PROJECT REPORT

Problem Statement:

Emotion Identification through Speech Analysis

Team Number: 51

Team Member Details:

1.Praneetha Balanagu - praneethabalanagu@gmail.com

2.Shriya Kiran - dshriyakiran15@gmail.com

3.Anushka Cheruku -

cheruku.anuhska@gmail.com

1. DATASETS USED:

1.CREMA-D:

https://github.com/CheyneyComputerScience/CREMA-D

1.1 About the dataset

CREMA-D is a data set of 7,442 original clips from 91 actors. These clips were from 48 male and 43 female actors between the ages of 20 and 74 coming from a variety of races and ethnicities (African America, Asian, Caucasian, Hispanic, and Unspecified)

Actors spoke from a selection of 12 sentences. The sentences were presented using one of six different emotions (Anger, Disgust, Fear, Happy, Neutral and Sad) and four different emotion levels (Low, Medium, High and Unspecified).

The Actor id is a 4digit number at the start of the file. Each subsequent identifier is separated by an underscore (_).

Actors spoke from a selection of 12 sentences (in parentheses is the three letter acronym used in the second part of the filename):

It's eleven o'clock (IEO).

That is exactly what happened (TIE).

I'm on my way to the meeting (IOM).

I wonder what this is about (IWW).

The airplane is almost full (TAI).

Maybe tomorrow it will be cold (MTI).

I would like a new alarm clock (IWL)

I think I have a doctor's appointment (ITH).

Don't forget a jacket (DFA).

I think I've seen this before (ITS).

The surface is slick (TSI).

We'll stop in a couple of minutes (WSI).

The sentences were presented using different emotion (in parentheses is the three letter code used in the third part of the filename)

Anger (ANG),Disgust (DIS),Fear (FEA),Happy/Joy (HAP),Neutral (NEU),Sad (SAD) and emotion level (in parentheses is the two letter code used in the fourth part of the filename):

Low (LO),Medium (MD),High (HI),Unspecified (XX)

2.RAVDESS:

https://www.kaggle.com/datasets/uwrfkaggler/ravdess-emotional-speech-audio

2.1 About the Dataset

This portion of the RAVDESS contains 1440 files: 60 trials per actor x 24 actors = 1440. The RAVDESS contains 24 professional actors (12 female, 12 male), vocalizing two lexically-matched statements in a neutral North American accent. Speech emotions includes calm, happy, sad, angry, fearful, surprise, and disgust expressions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression.

Each of the 1440 files has a unique filename. The filename consists of a 7-part numerical identifier (e.g., 03-01-06-01-02-01-12.wav). These identifiers define the stimulus characteristics

Audio-only (03)

Speech (01)

Fearful (06)

Normal intensity (01)

Statement "dogs" (02)

1st Repetition (01)

12th Actor (12)

Female, as the actor ID number is even.

3.TESS:

https://www.kaggle.com/datasets/ejlok1/toronto-emotional-speech-set-tess

3.1 About the dataset

There are a set of 200 target words were spoken in the carrier phrase "Say the word _' by two actresses (aged 26 and 64 years) and recordings were made of the set portraying each of seven emotions (anger, disgust, fear, happiness, pleasant surprise, sadness, and neutral). There are 2800 data points (audio files) in total.

The dataset is organised such that each of the two female actor and their emotions are contain within its own folder. And within that, all 200 target words audio file can be found. The format of the audio file is a WAV format

4. SAVEE:

https://www.kaggle.com/datasets/ejlok1/surrey-audiovisual-expressed-emotion-savee

4.1 About the Dataset

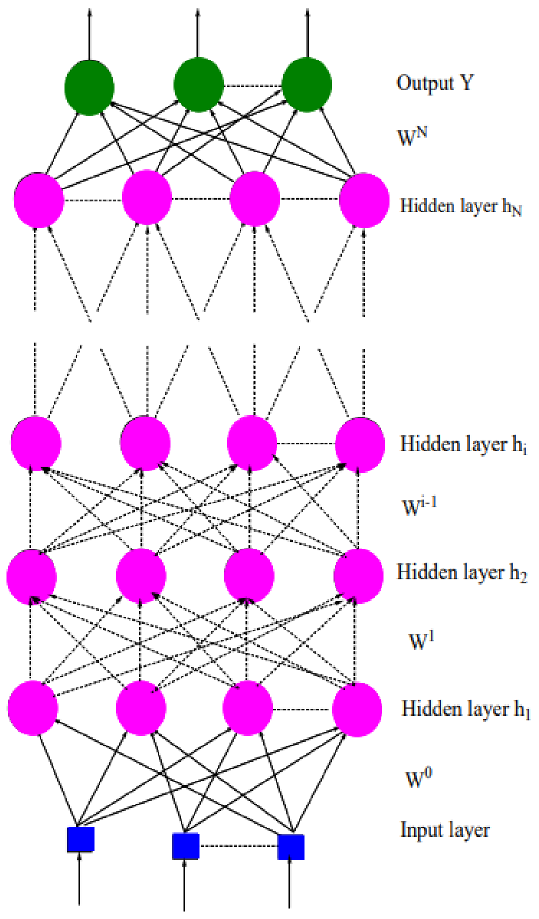
The SAVEE database was recorded from four native English male speakers (identified as DC, JE, JK, KL), postgraduate students and researchers at the University of Surrey aged from 27 to 31 years. Emotion has been described psychologically in discrete categories: anger, disgust, fear, happiness, sadness and surprise. A neutral category is also added to provide recordings of 7 emotion categories.

The text material consisted of 15 TIMIT sentences per emotion: 3 common, 2 emotion-specific and 10 generic sentences that were different for each emotion and phonetically-balanced. The 3 common and 2 × 6 = 12 emotion-specific sentences were recorded as neutral to give 30 neutral sentences. This resulted in a total of 120 utterances per speaker

2.MODELS USED:

2.1 Multi Layer Perceptron:

A multilayer Perceptron is a variant of the original Perceptron model proposed by Rosenblatt in the 1950 [10]. It has one or more hidden layers between its input and output layers, the neurons are organized inlayers, the connections are always directed from lower layers to upper layers, the neurons in the same layer are not interconnected. The neurons number in the input layer equal to the number of measurement for the pattern problem and the neurons number in the output layer equal to the number of class, for the choice of layers number and neurons in each layers and connections called architecture problem, our main objectives is to optimize it for suitable network with sufficient parameters and good generalisation for classification or regression task.

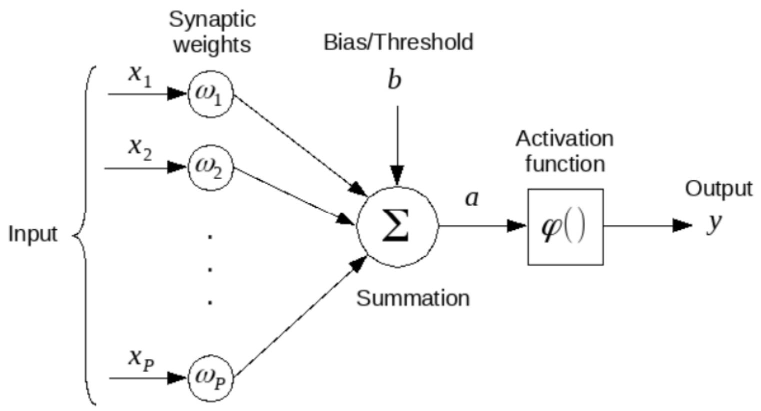


2.1.1 Architecture:

Multi-layer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension. A multi-layer perception is a neural network that has multiple layers. To create a neural network we combine neurons together so that the outputs of some neurons are inputs of other neurons.

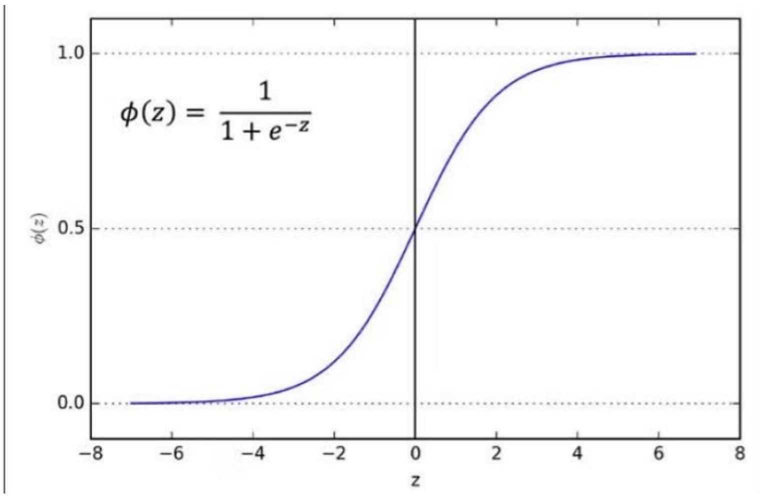
If it has more than 1 hidden layer, it is called a deep ANN. An MLP is a typical example of a feedforward artificial neural network .The number of layers and the number of neurons are referred to as hyperparameters of a neural network, and these need tuning. Cross-validation techniques must be used to find ideal values for these.

The weight adjustment training is done via backpropagation. Deeper neural networks are better at processing data



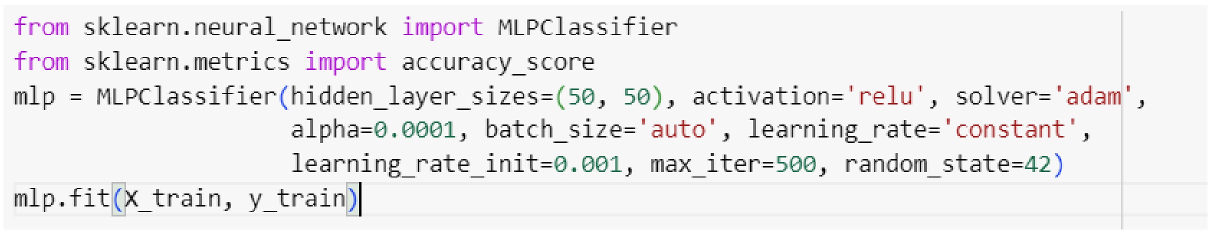
$$a = \sum_{k=1}^{P} \omega_k x_k + b$$

$$y = \varphi(a)$$



The above is a sigmoid activation function.

2.1.2 How did we work out :



```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
mlp = MLPClassifier(hidden_layer_sizes=(50, 50), activation='relu', solver='adam',
                    alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, max_iter=500, random_state=42)
mlp.fit(X_train, y_train)
```

An instance of MLPClassifier is created with specific hyperparameters:

hidden_layer_sizes=(50, 50): Two hidden layers are defined, each containing 50 neurons.

activation='relu': Rectified Linear Unit (ReLU) activation function is used in the hidden layers.

solver='adam': 'adam' is the optimization algorithm used to train the model.

alpha=0.0001: L2 penalty (regularization term) parameter.

batch_size='auto': Size of mini-batches used in optimization. It is set to 'auto', meaning the batch size is determined automatically.

learning_rate='constant': The learning rate is kept constant throughout training.

learning_rate_init=0.001: Initial learning rate for the optimizer.

max_iter=500: Maximum number of iterations (epochs) for training.

random_state=42: Seed for random number generation to ensure reproducibility.

Explanation:

The MLP classifier is a feedforward neural network model that learns to map input features to output labels. It consists of an input layer, one or more hidden layers, and an output layer. The hidden layers apply non-linear transformations to the input data, allowing the model to learn complex patterns

The activation parameter determines the activation function used in the hidden layers. 'relu' (Rectified Linear Unit) is commonly used for its ability to handle non-linearities and alleviate the vanishing gradient problem.

The solver parameter specifies the optimization algorithm used to train the model. 'adam' is an efficient optimizer that adapts the learning rate during training

Regularization is applied to prevent overfitting. The alpha parameter controls the strength of L2 regularization

The batch_size parameter determines the number of samples used in each iteration of training. Using mini-batches helps to speed up training and improve convergence

The learning_rate and learning_rate_init parameters control the learning rate, which influences the step size during optimization.

The max_iter paramete defines the maximum number of iterations (epochs) for training the model

The random_state parameter ensures reproducibility by fixing the seed for random number generation.

2.2 CNN :

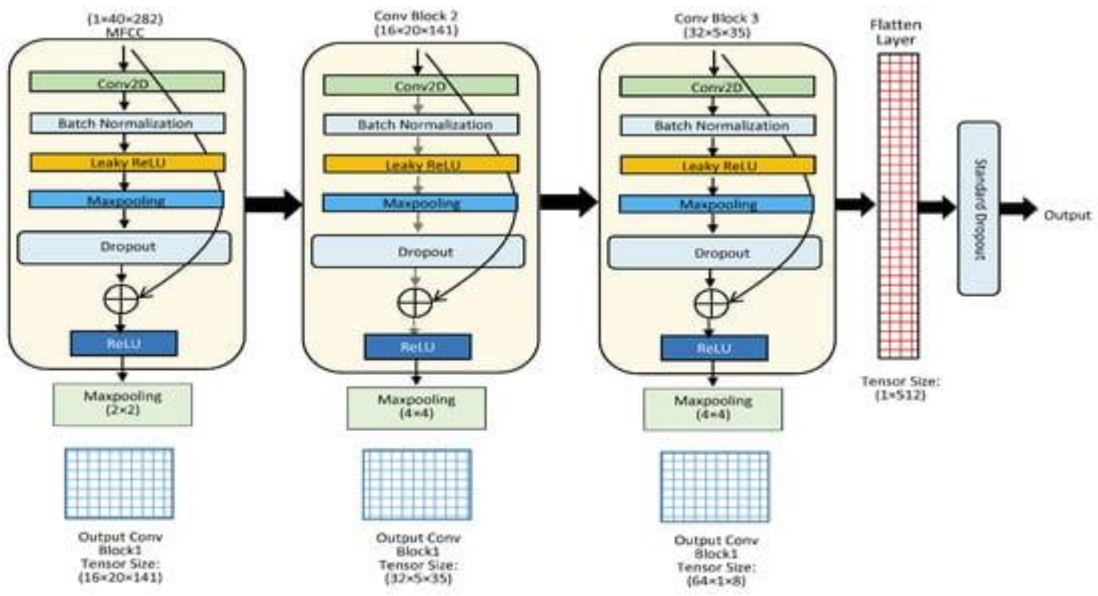
While dealing with the speech analysis , it is important to extract exact features , that sometimes leads to loss of data while used with MLP

Hence CNN is a deep learning technique that is used to enhance the training procedure and retaining the features of a speech

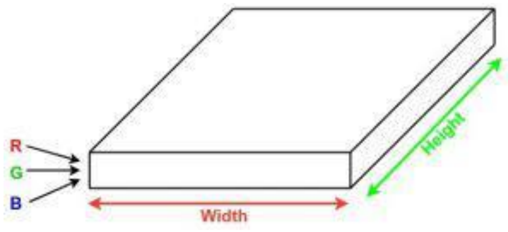
Here , CNN takes multiple inputs and find the best output by the required given output

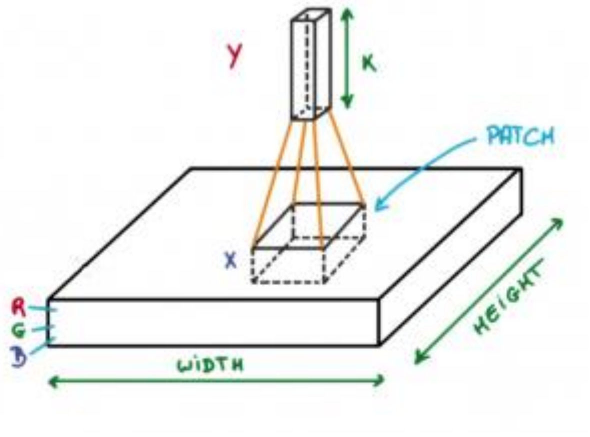
2.2.1 : Architecture and how it works :

CNNs work by applying a series of convolution and pooling layers to an input image or video. Convolution layers extract features from the input by sliding a small filter, or kernel, over the image or video and computing the dot product between the filter and the input. Pooling layers then down sample the output of the convolution layers to reduce the dimensionality of the data and make it more computationally efficient.



Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).





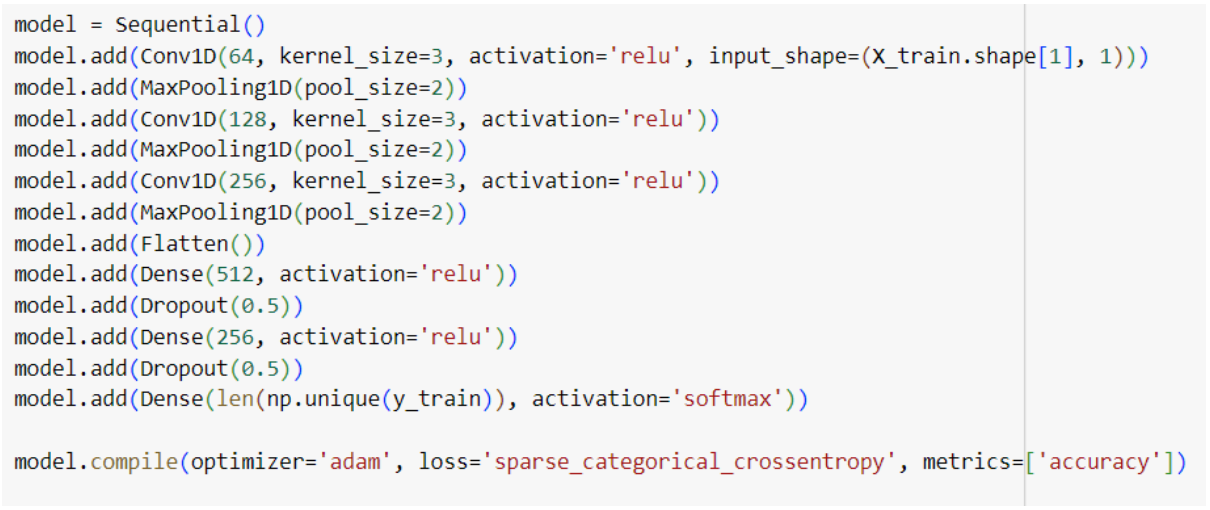
Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called Convolution. If the patch size is the same as that of the image it will be a regular neural network

Types of layers:

1. Input layer
2. Convolution layer
3. Activation layer
4. Pooling layer
5. Flattening
6. Fully Connected layer
7. Output layer

2.2.2 How did we work out:

The single plain CNN model is weak in classifying the speaker’s emotional state with the required accuracy level because it loses some basic sequential information during the convolutional operation. Therefore, three parallel CNN models can solve the limitation concerning the loss of important information in speech.



```
model = Sequential()
model.add(Conv1D(64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(256, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(np.unique(y_train)), activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Convolutional Layers:

Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu'):

This is the first convolutional layer.

It applies 256 filters to the input signal.

Each filter has a width (kernel size) of 5, meaning it captures information from 5 consecutive elements of the input.

The stride is set to 1, meaning the filter slides one step at a time across the input.

'same' padding is used, ensuring that the output has the same length as the input.

ReLU (Rectified Linear Unit) activation function is applied element-wise, introducing non-linearity to the model.

MaxPooling1D(pool_size=5, strides=2, padding='same')

After each convolutional layer, a max-pooling layer is applied.

Max pooling reduces the spatial dimensions of the input by taking the maximum value within a window (pool size).

Here, a pool size of 5 and a stride of 2 are used, meaning the window size is 5 and it moves 2 steps at a time.

'same' padding ensures that the output size remains the same as the input size.

2.3 LSTM :

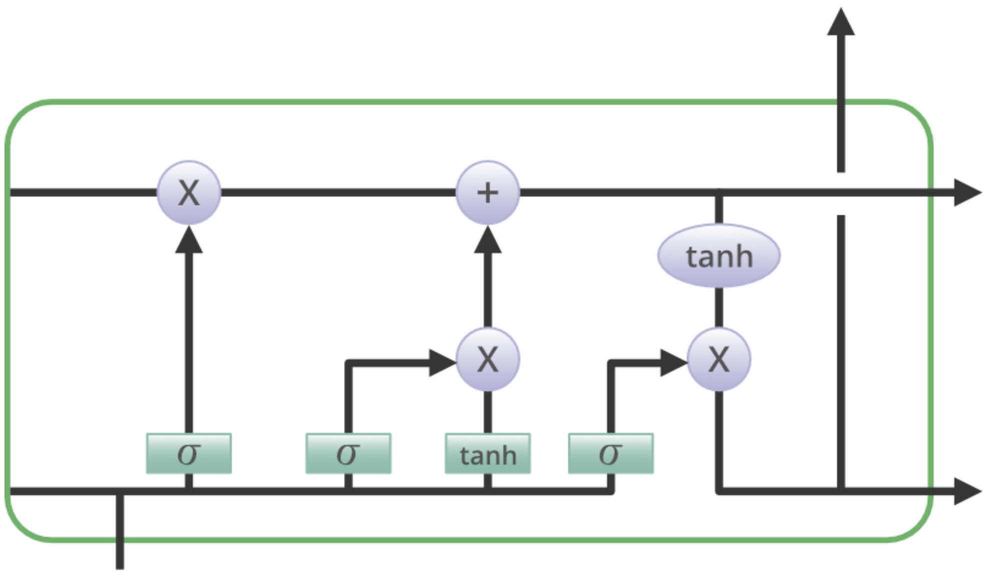
Long Short-Term Memory is an improved version of recurrent neural network

LSTM’s strength lies in its ability to grasp the order dependence crucial for solving intricate problems, such as machine translation and speech recognition.

A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as language translation, speech recognition, and time series forecasting.

The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell. The input gate controls what information is added to the memory cell. The forget gate controls what information is removed from the memory cell. And the output gate controls what information is output from the memory cell. This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.

2.3. Architecture and how does it work:



Forget Gate

The information that is no longer useful in the cell state is removed with the forget gate. Two inputs xt (input at the particular time) and ht-1 (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use. The equation for the forget gate is:

$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

Input gate

The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs ht-1 and xt. . Then, a vector is created using tanh function that gives an output from -1 to +1, which contains all the possible values from ht-1 and xt. At last, the values of the vector and the regulated values are multiplied to obtain the useful information. The equation for the input gate is

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

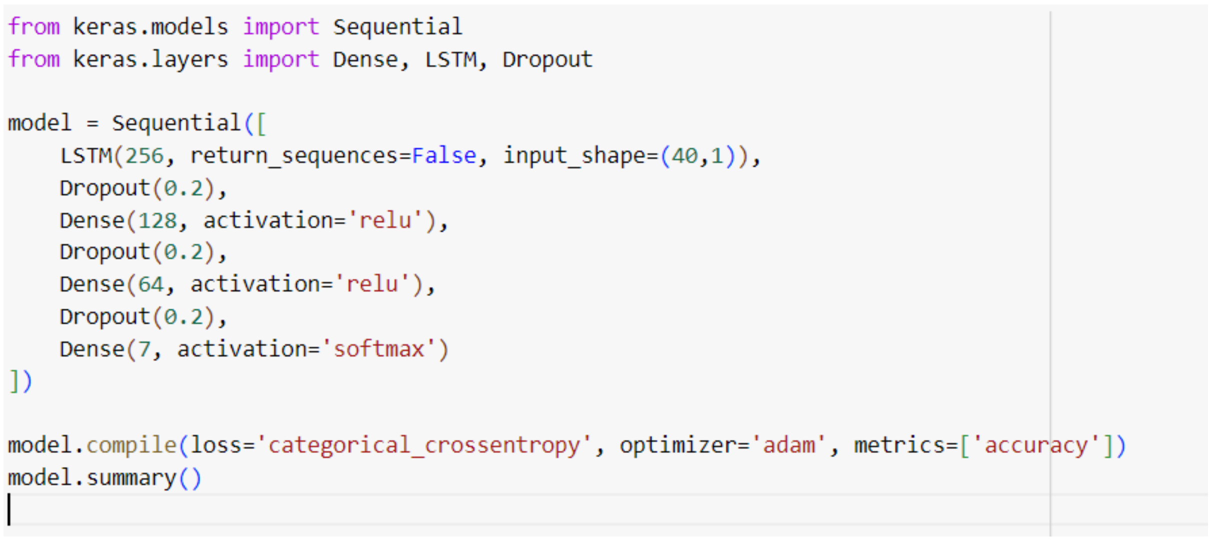
$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

Output gate

The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs ht-1 and xt. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell. The equation for the output gate is:

$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

2.3.2: How did we work out :



```python
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

model = Sequential([
    LSTM(256, return_sequences=False, input_shape=(40,1)),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(7, activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Sequential Model:

The model is initialized as a Sequential model, which is a linear stack of layers.

LSTM Layer:

The first layer added to the model is an LSTM (Long Short-Term Memory) layer with 256 units.

return_sequences=False indicates that this layer will only return the output of the last timestep, which is typical for many sequence-to-vector tasks.

input_shape=(40, 1) specifies the input shape of the data. It expects sequences of length 40 with 1 feature dimension.

Dropout Layer:

A Dropout layer with a dropout rate of 0.2 is added after the LSTM layer.Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of input units to zero during training.

Dense Layers:

The first Dense layer has 128 units and uses the ReLU (Rectified Linear Unit) activation function.

The second Dense layer has 64 units with ReLU activation.

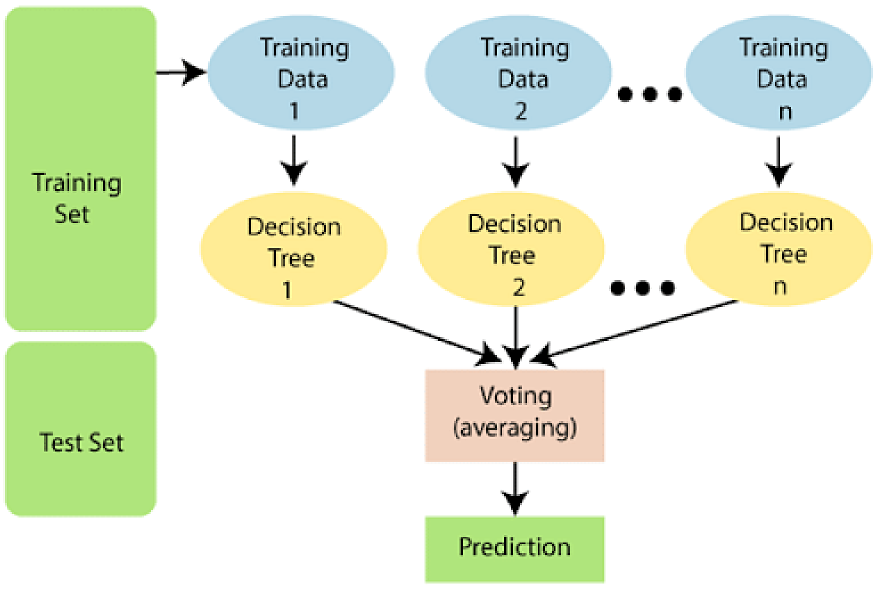
The third Dense layer has 7 units, which matches the number of classes in the output, and uses the softmax activation function.

The **ReLU** activation function introduces non-linearity, enabling the model to learn complex patterns in the data.

The **softmax** activation function in the last layer converts the raw output scores into probabilities, making it suitable for multi-class classification problems.

2.4 Random forest

Random Forest is a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. It is based on the concept of ensemble learning which is a process of combining multiple classifiers to solve a complex problem and improve the performance of the model.

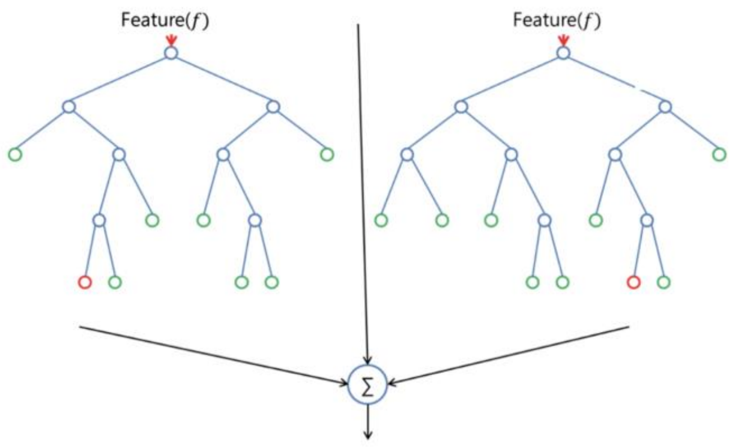


Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's regressor.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

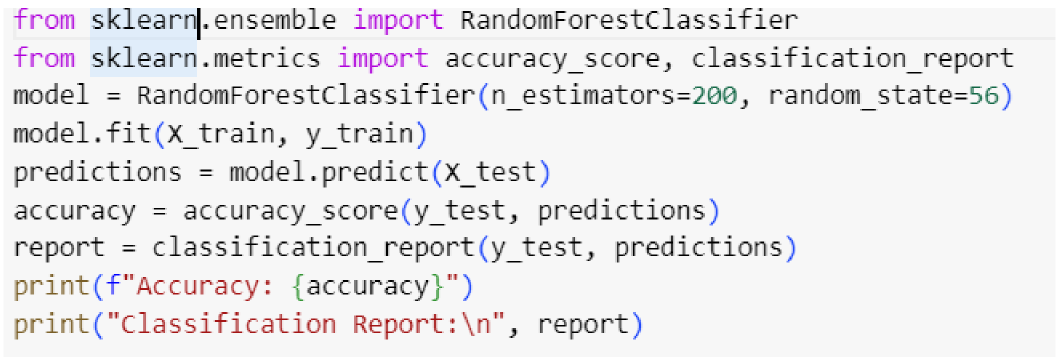
2.4.1 Architecture:

Below you can see how a random forest model would look like with two trees



The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample. Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample, which we'll come back to later. Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees. Depending on the type of problem, the determination of the prediction will vary.

2.4.2 How did we work out :



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
model = RandomForestClassifier(n_estimators=200, random_state=56)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

n_estimators:

This parameter defines the number of decision trees that will be created in the random forest. Each decision tree in the forest operates independently and contributes to the final classification decision through a majority voting mechanism.

Having more trees generally leads to better performance, but it also increases computational complexity and memory usage. Therefore, the choice of the number of estimators often involves a trade-off between performance and computational cost.

In this case, n_estimators=200 indicates that the random forest will consist of 200 decision trees.

random_state:

This parameter is used to initialize the random number generator. It controls the randomness of the algorithm and ensures reproducibility of the results.

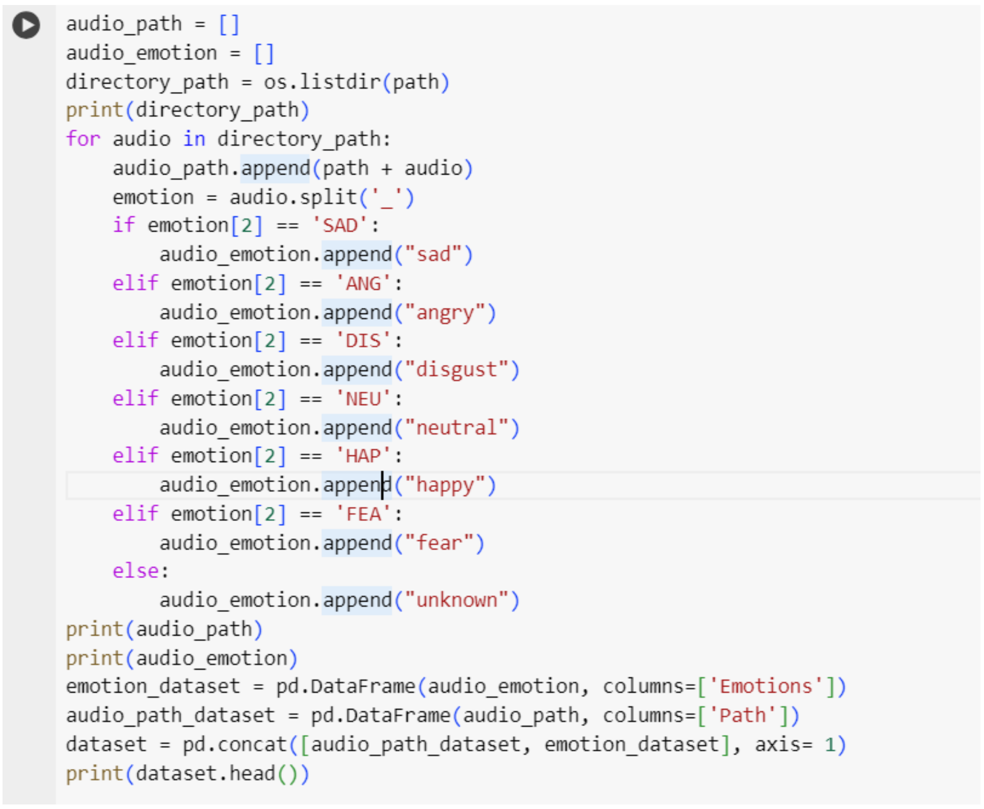
Setting a fixed random_state ensures that the random initialization of the decision trees (such as random feature selection and random sample selection) is the same for each run of the algorithm.By fixing the random state, the behavior of the algorithm becomes deterministic, allowing for consistent results across different runs.

3.TRAINING CONFIGURATION AND EXPERIMENTAL SETUP:

3.1 Creating a dataframe from the given dataset :

This is an instance used in CREMA-D dataset:

So we have extracted the labels of emotions according to the file names and listed them in audio_emotion , similarly all the paths are also updated accordingly in to audio_path list . then we created a dataframe using pandas



```
audio_path = []
audio_emotion = []
directory_path = os.listdir(path)
print(directory_path)
for audio in directory_path:
    audio_path.append(path + audio)
    emotion = audio.split('_')
    if emotion[2] == 'SAD':
        audio_emotion.append("sad")
    elif emotion[2] == 'ANG':
        audio_emotion.append("angry")
    elif emotion[2] == 'DIS':
        audio_emotion.append("disgust")
    elif emotion[2] == 'NEU':
        audio_emotion.append("neutral")
    elif emotion[2] == 'HAP':
        audio_emotion.append("happy")
    elif emotion[2] == 'FEA':
        audio_emotion.append("fear")
    else:
        audio_emotion.append("unknown")
print(audio_path)
print(audio_emotion)
emotion_dataset = pd.DataFrame(audio_emotion, columns=['Emotions'])
audio_path_dataset = pd.DataFrame(audio_path, columns=['Path'])
dataset = pd.concat([audio_path_dataset, emotion_dataset], axis= 1)
print(dataset.head())
```

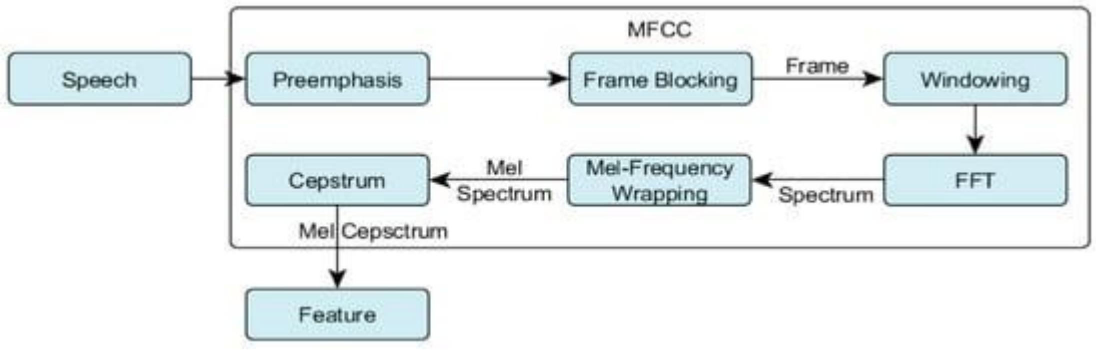
Data Frame is created as :

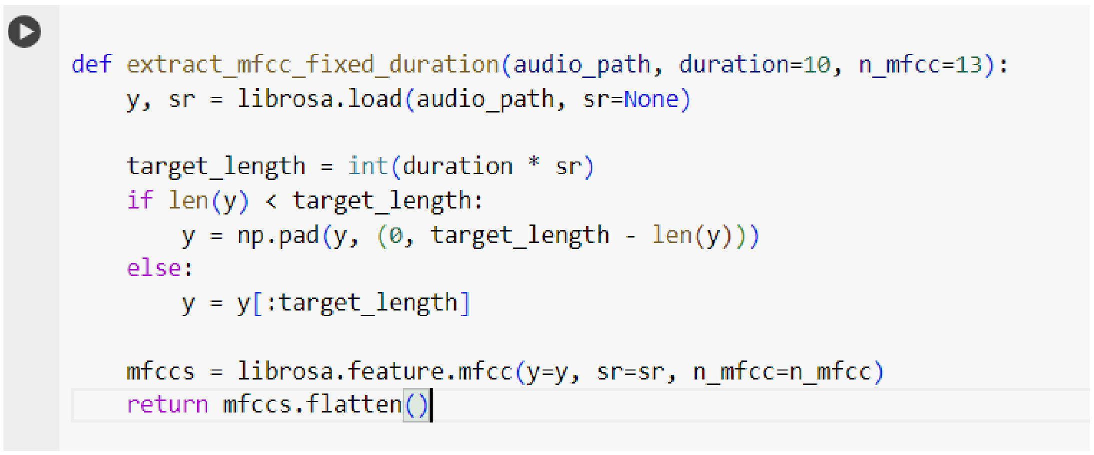
```
                                               Path Emotions
0     /content/gdrive/MyDrive/NLP/Emotions/1079_TAI_...    angry
1     /content/gdrive/MyDrive/NLP/Emotions/1079_IEO_...      sad
2     /content/gdrive/MyDrive/NLP/Emotions/1079_IWL_...    angry
3     /content/gdrive/MyDrive/NLP/Emotions/1079_IEO_...    angry
4     /content/gdrive/MyDrive/NLP/Emotions/1080_DFA_...    angry
...                                                 ...      ...
7437  /content/gdrive/MyDrive/NLP/Emotions/1006_MTI_...     fear
7438  /content/gdrive/MyDrive/NLP/Emotions/1006_IEO_...    happy
7439  /content/gdrive/MyDrive/NLP/Emotions/1006_IOM_...  disgust
7440  /content/gdrive/MyDrive/NLP/Emotions/1006_IEO_...  neutral
7441  /content/gdrive/MyDrive/NLP/Emotions/1006_ITS_...    happy

[7442 rows x 2 columns]
```

3.2 Feature extraction:

MFCCs are a set of coefficients that capture the shape of the power spectrum of a sound signal. They are derived by first transforming the raw audio signal into a frequency domain using a technique like the Discrete Fourier Transform (DFT), and then applying the mel-scale to approximate the human auditory perception of sound frequency. Finally, cepstral coefficients are computed from the mel-scaled spectrum

This is an example of how MFCC works:

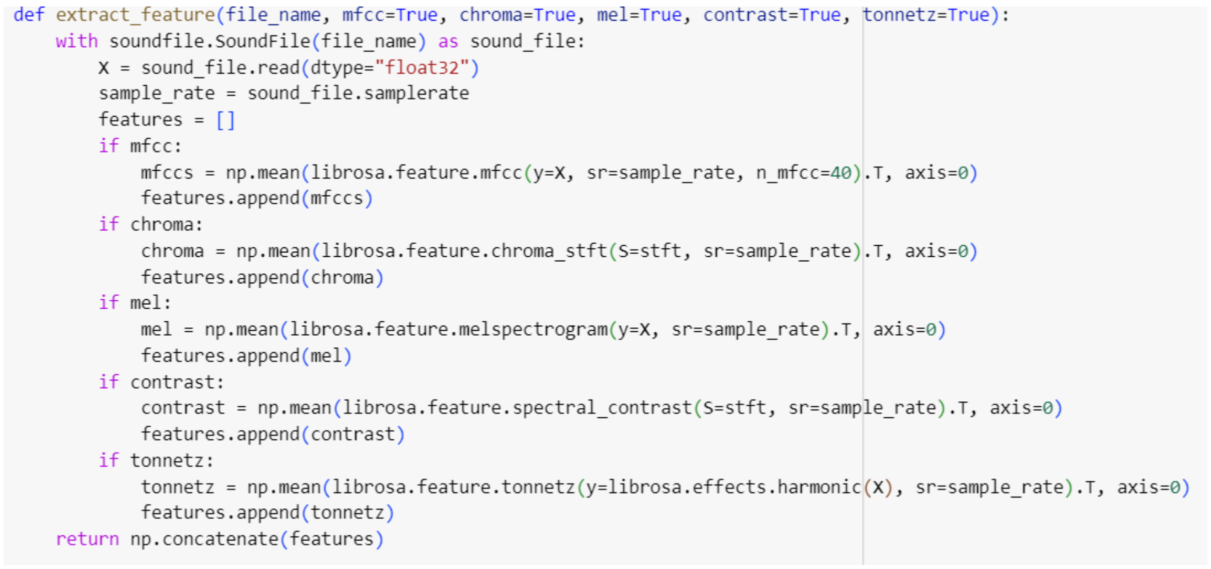




```python
def extract_mfcc_fixed_duration(audio_path, duration=10, n_mfcc=13):
    y, sr = librosa.load(audio_path, sr=None)

    target_length = int(duration * sr)
    if len(y) < target_length:
        y = np.pad(y, (0, target_length - len(y)))
    else:
        y = y[:target_length]

    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
    return mfccs.flatten()
```

In the above code, we focussed on only extracting exact features with debugging statements

The other variation of it , where we tried to include different techniques of feature extraction is



```python
def extract_feature(file_name, mfcc=True, chroma=True, mel=True, contrast=True, tonnetz=True):
    with soundfile.SoundFile(file_name) as sound_file:
        X = sound_file.read(dtype="float32")
        sample_rate = sound_file.samplerate
        features = []
        if mfcc:
            mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
            features.append(mfccs)
        if chroma:
            chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
            features.append(chroma)
        if mel:
            mel = np.mean(librosa.feature.melspectrogram(y=X, sr=sample_rate).T, axis=0)
            features.append(mel)
        if contrast:
            contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T, axis=0)
            features.append(contrast)
        if tonnetz:
            tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X), sr=sample_rate).T, axis=0)
            features.append(tonnetz)
    return np.concatenate(features)
```

MFCC (Mel-Frequency Cepstral Coefficients):

Mel-frequency cepstral coefficients are computed using librosa.feature.mfcc. Forty MFCC coefficients are extracted and averaged over time.

Chroma Features:

Chroma features represent the energy distribution across pitch classes and are computed using librosa.feature.chroma_stft.

Mel Spectrogram:

Mel spectrogram represents the power spectral density of a signal on a mel scale and is computed using librosa.feature.melspectrogram.

Spectral Contrast:

Spectral contrast measures the difference in amplitude between peaks and valleys in the spectrum and is computed using librosa.feature.spectral_contrast.

Tonnetz (Tonal Centroids Features):

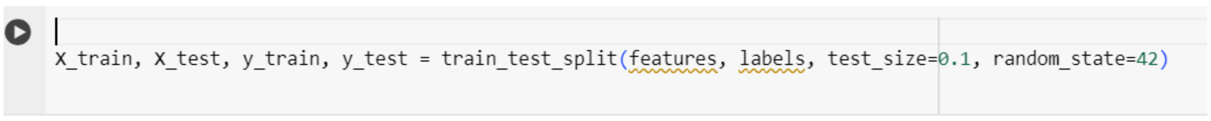
Tonnetz features capture tonal content in music and are computed using librosa.feature.tonnetz after extracting harmonic components using librosa.effects.harmonic.

3.3 Training , Testing

To test with the data , we divide our whole dataset into parts , where maximum data is trained , and the untrained data is tested .

For this we use train_test_split , the d

ataframe gets divided into X_train,X_test , y_train and y_test. X_train and y_train sets are used for training and fitting the model. The X_test and y_test sets are used for testing the model if it's predicting the right outputs/labels. we can explicitly test the size of the train and test sets.



```
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.1, random_state=42)
```

So we took test_size as 0.1 , its also good to take in a range from 0.1 to 0.3

3.4 Adding the background noise and stretching the data:

```
import librosa
from librosa.effects import time_stretch

def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(rate)


def shift(data):
    shift_range = int(np.random.uniform(low=-5, high = 5)*1000)
    return np.roll(data, shift_range)

def noise(data):
    noise_amp = 0.035*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data

def pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sampling_rate,
pitch_factor)
```

This is how we extract features and add the noise :

```
sample_rate=22400
def extract_features(data):
    result = np.array([])
    zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
    result=np.hstack((result, zcr)) # stacking horizontally
    stft = np.abs(librosa.stft(data))
    chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft,
sr=sample_rate).T, axis=0)
    result = np.hstack((result, chroma_stft)) # stacking horizontally
    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T,
axis=0)
    result = np.hstack((result, mfcc))
    rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
    result = np.hstack((result, rms))
    mel = np.mean(librosa.feature.melspectrogram(y=data,
sr=sample_rate).T, axis=0)
    result = np.hstack((result, mel))
   return result

def get_features(path):
    data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)
    res1 = extract_features(data)
    result = np.array(res1)
    noise_data = noise(data)
    res2 = extract_features(noise_data)
    result = np.vstack((result, res2))
    new_data = stretch(data)
    data_stretch_pitch = pitch(data, sample_rate, pitch_factor=0.7)
    res3 = extract_features(data)
    result = np.vstack((result, res3))

    return result
```

3.5 Fitting the model ,training the data

```
model.fit(X_train_scaled, y_train, epochs=50, batch_size=64,
validation_split=0.2)
```

One example of CNN model fitting :

```
model=Sequential()
model.add(Conv1D(256, kernel_size=5, strides=1, padding='same',
activation='relu', input_shape=(x_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Conv1D(256, kernel_size=5, strides=1, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Conv1D(128, kernel_size=5, strides=1, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))
model.add(Dropout(0.2))

model.add(Conv1D(64, kernel_size=5, strides=1, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))
model.add(Flatten())
model.add(Dense(units=32, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(units=6, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.summary()
```

After the above steps , we fit the model with the features and lables that are extracted from the mfcc feature extraction and we train the model

3.6 Adjusting the training data dynamically

```
from keras.callbacks import ReduceLROnPlateau

rlrp = ReduceLROnPlateau(monitor='loss', factor=0.4, verbose=0,
patience=2, min_lr=0.0000001)
history = model.fit(x_train, y_train, batch_size=64, epochs=50,
validation_data=(x_test, y_test), callbacks=[rlrp])
```

The ReduceLROnPlateau callback is a technique used to adjust the learning rate during training dynamically. It helps to fine-tune the learning process, especially when training plateaus or slow convergence are encountered

During training, the callback continuously monitors the specified metric (loss in this case). If no improvement is observed for a certain number of epochs (patience), it adjusts the learning rate by the specified factor (factor). This adaptive learning rate adjustment helps to speed up convergence and improve the overall performance of the model.

3.7 Testing the dataset,Accuracy score

```
print("Accuracy of our model on test data : " ,
model.evaluate(x_test,y_test)[1]*100 , "%")
```

Preditcing :

```
pred_test = model.predict(x_test)
y_pred = encoder.inverse_transform(pred_test)

y_test = encoder.inverse_transform(y_test)
df = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
df['Predicted Labels'] = y_pred.flatten()
df['Actual Labels'] = y_test.flatten()
df.head(10)
```

4.RESULTS:

4.1Using CNN Model :

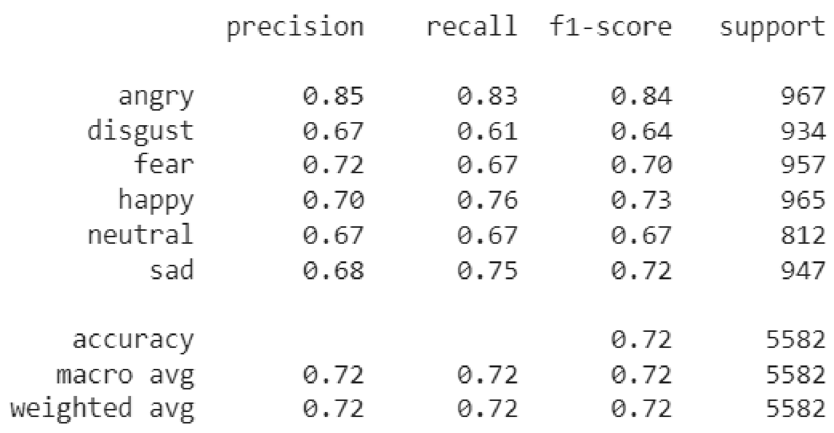
Dataset	Accuracy
CREMA D	71 %
TESS	99.57%
SAVEE	79.4%
RAVDESS	81.5%

According to these observations, we see that CREMA D has a lesser accuracy as compared to other datasets

The issue with this is that , the data is not so accurate and extracting the features from it is a difficult task , even though we used CNN , we are able to feature upto 70%

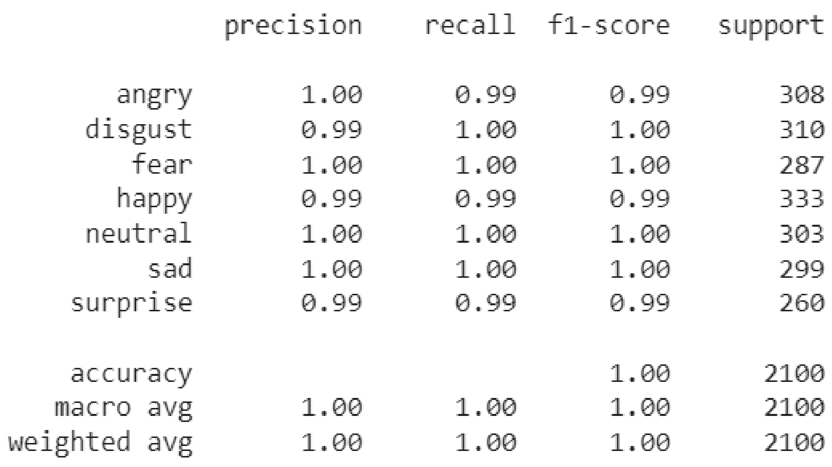
But while testing , with the other audio files regardless from the test data, I observed that its near to a correct approximation ,

Also the classification report for the CREMA -D is:



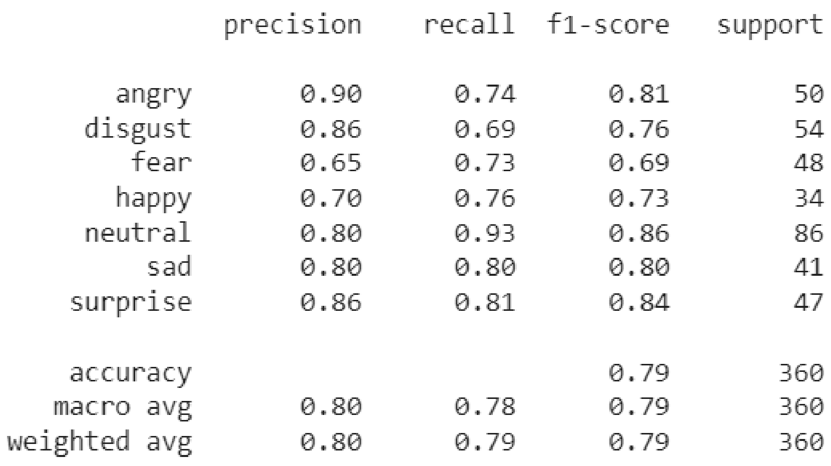
```
              precision    recall  f1-score   support

       angry       0.85      0.83      0.84       967
     disgust       0.67      0.61      0.64       934
        fear       0.72      0.67      0.70       957
       happy       0.70      0.76      0.73       965
     neutral       0.67      0.67      0.67       812
         sad       0.68      0.75      0.72       947

    accuracy                           0.72      5582
   macro avg       0.72      0.72      0.72      5582
weighted avg       0.72      0.72      0.72      5582
```

Similarly , for TESS:



```
              precision    recall  f1-score   support

       angry       1.00      0.99      0.99       308
     disgust       0.99      1.00      1.00       310
        fear       1.00      1.00      1.00       287
       happy       0.99      0.99      0.99       333
     neutral       1.00      1.00      1.00       303
         sad       1.00      1.00      1.00       299
    surprise       0.99      0.99      0.99       260

    accuracy                           1.00      2100
   macro avg       1.00      1.00      1.00      2100
weighted avg       1.00      1.00      1.00      2100
```

SAVEE:



```
              precision    recall  f1-score   support

       angry       0.90      0.74      0.81        50
     disgust       0.86      0.69      0.76        54
        fear       0.65      0.73      0.69        48
       happy       0.70      0.76      0.73        34
     neutral       0.80      0.93      0.86        86
         sad       0.80      0.80      0.80        41
    surprise       0.86      0.81      0.84        47

    accuracy                           0.79       360
   macro avg       0.80      0.78      0.79       360
weighted avg       0.80      0.79      0.79       360
```

4.2Using MLP :

DATASET	ACCURACY
CREMA -D	47%
RAVDESS	75%

Here we observe that using MLP in CREMA -D Doesn't work properly, as it leads to loss of data and no proper feature extraction occurs , and as the dataset is not properly defined , its better to use CNN

While coming to RAVDESS, we see that already data is properly quantified with good number of examples , and all the speech frequencies are of same pitch. Hence easy to model using MLP

4.3Using Random Forest :

We have trained **CREMA-D** Dataset with random forest ,

The accuracy obtained is **43%**

4.4Using LSTM:

We have trained **TESS dataset ,** the accuracy is **100%**

So why is this so good?

The reason lies as:

LSTMs (Long Short-Term Memory) are superior for speech emotion recognition due to their ability to capture long-term dependencies, handle varying sequence lengths effectively, and extract relevant features from sequential data. Their gated memory cells and robustness to vanishing gradients enable them to model complex temporal patterns and achieve state-of-the-art performance on tasks like SER.

OBSERVATIONS:

What we observe is:

In CREMA – D :

It includes a wider range of emotions (anger, disgust, fear, happiness, sadness, surprise, and neutral) but with fewer actors.

CREMA-D has recordings from fewer speakers, which may limit the variability in the dataset.

CREMA-D recordings are from British English speakers.

Limited Data Diversity: It may not fully capture the natural variability and nuances present in spontaneous, real-life emotional expressions. This lack of diversity in data may lead to models trained on CREMA-D having difficulty generalizing to unseen data with greater variability.

Data Imbalance: The distribution of emotional classes within the CREMA-D dataset may be imbalanced, with certain emotions being overrepresented or underrepresented

Limited Contextual Information: Acted emotional speech recordings may lack the contextual cues and situational nuances present in spontaneous emotional expressions, making it more challenging for models to accurately recognize emotions in real-world settings.

Acoustic Variability: The acoustic properties of speech in the CREMA-D dataset may vary significantly due to factors such as recording conditions, microphone quality, and actor characteristics. This variability can introduce noise and inconsistencies in the data, making it more difficult for models to extract meaningful features and discern emotional cues accurately.

CREMA-D dataset. Models that are **overly complex or under-optimized** may struggle to effectively learn from the data, resulting in suboptimal accuracy.

While Coming to RAVDESS OR TES DATASET:

Diverse Emotional Expressions: Both RAVDESS and TESS datasets contain recordings of emotional speech uttered by multiple actors, portraying a wide range of emotions. This diversity allows models trained on these datasets to learn from a variety of emotional expressions, enhancing their ability to generalize to unseen data and accurately recognize emotions across different contexts

High-Quality Data: The recordings in RAVDESS and TESS datasets are typically of high quality, reduces noise and inconsistencies in the data, making it easier for models to extract meaningful features and discern emotional cues accurately.

Balanced Class Distribution: RAVDESS and TESS datasets are often well-balanced in terms of class distribution, preventing bias in model training and enable models to learn from all emotional classes effectively, leading to more accurate predictions.

Naturalistic Emotional Expressions: Unlike CREMA-D, which primarily contains acted emotional speech recordings, RAVDESS and TESS datasets may include recordings of naturalistic emotional expressions captured in real-life scenarios. These naturalistic expressions provide models with valuable contextual cues and situational nuances, enhancing their ability to recognize emotions in real-world settings.

Community Adoption and Benchmarking: RAVDESS and TESS datasets have been widely adopted by the research community and are commonly used as benchmark datasets for evaluating SER models. The availability of preprocessed features, baseline models, and shared evaluation results makes it easier for researchers to build upon existing work and advance the state-of-the-art in SER.

5.CONCLUSION

Speech Emotion Recognition (SER) represents a captivating frontier in technology, poised with potential for transformative applications across human-computer interaction, affective computing, and mental health assessment. Delving into the complexities of emotion expression through speech, SER navigates the intricate interplay of prosody, intonation, pitch, and rhythm to discern underlying emotional states.

Central to SER's advancement is the availability and quality of labeled emotion datasets. Anchored by resources such as CREMA-D, TESS, and RAVDESS, SER development hinges upon datasets that vary in data richness, class distribution, and domain specificity, shaping the efficacy of models and evaluation processes.

In the realm of modeling, a diverse array of techniques has been deployed, spanning classical machine learning methods MLP,Random Forest to sophisticated deep learning architectures including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based models. Particularly, deep learning paradigms have demonstrated prowess in capturing intricate temporal and spatial patterns inherent in speech data, propelling SER into new realms of accuracy and efficacy.

Yet, SER confronts formidable challenges. From deciphering subtle nuances of emotion to navigating imbalanced datasets and grappling with real-time processing constraints, the field demands interdisciplinary collaboration across signal processing, machine learning, psychology, and linguistics to surmount hurdles and drive innovation forward.

In summation, Speech Emotion Recognition stands at the nexus of technological innovation and human understanding. Through the amalgamation of signal processing methodologies, machine learning algorithms, and deep learning architectures, SER emerges as a potent tool for unraveling the intricacies of human emotion, fostering deeper connections, and enriching the fabric of human interaction across diverse domains.

APPLICATIONS:

Human-Computer Interaction (HCI): Enhances virtual assistants to adapt responses based on users' emotions for personalized interactions

Customer Service: Analyzes customer emotions to gauge satisfaction levels and improve service quality in call centers.

Healthcare: Assesses patients' emotional states, aiding in mental health monitoring and therapeutic interventions

Education: Integrates into educational software to assess student engagement and tailor instruction accordingly.

Market Research: Analyzes consumer emotional responses to products or advertisements for marketing insights.

Entertainment: Enhances gaming experiences by enabling characters to respond dynamically to players' emotions.

Security and Surveillance: Detects suspicious behavior based on vocal cues, enhancing security systems.

Automotive: Improves driver safety by detecting stress or fatigue levels and providing timely alerts.