**Overview**

This project focuses on creating an advanced system that can detect human emotions from audio recordings. Using cutting-edge machine learning techniques, this system transforms raw audio data into meaningful emotional insights. It supports multiple audio formats, including MP3 and WAV, and uses the Toronto Emotional Speech Set (TESS) dataset, augmented with additional voices, for training.

**Project Structure**

The project is organized into several key scripts and directories:

- **config.py**: Defines configuration parameters for the project.

-**preprocess.py**: Prepares the audio data for model training by extracting features and performing data augmentation.

- **model.py**: Defines and trains the neural network model.

- **main.py**: Implements a Flask web server for deploying the model and handling user requests.

- **emotion\_data.pkl**: A pickle file storing the preprocessed audio data.

- **emotion\_model.keras**: The trained model saved in Keras format.

- **emotion\_model.weights.h5**:The model weights saved in HDF5 format.

- **uploads/**: Directory for storing uploaded audio files.

**Dataset**

The dataset used in this project includes the TESS dataset along with additional recordings from 10 male and 5 female voices form CREMA-D dataset. Each recording represents one of seven distinct emotions: happy, sad, angry, fearful, disgusted, surprised, and neutral. The dataset is organized into folders corresponding to each emotion. Speakers included in the dataset CREMA are1-15:

**Preprocessing**

Preprocessing is a crucial step that converts raw audio data into a structured format suitable for machine learning. This process involves several steps:

1. **Loading Audio Files**: The script uses `librosa` to load audio files and resample them to a consistent sampling rate of 16 kHz.

2. **Data Augmentation**: To increase the diversity and size of the training dataset, the script applies several audio augmentations using the `audiomentations` library.

3. **Feature Extraction**: Mel-Frequency Cepstral Coefficients (MFCCs) are extracted from the audio files. MFCCs are a set of features that represent the short-term power spectrum of sound and are crucial for capturing the nuances of speech.

4. **Normalization:** The MFCC features are standardized using `StandardScaler`. Normalization ensures that the data has a mean of zero and a standard deviation of one, which helps the neural network converge faster during training.

5. **Padding/Truncation:** All feature vectors are padded or truncated to a fixed length, ensuring uniform input size for the model.

6. **Saving Data**: The processed features and corresponding labels are saved to a pickle file (`emotion\_data.pkl`).

**Model Architecture**

The model architecture combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to leverage both spatial and temporal features in the audio data. Here is a brief overview of the layers:

- **Convolutional Layer**: Captures local patterns in the MFCC features. CNNs are effective at detecting spatial hierarchies in the data, such as phonemes in speech.

- **BatchNormalization Layer:** Normalizes activations for faster convergence. This helps in stabilizing and speeding up the training process.

- **MaxPooling Layer**: Reduces dimensionality and helps control overfitting by retaining the most significant features.

- **Bidirectional LSTM Layers**: Captures temporal dependencies and sequences in the audio data, essential for understanding the progression of emotions over time. Bidirectional LSTMs process the data in both forward and backward directions, providing a more comprehensive understanding of the temporal dynamics.

- **Dropout Layers**: Used for regularization to prevent overfitting by randomly setting a fraction of input units to zero during training.

- **Dense Layer**: The final dense layer uses a softmax activation function to produce a probability distribution over the emotion classes, enabling the model to predict the most likely emotion.

- **Adam Optimizer**:This optimizer is known for its adaptive learning rate, making it efficient and effective for training deep neural networks.

- **Sparse Categorical Cross-Entropy**: This loss function is ideal for multi-class classification tasks where labels are integers. It calculates the cross-entropy loss between the true labels and the predicted labels, encouraging the model to predict the correct class with higher confidence.

**Compilation**

The model is compiled with the Adam optimizer, known for its efficiency in training deep learning models. The loss function used is sparse categorical cross-entropy, suitable for multi-class classification problems, and the metric for evaluation is accuracy.

**Training and Evaluation**

**Data Augmentation**

The training dataset is augmented to increase its size twofold, helping the model generalize better by learning from a more diverse set of examples.

**Training**

The data is split into training and testing sets using an 80-20 split. Early stopping is employed to halt training when the validation loss stops improving, preventing overfitting and ensuring the model is well-generalized. The model is trained for up to 100 epochs with a batch size of 32, using early stopping to monitor validation loss.

**Evaluation Metrics**

The model is evaluated based on its accuracy on both the training and testing datasets. Accuracy measures the percentage of correctly classified samples. The trained model's performance is evaluated on both the training and testing sets to ensure its accuracy and generalization capability.

- Training Accuracy:95.52%

- Test Accuracy: 82.59%

**Web Interface with Flask**

The web interface is built using Flask, a lightweight web framework for Python. It provides two main endpoints:

- **/predict:** Accepts an uploaded audio file (in MP3 or WAV format) and returns the predicted emotion. The uploaded file is saved temporarily, processed to extract MFCC features, and then passed to the trained model for prediction.

- **/record**: Records audio from the user's microphone for a specified duration, processes the recorded audio, and returns the predicted emotion. This endpoint is useful for real-time emotion detection.

**How It Works?**

**Audio Upload and Processing**: Users can upload an audio file, which the server saves temporarily. The file is then processed to extract MFCC features, and these features are passed to the model to predict the emotion.

**Real-Time Recording:** The server can record audio chunks from the microphone, process each chunk in real-time, and provide immediate feedback on the detected emotion.

**Real-Time Predictions**

The real-time prediction capability allows users to interactively upload audio files or record their voices to get immediate feedback on the detected emotion.

**Usage**

To use this project, follow these steps:

1. **Preprocess the Data**: Run `preprocess.py` to process the audio data and generate `emotion\_data.pkl`.

2. **Train the Model**: Execute `model.py` to train the neural network and save the trained model as `emotion\_model.keras`.

3. **Start the Web Server:** Launch the Flask server by running `main.py`. The server will start listening for requests at the specified port.

4. **Make Predictions**:

- Use the `/predict` endpoint to upload an audio file and get the predicted emotion.

- Use the `/record` endpoint to record audio through the microphone and get the predicted emotion.

**Example Usage**

- To predict emotion from an uploaded file:

```bash

curl -X POST -F "file=@/path/to/audio.wav" http://127.0.0.1:5000/predict

```

- To record audio and predict emotion:

```bash

curl -X POST -H "Content-Type: application/json" -d '{"duration": 10, "chunk\_duration": 3}' http://127.0.0.1:5000/record

```

**Dataset License**

This project utilizes the Toronto Emotional Speech Set (TESS) dataset CREMA-D(Crowd-sourced Emotional Multimodal Actors Dataset):, which are publicly available for academic research. Please adhere to the dataset's license terms when using it.

**Code License**

The code in this project is licensed under the MIT License, which permits reuse, modification, and distribution of the code with proper attribution. This license encourages the open sharing and collaborative improvement of the project.

**Conclusion**

This project showcases an advanced and user-friendly emotion detection system using cutting-edge audio processing and machine learning techniques. By supporting various audio formats and providing a robust real-time prediction interface, it stands as a versatile tool for emotion recognition applications. From preprocessing and model training to deployment and real-time interaction, every aspect of this project is designed to deliver accurate and efficient emotion detection.

**A Glimpse of the Project**

**Preprocessing**: Converts raw audio data into structured format with MFCC feature extraction, data augmentation, normalization, and padding/truncation.

**Model Architecture**: Combines CNN and LSTM layers for capturing both spatial and temporal features in audio data, using a softmax dense layer for emotion prediction.

**Training & Evaluation**: Trains with an 80-20 data split, employing early stopping to prevent overfitting, achieving 90.96% training accuracy and 81.21% test accuracy.

**Web Interface**: Flask-based endpoints for predicting emotions from uploaded audio files or real-time recordings, offering immediate feedback on detected emotions.

**Usage**: Simple steps to preprocess data, train the model, start the web server, and make predictions via API endpoints, making the system accessible and practical for emotion detection applications.