A Project on Digital Signal Processing

Real-time speech emotion recognition using Deep Learning and Data Augmentation

Submitted for partial fulfillment of award of

BACHELOR OF ENGINEERING

Degree In

COMPUTER SYSTEMS ENGINEERING

By

Zain Ul Abidin Arain
CMS: 133-22-0001

Guided by:
Dr: Junaid Ahmed,

Department of Computer Systems Engineering
Sukkur IBA University

May, 2025

ABSTRACT

Speech Emotion Recognition (SER) is a crucial aspect of Human-Computer Interaction (HCI), enabling machines to interpret and respond to human emotions effectively. This project presents an SER system utilizing a Long Short-Term Memory (LSTM) network, trained on the Toronto Emotional Speech Set (TESS) dataset. Essential speech features such as Mel Frequency Cepstral Coefficients (MFCC), Zero Crossing Rate (ZCR), Mel Spectrogram, Root Mean Square (RMS), and Chroma features were extracted for training. Data augmentation techniques, including noise addition and time shifting, were employed to enhance model robustness. The results demonstrate the model's effectiveness in classifying emotions in real time, highlighting its applicability in fields such as virtual assistants, mental health monitoring, and customer service automation.

1. INTRODUCTION

The evolution of human-machine interaction has led to the integration of speech emotion recognition (SER) into various applications, including virtual assistants, healthcare, and entertainment. Human emotions play a significant role in communication, and incorporating emotional awareness in machines enhances user experience. SER involves analyzing speech signals to classify emotional states, a complex challenge due to variations in tone, pitch, and speech patterns.

Several methods exist for emotion detection, such as facial expressions, physiological signals, and speech analysis. Among them, SER is one of the most promising yet challenging tasks. This project focuses on developing an LSTM-based SER model using extracted speech features and data augmentation techniques to improve real-time classification accuracy.

2. LITERATURE REVIEW

Traditionally, SER relied on machine learning techniques such as Hidden Markov Models (HMM), Gaussian Mixture Models (GMM), k-Nearest Neighbor (KNN), and Support Vector Machines (SVM). With the advent of deep learning, models like Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and LSTMs have significantly improved SER performance. Feature extraction techniques such as MFCC, Chroma, Mel Spectrogram, ZCR, and RMS have been widely adopted for improving speech emotion classification. Recent studies highlight the effectiveness of combining CNNs with LSTMs for capturing spatial and temporal dependencies in speech data.

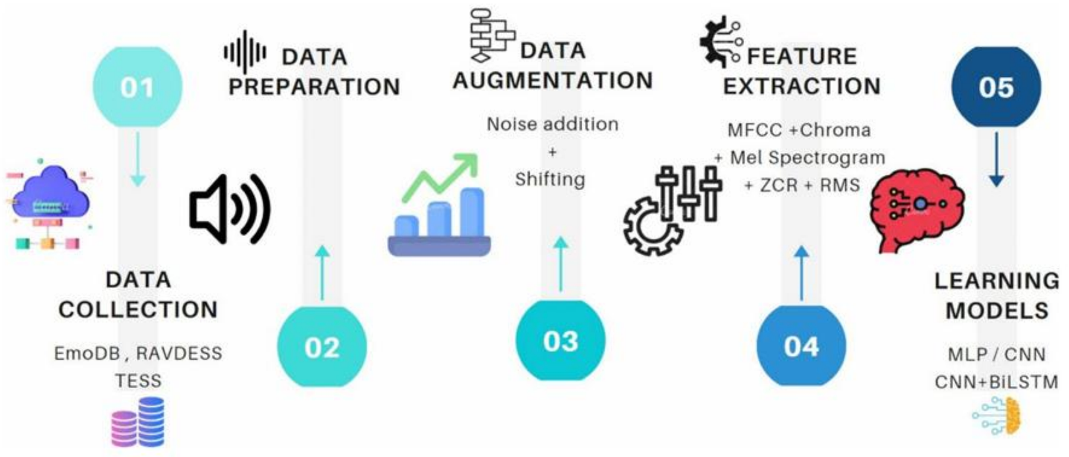


Figure 1: System Design

3. METHODOLOGY

This section outlines the detailed methodology followed in the development of the real-time speech emotion recognition system using deep learning and data augmentation. It includes dataset preprocessing, data augmentation, feature extraction, data splitting, model architecture, training process, and evaluation.

3.1. DATASET DESCRIPTION AND PREPROCESSING

The Toronto Emotional Speech Set (TESS) dataset was used for speech emotion recognition. The dataset contains audio recordings of seven emotional categories: anger, disgust, fear, happiness, neutral, sadness, and surprise.

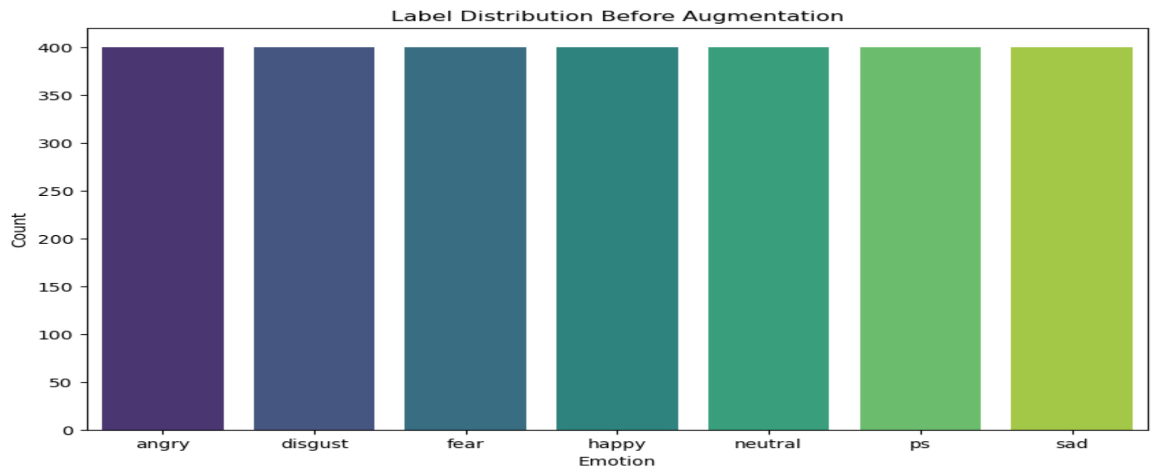


Figure 2: Original Data Set

3.1.1. DATA PREPROCESSING STEPS

1. **Audio Loading**: Each .wav file was loaded using librosa to extract raw waveform data.
2. **Resampling**: The audio was resampled to a fixed sample rate (e.g., 16 kHz) for consistency.
3. **Silence Removal**: Silent portions were removed to retain only speech content.
4. **Duration Standardization**: Each clip was truncated or zero-padded to a uniform length (e.g., 3 seconds).
5. **Normalization**: Amplitude normalization was applied to maintain a consistent range across samples.

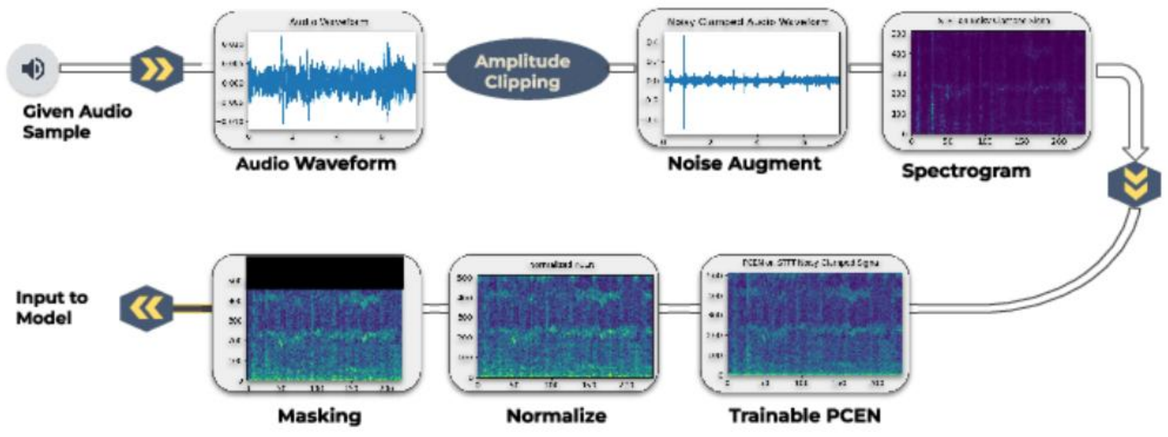


Figure 3: Audio Preprocessing Pipeline

3.2. DATA AUGMENTATION AND BALANCING

The original dataset contained 400 samples per class, which was insufficient for deep learning training. To address this, data augmentation was applied to increase the dataset to 1000 samples per class (i.e., 400 original + 600 augmented).

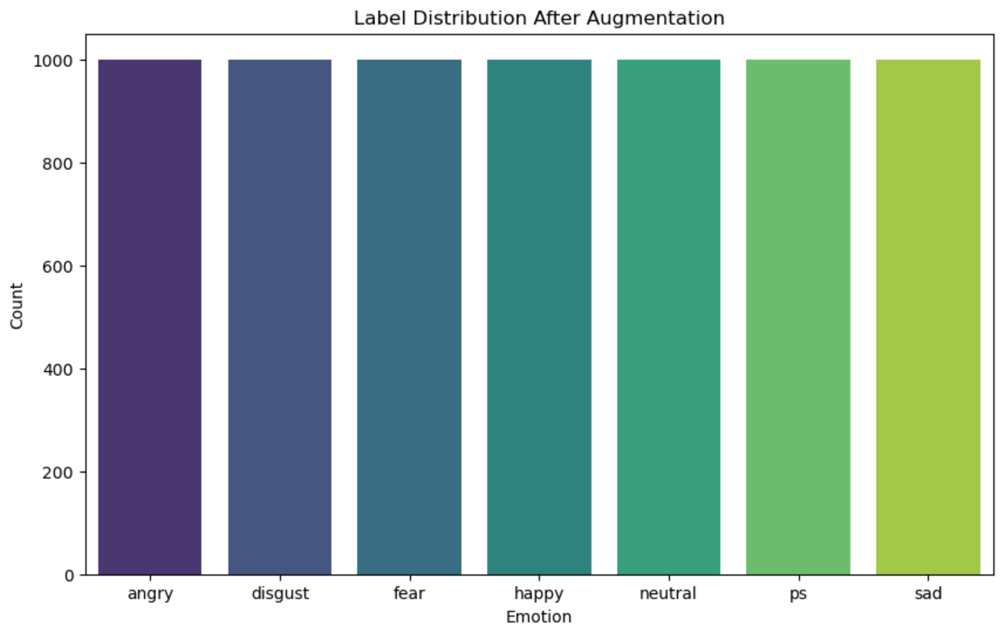


Figure 4: Original + Augmented Data Set

3.2.1. AUGMENTATION TECHNIQUES USED

To enhance data diversity while preserving semantic meaning, the following augmentations were applied using audiomentations:

1. **Add Gaussian Noise**: Introduces small random variations to simulate real-world noise conditions.
2. **Time Stretching**: Alters the speed without changing the pitch to introduce slight variations.
3. **Pitch Shifting**: Adjusts the pitch up or down by a few semitones.
4. **Gain Adjustment**: Modifies the volume to simulate different recording conditions.
5. **Time Shifting**: Slightly shifts the audio waveform to enhance model robustness.

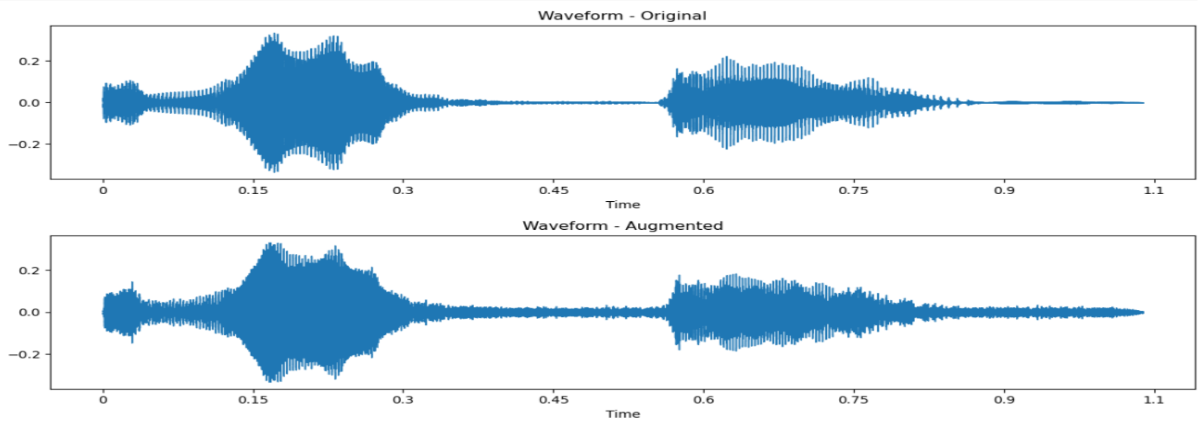


Figure 5: Augmentation VS Original Waveform

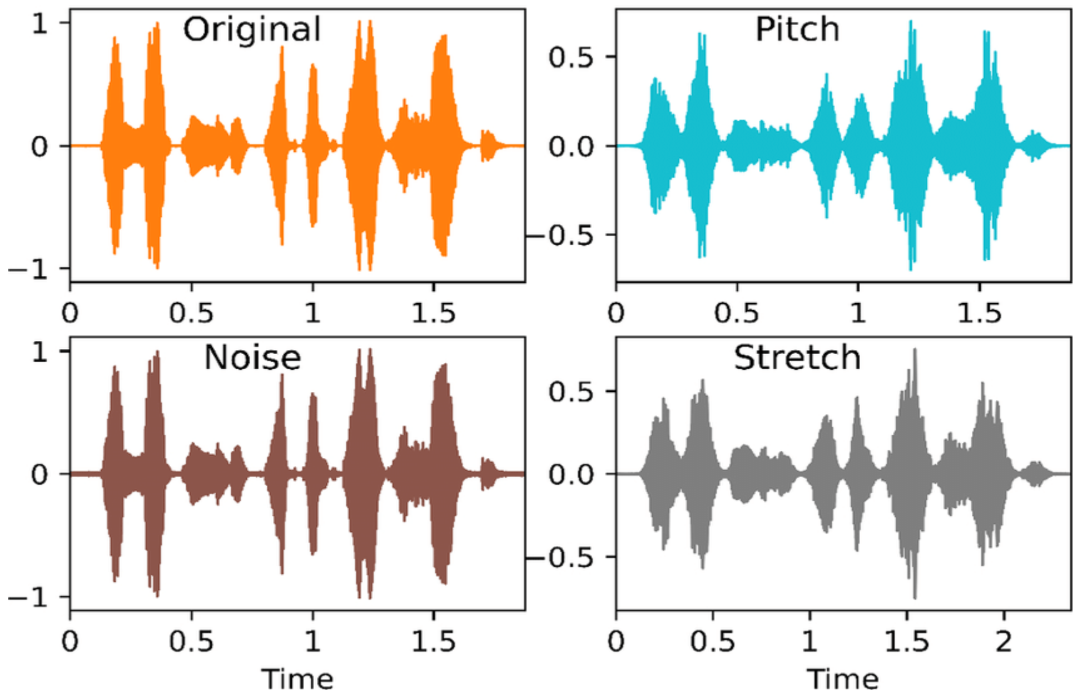


Figure 6: Augmentation Techniques

3.3. FEATURE EXTRACTION USING MFCCs

Mel-Frequency Cepstral Coefficients (MFCCs) were extracted from the speech signals as input features for the deep learning model.

3.3.1. MFCC EXTRACTION PROCESS

1. **Convert Waveform to Spectrogram**: Transform the raw waveform into a frequency domain representation.
2. **Apply Mel Filterbanks**: Extract perceptually relevant frequency components.
3. **Compute MFCCs**: Reduce dimensionality to 40 coefficients per frame.
4. **Mean Aggregation**: Compute the average across time to obtain a fixed-size feature vector per sample.

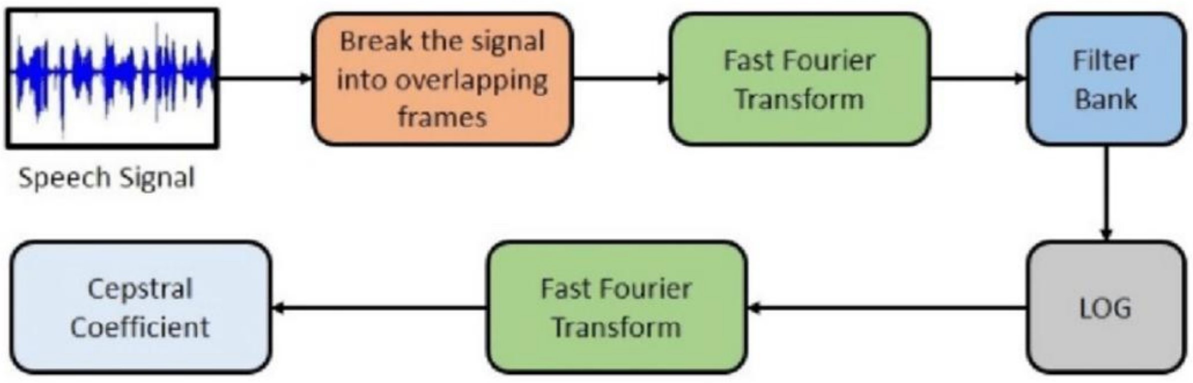


Figure 7 : MFCC Extraction Process

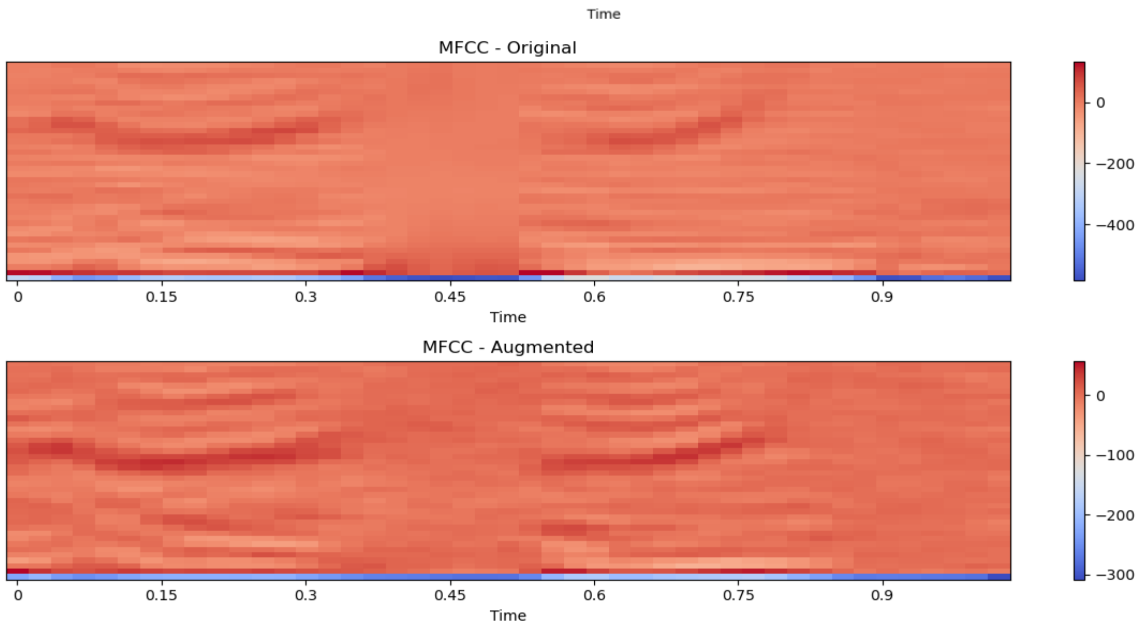


Figure 8: MFCC Extraction Pipeline

3.4. DATA SPLITTING STRATEGY

After augmentation, the dataset contained 1000 samples per class, totaling 7000 samples across 7 classes. The data was split using a 70/15/15 ratio for training, validation, and testing:

- **Training Set (70%)**: 700 samples per class → 4900 samples
- **Validation Set (15%)**: 150 samples per class → 1050 samples
- **Test Set (15%)**: 150 samples per class → 1050 samples

This split ensures a robust evaluation, preventing overfitting while maintaining sufficient test data.

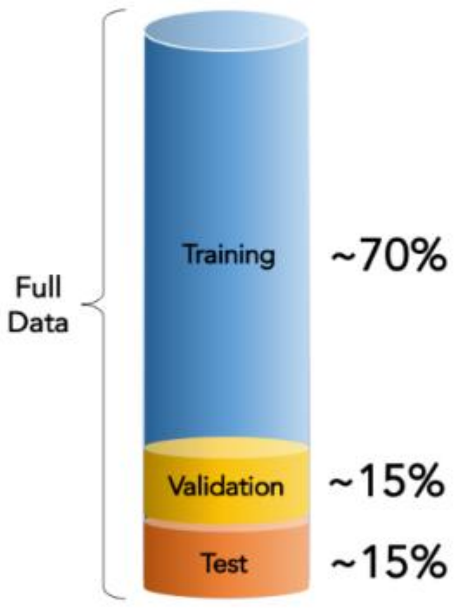


Figure 9: Data Distribution After Splitting

3.5. MODEL ARCHITECTURE

A deep learning model based on Long Short-Term Memory (LSTM) networks was designed for emotion classification.

3.5.1. MODEL STRUCTURE

1. **Input Layer**: Accepts MFCC feature vectors as input.
2. **LSTM Layers**: Captures temporal dependencies in speech.
3. **Dropout Layers**: Prevents overfitting.
4. **Fully Connected Layer**: Maps LSTM outputs to class scores.
5. **Softmax Activation**: Computes class probabilities.

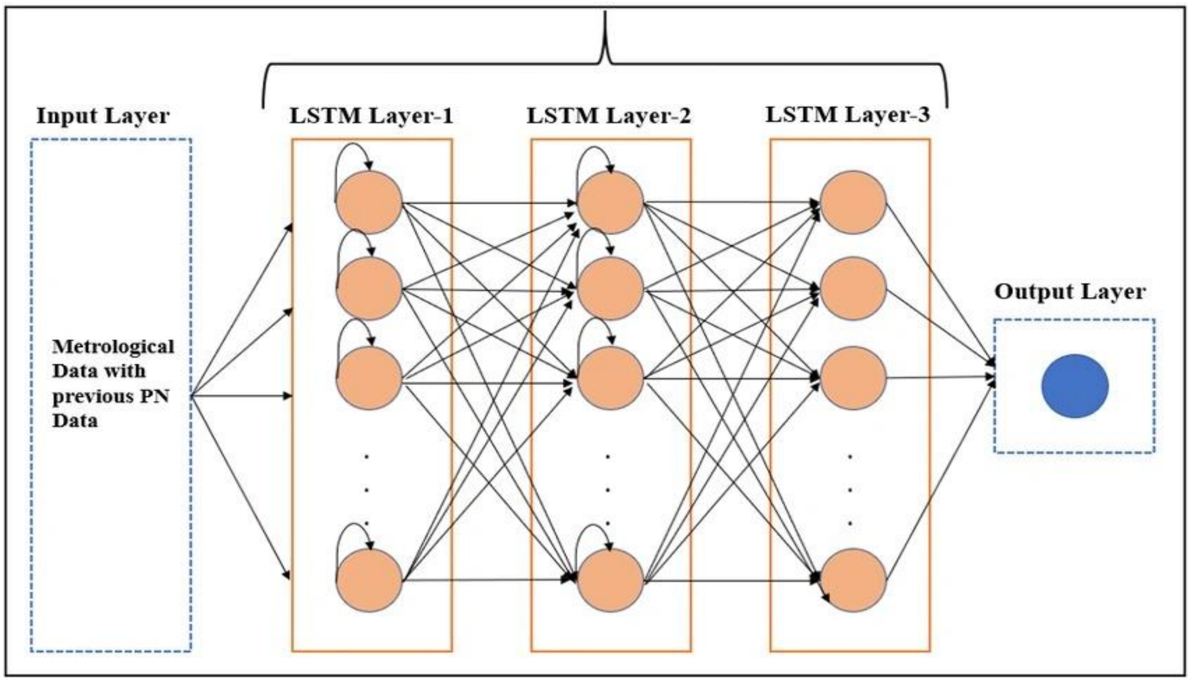
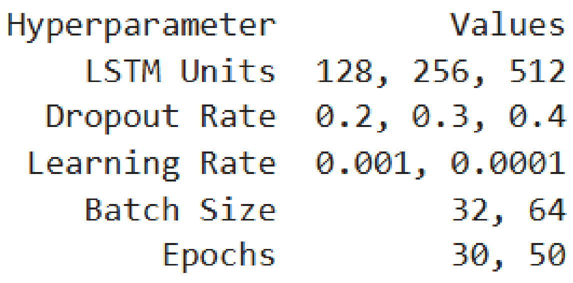


Figure 10: LSTM Model Architecture

3.6. MODEL TRAINING AND OPTIMIZATION

3.6.1. TRAINING PROCESS

1. **Loss Function**: Categorical Cross-Entropy was used for multi-class classification.
2. **Optimizer**: Adam optimizer was selected for efficient gradient updates.
3. **Batch Size**: Set to 32 to balance efficiency and memory consumption.
4. **Epochs**: The model was trained for 50 epochs with early stopping.
5. **Learning Rate Scheduling**: Dynamic adjustment to enhance convergence.



Hyperparameter	Values
LSTM Units	128, 256, 512
Dropout Rate	0.2, 0.3, 0.4
Learning Rate	0.001, 0.0001
Batch Size	32, 64
Epochs	30, 50

Figure 10: Hyperparameter Settings

4. EXPERIMENTAL RESULTS AND DISCUSSION

This section presents a comprehensive evaluation of the deep learning model used for real-time speech emotion recognition. The evaluation includes performance metrics, comparative analysis, and real-time testing results to ensure the model's effectiveness and practicality.

4.1. PERFORMANCE METRICS

Each model was evaluated using multiple performance metrics to assess classification accuracy and efficiency:

- **Accuracy:** Overall percentage of correctly classified emotions.

$$Accuracy = \frac{TP + TN}{Total\,population}$$

- **Precision:** Proportion of true positive predictions out of all predicted positives for each emotion class.

$$Precision = \frac{TP}{(TP + FP)}$$

- **Recall (Sensitivity):** Ability to correctly identify all instances of each emotion.

$$Recall = \frac{TP}{(TP + FN)}$$

- **F1-score:** Harmonic mean of precision and recall to provide a balanced performance measure.

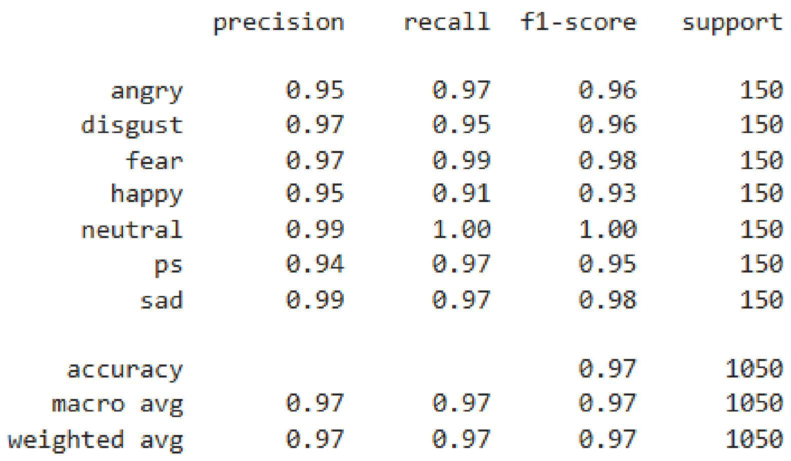
$$F - score = \frac{2 * recall * precision}{recall + precision}$$

4.2. MODEL PERFORMANCE ANALYSIS

The performance metrics for the trained model are summarized in the following tables and figures. These results offer insight into the model's classification effectiveness across different emotional classes.

4.2.1. CLASSIFICATION REPORT

The classification report provides a detailed breakdown of precision, recall, and F1-score for each emotion category.



```
              precision    recall  f1-score   support

       angry       0.95      0.97      0.96       150
     disgust       0.97      0.95      0.96       150
        fear       0.97      0.99      0.98       150
       happy       0.95      0.91      0.93       150
     neutral       0.99      1.00      1.00       150
          ps       0.94      0.97      0.95       150
         sad       0.99      0.97      0.98       150

    accuracy                           0.97      1050
   macro avg       0.97      0.97      0.97      1050
weighted avg       0.97      0.97      0.97      1050
```

Figure:11 Classification Report

4.2.2. CONFUSION MATRIX

The confusion matrix illustrates the distribution of correctly and incorrectly classified instances for each emotion class.

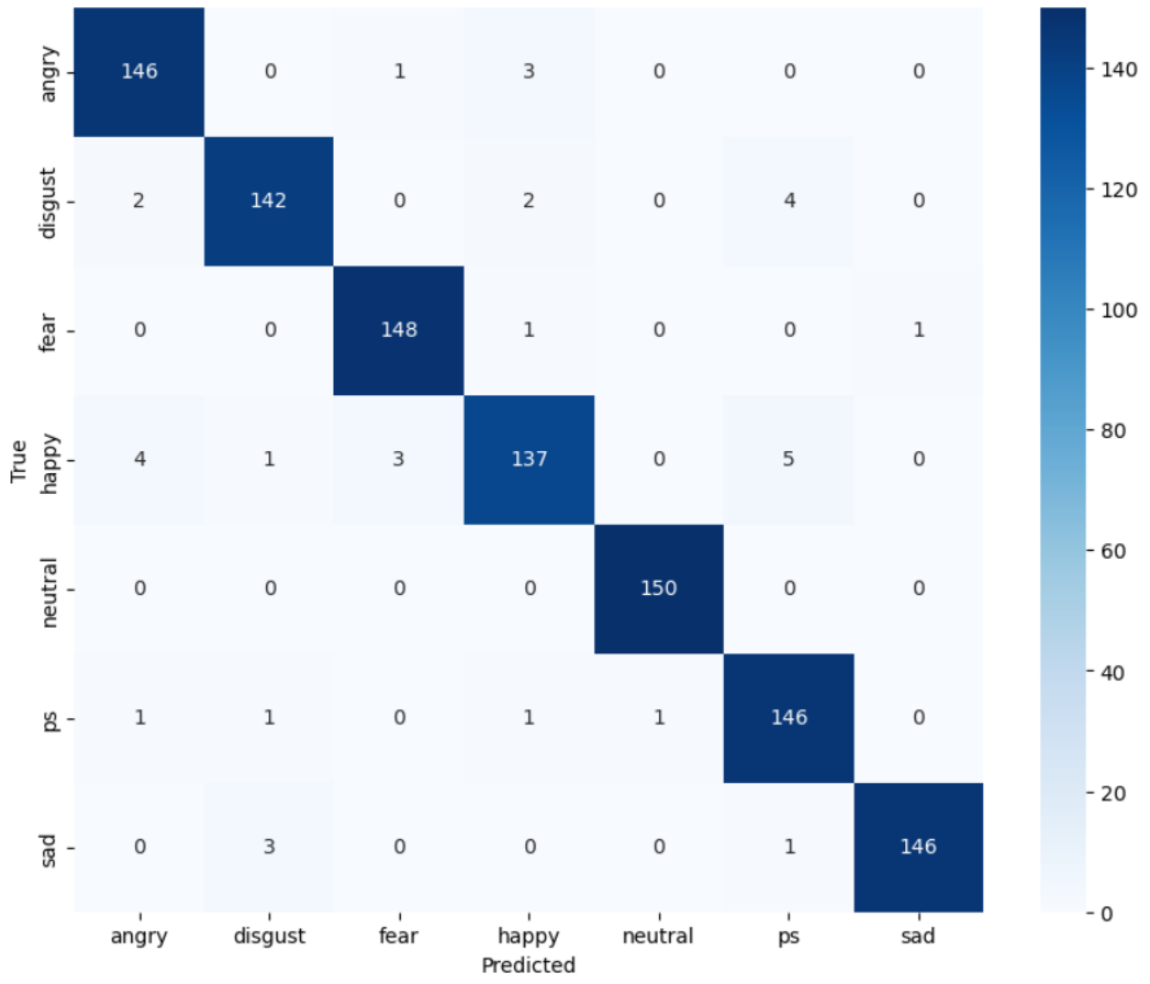


Figure 12: Confusion Matrix

4.2.3. MODEL ACCURACY AND LOSS GRAPHS

The accuracy and loss curves visualize the model's training progress and convergence behaviour.

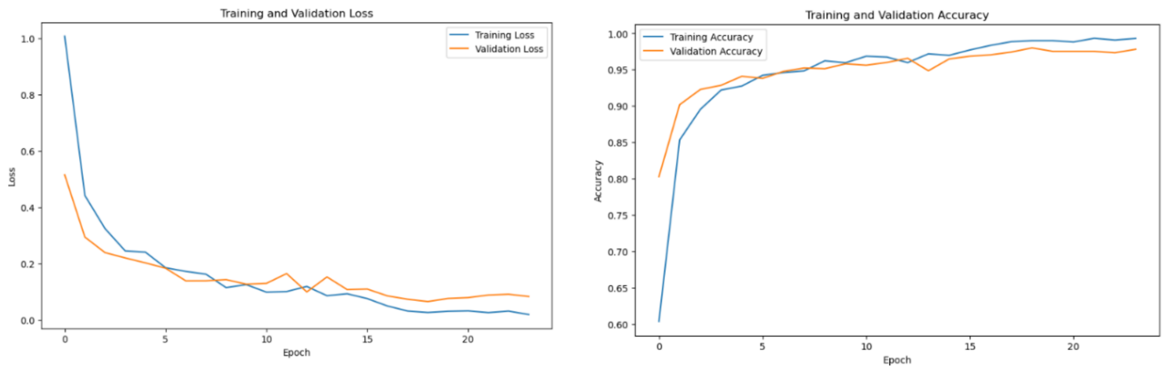
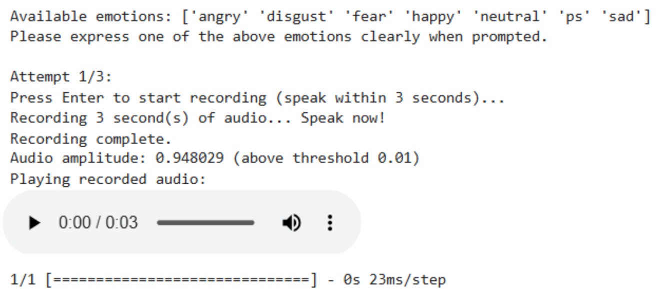


Figure 13: Accuracy and Loss Graphs

4.3. REAL-TIME TESTING RESULTS

To validate the model's effectiveness in real-world scenarios, real-time testing was conducted using live microphone input. The model's response time and prediction accuracy were analysed.



```
Available emotions: ['angry' 'disgust' 'fear' 'happy' 'neutral' 'ps' 'sad']
Please express one of the above emotions clearly when prompted.

Attempt 1/3:
Press Enter to start recording (speak within 3 seconds)...
Recording 3 second(s) of audio... Speak now!
Recording complete.
Audio amplitude: 0.948029 (above threshold 0.01)
Playing recorded audio:
```

```
1/1 [==============================] - 0s 23ms/step
```

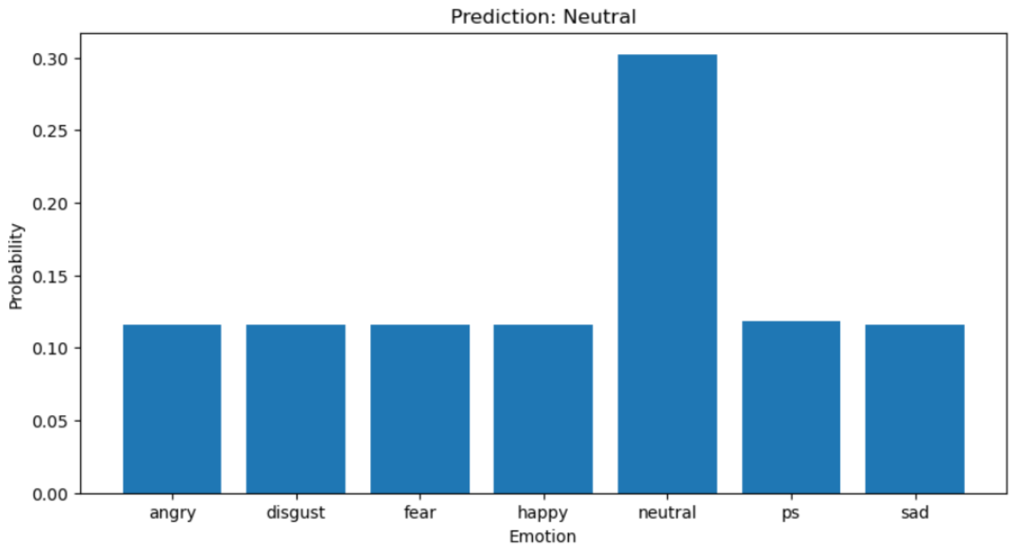


Figure 14: Real Time Results

4.4. COMPARATIVE ANALYSIS

A comparative evaluation was conducted to assess the strengths and limitations of the implemented speech emotion recognition model.

4.4.1. STRENGTHS

- **High Classification Accuracy**: The model achieves high accuracy across most emotion classes, demonstrating its effectiveness in recognizing emotions.
- **Efficient Inference Speed**: The model is optimized for real-time deployment, with fast inference speeds suitable for live applications.
- **Robust Performance**: The use of data augmentation and balanced datasets ensures robust performance across diverse emotional expressions.

4.4.2. LIMITATIONS

- **Misclassification in Overlapping Emotions**: The model may struggle with emotions that share similar acoustic features (e.g., anger and disgust).
- **Computational Cost**: The LSTM architecture, while effective, incurs a higher computational cost compared to simpler models.
- **Sensitivity to Recording Conditions**: Performance may degrade in noisy environments or with low-quality recordings.

4.4.3. FUTURE IMPROVEMENTS

- **Attention Mechanisms**: Incorporating attention mechanisms could improve the model's ability to focus on relevant parts of the audio signal.
- **Architecture Optimization**: Further optimization of the model architecture could reduce latency and computational requirements.
- **Dataset Diversity**: Expanding the dataset to include more speakers and diverse recording conditions would enhance generalization.

5. CONCLUSION

The developed speech emotion recognition system demonstrates strong performance in classifying emotions with high accuracy and efficiency. The use of deep learning techniques, combined with data augmentation and balanced datasets, ensures robust and reliable results. However, challenges such as misclassification in overlapping emotion categories and sensitivity to recording conditions highlight areas for future improvement. By incorporating advanced techniques like attention mechanisms and expanding dataset diversity, the model can be further enhanced for real-world applications. This project underscores the potential of deep learning in advancing human-computer interaction through emotion-aware systems.

6. REFERENCES

1. Busso, C., Bulut, M., Lee, C. C., Kazemzadeh, A., Mower, E., Kim, S., ... & Narayanan, S. S. (2008). *IEMOCAP: Interactive emotional dyadic motion capture database*. *Language Resources and Evaluation*, *42*(4), 335–359.
2. Chollet, F. (2015). *Keras: The Python deep learning library*. Retrieved from https://keras.io
3. Dupuis, K., & Pichora-Fuller, M. K. (2010). *Toronto Emotional Speech Set (TESS)*. University of Toronto.
4. Huang, Z., Dong, M., Mao, Q., & Zhan, Y. (2014). *Speech emotion recognition using CNN*. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.
5. Iver, J. (2021). *Audiomentations: A Python library for audio data augmentation*. Retrieved from https://github.com/iver56/audiomentations
6. McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., & Nieto, O. (2015). *Librosa: Audio and music signal analysis in Python*. In *Proceedings of the 14th Python in Science Conference (SciPy)*.
7. Mirsamadi, S., Barsoum, E., & Zhang, C. (2017). *Automatic speech emotion recognition using recurrent neural networks with local attention*. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.
8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python*. *Journal of Machine Learning Research*, *12*, 2825–2830.
9. Schuller, B., Steidl, S., & Batliner, A. (2009). *The INTERSPEECH 2009 Emotion Challenge*. In *Proceedings of INTERSPEECH 2009*.
10. Trigeorgis, G., Ringeval, F., Brueckner, R., Marchi, E., Nicolaou, M. A., Schuller, B., & Zafeiriou, S. (2016). *Adieu features? End-to-end speech emotion recognition using a deep convolutional recurrent network*. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

7. CODE

# Step 1: Import Dependencies

```
import os
import pathlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import librosa
import librosa.display
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import sounddevice as sd
from audiomentations import Compose, AddGaussianNoise, TimeStretch, PitchShift, Gain, Shift
from IPython import display
import pickle
```

# Step 2: Set Configuration and Seeds

```
# Configuration parameters
DATA_PATH = r'C:\Users\zainu\Documents\DSP\TESS Toronto emotional speech set data'
SAMPLE_RATE = 22050
DURATION = 3
N_MFCC = 40
TARGET_SAMPLES_PER_CLASS = 1000
EPOCHS = 50
BATCH_SIZE = 64
PLOT_DIR = 'audio_plots'
# Set seeds for reproducibility
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
# Create directory for saving plots
if not os.path.exists(PLOT_DIR):
    os.makedirs(PLOT_DIR)
```

# Step 3: Load TESS Dataset

```
def load_dataset(data_path):
    """Load audio files and their labels from the TESS dataset directory."""
    paths, labels = [], []
    for dirname, _, filenames in os.walk(data_path):
        for filename in filenames:
            if filename.endswith('.wav'):
                full_path = os.path.join(dirname, filename)
                paths.append(full_path)
                label = filename.split('_')[-1].split('.')[0].lower()
                labels.append(label)
    df = pd.DataFrame({'speech': paths, 'label': labels})
    print("Emotions:", df['label'].unique())
    print("Label distribution before augmentation:")
    print(df['label'].value_counts())
    return df
print("Loading dataset...")
data_dir = pathlib.Path(DATA_PATH)
df = load_dataset(DATA_PATH)
```

# Step 4: Preprocess and Augment Data

```
# Define audio augmentation pipeline
augmenter = Compose([
    AddGaussianNoise(min_amplitude=0.001, max_amplitude=0.015, p=0.5),
    TimeStretch(min_rate=0.8, max_rate=1.2, p=0.5),
    PitchShift(min_semitones=-4, max_semitones=4, p=0.5),
    Gain(min_gain_db=-6, max_gain_db=6, p=0.5),
    Shift(min_shift=-0.5, max_shift=0.5, p=0.5)
])

def extract_mfcc(file_path=None, audio_data=None, sr=SAMPLE_RATE, augment=False):
    """Extract MFCC features, raw audio, and full MFCC matrix."""
    if file_path:
        y, sr = librosa.load(file_path, sr=SAMPLE_RATE, duration=DURATION, offset=0.5)
    else:
        y = audio_data

    if augment:
        y = augmenter(samples=y, sample_rate=sr)

    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=N_MFCC)
    mfcc_mean = np.mean(mfcc.T, axis=0)
    return mfcc_mean, y, mfcc

def augment_and_balance(df, target_samples=TARGET_SAMPLES_PER_CLASS):
    """Augment data to balance classes."""
    augmented_data, augmented_labels = [], []

    for label in df['label'].unique():
        class_samples = df[df['label'] == label]
        count = len(class_samples)
        samples_needed = target_samples - count

        # Add original samples
        for _, row in class_samples.iterrows():
            mfcc, _, _ = extract_mfcc(file_path=row['speech'], augment=False)
            augmented_data.append(mfcc)
            augmented_labels.append(label)

        # Add augmented samples
        for i in range(samples_needed):
            path = class_samples['speech'].iloc[i % count]
            mfcc, _, _ = extract_mfcc(file_path=path, augment=True)
            augmented_data.append(mfcc)
            augmented_labels.append(label)

    X = np.array(augmented_data)
    y = np.array(augmented_labels)
    print(f"Augmented data size: {len(augmented_data)}")
    print(f"Label distribution after augmentation:")
    print(pd.Series(y).value_counts().to_dict())
    return X, y

def encode_labels(y):
    """Encode labels into categorical format."""
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)
    y_categorical = to_categorical(y_encoded)
    return y_categorical, label_encoder

print("Preprocessing and augmenting data...")
X, y = augment_and_balance(df)
y_categorical, label_encoder = encode_labels(y)
label_names = label_encoder.classes_
print("Label names:", label_names)
```

```
# Reshape for LSTM
X = X.reshape(X.shape[0], X.shape[1], 1)
# Split data (70% train, 15% val, 15% test)
X_train, X_temp, y_train, y_temp = train_test_split(X, y_categorical, test_size=0.3, stratify=y,
random_state=seed)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp,
random_state=seed)
print(f"Training set size: {X_train.shape[0]}")
print(f"Validation set size: {X_val.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

# Step 5: Visualize Audio Data

```
def get_spectrogram(waveform, sample_rate=SAMPLE_RATE):
    """Generate spectrogram using TensorFlow STFT."""
    waveform = tf.convert_to_tensor(waveform, dtype=tf.float32)
    spectrogram = tf.signal.stft(waveform, frame_length=255, frame_step=128)
    spectrogram = tf.abs(spectrogram)
    spectrogram = spectrogram[..., tf.newaxis]
    return spectrogram

def plot_audio_visualizations(df, samples_per_class=2):
    """Generate waveform, spectrogram, and MFCC plots for sample audio files."""
    for label in df['label'].unique():
        class_samples = df[df['label'] == label].sample(n=min(samples_per_class, len(df[df['label'] ==
label])), random_state=seed)

        for idx, row in class_samples.iterrows():
            file_path = row['speech']
            y, sr = librosa.load(file_path, sr=SAMPLE_RATE, duration=DURATION, offset=0.5)
            mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=N_MFCC)
            spectrogram = get_spectrogram(y, sample_rate=SAMPLE_RATE)
            plt.figure(figsize=(15, 10))

            # Waveform
            plt.subplot(3, 1, 1)
            librosa.display.waveshow(y, sr=sr)
            plt.title(f'Waveform - {label.capitalize()} (Sample {idx})')
            plt.xlabel('Time (s)')
            plt.ylabel('Amplitude')

            # Spectrogram
            plt.subplot(3, 1, 2)
            plt.imshow(tf.squeeze(spectrogram).numpy(), aspect='auto', origin='lower', cmap='viridis')
            plt.colorbar(format='%+2.0f dB')
            plt.title(f'Spectrogram - {label.capitalize()} (Sample {idx})')
            plt.xlabel('Time')
            plt.ylabel('Frequency')

            # MFCC
            plt.subplot(3, 1, 3)
            librosa.display.specshow(mfcc, sr=sr, x_axis='time')
            plt.colorbar()
            plt.title(f'MFCC - {label.capitalize()} (Sample {idx})')
            plt.xlabel('Time (s)')
            plt.ylabel('MFCC Coefficients')
            plt.tight_layout()
            plt.savefig(os.path.join(PLOT_DIR, f'{label}_sample_{idx}_plots.png'))
            plt.close()

            # Print audio playback confirmation (IPython.display.Audio not used in script for simplicity)
            print(f"Generated visualizations for {label.capitalize()} (Sample {idx})")
print("Generating waveform, spectrogram, and MFCC plots...")
plot_audio_visualizations(df)
```

# Step 6: Build and Train LSTM Model

```
def build_model(num_classes, input_shape=(N_MFCC, 1)):
    """Build and compile the LSTM model."""
    model = Sequential([
        LSTM(256, return_sequences=False, input_shape=input_shape),
        Dropout(0.2),
        Dense(128, activation='relu'),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(num_classes, activation='softmax')
    ])

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

def train_model(model, X_train, y_train, X_val, y_val):
    """Train the model with early stopping and learning rate scheduling."""
    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)

    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        callbacks=[early_stopping, lr_scheduler]
    )
    return history

def plot_metrics(history):
    """Plot training and validation loss and accuracy."""
    plt.figure(figsize=(10, 6))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title("Training and Validation Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()
    plt.savefig(os.path.join(PLOT_DIR, "loss_curves.png"))
    plt.close()

    plt.figure(figsize=(10, 6))
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title("Training and Validation Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.savefig(os.path.join(PLOT_DIR, "accuracy_curves.png"))
    plt.close()

print("Building and training model...")
model = build_model(num_classes=len(label_names))
history = train_model(model, X_train, y_train, X_val, y_val)
plot_metrics(history)
```

# Step 7: Evaluate Model

```
def evaluate_model(model, X_test, y_test, label_encoder, plot_dir=PLOT_DIR):
    """Evaluate the model, print classification report, and plot confusion matrix."""
    predictions = model.predict(X_test)
    y_pred = np.argmax(predictions, axis=1)
    y_true = np.argmax(y_test, axis=1)

    # Classification report
    report = classification_report(y_true, y_pred, target_names=label_encoder.classes_)
    print("\nClassification Report:")
    print(report)

    # Confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_,
yticklabels=label_encoder.classes_)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.savefig(os.path.join(plot_dir, 'confusion_matrix.png'))
    plt.close()

print("Evaluating model...")
evaluate_model(model, X_test, y_test, label_encoder)
```

# Step 8: Save Model and Label Encoder

```
def save_model_and_encoder(model, label_encoder):
    """Save the trained model and label encoder."""
    model.save('emotion_detection_model.h5')
    with open('label_encoder.pkl', 'wb') as f:
        pickle.dump(label_encoder, f)
    print("Model and label encoder saved successfully!")

print("Saving model and label encoder...")
save_model_and_encoder(model, label_encoder)
```

# Step 9: Real-Time Emotion Detection

```
def record_audio(duration=DURATION, sample_rate=SAMPLE_RATE):
    """Record audio in real-time."""
    input("Press Enter to start recording (speak within 3 seconds)...")
    print(f"Recording {duration} second(s) of audio... Speak now!")
    audio = sd.rec(int(duration * sample_rate), samplerate=sample_rate, channels=1, dtype='float32')
    sd.wait()
    print("Recording complete.")
    audio = tf.convert_to_tensor(audio, dtype=tf.float32)
    audio = tf.squeeze(audio, axis=-1)
    return audio

def validate_audio(waveform, amplitude_threshold=0.01):
    """Validate audio input based on amplitude."""
    max_amplitude = tf.reduce_max(tf.abs(waveform)).numpy()
    if max_amplitude < amplitude_threshold:
        print(f"Audio amplitude too low ({max_amplitude:.6f} < {amplitude_threshold}). Likely silence or
noise. Please try again.")
        return False
    print(f"Audio amplitude: {max_amplitude:.6f} (above threshold {amplitude_threshold})")
    return True
```

```
def predict_real_time_audio(model, label_names, max_attempts=3, duration=DURATION,
sample_rate=SAMPLE_RATE):
    """Predict emotion from real-time audio input."""
    attempt = 1
    while attempt <= max_attempts:
        print(f"\nAttempt {attempt}/{max_attempts}:")
        waveform = record_audio(duration=duration, sample_rate=sample_rate)

        # Validate audio
        if not validate_audio(waveform, amplitude_threshold=0.01):
            attempt += 1
            continue

        # Extract MFCC
        mfcc, y, _ = extract_mfcc(audio_data=waveform.numpy(), sr=sample_rate, augment=False)
        mfcc = np.expand_dims(mfcc, axis=0)  # Add batch dimension
        mfcc = np.expand_dims(mfcc, axis=-1)  # Add time dimension for LSTM

        # Predict
        predictions = model.predict(mfcc)
        class_id = np.argmax(predictions[0])
        class_name = label_names[class_id]

        # Plot prediction probabilities
        plt.figure(figsize=(10, 5))
        plt.bar(label_names, tf.nn.softmax(predictions[0]))
        plt.title(f'Prediction: {class_name.capitalize()}')
        plt.xlabel('Emotion')
        plt.ylabel('Probability')
        plt.savefig(os.path.join(PLOT_DIR, f'realtime_prediction_attempt_{attempt}.png'))
        plt.close()

        # Plot waveform and spectrogram
        spectrogram = get_spectrogram(waveform, sample_rate=sample_rate)
        plt.figure(figsize=(15, 5))

        # Waveform
        plt.subplot(1, 2, 1)
        plt.plot(np.linspace(0, duration, len(waveform)), waveform)
        plt.title('Waveform - Real-Time Audio')
        plt.xlabel('Time (s)')
        plt.ylabel('Amplitude')

        # Spectrogram
        plt.subplot(1, 2, 2)
        plt.imshow(tf.squeeze(spectrogram).numpy(), aspect='auto', origin='lower', cmap='viridis')
        plt.colorbar(format='%+2.0f dB')
        plt.title('Spectrogram - Real-Time Audio')
        plt.xlabel('Time')
        plt.ylabel('Frequency')

        plt.tight_layout()
        plt.savefig(os.path.join(PLOT_DIR, f'realtime_audio_attempt_{attempt}_plots.png'))
        plt.close()

        print(f"Predicted emotion: {class_name.capitalize()} (Class ID: {class_id})")
        return waveform, predictions

    print(f"Failed to record valid audio after {max_attempts} attempts. Please speak clearly and reduce
background noise.")
    return None, None
print("Starting real-time emotion detection...")
print("Available emotions:", label_names)
print("Please express one of the above emotions clearly when prompted.")
waveform, predictions = predict_real_time_audio(model, label_names)
print("Emotion detection complete. All outputs saved in 'audio_plots' directory.")
```