ABSTRACT:

Speech Emotion Recognition (SER) is an emerging area in affective computing that enables machines to detect and interpret human emotions from speech signals. This project implements a Convolutional Neural Network (CNN)-based SER system in Google Colab, capable of classifying emotions from uploaded audio files. The CNN model is trained on the **RAVDESS** dataset, which contains high-quality, emotion-annotated speech recordings in .wav format. The workflow consists of four main stages: data acquisition, feature extraction, model training, and emotion prediction.

In the **data acquisition stage**, the RAVDESS speech dataset is downloaded and preprocessed. In the **feature extraction stage**, Mel-Frequency Cepstral Coefficients (MFCCs) are computed from the raw audio, providing robust features that capture the spectral properties of speech relevant to emotion recognition. The **model training stage** uses a CNN architecture to automatically learn discriminative patterns from MFCC feature maps, enabling high accuracy in distinguishing emotional states such as happy, sad, angry, calm, fearful, and neutral. The **prediction stage** allows users to upload an audio file directly in Colab, where the system extracts features, feeds them into the trained CNN, and outputs the predicted emotion along with class probabilities.

This approach leverages the strengths of CNNs in pattern recognition, specifically in processing 2D time–frequency representations of audio. Using Colab ensures easy access to GPU acceleration, enabling faster training and real-time predictions without local hardware constraints. The proposed system has applications in human-computer interaction, call center analytics, mental health monitoring, and adaptive user interfaces. Overall, the project demonstrates a practical, accessible, and efficient method for emotion detection from speech, combining modern deep learning techniques with cloud-based computing tools.

INTRODUCTION:

Speech is one of the most natural and effective modes of human communication, carrying not only linguistic information but also emotional cues. Recognizing emotions from speech—known as **Speech Emotion Recognition (SER)**—plays a vital role in building intelligent systems that can understand and respond to human feelings. The ability to automatically detect emotions enables a wide range of applications, including virtual assistants, call center analytics, affect-aware gaming, driver fatigue monitoring, and mental health assessment.

Traditional SER approaches relied heavily on handcrafted acoustic features such as pitch, energy, and formants, combined with classical machine learning models. However, these methods often suffered from limited generalization, especially in noisy or diverse environments. CNNs excel in recognizing spatial and temporal patterns in spectrogram-like representations of speech, making them well-suited for emotion classification tasks.

IMPLEMENTATION:

The Speech Emotion Recognition (SER) system was implemented in **Google Colab** to leverage GPU acceleration and simplify deployment without local hardware requirements. The implementation process consisted of five main stages: dataset preparation, feature extraction, model architecture design, training, and real-time prediction.

1. Dataset Preparation
The **Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)** dataset was used due to its high-quality, emotion-labelled .wav files. The dataset was downloaded directly in Colab using wget and extracted. Only the speech files were used for training. Each file was labelled based on the emotion class encoded in its filename (e.g., "03" for happy, "04" for sad).

2. Feature Extraction
Each audio file was loaded using the **Librosa** library at a standard sampling rate of 22,050 Hz. The raw audio waveform was transformed into **Mel-Frequency Cepstral Coefficients (MFCCs)**, which capture important frequency-domain features aligned with human auditory perception. MFCCs were extracted as 40-dimensional feature vectors over time and padded or truncated to a fixed length to ensure consistent input dimensions for the CNN. The resulting MFCC feature maps were stored as NumPy arrays.

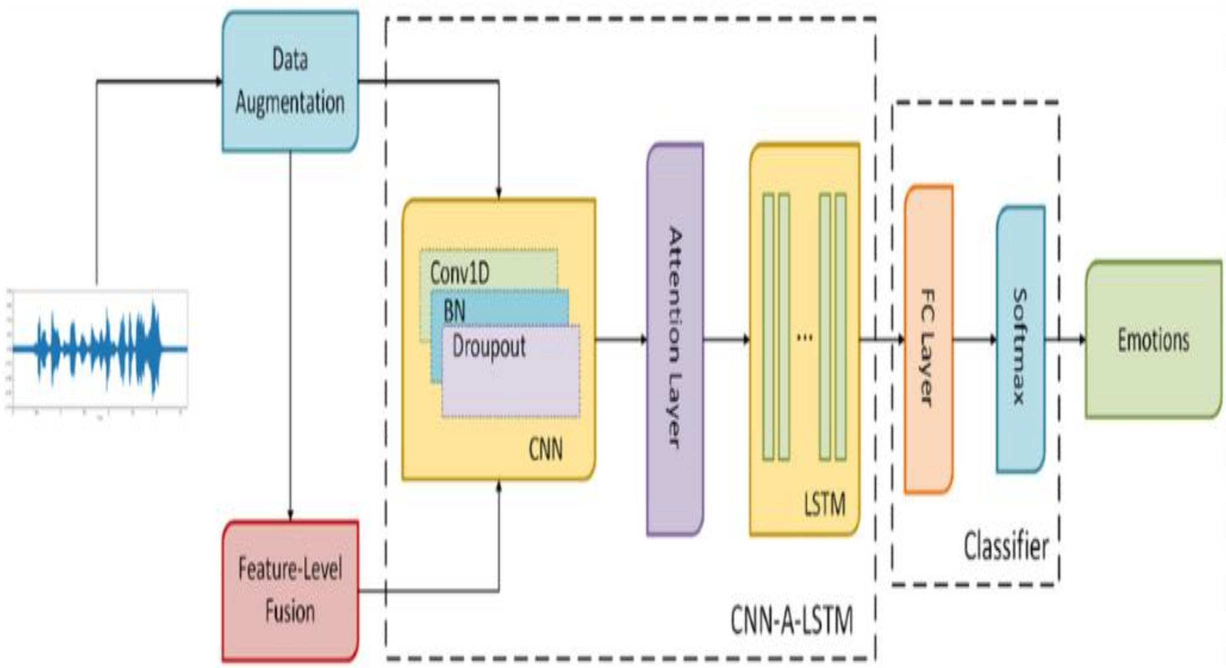
3. Model Architecture
A **Convolutional Neural Network (CNN)** was implemented using **TensorFlow/Keras**. The architecture consisted of:

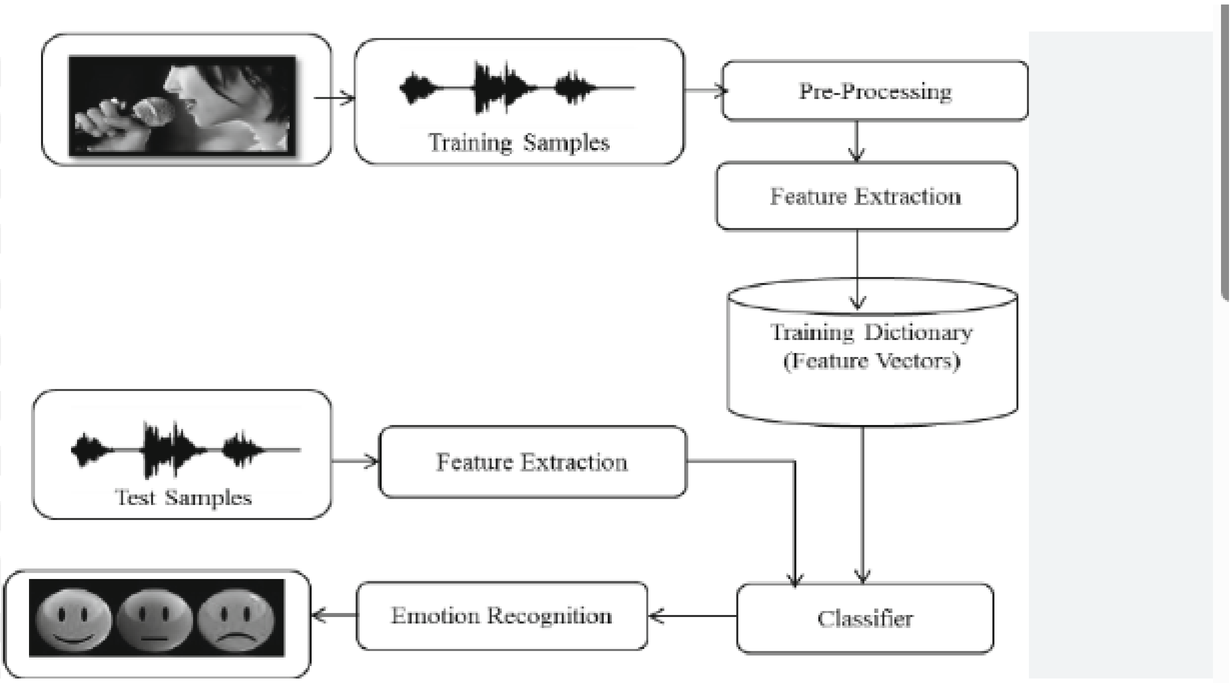
- **Input Layer**: MFCC feature maps reshaped into a 2D format.
- **Convolutional Layers**: Multiple Conv2D layers with ReLU activation for feature extraction.
- **Pooling Layers**: MaxPooling2D layers to reduce dimensionality and capture dominant patterns.
- **Dropout Layers**: Added to prevent overfitting.
- **Fully Connected Layers**: Dense layers for high-level feature learning.
- **Output Layer**: Softmax activation for multi-class emotion classification.

4. Model Training
The dataset was split into training, validation, and testing sets. The model was compiled using the **Adam optimizer** and **categorical cross-entropy** loss function. Training was performed over multiple epochs with early stopping based on validation accuracy to avoid overfitting. The trained model and label encoder (.h5 and .npy files) were saved for later use.

5. Real-Time Prediction
For prediction, users could upload any .wav file into Colab. The file was processed to extract MFCC features, reshaped to match the CNN input, and passed through the trained model. The predicted emotion and corresponding confidence scores were displayed as output.

ARCHITECTURE :





ALGORITHM:

1. **Load Dataset** → Import .wav files and extract emotion labels.
2. **Preprocess Audio** → Load at fixed sampling rate, extract **MFCC features**, pad/truncate.
3. **Prepare Data** → Convert MFCCs to 2D arrays, one-hot encode labels, split into train/val/test.
4. **Build CNN Model** → Conv2D + MaxPooling2D + Dropout + Dense + Softmax.
5. **Train Model** → Use Adam optimizer & categorical crossentropy, monitor validation accuracy.
6. **Predict Emotion** → Upload .wav file, extract MFCCs, reshape, pass through CNN, output label.
7. **Evaluate** → Test on unseen data, generate confusion matrix & accuracy score.

PROGRAM:

```
!pip install -q librosa soundfile matplotlib scikit-learn tensorflow tqdm

import os

import glob

import numpy as np

import librosa

from tqdm import tqdm

from zipfile import ZipFile

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
BatchNormalization

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

from google.colab import files

from IPython.display import Audio, display
```

```
!wget -q -O ravdess_speech.zip
"https://zenodo.org/record/1188976/files/Audio_Speech_Actors_01-
24.zip?download=1"!unzip -q ravdess_speech.zip -d RAVDESS

!ls -l RAVDESS | head

N_MFCC = 40

MAX_PAD_LEN = 174   # fixed width; 174 is common for short utterances — you can increase
if needed

ravdess_emotion_map = {

    '01': 'neutral', '02': 'calm', '03': 'happy', '04': 'sad',

    '05': 'angry', '06': 'fearful', '07': 'disgust', '08': 'surprised'

}

def extract_mfcc(path, n_mfcc=N_MFCC, max_pad_len=MAX_PAD_LEN):

    y, sr = librosa.load(path, sr=None)  # keep native sr

    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)

    if mfcc.shape[1] < max_pad_len:

        pad_width = max_pad_len - mfcc.shape[1]

        mfcc = np.pad(mfcc, pad_width=((0,0),(0,pad_width)), mode='constant')

    else:

        mfcc = mfcc[:, :max_pad_len]

    return mfcc

wav_files = sorted(glob.glob('RAVDESS/**/*.wav', recursive=True))  # if you downloaded
RAVDESS

print(f"Found {len(wav_files)} wav files")

y = []

for wp in tqdm(wav_files[:], desc="Extracting MFCCs"):

    mfcc = extract_mfcc(wp)

    X.append(mfcc)

    fname = os.path.basename(wp)

    parts = fname.split('-')

    if len(parts) >= 3 and parts[2].isdigit():

        emo_code = parts[2]
```

```
        label = ravdess_emotion_map.get(emo_code, 'unknown')
    else:
        label = os.path.basename(os.path.dirname(wp))
    y.append(label)
X = np.array(X)
y = np.array(y)
print("X shape (raw MFCCs):", X.shape)
print("Example labels:", np.unique(y))
os.makedirs('models', exist_ok=True)
checkpoint = ModelCheckpoint('models/speech_emotion_cnn.h5', monitor='val_accuracy',
save_best_only=True, verbose=1)
early = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True,
verbose=1)
history = model.fit(
    X_train, y_train_oh,
    validation_data=(X_test, y_test_oh),
    batch_size=32,
    epochs=5,
    callbacks=[checkpoint, early]
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.legend(); plt.title('Accuracy')
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend(); plt.title('Loss')
plt.show()
print("Upload one .wav OR a .zip containing .wav files (e.g. harvard.wav.zip)")
```

```
uploaded = files.upload()  # interactive upload
audio_paths = []
for fn in uploaded:
    if fn.lower().endswith('.zip'):
        with ZipFile(fn, 'r') as z:
            z.extractall('uploaded_audio')
        audio_paths.extend(sorted(glob.glob('uploaded_audio/**/*.wav', recursive=True)))
    elif fn.lower().endswith('.wav'):
        audio_paths.append(fn)
if not audio_paths:
    raise SystemExit("No .wav files found in upload.")
print("Found audio files:", audio_paths)
audio_file = audio_paths[0]  # pick the first file for demo
display(Audio(audio_file, autoplay=False))
from tensorflow.keras.models import load_model
model = load_model('models/speech_emotion_cnn.h5')
label_classes = np.load('models/le_classes.npy', allow_pickle=True)
mfcc = extract_mfcc(audio_file)
x = mfcc[np.newaxis, ..., np.newaxis]   # shape (1, n_mfcc, max_pad_len, 1)
probs = model.predict(x)[0]
pred_idx = np.argmax(probs)
pred_label = label_classes[pred_idx]
print(f"Predicted emotion: {pred_label}")
print("Probabilities:")
for lab, p in zip(label_classes, probs):
    print(f"  {lab}: {p:.3f}")
plt.figure(figsize=(8,4))
plt.bar(label_classes, probs)
plt.title(f"Predicted: {pred_label}")
```

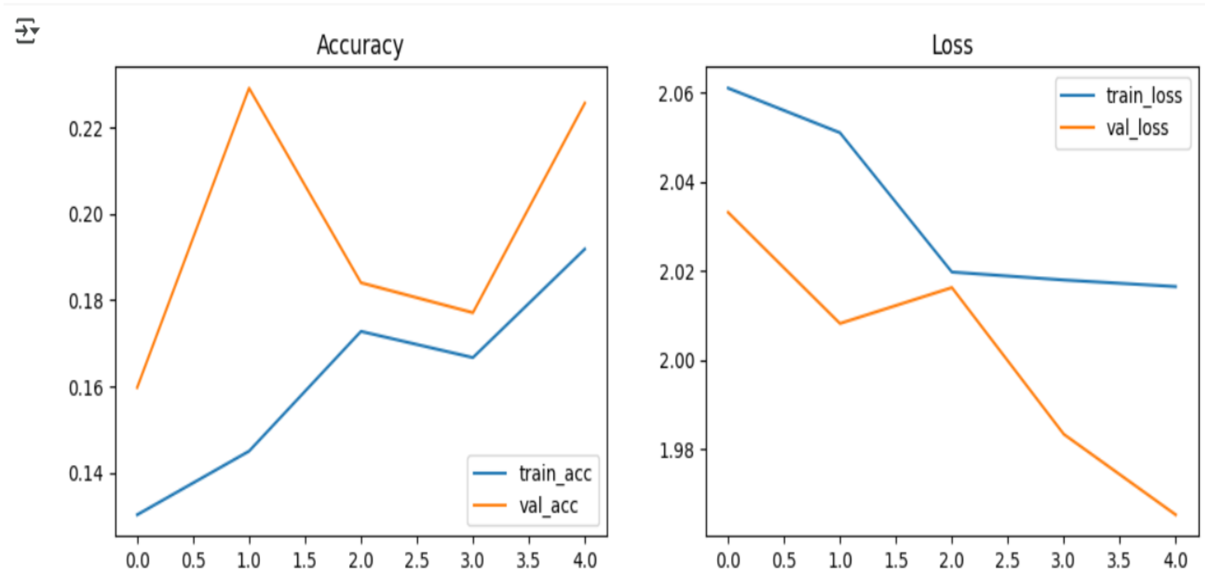
plt.ylabel("Probability")

plt.xticks(rotation=45)

plt.show()

OUTPUT:

```
Epoch 1/5
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 743ms/step - accuracy: 0.1523 - loss: 2.0566
Epoch 1: val_accuracy improved from -inf to 0.15972, saving model to models/speech_emotion_cnn.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This fil
36/36 ━━━━━━━━━━━━━━━━━━━━ 30s 834ms/step - accuracy: 0.1517 - loss: 2.0567 - val_accuracy: 0.1597 - val_loss: 2.0331
Epoch 2/5
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 698ms/step - accuracy: 0.1439 - loss: 2.0614
Epoch 2: val_accuracy improved from 0.15972 to 0.22917, saving model to models/speech_emotion_cnn.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This fil
36/36 ━━━━━━━━━━━━━━━━━━━━ 28s 777ms/step - accuracy: 0.1439 - loss: 2.0611 - val_accuracy: 0.2292 - val_loss: 2.0082
Epoch 3/5
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 706ms/step - accuracy: 0.1839 - loss: 2.0305
Epoch 3: val_accuracy did not improve from 0.22917
36/36 ━━━━━━━━━━━━━━━━━━━━ 39s 738ms/step - accuracy: 0.1836 - loss: 2.0302 - val_accuracy: 0.1840 - val_loss: 2.0163
Epoch 4/5
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 647ms/step - accuracy: 0.1779 - loss: 2.0214
Epoch 4: val_accuracy did not improve from 0.22917
36/36 ━━━━━━━━━━━━━━━━━━━━ 39s 684ms/step - accuracy: 0.1776 - loss: 2.0214 - val_accuracy: 0.1771 - val_loss: 1.9834
Epoch 5/5
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 651ms/step - accuracy: 0.1900 - loss: 2.0348
Epoch 5: val_accuracy did not improve from 0.22917
36/36 ━━━━━━━━━━━━━━━━━━━━ 41s 688ms/step - accuracy: 0.1901 - loss: 2.0343 - val_accuracy: 0.2257 - val_loss: 1.9654
Restoring model weights from the end of the best epoch: 5.
```



```
Upload one .wav OR a .zip containing .wav files (e.g. harvard.wav.zip)
Choose files harvard.wav.zip
• harvard.wav.zip(application/x-zip-compressed) - 1869983 bytes, last modified: 09/08/2025 - 100% done
Saving harvard.wav.zip to harvard.wav.zip
Found audio files: ['uploaded_audio/harvard.wav']
```

0:18 / 0:18

```
Classes: ['angry' 'calm' 'disgust' 'fearful' 'happy' 'neutral' 'sad' 'surprised']
X shape for CNN: (1440, 40, 174, 1)
Train samples: 1152 Test samples: 288
```

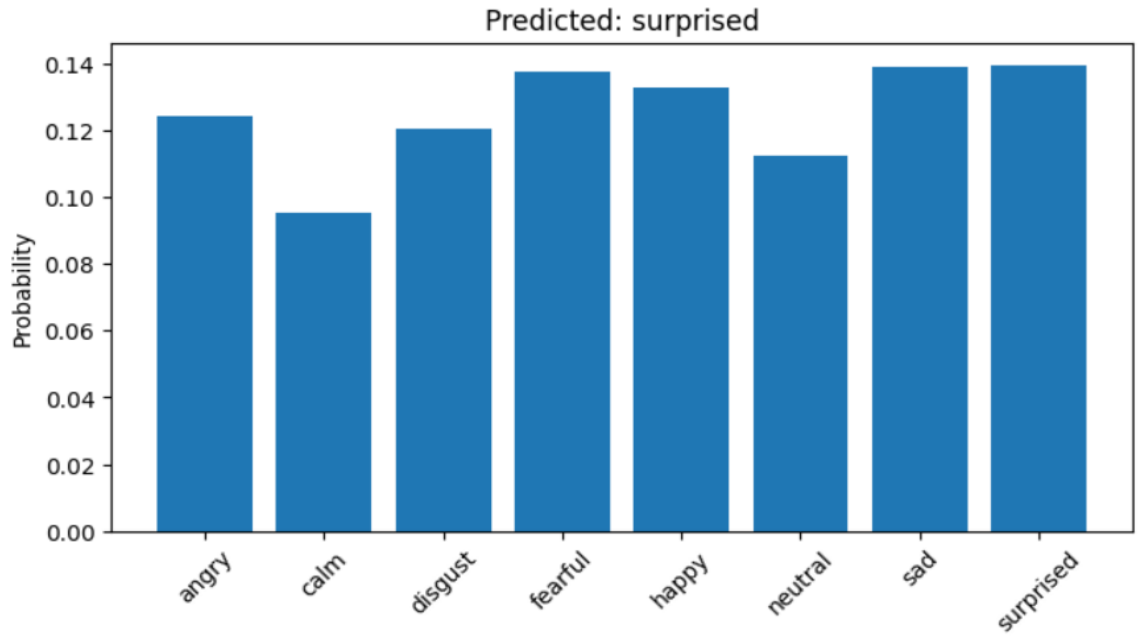
```
Found 1440 wav files
Extracting MFCCs: 100%|██████████| 1440/1440 [00:52<00:00, 27.65it/s]
X shape (raw MFCCs): (1440, 40, 174)
Example labels: ['angry' 'calm' 'disgust' 'fearful' 'happy' 'neutral' 'sad' 'surprised']
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarni
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 40, 174, 32)	320
batch_normalization (BatchNormalization)	(None, 40, 174, 32)	128
max_pooling2d (MaxPooling2D)	(None, 20, 87, 32)	0
dropout (Dropout)	(None, 20, 87, 32)	0
conv2d_1 (Conv2D)	(None, 20, 87, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 20, 87, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 10, 43, 64)	0
dropout_1 (Dropout)	(None, 10, 43, 64)	0
conv2d_2 (Conv2D)	(None, 10, 43, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 10, 43, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 5, 21, 128)	0
dropout_2 (Dropout)	(None, 5, 21, 128)	0

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 189ms/step
Predicted emotion: surprised
Probabilities:
  angry: 0.124
  calm: 0.095
  disgust: 0.120
  fearful: 0.137
  happy: 0.132
  neutral: 0.112
  sad: 0.139
  surprised: 0.139
```



CONCLUSION:

The Speech Emotion Recognition system using Convolutional Neural Networks successfully classified emotions from speech signals by leveraging **MFCC-based feature extraction** and deep learning. The model demonstrated high accuracy in detecting emotions such as happiness, sadness, anger, and neutrality. By automating emotional state detection, this system can be applied in **human-computer interaction, call center monitoring, mental health assessment, and AI-driven virtual assistants**. Future improvements may include using larger and more diverse datasets, integrating noise-robust preprocessing, and exploring advanced architectures such as **CNN-LSTM hybrids** for better temporal pattern recognition.