1-emotion recognition

- data preprocessing

```
import os
import librosa
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Function to extract MFCC features from audio
def extract_features(file_path, max_pad_len=174):
    try:
        audio, sample_rate = librosa.load(file_path, res_type='kaiser_fast')
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
        pad_width = max_pad_len - mfccs.shape[1]
        mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant')
        return mfccs
    except Exception as e:
        print(f"Error encountered while parsing file: {file_path}")
        return None

# Define the emotions to classify
emotion_labels = {
    '01': 'neutral',
    '02': 'calm',
    '03': 'happy',
    '04': 'sad',
    '05': 'angry',
    '06': 'fearful',
    '07': 'disgust',
    '08': 'surprised'
}

# Directory where the dataset is stored
dataset_path = 'path_to_ravdess_data'

# Prepare dataset
def load_data():
    features = []
    labels = []

    for root, _, files in os.walk(dataset_path):
        for file in files:
            if file.endswith('.wav'):
                file_path = os.path.join(root, file)
                emotion = emotion_labels[file.split('-')[2]]
                feature = extract_features(file_path)
                if feature is not None:
```

```
                features.append(feature)
                labels.append(emotion)

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

# Load data
X, y = load_data()

# Encode labels
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
y = to_categorical(y)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- model building

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Reshape data for CNN input
X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]

# Build the CNN model
model = Sequential()

# Convolutional Layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(40, 174, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Second Convolutional Layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Flatten Layer
model.add(Flatten())

# Dense Layer
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
```

```
# Output Layer
model.add(Dense(len(emotion_labels), activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Summary of the model
model.summary()
```

- model training

```
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Save the model
model.save('emotion_recognition_model.h5')
```

- evaluation

```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'Test accuracy: {test_acc}')
```

- make prediction

```
# Load model for prediction
model = tf.keras.models.load_model('emotion_recognition_model.h5')

# Predict on new audio
new_audio_path = 'path_to_new_audio.wav'
new_feature = extract_features(new_audio_path)
new_feature = new_feature.reshape(1, 40, 174, 1)

# Predict emotion
predicted_emotion = model.predict(new_feature)
emotion = le.inverse_transform([np.argmax(predicted_emotion)])
print(f'The predicted emotion is: {emotion[0]}')
```