**Report**

This is the project report of task 1(emotion recognition from speech) that I completed under the summer internship for the month of July-august under Codealpha.

This report file contains the detailed explanation of the project. For video explanation of this project you may checkout the following link:-

<https://www.linkedin.com/posts/pranita-padaliya-46453b278_emotion-recognition-from-speech-source-code-activity-7222115184381763584-x8zO?utm_source=share&utm_medium=member_android>

**Prerequisite –**

* Basics of Phyton language
* Installation of python on the system
* RAVDESS Dataset (to work upon)
* Installation of following packages:-

1. Pandas
2. Numpy
3. Scikit
4. Librosa
5. tensorflow

Note- install the packages in the windows terminal by using the following syntax :-

py -m pip install package\_name

and in macos use the following syntax :-

python3 pip install package\_name

I have made the project using 13 basic steps:-

Step 1- imports and dependencies

import os

import numpy as np

import librosa

import librosa.feature

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Conv1D, MaxPooling1D, Flatten

from tensorflow.keras.utils import to\_categorical

the libraries used in the project and their uses are-

* Os – for file hndling
* Numpy – for numerical operation
* Librosa – for audio processing
* Tenserflow – for building and training neural networks
* Sklearn – data processing and splitting

Step 2- function to load and preprocess the RAVDESS dataset

def load\_ravdess\_data(data\_path):

    labels = []

    features = []

    emotions = {

        '01': 'neutral',

        '02': 'calm',

        '03': 'happy',

        '04': 'sad',

        '05': 'angry',

        '06': 'fearful',

        '07': 'disgust',

        '08': 'surprised'

This function initializes empty lists labels and features and a dictionary emotions to map emotion codes to their respective labels.

Step 3 – iteration over audio files

for root, \_, files in os.walk(data\_path):

        for file in files:

            if file.endswith(".wav"):

                file\_path = os.path.join(root, file)

                emotion = None

                try:

                    emotion\_code = file.split("-")[2]

                    emotion = emotions[emotion\_code]

                except KeyError:

                    print(f"skipping file {file\_path}: unrecognized emotion code {emotion\_code}")

                    continue

                except IndexError:

                     print(f"skipping file {file\_path}: file name format is incorrect")

                     continue

this part walks though the directories in the data\_path, looking for ‘.wav’ files. It extracts the emotion code from the filename and maps it to the corresponding emotion using emotions dictionary.

Step 4 – processing audio files

 if emotion:

                    try:

                         y, sr = librosa.load(file\_path, duration=2.5, offset=0.5)

                         if y is None or sr is None :

                              print(f"skipping file {file\_path}: failed to load audio")

                              continue

                         mfccs = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=13)

                         mfccs\_scaled = np.mean(mfccs.T, axis=0)

                         features.append(mfccs\_scaled)

                         labels.append(emotion)

                    except Exception as e:

                        print(f"skipping file {file\_path}: {e}")

    return np.array(features), np.array(labels)

* Loads the audio file with librosa.load, trimming it to 2.5seconds starting at 0.5seconds
* Extracts MFCC features from the audio
* Averages the MFCCs over time and appends them to the features list
* Appends the corresponding emotion to the labels list
* Handles any errors that occur during this process and skips the problematic file.

Step 5 – path and data loading

data\_path = "Audio\_Speech\_Actors\_01-24"

X, y = load\_ravdess\_data(data\_path)

if X.size == 0 or y.size == 0:

    raise ValueError("no data loaded. please check the data pathor file format.")

defines the path of the dataset and loads the data using load\_ravdess\_data function. It raises an error if no data is loaded.

Step 6- encoding labels

encoder = LabelEncoder()

y\_encoder = encoder.fit\_transform(y)

y\_categorical = to\_categorical(y\_encoder)

encodes the emotion labels as integers and then converts them to one-hot encoded vectors using labelencoder and to\_categorical.

Step 7 – splitting data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_categorical, test\_size=0.2,random\_state=42)

Splits the data into training and testing sets with 80% for training and 20% for testing

Step 8 – building the model

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(Dropout(0.3))

model.add(Conv1D(128,kernel\_size=3, activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(len(np.unique(y)),activation='softmax'))

builds a convolutional neural network(CNN) model using sequential API:-

* First conv1D layer with 64 filters, kernel size =3, and relu activation
* Maxpooling1D layer to reduce the dimentionality
* Dropout layer to prevent overfiting
* Another conv1D layer with 128 filters
* Another Maxpooling1D layer and Dropout layer
* A flatten layer to convert the 2D output to 1D
* A dense layer with 128 neurons and relu activation
* Another Dropout layer
* Final dense layer with number of outputs neurons equal to the number of unique emotions.

Step 9 – compiling the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()

compiles the model using categorical cross entropy as loss function, the adam optimizer and the accuracy as the evaluation metric. It also prints the model summary.

Step 10 – reshaping the data

X\_train\_reshaped = np.expand\_dims(X\_train, axis=2)

X\_test\_reshaped = np.expand\_dims(X\_test, axis=2)

Reshapes the data to add an extra dimension, which is required for conv1d layer.

Step 11- training the model

history = model.fit(X\_train\_reshaped, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test\_reshaped, y\_test))

trains the model for 50 epochs with a batch size of 32

Step 12- evaluating the model

loss, accuracy = model.evaluate(X\_test\_reshaped, y\_test)

print(f"Test Accuracy: {accuracy\* 100:.2f}%")

evaluates the model on the testing set and prints the test accuracy as percentage.

Step 13- saving the model

model.save('emotion\_recognition\_model.keras')

saves the train model to a file.

This is all that I used in the code and their respective functions.

Hope it’s helpful.

Thank you