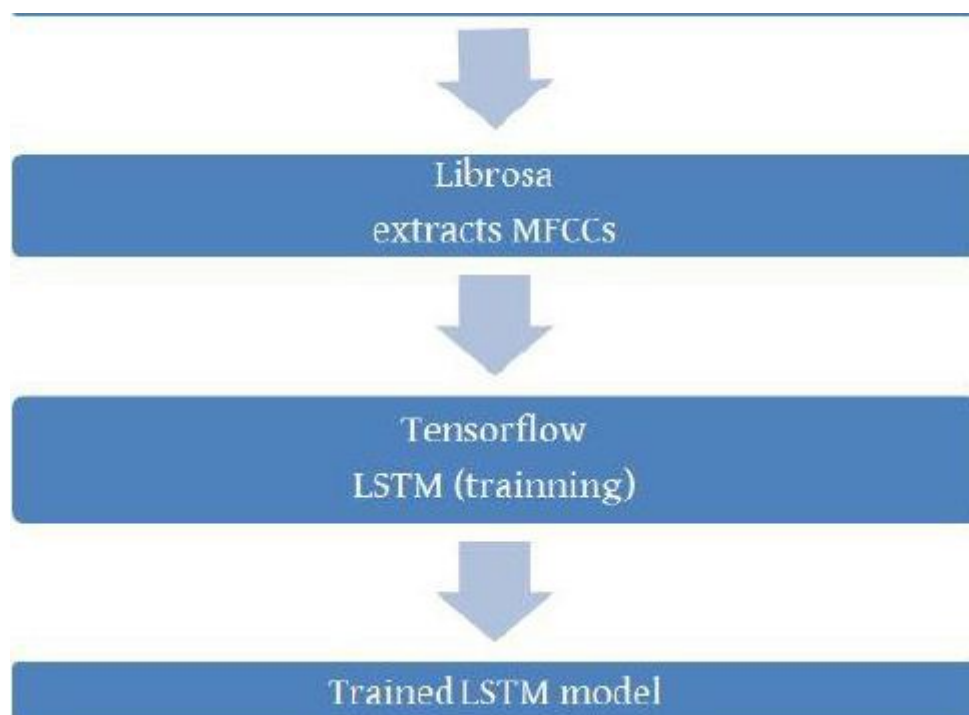
Loading and Splitting the data:

We are going now to split the labeled dataset using the `train_test_split()` function:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Defining the model:

- **LSTM:** We use the Ravdess dataset to train our system. We apply the Librosa library to extract audio features, i.e. the Mel Frequency cepstral coefficients (MFCC), from the raw data. The extracted features are input to the Long Short Term Memory (LSTM) neural network model for training. Our LSTM are built with Keras and Tensorflow.



Experimental Results

The model is trained with a dataset in 340 epochs. The accuracy and loss functions are taken into the consideration for evaluation of the model. To do this we have to utilize the model to predict the suitable result on the evaluation dataset and compare that result with the predicted target with an actual answer.

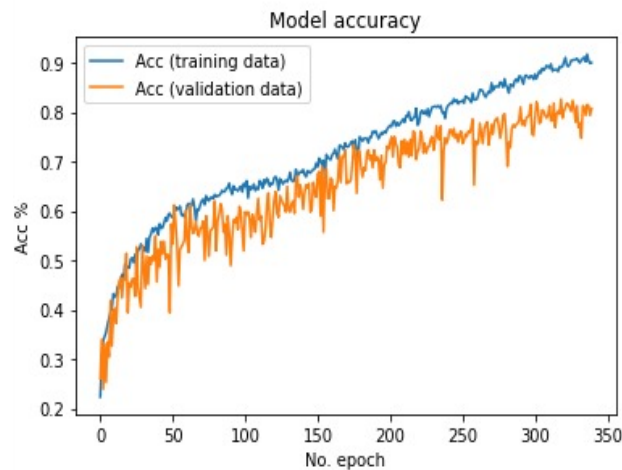


Figure above compares the model's accuracies as achieved on running it on training and testing data set. On the training dataset, the model achieved an accuracy of 91.3% while on the test data, it had an average accuracy of .07% at the end of 340 epochs.

The accuracy is low. So, we decide to use another model:

- **CNN:** In the example model below, a 1D **Convolutional Layer** (Conv1D) unit is the portion that learns the *translation invariant spatial patterns* and their *spatial hierarchies*.
 - ❖ The **Max Pooling Layer** halves the size of the feature maps by downsampling them to the max value inside a window. Why downsample? Because otherwise it would result in a ginormous number of parameters and your computer would blow up and after all that the model would massively overfit the data. This magical layer is the reason that a CNN can handle the huge amounts of data in images. Max Pooling does a model good.

- ❖ The **Dropout Layer** guards against overfitting by randomly setting the weights of a portion of the data to zero, and the **Dense** units contain hidden layers tied to the degrees of freedom the model has to try and fit the data. The more complex the data, the more degrees of freedom the model needs. *Take care not to add a bunch of these and end up overfitting the data.*
- ❖ The **Flatten Layer** squishes all the feature map information into a single column in order to feed it into a Dense layer, the last of which outputs the 8 species that the model is supposed to classify the audio recordings into.

A sample CNN model architecture

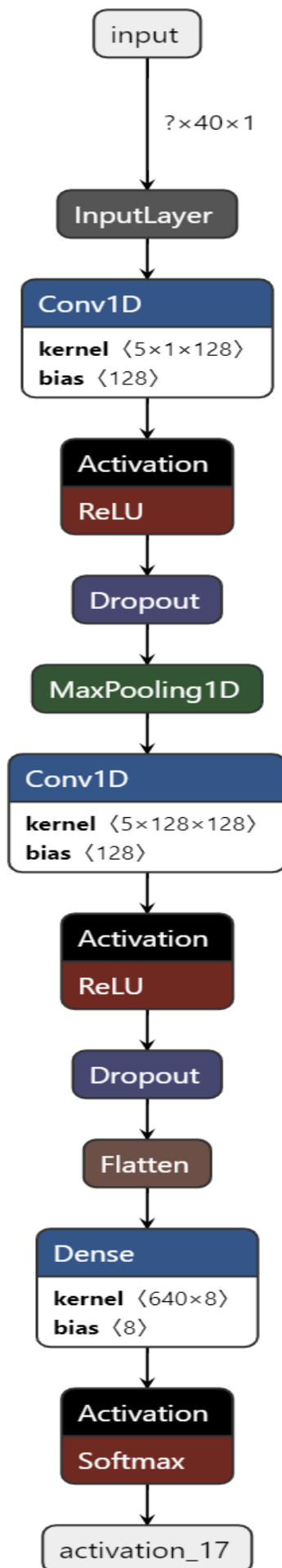
The **activation** functions give the model the ability to add nonlinearity to the model. Here, the **relu** function is used, which zeros out negative weights. This is a good one to start with. The last **Dense** layer's activation function type is **softmax**, which outputs a probability for each class.

```
model = Sequential()

model.add(Conv1D(128, 5, padding='same', input_shape=(40,1)))
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(Dropout(0.1))
model.add(Flatten())

model.add(Dense(8))
model.add(Activation('softmax'))
```



Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 40, 128)	768
activation (Activation)	(None, 40, 128)	0
dropout (Dropout)	(None, 40, 128)	0
max_pooling1d (MaxPooling1D)	(None, 5, 128)	0
conv1d_1 (Conv1D)	(None, 5, 128)	82048
activation_1 (Activation)	(None, 5, 128)	0
dropout_1 (Dropout)	(None, 5, 128)	0
flatten (Flatten)	(None, 640)	0
dense (Dense)	(None, 8)	5128
activation_2 (Activation)	(None, 8)	0

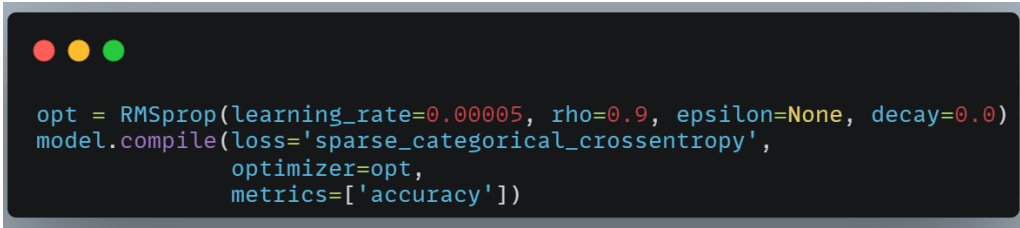
Total params: 87,944
Trainable params: 87,944
Non-trainable params: 0

A network using Convolution layers was used to build classifier, network architecture is shown in figure above. The number of filters for both convolution was 128 and filter_size was 5 and 2 for respective layers followed by 1 fully connected layer.

Max pooling was used after each convolution layer. During Training overfitting was observed, to handle that dropout 10% (keep) was used after each convolution layer. Input shape was fixed as (40,1) while 40 represents the number of MFCCS for each audio file with 1 channel.

During training it was also observed that, without downsampling, the data model was not able to generalize well. Adding downsampling techniques helped the model in generalization.

Compile the model:



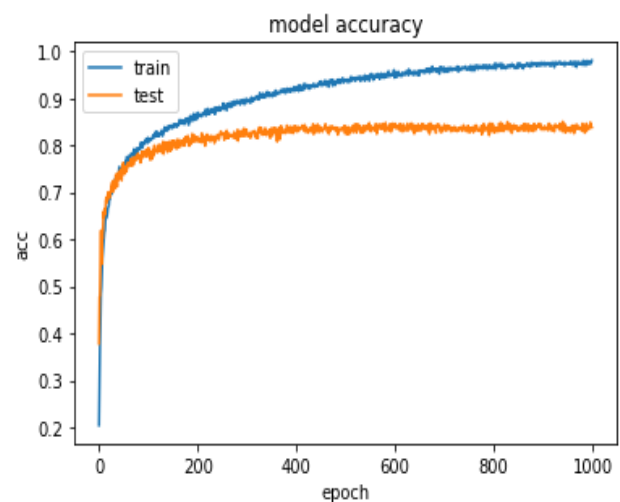
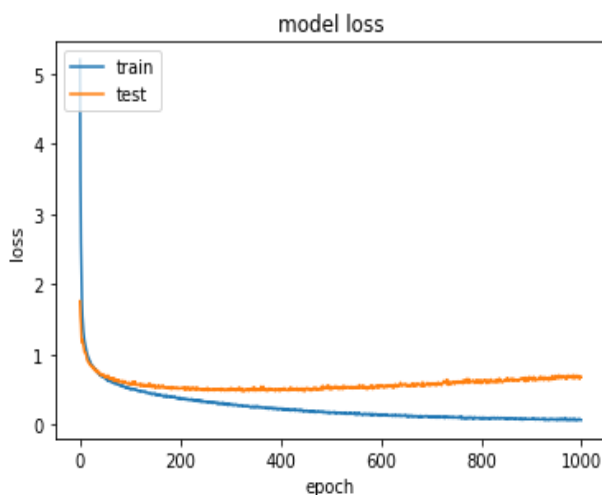
```
opt = RMSprop(learning_rate=0.00005, rho=0.9, epsilon=None, decay=0.0)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

The **RMSprop** optimizer manages the learning rate , the **loss** function is used to evaluate how different the predicted and actual data are and penalizes the model for poor predictions. In this example, the loss function is **SparseCategoricalCrossentropy**, which is used when each sample belongs to one label, as opposed to more than one, **and** it's not binary classification. This is an appropriate choice because each audio sample belongs to one species and there are 8 of them.

Fit the model:

```
cnnhistory=model.fit(x_traincnn, y_train, batch_size=16, epochs=1000, validation_data=(x_testcnn, y_test))
```

Model Prediction Accuracy Score:



PERFORMANCE: Training was done for 1000 epochs using **Root Mean Squared Propagation** (RMSprop) as optimizer with learning rate of 0.00005. Figure above displays accuracy during training.

□ Training Accuracy: 99.88%

□ Testing Accuracy: 84.75%

step 5: Final Testing & deployment

Finally once the **model** is created we save it locally. We don't want to create a model every time we want to test something. Whenever we want to perform some test we load the model and record an audio.



Emotion_Voice_Detection_Model66.h5

18/06/2022 17:01

H5 File

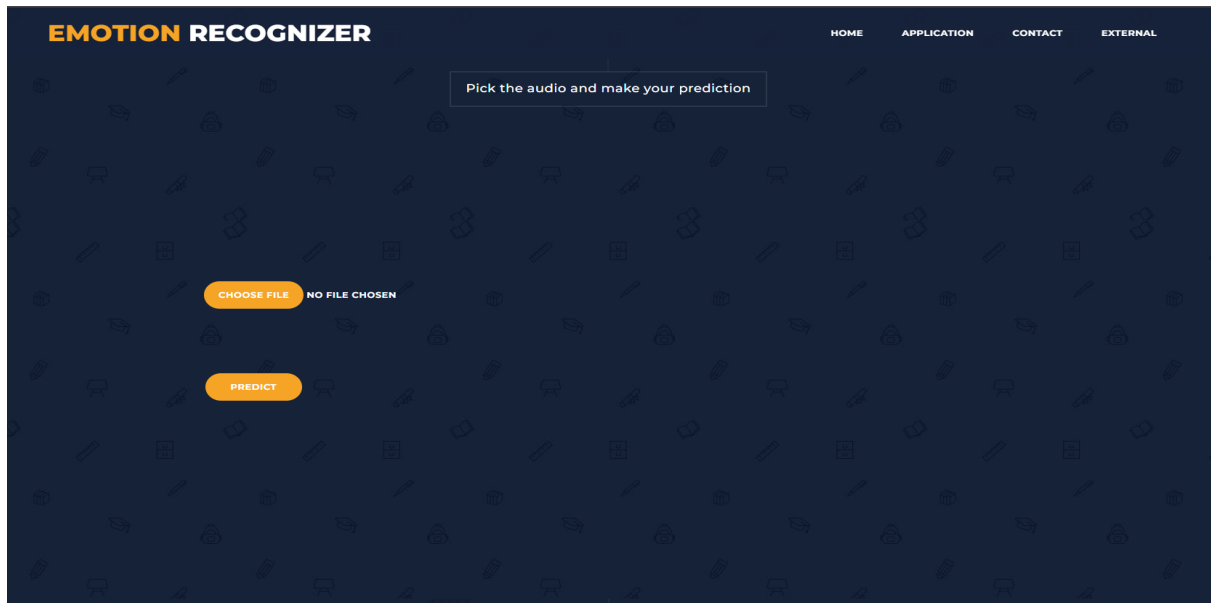
729 KB

Now let us have a look at the final result, how everything looks once we deploy it, first I will deploy it in my local machine and later we will deploy it in a public server. Deployment is very simple you simply set up your python environment and run the app.py application, since I already have Anaconda installed in my computer I will run the app.py file from there, otherwise you can create your separate virtual environment install all the dependencies their and run it from there, you don't need the entire anaconda package to run it, you just need python and the libraries that were used in the code.

```
C:\Users\S1-S2\emotion-classification-from-audio-files\legacy_code\Emotion Recognition>python app.py
2022-06-23 17:46:53.383224: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlderror: cudart64_110.dll not found
2022-06-23 17:46:53.383396: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
* Restarting with stat
```

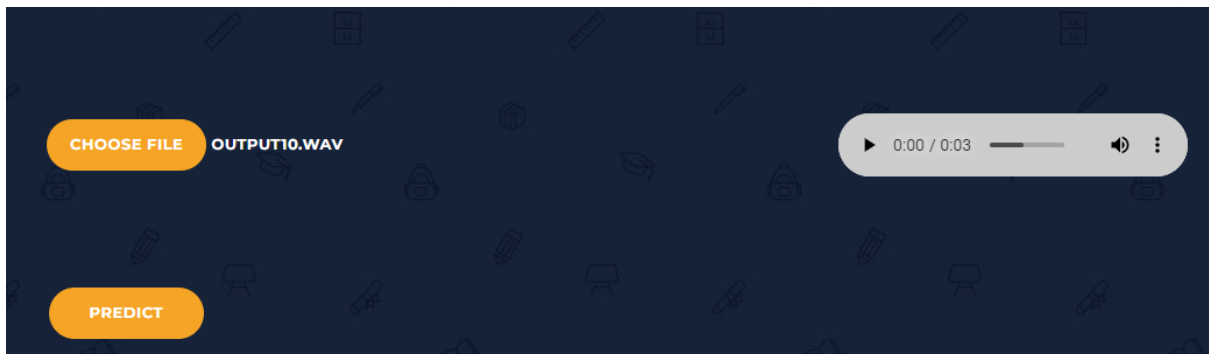
This how our app look like:



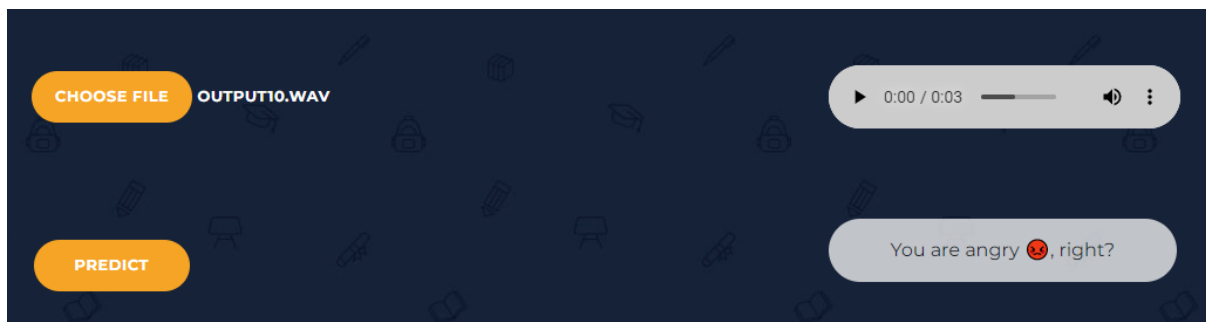


Now let us import a file from our local machine and see how our prediction page looks like.

After importing the file, it will be displayed.



After clicking on the “PREDICT” button, we got our prediction.



Future perspectives:

- improving model accuracy by:
 - Adding more data
 - Updating the model
 - Adding more features(Audio filters)
- Implementing the model in phone application
- Handling long audio files

...