TORONTO EMOTIONAL SPEECH SET (TESS) PROJECT REPORT

RAJESH RAGI

JULY - 2024

Contents

---

1. Introduction

1.1 **Overview of the Toronto Emotional Speech Set (TESS)**

The Toronto Emotional Speech Set (TESS) is a dataset designed for emotion recognition from speech. It comprises audio recordings where actors express a range of emotions. It is commonly used in research on speech emotion recognition.

1.2 **Project Objective**

This project aims to analyze the TESS dataset, extract relevant features, and build a machine learning model to accurately classify emotions based on speech samples.

2. Environment Setup

2.1 **Platform Used**

The project is implemented using Google Colab, providing the necessary computational resources and GPU acceleration to facilitate efficient model training.

2.2 **Computational Resources**

Google Colab provides a free GPU which is used to speed up the training process of the machine learning models.

3. Libraries Used

3.1 **Overview of Key Libraries**

- **Pandas**: For data manipulation and analysis.
- **NumPy**: For numerical computations.
- **Seaborn and Matplotlib**: For data visualization.
- **Librosa**: For audio analysis and feature extraction.
- **Keras**: For building and training neural networks.
- **Scikit-learn**: For data preprocessing and evaluation metrics.

4. Loading the Dataset

4.1 **Source and Structure**

The dataset is obtained from Kaggle, containing audio recordings of actors expressing different emotions. Each file name includes information about the emotion being expressed.

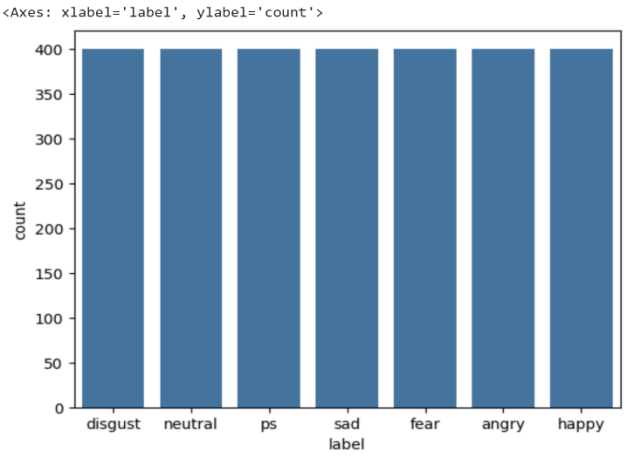
```
['/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/YAF_disgust/YAF_gaze_disgust.wav',
 '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/YAF_disgust/YAF_thin_disgust.wav',
 '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/YAF_disgust/YAF_food_disgust.wav',
 '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/YAF_disgust/YAF_vote_disgust.wav',
 '/content/tess toronto emotional speech set data/TESS Toronto emotional speech set data/YAF_disgust/YAF_mouse_disgust.wav']
```

4.2 **Label Extraction**

The emotion labels are extracted from the file names and used for labeling the dataset.

	speech	label
0	/content/tess toronto emotional speech set dat...	disgust
1	/content/tess toronto emotional speech set dat...	disgust
2	/content/tess toronto emotional speech set dat...	disgust
3	/content/tess toronto emotional speech set dat...	disgust
4	/content/tess toronto emotional speech set dat...	disgust
...	...	...
2795	/content/tess toronto emotional speech set dat...	neutral
2796	/content/tess toronto emotional speech set dat...	neutral
2797	/content/tess toronto emotional speech set dat...	neutral
2798	/content/tess toronto emotional speech set dat...	neutral
2799	/content/tess toronto emotional speech set dat...	neutral

2800 rows × 2 columns



5. Exploratory Data Analysis (EDA)

5.1 **Creating a DataFrame**

A DataFrame is constructed to store file paths and corresponding emotion labels.

	speech	label	label_count
0	/content/tess toronto emotional speech set dat...	disgust	NaN
1	/content/tess toronto emotional speech set dat...	disgust	NaN
2	/content/tess toronto emotional speech set dat...	disgust	NaN
3	/content/tess toronto emotional speech set dat...	disgust	NaN
4	/content/tess toronto emotional speech set dat...	disgust	NaN
...	...	...	...
2795	/content/tess toronto emotional speech set dat...	neutral	NaN
2796	/content/tess toronto emotional speech set dat...	neutral	NaN
2797	/content/tess toronto emotional speech set dat...	neutral	NaN
2798	/content/tess toronto emotional speech set dat...	neutral	NaN
2799	/content/tess toronto emotional speech set dat...	neutral	NaN

2800 rows × 3 columns

5.2 **Distribution Visualization**

Visualizations are generated to identify any imbalances in the dataset.

5.3 **Observations**

- The dataset includes seven distinct emotions: angry, disgust, fear, happy, neutral, pleasant surprise (ps), and sad.
- The distribution of samples across emotions is relatively balanced, benefiting model training without bias.

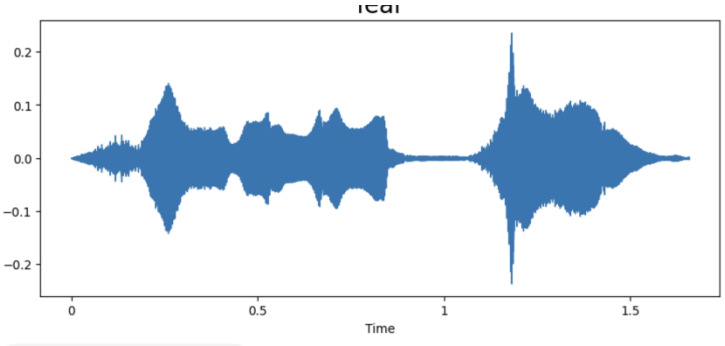
6. Audio Visualization

6.1 **Waveforms**

Waveforms display the amplitude variations of audio signals over time.

Code :-

```
def waveplot(data, sr, emotion):
    plt.figure(figsize=(10,4))
    plt.title(emotion, size=20)
    librosa.display.waveshow(data, sr=sr)
    plt.show()
```

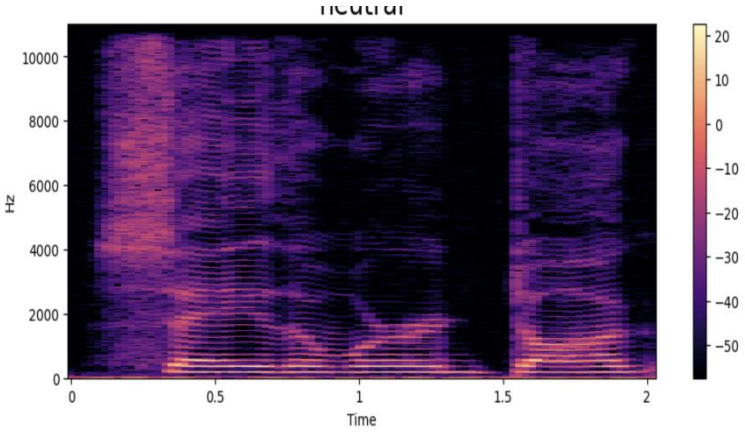


6.2 **Spectrograms**

Spectrograms show the frequency components of the audio signals.

Code :-

```
def spectogram(data, sr, emotion):
    x = librosa.stft(data)
    xdb = librosa.amplitude_to_db(abs(x))
    plt.figure(figsize=(11,4))
    plt.title(emotion, size=20)
    librosa.display.specshow(xdb, sr=sr,
x_axis='time', y_axis='hz')
    plt.colorbar()
```



6.3 **Observations**

- Different emotions exhibit distinct waveform and spectrogram patterns.
- Significant variations in frequency components across emotions can be leveraged for classification.

7. Feature Extraction

7.1 Mel Frequency Cepstral Coefficients (MFCCs)

MFCCs capture the timbral aspects of audio signals, effectively representing the nuances of emotional expressions in speech.

Code :-

```
def extract_mfcc(filename):
    y, sr = librosa.load(filename, duration=3, offset=0.5)
    mfcc = np.mean(librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40).T, axis=0)
    return mfcc
```

```
extract_mfcc(df['speech'][0])
```

```
array([-4.2960641e+02,  6.0449196e+01,  1.2524098e+01,  2.9926033e+01,
        3.5471654e+00,  2.4861872e-02, -1.0219503e+01, -1.0065464e+01,
       -1.4282920e+01, -4.3911285e+00, -6.9359655e+00, -6.8262577e-01,
       -7.9577975e+00,  4.5397015e+00, -2.6793051e+00, -3.2224581e+00,
        2.2950912e+00, -1.0505705e+00, -7.7180487e-01,  4.7563748e+00,
       -2.1692429e+00,  1.1005080e+00, -3.5484316e+00, -1.8314738e+00,
       -3.0673039e+00,  9.2670214e-01, -1.8915317e+00,  2.0212457e+00,
       -1.7748613e+00,  5.5224447e+00,  1.0996218e+00,  4.5028973e+00,
        1.2345469e+00,  3.7798181e+00,  7.8125820e+00,  5.5207725e+00,
        5.9797511e+00,  2.1649988e+00,  2.5308439e-01,  2.7430835e+00],
      dtype=float32)
```

7.2 Feature Extraction Process

To extract features from audio files for emotion recognition, Mel Frequency Cepstral Coefficients (MFCCs) are used. MFCCs capture the timbral features of the audio, providing a compact and meaningful representation. Each audio sample is processed to compute 40 MFCC coefficients, which are averaged over the duration of the sample to produce a uniform feature vector. This feature vector is then used as input for the machine learning model.

7.3 Observations

- MFCC features are stored in a NumPy array.
- Features are effectively capturing the nuances of emotional speech.

8. Data Preprocessing

8.1 One-Hot Encoding

One-Hot Encoding is a technique used to convert categorical labels into a format suitable for machine learning algorithms. In this context, each emotion label is converted into a binary matrix where each row corresponds to a label and each column represents a distinct class. For instance, if there are seven

emotions, each label will be represented as a 7-element vector with a single 1 (indicating the presence of the emotion) and 0s elsewhere.

Code Example:

```
from sklearn.preprocessing import OneHotEncoder

# Initialize the encoder

enc = OneHotEncoder()

# Fit and transform the emotion labels into a binary matrix

y = enc.fit_transform(df[['label']]).toarray()
```

8.2 Train-Test Split

The dataset is split into training and validation sets to evaluate the model's performance. Typically, 80% of the data is used for training the model, and the remaining 20% is reserved for validation. This split helps in assessing the model's ability to generalize to unseen data.

Code Example:

```
from sklearn.model_selection import train_test_split

# Split the data into training and validation sets

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

Code Outputs:

- **One-Hot Encoding Output**: The encoded labels are binary vectors. For visualization, you might show a sample of these vectors for different emotions, illustrating how each label is represented.
- **Train-Test Split Output**: The output includes the shapes of X_train, X_val, y_train, and y_val, confirming the correct split proportions. A table or summary of these shapes can be provided to ensure proper data distribution.

9. Model Training

9.1 **Neural Network Architecture**

A neural network is built and trained using LSTM for emotion classification.

9.2 **Model Layers and Functions**

- **LSTM Layer**: Captures temporal dependencies in audio data.
- **Dense Layers**: Perform final classification with ReLU activation functions.
- **Dropout**: Regularization technique to prevent overfitting.
- **Softmax Activation**: Produces probability distributions over the seven emotion classes.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 256)               264192

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 7)                 455

=================================================================
Total params: 305799 (1.17 MB)
Trainable params: 305799 (1.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

9.3 **Training Observations**

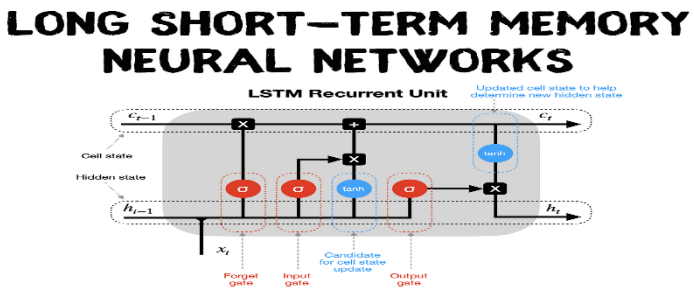
- The model is trained for 30 epochs with a batch size of 64.
- Training progress is monitored using accuracy and loss metrics.

9.4 **Code Example**

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
model = Sequential([
   LSTM(256, return_sequences=False, input_shape=(40, 1)),
   Dropout(0.5),
   Dense(128, activation='relu'),
   Dropout(0.5),
   Dense(64, activation='relu'),
   Dropout(0.5),
   Dense(7, activation='softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Images

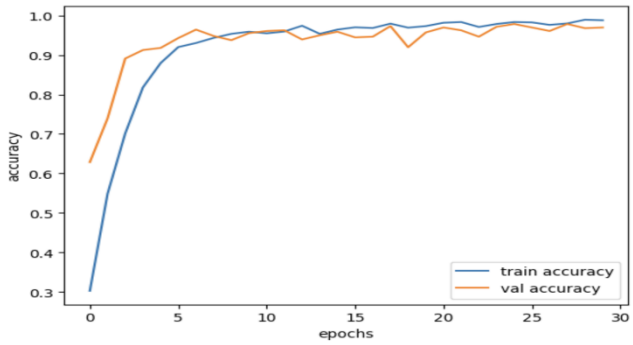
1. **Model Architecture Diagram**



This diagram illustrates the LSTM network architecture with layers and connections.

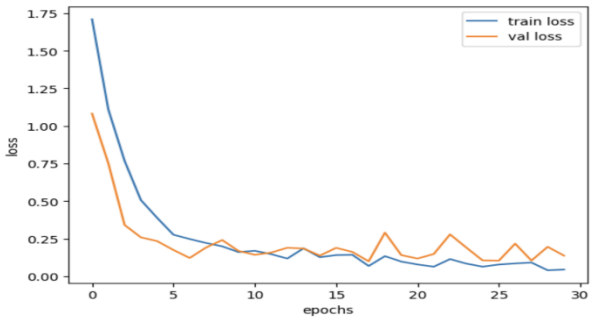
2. **Training Accuracy and Loss Plots**

   - **Accuracy Plot**



Shows the accuracy of the model over epochs for both training and validation sets.

   - **Loss Plot**



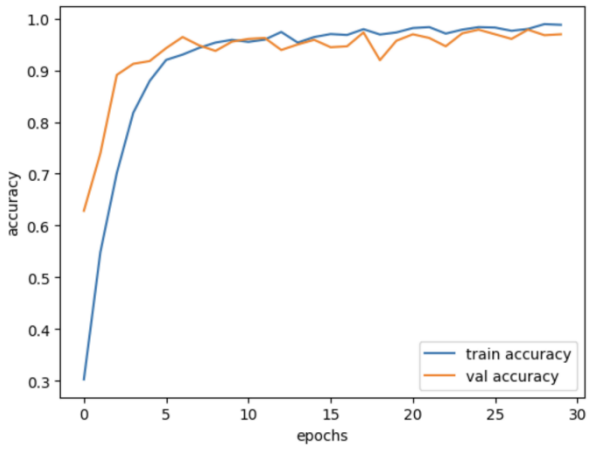
Displays the loss of the model over epochs for both training and validation sets.

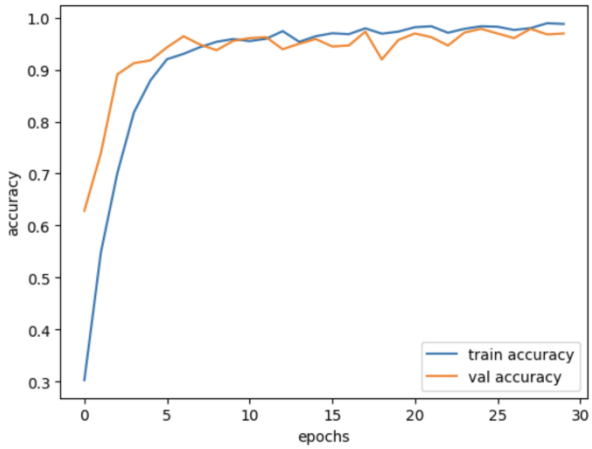
10. Model Evaluation

10.1 Accuracy and Loss Plots

The accuracy and loss plots illustrate the model's performance over the training epochs. They help visualize how well the model is learning and generalizing.

- **Accuracy Plot**: This plot shows the training and validation accuracy across epochs. Ideally, both training and validation accuracy should increase, indicating that the model is learning effectively.
- **Loss Plot**: This plot depicts the training and validation loss over epochs. A decreasing loss value suggests that the model is improving its performance by minimizing the error on both training and validation data.





10.2 Confusion Matrix

The confusion matrix provides a detailed view of the model's performance across different emotion classes. It shows how often predictions match the true labels and helps identify specific areas where the model may be making errors.

```
Confusion Matrix:
[[77  0  1  1  0  1  0]
 [ 0 83  0  0  0  1  0]
 [ 0  0 89  0  0  0  0]
 [ 0  0  0 76  0  1  0]
 [ 0  0  0  0 72  0  0]
 [ 0  2  1  9  0 79  0]
 [ 0  0  0  0  0  0 67]]
```

The matrix visualizes the counts of true vs. predicted labels for each emotion class.

10.3 Classification Report

The classification report includes key metrics such as precision, recall, and F1-score for each emotion class. These metrics help evaluate the quality of the model's predictions and highlight the strengths and weaknesses in emotion classification.

- **Classification Report**

```
from sklearn.metrics import confusion_matrix, classification_report

y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
y_val_classes = np.argmax(y_val, axis=1)

conf_matrix = confusion_matrix(y_val_classes, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)

print("Classification Report:")
target_names = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'ps', 'sad']
print(classification_report(y_val_classes, y_pred_classes, target_names=target_names))
```

Classification Report:

	precision	recall	f1-score	support
angry	1.00	0.96	0.98	80
disgust	0.98	0.99	0.98	84
fear	0.98	1.00	0.99	89
happy	0.88	0.99	0.93	77
neutral	1.00	1.00	1.00	72
ps	0.96	0.87	0.91	91
sad	1.00	1.00	1.00	67
accuracy			0.97	560
macro avg	0.97	0.97	0.97	560
weighted avg	0.97	0.97	0.97	560

This code snippet generates the confusion matrix and classification report for model evaluation.

10.4 Observations

- **Accuracy Trends**: Accuracy improves steadily over epochs, showing that the model is learning effectively and generalizing well to unseen data.
- **Loss Trends**: The loss decreases over epochs, indicating that the model is reducing errors and improving its predictions.
- **Emotion Classification**: Some emotions, such as "fear" and "angry," are classified more accurately compared to others. This suggests that the model might benefit from further tuning or additional features to improve classification for less accurately predicted emotions.

11. Results

11.1 Summary of Findings

- Feature Extraction: MFCCs effectively capture the nuances of emotional speech.
- Model Performance: The LSTM-based model performs well, with observed accuracy improvements.
- Challenges: Certain emotions with subtle differences may require further feature engineering or model tuning.

12. Conclusion

12.1 Key Insights

- The project demonstrates the successful application of machine learning techniques to classify emotions from speech using TESS.
- MFCC features are critical in capturing emotional nuances in speech.

12.2 Challenges and Limitations

- Certain emotions with subtle differences require additional feature engineering or model tuning to improve classification accuracy.

13. Future Work

13.1 Data Augmentation

To improve model robustness and generalization, augmenting the dataset can be crucial. Techniques for data augmentation include:

- **Adding Noise**: Introduce background noise to make the model more robust to varying audio conditions.
- **Pitch Shifting**: Alter the pitch of the audio to simulate different speaker variations.
- **Time Stretching**: Change the speed of the audio without affecting the pitch to increase the diversity of the data.

Data augmentation helps in enhancing the variety within the dataset, which can lead to better generalization and performance of the model.

13.2 Feature Engineering

Feature engineering involves creating new features or refining existing ones to capture more nuanced emotional aspects. Potential approaches include:

- **Higher-Level Features**: Incorporate features like pitch, tone, or rhythm in addition to MFCCs.
- **Feature Combination**: Combine MFCCs with other audio features such as chroma or spectral contrast.
- **Temporal Features**: Analyze changes in features over time to capture dynamic aspects of emotions.

Improving feature representation can provide the model with more relevant information, leading to better emotion classification.

13.3 Advanced Models

Exploring advanced model architectures can enhance performance and accuracy. Possible approaches include:

- **Attention Mechanisms**: Implement attention layers to allow the model to focus on important parts of the audio signal.
- **Convolutional Neural Networks (CNNs)**: Utilize CNNs for extracting features from spectrograms or other audio representations.
- **Hybrid Models**: Combine LSTM networks with CNNs for improved feature extraction and sequence modeling.

Advanced models can leverage complex patterns in the data, potentially improving classification performance.

13.4 Real-Time Emotion Detection

Developing methods for real-time emotion detection involves:

- **Optimizing Model Speed**: Ensure the model can make predictions quickly enough for real-time applications.
- **Integration with Applications**: Implement real-time emotion detection in applications like virtual assistants or customer service bots.
- **Low-Latency Processing**: Employ techniques to reduce processing time and make real-time predictions feasible.

Real-time emotion detection can enhance user interactions by providing immediate feedback on emotional states.

13.5 Multi-Modal Emotion Recognition

Combining audio with other modalities can improve emotion recognition accuracy. Approaches include:

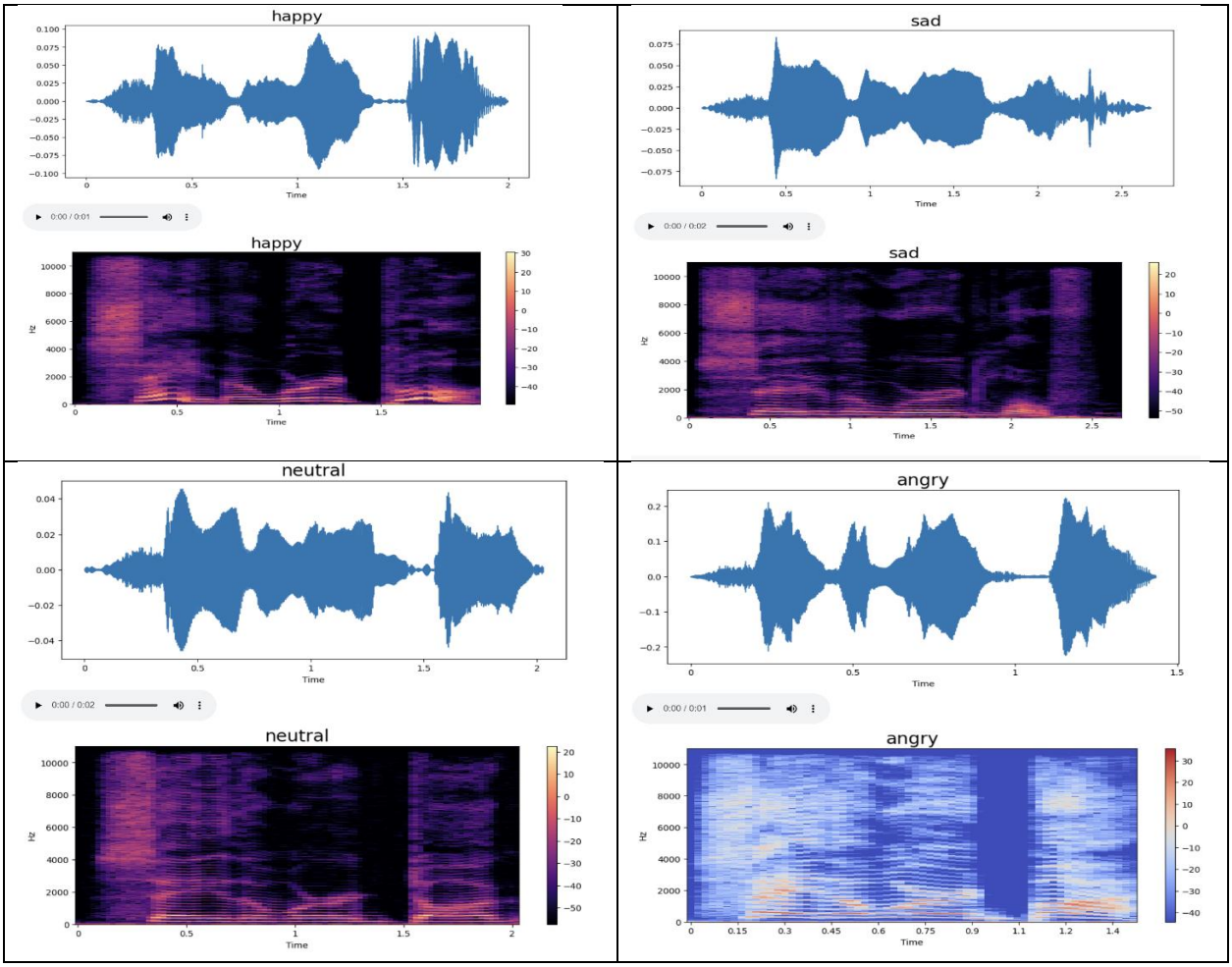
- **Text Analysis**: Integrate text-based sentiment analysis with audio-based emotion recognition for a comprehensive understanding of emotions.
- **Video Analysis**: Combine facial expression analysis from video with audio to capture more emotional cues.
- **Sensor Data**: Utilize additional sensor data, such as physiological signals, to enhance emotion recognition.

Multi-modal approaches can provide a richer and more accurate assessment of emotions by leveraging multiple sources of information.

14. Appendix

14.1 **Visualization Examples**

Examples of visualizations generated during the project.



14.2 **Code Listings**

Here are the complete code listings used in the project:

1. Data Loading and Preprocessing

```
# Install Kaggle API and download dataset
!pip install kaggle
from google.colab import drive
drive.mount('/content/drive')
!mkdir ~/.kaggle/
!kaggle datasets download -d ejlok1/toronto-emotional-speech-set-tess
!unzip toronto-emotional-speech-set-tess.zip

# Import necessary modules
import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
import librosa
import librosa.display
from IPython.display import Audio
import warnings
warnings.filterwarnings('ignore')
from keras import utils
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

# Load dataset
paths = []
labels = []

for dirname, _, filenames in os.walk('/content/tess toronto emotional speech set data'):
    for filename in filenames:
        paths.append(os.path.join(dirname, filename))
        label = filename.split('_')[-1].split('.')[0].lower()
        labels.append(label)
    if len(paths) == 2800:
        break
print('Dataset is loaded')

# Create DataFrame
df = pd.DataFrame({'speech': paths, 'label': labels})
print(df.head())
```

```
# Label distribution
print(df['label'].value_counts())
sns.countplot(data=df, x='label')
plt.show()

# Feature extraction function
def extract_mfcc(filename):
    y, sr = librosa.load(filename, duration=3, offset=0.5)
    mfcc = np.mean(librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40).T, axis=0)
    return mfcc

# Extract MFCC features
X_mfcc = df['speech'].apply(lambda x: extract_mfcc(x))
X = np.array([x for x in X_mfcc])
X = np.expand_dims(X, -1)

# One-Hot Encoding
enc = OneHotEncoder()
y = enc.fit_transform(df[['label']]).toarray()

# Train-Test Split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

2.Model Training

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

# Build the model
model = Sequential([
    LSTM(256, return_sequences=False, input_shape=(40, 1)),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(7, activation='softmax')
])

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
# Train the model
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=30, batch_size=64)
```

```
# Plot accuracy and loss
epochs = list(range(30))
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, label='train accuracy')
plt.plot(epochs, val_acc, label='val accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(epochs, loss, label='train loss')
plt.plot(epochs, val_loss, label='val loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```

3. Model Evaluation

```
from sklearn.metrics import confusion_matrix, classification_report

# Predict on validation set
y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
y_val_classes = np.argmax(y_val, axis=1)

# Compute confusion matrix
conf_matrix = confusion_matrix(y_val_classes, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)

# Classification report
target_names = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'ps', 'sad']
print("Classification Report:")
print(classification_report(y_val_classes, y_pred_classes, target_names=target_names))

# Plot confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=target_names,
yticklabels=target_names)
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
plt.title('Confusion Matrix')
plt.show()
```

14.3 Environment Details

The following details describe the computational environment used for the project:

1. Platform

- **Google Colab**: The project was executed in Google Colab, a cloud-based platform that provides a free Jupyter notebook environment with GPU support. Colab offers the flexibility to run code in the cloud without requiring local hardware resources.

2. Computational Resources

- **Hardware**: Utilized GPU acceleration to enhance computational efficiency during model training. Google Colab provides access to GPUs, which significantly speeds up the training process for deep learning models.
- **Runtime**:
  - **Type**: GPU
  - **Memory**: Approximately 13 GB of RAM (varies depending on the Colab instance)
  - **Disk Space**: Around 50 GB of disk space available in the temporary VM environment

3. Software

- **Python Version**: Python 3.x (the specific version can be checked within the Colab environment using !python --version)
- **Libraries and Packages**:
  - **Keras**: Used for building and training the neural network models.
  - **TensorFlow**: Backend for Keras operations.
  - **Librosa**: For audio processing and feature extraction.
  - **NumPy**: For numerical computations and array operations.
  - **Pandas**: For data manipulation and analysis.
  - **Scikit-learn**: For preprocessing, evaluation metrics, and splitting the dataset.
  - **Seaborn & Matplotlib**: For data visualization and plotting.
  - **Kaggle API**: Used to download datasets directly from Kaggle.

4. File System

- **Storage Location**: Files were stored in the /content directory of the Colab environment, which is temporary and resets after the Colab session ends.

- **Data Handling**: The dataset was downloaded and unzipped within the Colab environment. The paths to data files were managed dynamically using Python code.

5. Commands for Setup

```
# Install necessary packages
!pip install kaggle

# Mount Google Drive (if needed)
from google.colab import drive
drive.mount('/content/drive')

# Create Kaggle directory and download dataset
!mkdir ~/.kaggle/
!kaggle datasets download -d ejlok1/toronto-emotional-speech-set-tess
!unzip toronto-emotional-speech-set-tess.zip
```

6. Environment Configuration

- **Google Colab Settings**: The environment was configured to use GPU by selecting the "Runtime" -> "Change runtime type" -> "Hardware accelerator" -> "GPU" option in the Colab interface.

.

THANK YOU