Multimodal Emotion Recognition from Audio and Transcript

Objective

Design and train a deep learning model that classifies **human emotions** using both:

- **Audio data** (processed visually as spectrograms for CNN input)
- **Generated text transcripts** (via speech-to-text for NLP modeling)

This challenge aims to apply **Convolutional Neural Networks (CNNs)**, **Recurrent Neural Networks (RNNs)**, and optionally **Transformers**, while exploring multimodal learning from real-world emotional speech data.

Dataset

RAVDESS Emotional Speech Audio
Kaggle Link

- Audio-only version of RAVDESS
- 1440 speech clips (male and female actors)
- 8 emotions: **neutral, calm, happy, sad, angry, fearful, disgust, surprised**

Phases of the Task

Phase 1 – Unimodal Pipelines

- **Audio CNN**:
    - Convert audio to **spectrograms or MFCCs**

    - Train a CNN to classify emotions from these 2D visual representations

- **Text RNN**:
    - Generate transcripts from the audio (e.g., using **Whisper** or **Google Speech-to-Text**)
    - Train an RNN (e.g., LSTM or GRU) on these transcripts to classify emotions

Note: If transcripts are poor, you can resort to using simulated sentences based on emotion labels (e.g., "I am very angry!")

Phase 2 – Multimodal Fusion

- Merge the visual features from the CNN and text features from the RNN
- Explore different fusion strategies:
    - Early fusion: concatenate embeddings before classification
    - Late fusion: average/logit voting

Bonus Phase – Transformers

- Replace the RNN with a Transformer-based text model (e.g., **DistilBERT**, **BERT**, or **Whisper** encoder…any model of your choice is acceptable)
- Explore using **CNN + BERT** or even **audio transformer** models like **AST**

Tools You Can Use

- **Audio processing**: Librosa, torchaudio, OpenCV (for spectrograms)
- **Modeling**: PyTorch / TensorFlow, Hugging Face Transformers

- **Speech-to-text**: Whisper (open-source), Google Speech API, etc.

- **Other**: matplotlib/seaborn, scikit-learn, NumPy, pandas

Deliverables

- .py or .ipynb files containing your code implementation. Ensure the files are well-documented and modular. Providing setup details via a requirements.txt or using a dependency managing tool like **Poetry** will be given extra merit.
- Share the training and validation accuracies of your models in a .docx or .pdf. Using more detailed evaluation metrics and drawing inferences is encouraged.