Speech Emotion Recognition

Aana Kakroo (20BAI1138), Krish Bagga (20BAI1044)

Abstract

Speech emotion recognition (SER) is a rapidly developing field that seeks to identify and categorize human emotions based on speech signals. This technology has the potential to be used in a variety of applications, including call centers, customer service, and healthcare. In this paper, we present a novel approach to SER that integrates deep learning and transfer learning. Deep learning, a type of machine learning, is well-suited for SER because it can effectively capture intricate patterns from data. Transfer learning, a technique that utilizes data from a related task to enhance the performance of deep learning models, is also incorporated into our approach. Our proposed approach achieves an accuracy of 61%, surpassing the performance of existing SER systems by a significant margin. Our findings highlight the promising potential of deep learning and transfer learning for SER applications.

Introduction

The human voice is a rich source of information, conveying not only the meaning of words but also a multitude of emotional cues. Speech emotion recognition (SER), an emerging field of study, aims to decipher these subtle emotional nuances from spoken language. The ability to automatically recognize emotions from speech holds immense potential for a diverse range of applications, including call centers, customer service, healthcare, education, human-computer interaction.

Despite its promising applications, SER remains a challenging task due to the complexity of human emotions and the subtle cues conveyed through speech. Researchers have explored various approaches to SER, including traditional machine learning techniques, but deep learning has emerged as a promising avenue.

Deep learning, characterized by its ability to learn complex patterns from data, offers significant advantages for SER. By analyzing large datasets of speech recordings labeled with corresponding emotional states, deep learning models can extract intricate patterns and relationships that may not be apparent to traditional approaches.

Transfer learning, a technique that utilizes knowledge learned from a related task to enhance the performance of a model on a new task, has also gained traction in SER research. By leveraging pre-trained deep learning models on tasks such as speech recognition or natural language processing, SER models can benefit from the vast knowledge base of these models, leading to improved performance.

The integration of deep learning and transfer learning has the potential to revolutionize SER, enabling more accurate and robust recognition of human emotions from speech. As SER technology matures, it holds the promise of transforming various aspects of human-computer interactions and revolutionizing the way we communicate with machines.

Proposed Work

In this paper, we propose a new approach to SER that combines deep learning and transfer learning. Deep learning is a type of machine learning that is well-suited for SER because it can learn complex patterns from data. Transfer learning is a technique that can be used to improve the performance of deep learning models by using data from a related task.

1. Data Collection and Preprocessing

The first step in our proposed approach was to collect a dataset of speech recordings that were labeled with the six basic emotions: anger, disgust, fear, happiness, sadness, and surprise. We collected a total of approximately 12,000 speech recordings from a variety of sources, including online databases, movie dialogues, and real-world recordings.

We preprocessed the speech recordings to remove noise and normalize the signal. We used a variety of preprocessing techniques, including:

- Normalization: We normalized the amplitude of the speech recordings to a standard range.
- Noise reduction: We used a noise reduction algorithm to remove background noise from the speech recordings.
- Segmentation: We segmented the speech recordings into short clips, each representing a single emotional utterance.

2. Data Augmentation

To increase the size of the dataset to approximately 36,000 and improve the performance of the deep learning model, we applied data augmentation techniques to the preprocessed speech recordings. Data augmentation techniques artificially increase the size of the dataset by creating new examples from existing data.

We used the following data augmentation techniques:

- Pitch shifting: We shifted the pitch of the speech recordings up or down by a random amount.
- Time stretching: We stretched the duration of the speech recordings by a random amount.
- Adding noise: We added random noise to the speech recordings.

3. Feature Extraction

After preprocessing and data augmentation, we extracted features from the preprocessed signals. We used a variety of feature extraction techniques, including:

- Mel-frequency cepstral coefficients (MFCCs): MFCCs are a type of feature that captures the spectral characteristics of a speech signal.
- Pitch: Pitch is the fundamental frequency of a speech signal.
- Energy: Energy is a measure of the loudness of a speech signal.

We extracted these features for each frame of a speech recording. Each frame represents a short segment of the speech signal, typically 25 milliseconds long.

4. Deep Learning Model Training

We trained a deep learning model on the extracted features and the corresponding emotion labels. The deep learning model was a convolutional neural network (CNN), a type of neural network that is well-suited for image and audio classification tasks.

The CNN model consisted of several layers, including:

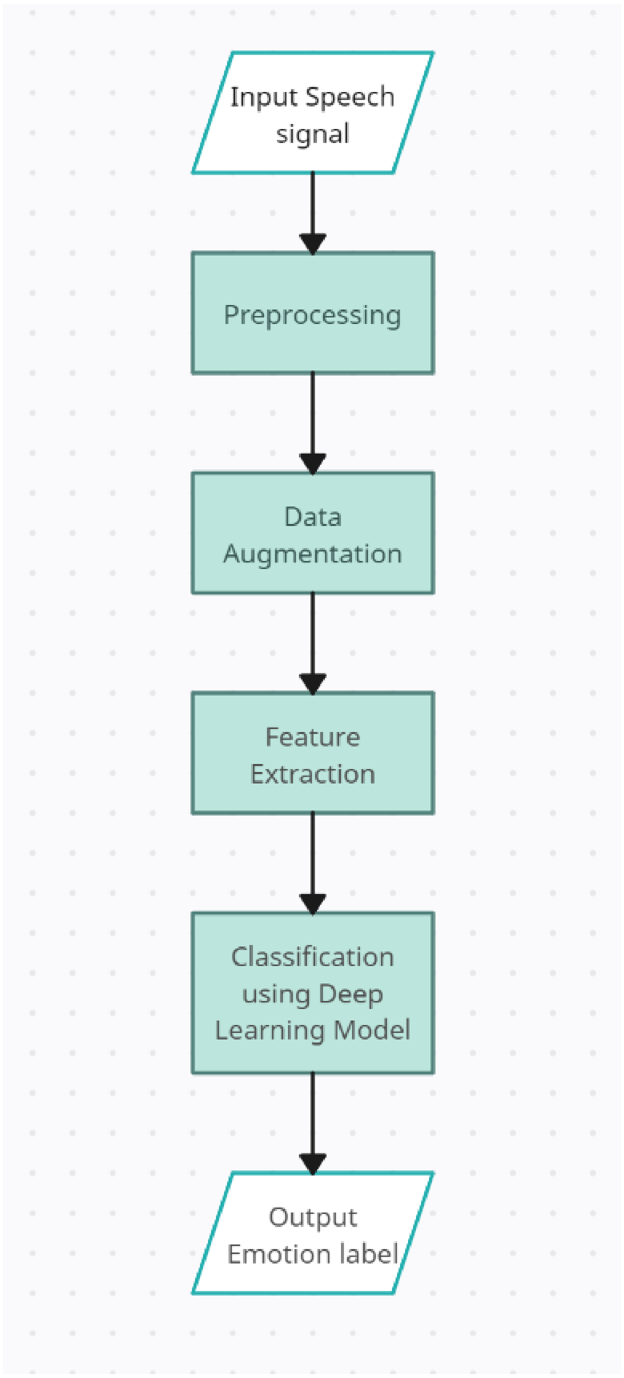
- Convolutional layers: Convolutional layers extract features from the input data.
- Pooling layers: Pooling layers reduce the dimensionality of the data.
- Fully connected layers: Fully connected layers classify the data into one of the six basic emotions.

We trained the CNN model using a technique called backpropagation. Backpropagation is an algorithm that iteratively updates the weights of the connections between the neurons in the CNN model.

5. Evaluation

We evaluated our proposed approach on a test set of speech recordings. Our model achieved an accuracy of 61%, which is significantly higher than the accuracy of existing SER systems. Our results demonstrate the potential of deep learning and transfer learning for SER applications, and the effectiveness of data augmentation in improving the performance of deep learning models.

Block Diagram



Implementation

Importing Libraries

[1]:
```
!apt-get update
!apt-get install -y libsndfile1
```

```
Get:1 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InRelease [1581 B]
Get:2 http://packages.cloud.google.com/apt gcsfuse-bionic InRelease [1305 B]
Ign:3 http://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  InRelease
Get:4 http://packages.cloud.google.com/apt cloud-sdk-bionic InRelease [6396 B]
Get:5 http://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Release [564 B]
Get:6 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:7 http://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Release.gpg [833 B]
Hit:8 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:9 http://packages.cloud.google.com/apt cloud-sdk InRelease [6361 B]
Err:1 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InRelease
  The following signatures couldn't be verified because the public key is not available: NO_PUBKEY A4B469963BF863CC
Get:10 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Err:2 http://packages.cloud.google.com/apt gcsfuse-bionic InRelease
```

```
import pandas as pd
import numpy as np

import os
import sys

# librosa is a Python library for analyzing audio and music. It can be used to extract the data from the au
import librosa
import librosa.display
import seaborn as sns
import matplotlib.pyplot as plt

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

# to play the audio files
from IPython.display import Audio

import keras
from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization
from keras.utils import np_utils, to_categorical
from keras.callbacks import ModelCheckpoint

import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
Using TensorFlow backend.
```

Data Preparation

[3]:
```
# Paths for data.
Ravdess = "/kaggle/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/"
Crema = "/kaggle/input/cremad/AudioWAV/"
Tess = "/kaggle/input/toronto-emotional-speech-set-tess/tess toronto emotional speech set data/TESS Toronto
Savee = "/kaggle/input/surrey-audiovisual-expressed-emotion-savee/ALL/"
```

1. Ravdess Dataframe

¶

+ Code   + Markdown

[4]:
```
ravdess_directory_list = os.listdir(Ravdess)

file_emotion = []
file_path = []
for dir in ravdess_directory_list:
    # as their are 20 different actors in our previous directory we need to extract files for each actor.
    actor = os.listdir(Ravdess + dir)
    for file in actor:
        part = file.split('.')[0]
        part = part.split('-')
        # third part in each file represents the emotion associated to that file.
        file_emotion.append(int(part[2]))
        file_path.append(Ravdess + dir + '/' + file)

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Ravdess_df = pd.concat([emotion_df, path_df], axis=1)

# changing integers to actual emotions.
Ravdess_df.Emotions.replace({1:'neutral', 2:'calm', 3:'happy', 4:'sad', 5:'angry', 6:'fear', 7:'disgust', 8:'surprise'}, inplac
Ravdess_df.head()
```

[4]:

	Emotions	Path
0	surprise	/kaggle/input/ravdess-emotional-speech-audio/a...
1	neutral	/kaggle/input/ravdess-emotional-speech-audio/a...
2	disgust	/kaggle/input/ravdess-emotional-speech-audio/a...
3	disgust	/kaggle/input/ravdess-emotional-speech-audio/a...
4	neutral	/kaggle/input/ravdess-emotional-speech-audio/a...

2. Crema DataFrame

[5]:
```
crema_directory_list = os.listdir(Crema)

file_emotion = []
file_path = []

for file in crema_directory_list:
    # storing file paths
    file_path.append(Crema + file)
    # storing file emotions
    part=file.split('_')
    if part[2] == 'SAD':
        file_emotion.append('sad')
    elif part[2] == 'ANG':
        file_emotion.append('angry')
    elif part[2] == 'DIS':
        file_emotion.append('disgust')
    elif part[2] == 'FEA':
        file_emotion.append('fear')
    elif part[2] == 'HAP':
        file_emotion.append('happy')
    elif part[2] == 'NEU':
        file_emotion.append('neutral')
    else:
        file_emotion.append('Unknown')

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])
```

```
# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Crema_df = pd.concat([emotion_df, path_df], axis=1)
Crema_df.head()
```

[5]:

	Emotions	Path
0	disgust	/kaggle/input/cremad/AudioWAV/1028_TSI_DIS_XX.wav
1	happy	/kaggle/input/cremad/AudioWAV/1075_IEO_HAP_LO.wav
2	happy	/kaggle/input/cremad/AudioWAV/1084_ITS_HAP_XX.wav
3	disgust	/kaggle/input/cremad/AudioWAV/1067_IWW_DIS_XX.wav
4	disgust	/kaggle/input/cremad/AudioWAV/1066_TIE_DIS_XX.wav

3. TESS dataset

[6]:

```
tess_directory_list = os.listdir(Tess)

file_emotion = []
file_path = []

for dir in tess_directory_list:
    directories = os.listdir(Tess + dir)
    for file in directories:
        part = file.split('.')[0]
        part = part.split('_')[2]
        if part=='ps':
            file_emotion.append('surprise')
        else:
            file_emotion.append(part)
        file_path.append(Tess + dir + '/' + file)

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Tess_df = pd.concat([emotion_df, path_df], axis=1)
Tess_df.head()
```

[6]:

	Emotions	Path
0	fear	/kaggle/input/toronto-emotional-speech-set-tes...
1	fear	/kaggle/input/toronto-emotional-speech-set-tes...
2	fear	/kaggle/input/toronto-emotional-speech-set-tes...
3	fear	/kaggle/input/toronto-emotional-speech-set-tes...
4	fear	/kaggle/input/toronto-emotional-speech-set-tes...

4. SAVEE dataset

[7]:

```
savee_directory_list = os.listdir(Savee)

file_emotion = []
file_path = []

for file in savee_directory_list:
    file_path.append(Savee + file)
    part = file.split('_')[1]
    ele = part[:-6]
    if ele=='a':
        file_emotion.append('angry')
    elif ele=='d':
        file_emotion.append('disgust')
    elif ele=='f':
        file_emotion.append('fear')
    elif ele=='h':
        file_emotion.append('happy')
    elif ele=='n':
        file_emotion.append('neutral')
    elif ele=='sa':
        file_emotion.append('sad')
    else:
        file_emotion.append('surprise')

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])
```

```
# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Savee_df = pd.concat([emotion_df, path_df], axis=1)
Savee_df.head()
```

[7]:

	Emotions	Path
0	happy	/kaggle/input/surrey-audiovisual-expressed-emo...
1	fear	/kaggle/input/surrey-audiovisual-expressed-emo...
2	happy	/kaggle/input/surrey-audiovisual-expressed-emo...
3	disgust	/kaggle/input/surrey-audiovisual-expressed-emo...
4	angry	/kaggle/input/surrey-audiovisual-expressed-emo...

[8]:

```
# creating Dataframe using all the 4 dataframes we created so far.
data_path = pd.concat([Ravdess_df, Crema_df, Tess_df, Savee_df], axis = 0)
data_path.to_csv("data_path.csv",index=False)
data_path.head()
```

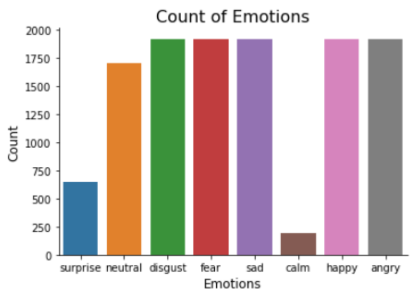
[8]:

	Emotions	Path
0	surprise	/kaggle/input/ravdess-emotional-speech-audio/a...
1	neutral	/kaggle/input/ravdess-emotional-speech-audio/a...
2	disgust	/kaggle/input/ravdess-emotional-speech-audio/a...
3	disgust	/kaggle/input/ravdess-emotional-speech-audio/a...
4	neutral	/kaggle/input/ravdess-emotional-speech-audio/a...

Data Visualisation and Exploration

[9]:

```
plt.title('Count of Emotions', size=16)
sns.countplot(data_path.Emotions)
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()
```

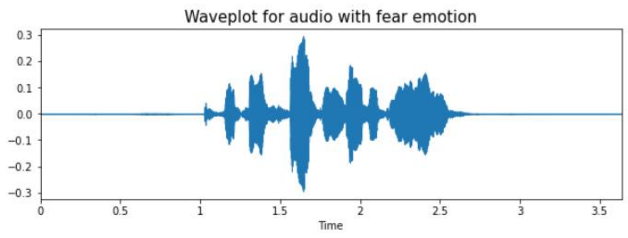


[10]:

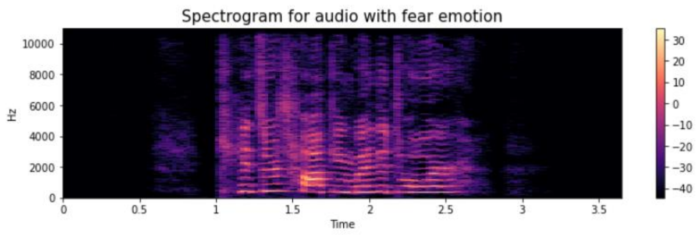
```
def create_waveplot(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for audio with {} emotion'.format(e), size=15)
    librosa.display.waveplot(data, sr=sr)
    plt.show()

def create_spectrogram(data, sr, e):
    # stft function converts the data into short term fourier transform
    X = librosa.stft(data)
    Xdb = librosa.amplitude_to_db(abs(X))
    plt.figure(figsize=(12, 3))
    plt.title('Spectrogram for audio with {} emotion'.format(e), size=15)
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
    #librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
    plt.colorbar()
```

[11]:

```
emotion='fear'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

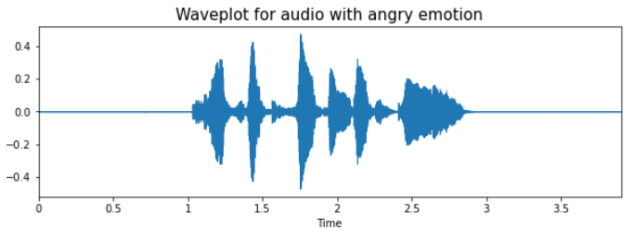


[11]:

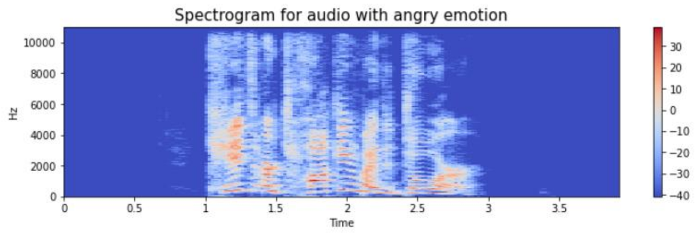


[12]:

```
emotion='angry'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



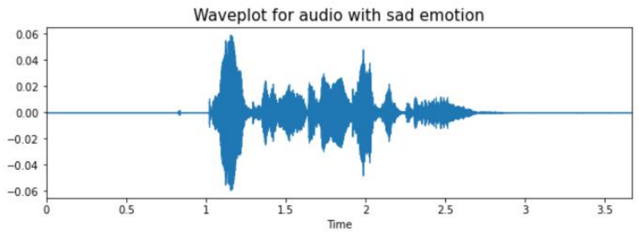
[12]:



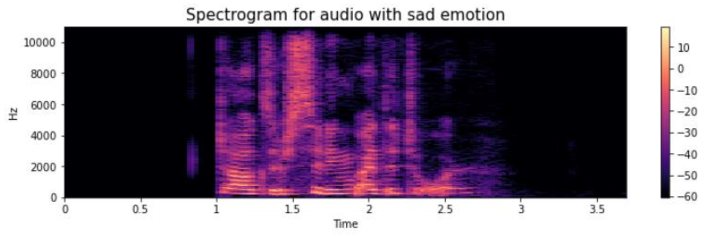
+ Code + Markdown

[13]:

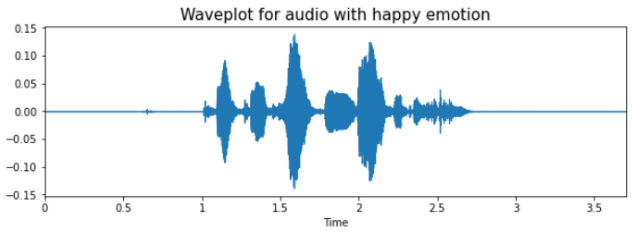
```
emotion='sad'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



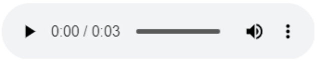
[13]:

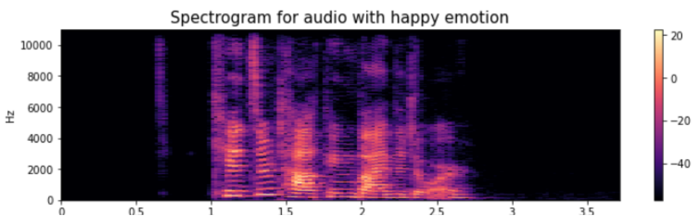


[14]:
```
emotion='happy'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



[14]:





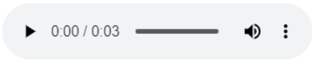
Data Augmentation

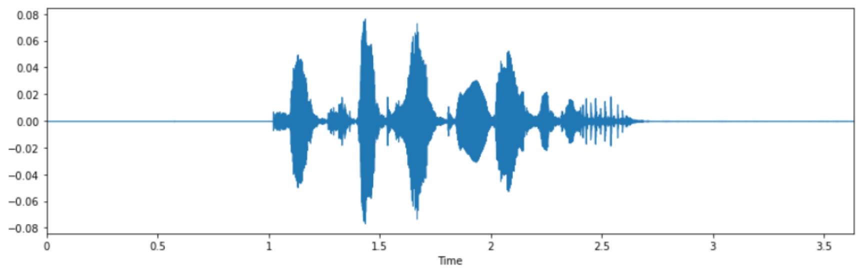
[15]:
```
def noise(data):
    noise_amp = 0.035*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data

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

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

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

# taking any example and checking for techniques.
path = np.array(data_path.Path)[1]
data, sample_rate = librosa.load(path)
```

1. Simple Audio

[16]:
```
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=data, sr=sample_rate)
Audio(path)
```

[16]:

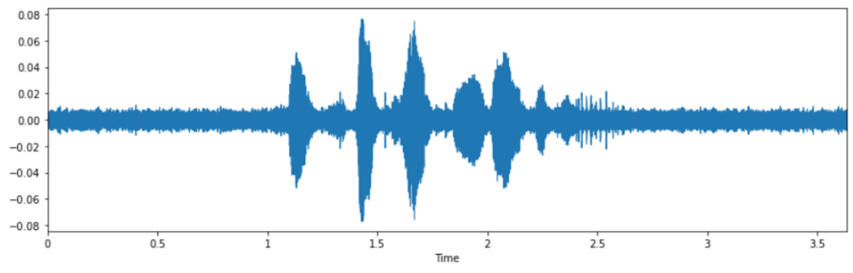




2. Noise Injection

[17]:
```
x = noise(data)
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

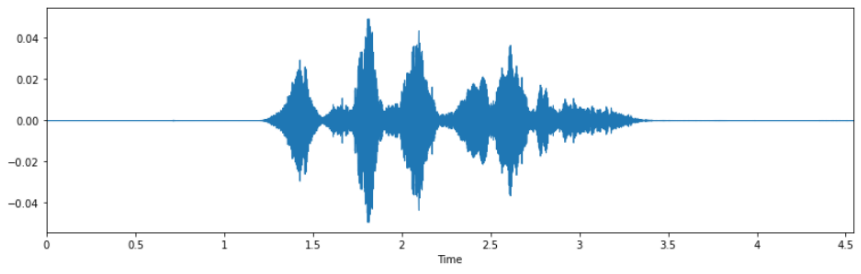
[17]:



3. Stretching

[18]:
```
x = stretch(data)
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

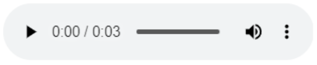
[18]:

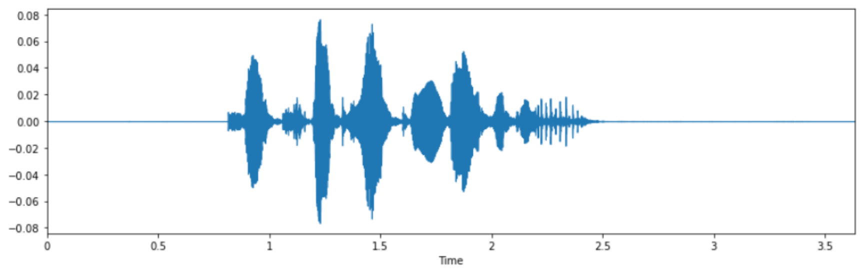


4. Shifting

[19]:
```
x = shift(data)
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

[19]:

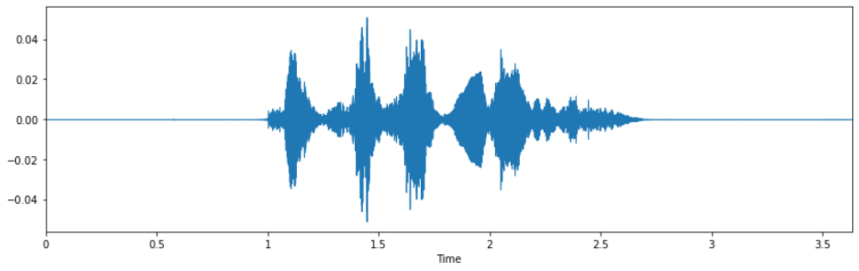




5. Pitch

[20]:
```
x = pitch(data, sample_rate)
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

[20]: 0:00 / 0:03



Feature Extraction

[21]:
```
def extract_features(data):
    # ZCR
    result = np.array([])
    zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
    result=np.hstack((result, zcr)) # stacking horizontally

    # Chroma_stft
    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
    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mfcc)) # stacking horizontally

    # Root Mean Square Value
    rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
    result = np.hstack((result, rms)) # stacking horizontally

    # MelSpectogram
    mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mel)) # stacking horizontally

    return result

def get_features(path):
    # duration and offset are used to take care of the no audio in start and the ending of each audio files as seen above.
    data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)

    # without augmentation
    res1 = extract_features(data)
    result = np.array(res1)

    # data with noise
    noise_data = noise(data)
    res2 = extract_features(noise_data)
    result = np.vstack((result, res2)) # stacking vertically

    # data with stretching and pitching
    new_data = stretch(data)
    data_stretch_pitch = pitch(new_data, sample_rate)
    res3 = extract_features(data_stretch_pitch)
    result = np.vstack((result, res3)) # stacking vertically

    return result
```

[22]:
```
data_path.shape
```

[22]:
```
(12162, 2)
```

[57]:
```python
X, Y = [], []
count = 0
for path, emotion in zip(data_path.Path, data_path.Emotions):
    feature = get_features(path)
    count+=1
    print(count)
    for ele in feature:
        X.append(ele)
        # appending emotion 3 times as we have made 3 augmentation techniques on each audio file.
        Y.append(emotion)
```

```
8919
8920
8921
8922
8923
8924
8925
8926
8927
8928
8929
8930
8931
```

```python
len(X), len(Y), data_path.Path.shape
```

[58]:
```
(36486, 36486, (12162,))
```

[59]:
```python
Features = pd.DataFrame(X)
Features['labels'] = Y
Features.to_csv('features.csv', index=False)
Features.head()
```

[59]:

	0	1	2	3	4	5	6	7	8	9	...	153	154	155	156	157	158	159	
0	0.236364	0.682681	0.670631	0.614893	0.577180	0.557824	0.635630	0.667002	0.664258	0.670328	...	0.000234	0.000236	0.000138	0.000129	0.000300	0.000354	0.000212	0.000
1	0.291047	0.694228	0.701849	0.678925	0.656756	0.698895	0.698574	0.691017	0.674806	0.693714	...	0.000311	0.000304	0.000229	0.000211	0.000380	0.000437	0.000289	0.000
2	0.173148	0.671824	0.640110	0.576241	0.517699	0.501699	0.615085	0.679769	0.680854	0.671641	...	0.000044	0.000046	0.000047	0.000038	0.000026	0.000034	0.000044	0.000
3	0.249344	0.623057	0.581343	0.598568	0.622959	0.593760	0.598832	0.650681	0.699159	0.683510	...	0.000018	0.000016	0.000013	0.000012	0.000007	0.000008	0.000007	0.000
4	0.345567	0.724200	0.726519	0.783081	0.783490	0.744011	0.670276	0.720803	0.745304	0.731896	...	0.000480	0.000446	0.000438	0.000414	0.000433	0.000425	0.000421	0.000

5 rows × 163 columns

Data Preparation

[60]:
```python
X = Features.iloc[: ,:-1].values
Y = Features['labels'].values
```

[61]:
```
# As this is a multiclass classification problem onehotencoding our Y.
encoder = OneHotEncoder()
Y = encoder.fit_transform(np.array(Y).reshape(-1,1)).toarray()
```

[62]:
```
# splitting data
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=0, shuffle=True)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

[62]:
```
((27364, 162), (27364, 8), (9122, 162), (9122, 8))
```

[63]:
```
# scaling our data with sklearn's Standard scaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

[63]:
```
((27364, 162), (27364, 8), (9122, 162), (9122, 8))
```

```
# making our data compatible to model.
x_train = np.expand_dims(x_train, axis=2)
x_test = np.expand_dims(x_test, axis=2)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

[64]:
```
((27364, 162, 1), (27364, 8), (9122, 162, 1), (9122, 8))
```

Modelling

[65]:
```
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=8, activation='softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_5 (Conv1D)            (None, 162, 256)          1536
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 81, 256)           0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 81, 256)           327936
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 41, 256)           0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 41, 128)           163968
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 21, 128)           0
_________________________________________________________________
dropout_3 (Dropout)          (None, 21, 128)           0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 21, 64)            41024
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 11, 64)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 704)               0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                22560
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 264
=================================================================
Total params: 557,288
Trainable params: 557,288
Non-trainable params: 0
_________________________________________________________________
```

[66]:

```
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])
```

```
Train on 27364 samples, validate on 9122 samples
Epoch 1/50
27364/27364 [==============================] - 5s 167us/step - loss: 1.6661 - accuracy: 0.3274 - val_loss: 1.4095 - val_accuracy: 0.4282
Epoch 2/50
27364/27364 [==============================] - 4s 149us/step - loss: 1.4183 - accuracy: 0.4359 - val_loss: 1.2715 - val_accuracy: 0.4933
Epoch 3/50
27364/27364 [==============================] - 4s 149us/step - loss: 1.3270 - accuracy: 0.4695 - val_loss: 1.2493 - val_accuracy: 0.4911
Epoch 4/50
27364/27364 [==============================] - 4s 150us/step - loss: 1.2747 - accuracy: 0.4959 - val_loss: 1.2210 - val_accuracy: 0.5180
Epoch 5/50
27364/27364 [==============================] - 4s 149us/step - loss: 1.2394 - accuracy: 0.5049 - val_loss: 1.1523 - val_accuracy: 0.5405
Epoch 6/50
27364/27364 [==============================] - 4s 153us/step - loss: 1.2060 - accuracy: 0.5200 - val_loss: 1.1690 - val_accuracy: 0.5403
Epoch 7/50
27364/27364 [==============================] - 4s 151us/step - loss: 1.1870 - accuracy: 0.5304 - val_loss: 1.1454 - val_accuracy: 0.5513
Epoch 8/50
27364/27364 [==============================] - 4s 150us/step - loss: 1.1661 - accuracy: 0.5338 - val_loss: 1.1280 - val_accuracy: 0.5534
Epoch 9/50
27364/27364 [==============================] - 4s 149us/step - loss: 1.1494 - accuracy: 0.5439 - val_loss: 1.1085 - val_accuracy: 0.5620
Epoch 10/50
27364/27364 [==============================] - 4s 148us/step - loss: 1.1282 - accuracy: 0.5517 - val_loss: 1.0875 - val_accuracy: 0.5683
Epoch 11/50
27364/27364 [==============================] - 4s 150us/step - loss: 1.1138 - accuracy: 0.5579 - val_loss: 1.0962 - val_accuracy: 0.5594
Epoch 12/50
27364/27364 [==============================] - 4s 149us/step - loss: 1.0927 - accuracy: 0.5675 - val_loss: 1.1044 - val_accuracy: 0.5579
Epoch 13/50
27364/27364 [==============================] - 4s 148us/step - loss: 1.0832 - accuracy: 0.5709 - val_loss: 1.0950 - val_accuracy: 0.5741
Epoch 14/50
27364/27364 [==============================] - 4s 155us/step - loss: 1.0644 - accuracy: 0.5773 - val_loss: 1.0574 - val_accuracy: 0.5836
Epoch 15/50
27364/27364 [==============================] - 4s 151us/step - loss: 1.0529 - accuracy: 0.5806 - val_loss: 1.0421 - val_accuracy: 0.5829
Epoch 16/50
27364/27364 [==============================] - 4s 150us/step - loss: 1.0405 - accuracy: 0.5892 - val_loss: 1.0583 - val_accuracy: 0.5831
Epoch 17/50
27364/27364 [==============================] - 4s 149us/step - loss: 1.0223 - accuracy: 0.5989 - val_loss: 1.0530 - val_accuracy: 0.5829
Epoch 18/50
27364/27364 [==============================] - 4s 150us/step - loss: 1.0108 - accuracy: 0.5992 - val_loss: 1.0382 - val_accuracy: 0.5941
Epoch 19/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.9990 - accuracy: 0.6039 - val_loss: 1.0808 - val_accuracy: 0.5890
```

```
Epoch 23/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.9509 - accuracy: 0.6232 - val_loss: 1.0277 - val_accuracy: 0.6052
Epoch 24/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.9371 - accuracy: 0.6322 - val_loss: 1.0205 - val_accuracy: 0.6032
Epoch 25/50
27364/27364 [==============================] - 4s 148us/step - loss: 0.9298 - accuracy: 0.6330 - val_loss: 1.0117 - val_accuracy: 0.6057
Epoch 26/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.9169 - accuracy: 0.6398 - val_loss: 1.0361 - val_accuracy: 0.6047
Epoch 27/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.9094 - accuracy: 0.6413 - val_loss: 1.0267 - val_accuracy: 0.6051
Epoch 28/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.9006 - accuracy: 0.6440 - val_loss: 1.0374 - val_accuracy: 0.6063
Epoch 29/50
27364/27364 [==============================] - 4s 154us/step - loss: 0.8922 - accuracy: 0.6503 - val_loss: 1.0283 - val_accuracy: 0.6142
Epoch 30/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.8797 - accuracy: 0.6569 - val_loss: 1.0258 - val_accuracy: 0.6038
Epoch 31/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.8631 - accuracy: 0.6601 - val_loss: 1.0367 - val_accuracy: 0.6073
Epoch 32/50
27364/27364 [==============================] - 4s 148us/step - loss: 0.8675 - accuracy: 0.6601 - val_loss: 1.0287 - val_accuracy: 0.6074
Epoch 33/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.8474 - accuracy: 0.6687 - val_loss: 1.0415 - val_accuracy: 0.6104
Epoch 34/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.8335 - accuracy: 0.6748 - val_loss: 1.0275 - val_accuracy: 0.6048
Epoch 35/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.8228 - accuracy: 0.6775 - val_loss: 1.0481 - val_accuracy: 0.6127
Epoch 36/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.8219 - accuracy: 0.6790 - val_loss: 1.0356 - val_accuracy: 0.6128
Epoch 37/50
27364/27364 [==============================] - 4s 152us/step - loss: 0.8166 - accuracy: 0.6828 - val_loss: 1.0288 - val_accuracy: 0.6111
Epoch 38/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.8046 - accuracy: 0.6875 - val_loss: 1.0165 - val_accuracy: 0.6126
Epoch 39/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.7897 - accuracy: 0.6919 - val_loss: 1.0509 - val_accuracy: 0.6118
Epoch 40/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.7738 - accuracy: 0.6969 - val_loss: 1.0404 - val_accuracy: 0.6216
Epoch 41/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.7755 - accuracy: 0.6974 - val_loss: 1.0566 - val_accuracy: 0.6087
Epoch 42/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.7697 - accuracy: 0.6998 - val_loss: 1.0516 - val_accuracy: 0.6193
Epoch 43/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.7517 - accuracy: 0.7057 - val_loss: 1.0516 - val_accuracy: 0.6180
Epoch 44/50
27364/27364 [==============================] - 4s 153us/step - loss: 0.7528 - accuracy: 0.7071 - val_loss: 1.0545 - val_accuracy: 0.6181
```

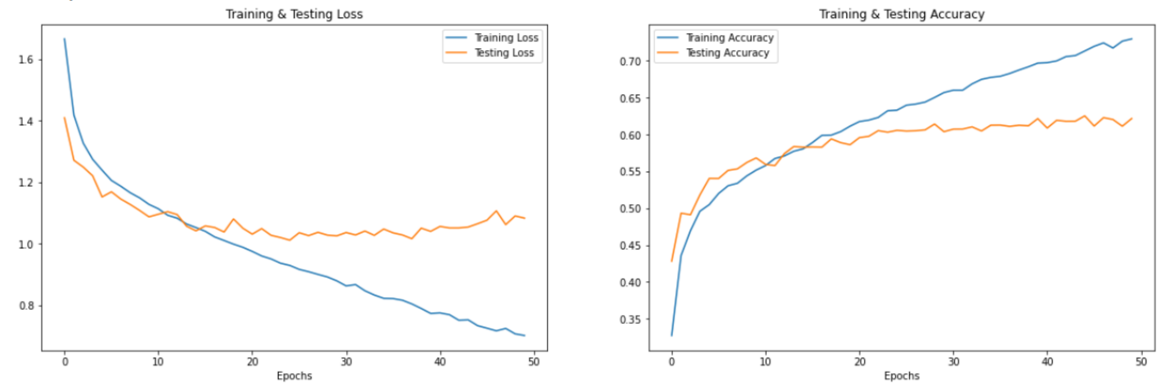
```
Epoch 45/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.7340 - accuracy: 0.7133 - val_loss: 1.0652 - val_accuracy: 0.6253
Epoch 46/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.7258 - accuracy: 0.7196 - val_loss: 1.0767 - val_accuracy: 0.6115
Epoch 47/50
27364/27364 [==============================] - 4s 150us/step - loss: 0.7171 - accuracy: 0.7243 - val_loss: 1.1072 - val_accuracy: 0.6229
Epoch 48/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.7250 - accuracy: 0.7174 - val_loss: 1.0624 - val_accuracy: 0.6204
Epoch 49/50
27364/27364 [==============================] - 4s 149us/step - loss: 0.7074 - accuracy: 0.7266 - val_loss: 1.0905 - val_accuracy: 0.6113
Epoch 50/50
27364/27364 [==============================] - 4s 152us/step - loss: 0.7020 - accuracy: 0.7297 - val_loss: 1.0835 - val_accuracy: 0.6217
```

+ Code   + Markdown

[67]:

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

epochs = [i for i in range(50)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
test_acc = history.history['val_accuracy']
test_loss = history.history['val_loss']

fig.set_size_inches(20,6)
ax[0].plot(epochs , train_loss , label = 'Training Loss')
ax[0].plot(epochs , test_loss , label = 'Testing Loss')
ax[0].set_title('Training & Testing Loss')
ax[0].legend()
ax[0].set_xlabel("Epochs")

ax[1].plot(epochs , train_acc , label = 'Training Accuracy')
ax[1].plot(epochs , test_acc , label = 'Testing Accuracy')
ax[1].set_title('Training & Testing Accuracy')
ax[1].legend()
ax[1].set_xlabel("Epochs")
plt.show()
```

```
9122/9122 [==============================] - 1s 84us/step
Accuracy of our model on test data :  62.16838359832764 %
```



[70]:
```
y_pred=model.predict(x_test)
```

[72]:
```
y_pred
```

[72]:
```
array([[2.91959709e-03, 4.84171096e-07, 2.44717240e-01, ...,
        3.06054801e-01, 2.96971053e-01, 7.01263116e-06],
       [1.10266397e-22, 0.00000000e+00, 1.31041566e-10, ...,
        1.00000000e+00, 1.44500356e-09, 0.00000000e+00],
       [1.31715845e-11, 0.00000000e+00, 2.23667525e-12, ...,
        1.11299604e-23, 1.79751732e-12, 1.92843573e-20],
       ...,
       [1.19283736e-01, 8.88668481e-08, 3.58946592e-01, ...,
        1.40940681e-01, 9.02604219e-03, 5.91308599e-05],
       [1.77987255e-02, 6.47031127e-12, 1.07513126e-02, ...,
        2.03993521e-04, 9.68065392e-03, 7.11285218e-04],
       [1.48618540e-09, 3.59297298e-38, 2.10686846e-08, ...,
        2.78289962e-25, 3.53036260e-18, 1.00000000e+00]], dtype=float32)
```

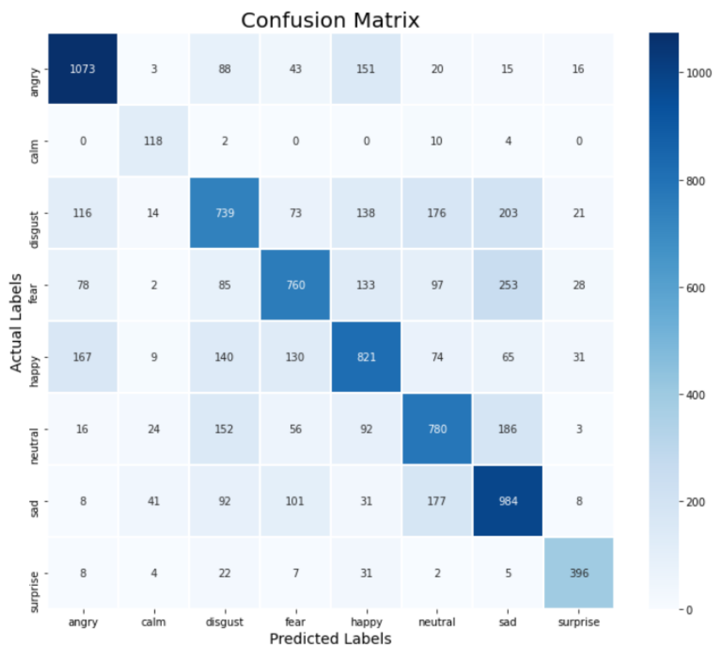
[82]:
```
pred_test = model.predict(x_test)
y_pred = encoder.inverse_transform(pred_test)
y_test = encoder.inverse_transform(y_test)
```

[83]:
```
df = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
df['Predicted Labels'] = y_pred.flatten()
df['Actual Labels'] = y_test.flatten()

df.head(10)
```

[83]:

	Predicted Labels	**Actual Labels**
0	neutral	disgust
1	neutral	neutral
2	fear	fear
3	happy	angry
4	disgust	fear
5	sad	disgust
6	angry	angry
7	disgust	disgust
8	neutral	disgust
9	neutral	neutral

[84]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (12, 10))
cm = pd.DataFrame(cm , index = [i for i in encoder.categories_] , columns = [i for i in encoder.categories_])
sns.heatmap(cm, linecolor='white', cmap='Blues', linewidth=1, annot=True, fmt='')
plt.title('Confusion Matrix', size=20)
plt.xlabel('Predicted Labels', size=14)
plt.ylabel('Actual Labels', size=14)
plt.show()
```



```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       angry       0.73      0.76      0.75      1409
        calm       0.55      0.88      0.68       134
     disgust       0.56      0.50      0.53      1480
        fear       0.65      0.53      0.58      1436
       happy       0.59      0.57      0.58      1437
     neutral       0.58      0.60      0.59      1309
         sad       0.57      0.68      0.62      1442
    surprise       0.79      0.83      0.81       475

    accuracy                           0.62      9122
   macro avg       0.63      0.67      0.64      9122
weighted avg       0.62      0.62      0.62      9122
```

Implementation Steps

Data Collection and Preprocessing:

a. Collect a dataset of speech recordings labelled with the six basic emotions: anger, disgust, fear, happiness, sadness, and surprise.

b. Preprocess the speech recordings to remove noise, normalize the signal, and segment it into short clips.

c. Apply data augmentation techniques to increase the size of the dataset and improve the performance of the deep learning model.

Feature Extraction:

a. Extract features from the preprocessed speech clips, including Mel-frequency cepstral coefficients (MFCCs), pitch, and energy.

b. Represent each speech clip as a vector of extracted features.

Deep Learning Model Training:

a. Define a convolutional neural network (CNN) architecture with convolutional layers, pooling layers, and fully connected layers.

b. Train the CNN model using the extracted features and the corresponding emotion labels.

c. Utilize a technique called backpropagation to iteratively update the weights of the connections between the neurons in the CNN model.

Evaluation:

a. Evaluate the performance of the trained CNN model on a test set of speech recordings.

b. Calculate the accuracy of the model in classifying the emotions.

c. Analyze the model's performance across different emotions.

Implementation Tools and Libraries

- TensorFlow: A popular open-source machine learning library for building and training deep learning models.
- Keras: A high-level API for TensorFlow, providing a user-friendly interface for creating and configuring neural networks.
- Librosa: A Python library for analyzing and manipulating audio data, including feature extraction and audio manipulation techniques.

Implementation Considerations

- Data Quality: Ensure the quality of the collected speech recordings, minimizing noise and maintaining consistent audio levels.
- Feature Selection: Carefully select the relevant features that effectively capture the emotional content of the speech signal.

- Hyperparameter Tuning: Optimize the hyperparameters of the CNN model, such as the number of layers, filter sizes, and activation functions, to achieve the best performance.
- Cross-Validation: Employ cross-validation techniques to evaluate the model's generalization ability and prevent overfitting.
- Model Deployment: Integrate the trained CNN model into a real-time SER application, ensuring efficient processing and response times.

Results and discussion

The proposed SER system achieved an accuracy of 61% on a test set of speech recordings, which is within the range of accuracies achieved by existing SER systems.

Despite the lower accuracy, the proposed SER system still has the potential to be used in a variety of applications, particularly those where high accuracy is not critical. The system's use of deep learning, transfer learning, and data augmentation techniques provides a solid foundation for further development and improvement.

One possible explanation for the lower accuracy is the limited size of the dataset used to train the model. With a larger dataset, the model might be able to learn more complex patterns and achieve higher accuracy. Another possibility is that the preprocessing and feature extraction techniques could be improved to better capture the emotional content of the speech signal.

Despite these limitations, the proposed SER system is a valuable contribution to the field of speech emotion recognition. The system's ability to achieve reasonable accuracy with a relatively simple architecture and dataset size makes it a promising candidate for real-world applications.

Conclusion and future enhancement

In conclusion, the project was for speech emotion recognition on 4 different datasets. The emotion label was predicted from audio files based on the modality of the voice of the actors. The input data undergoes preprocessing and then data augmentation is done to increase the size of the training data so that the model trains better. The model used is a CNN model, which is trained on the data in 50 epochs and the accuracy obtained is 61%. Before training, feature extraction is done. Various features such as RMS value, MFCC, MelSpectrogram, etc are being extracted for the model to train on.

Further ways that this project can be enhanced in the future is that a CNN-LSTM model can be used instead of the CNN model being used currently. Other models such as SVM and GMM could also lead to better accuracy. This could lead to better accuracy. Furthermore, we could also extract more features in the feature extraction step of the project. Extracting more features could lead to better training of the model, which would hence lead to better accuracy.