Project Report

This report is submitted in partial fulfillment of the
requirements for
the award of the degree of

The Professional Master Of
Embedded And Cyber-Physical Systems

ECPS-211: Machine Learning and Data Mining

Speech Emotion Recognition Using Deep Learning

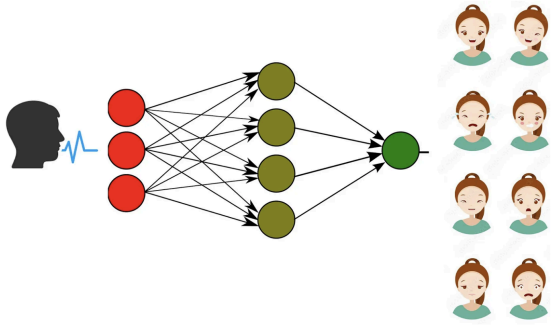
{Comparative Analysis of CNN, LSTM, and Transformer Models}

Under the guidance of
Prof. Marco Levorato

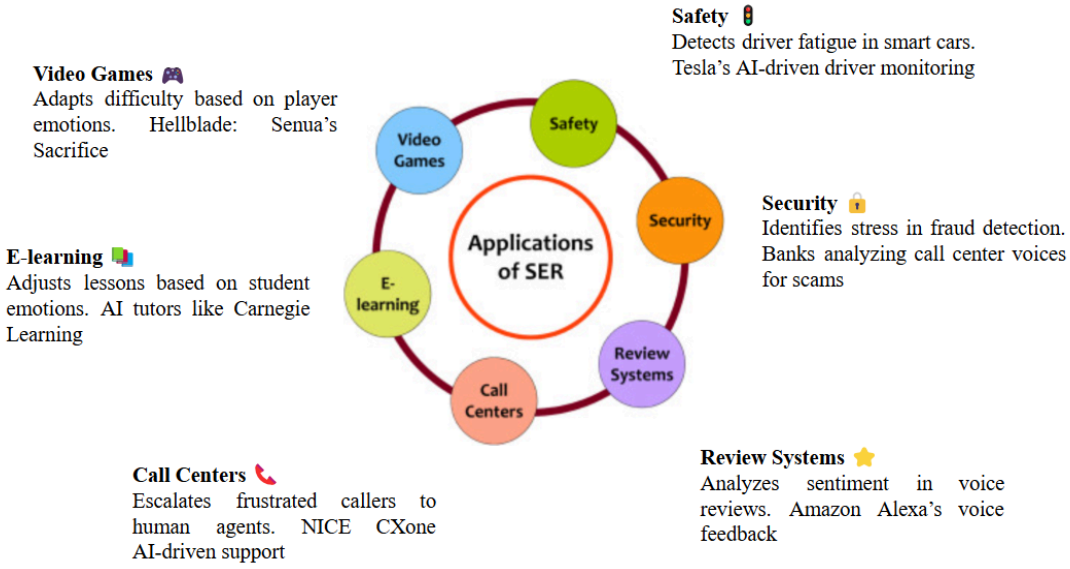
Name of Candidates	UCI Student ID
Sahil Mukeshbhai Kakadiya	93282494
Vaibhav Ashokbhai Gajera	89125790
Prateeksha Ranjan	28112912

❖ Project Description

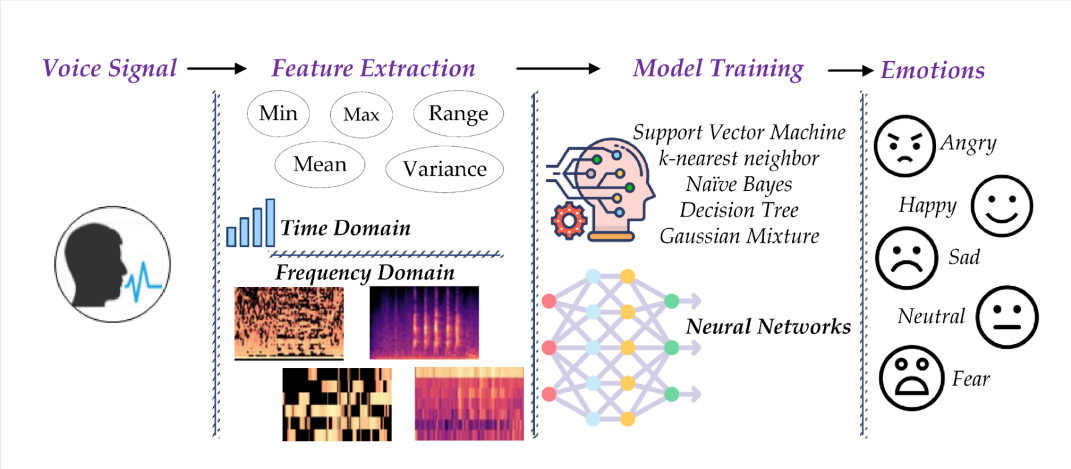
- This project focuses on Speech Emotion Recognition using deep learning techniques to classify emotions from audio speech data. The RAVDESS dataset is used for training and evaluation. The audio data is preprocessed and then the extraction of different features like: MFCC features to capture important speech characteristics.
- A baseline model using LSTM, CNN, and a Combination of both is implemented to classify emotions such as happy, sad, and angry. Additionally, a transformer-based model (e.g., Wav2Vec 2.0) available on Facebook-Hugging Face is explored to improve recognition accuracy. The models are trained and evaluated based on standard performance metrics, and their results are visualized for comparison.
- This study aims to analyze the effectiveness of CNN, LSTM, and transformer architectures for SER, highlighting the trade-offs in accuracy and computational efficiency.



❖ Applications of SER



❖ How Speech Emotion Recognition (SER) Works?



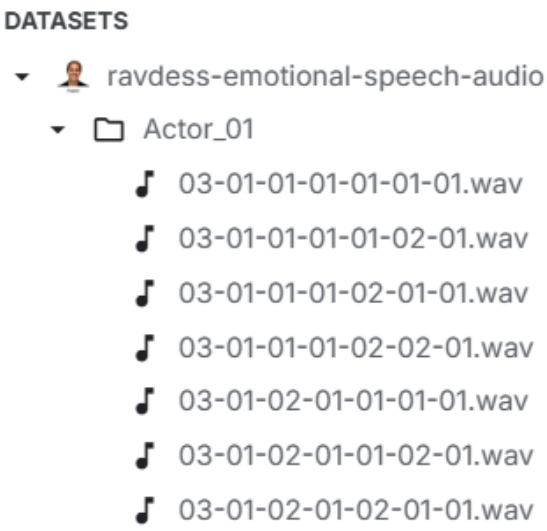
❖ Software/Library Requirements

- **Librosa** – For audio processing, feature extraction, and signal analysis.
- **NumPy & SciPy** – For numerical computations and handling multi-dimensional arrays.
- **Matplotlib & Seaborn** – For data visualization and plotting statistical insights.
- **Scikit-learn** – For feature extraction, preprocessing, dimensionality reduction, and classical ML models (e.g., Random Forest).
- **TensorFlow/Keras** – For building and training deep learning models like CNNs, LSTMs, and sequential networks.
- **Transformers (Hugging Face)** – For using pre-trained Wav2Vec2 models in speech processing tasks.
- **Torch (PyTorch)** – For deep learning and tensor computations.
- **IPython.display** – For audio playback within Jupyter notebooks.
- **OS & JSON** – For file handling, directory management, and metadata processing.

❖ Dataset Used

RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song)

- **Total Files:** 1,440 speech samples
- **Actors:** 24 (12 male, 12 female)
- **Emotions:** Neutral, Calm, Happy, Sad, Angry, Fearful, Disgust, Surprised
- **Audio Format**: 16-bit, 48kHz .wav files
- File Naming Convention:
  - **Example**: 03-01-06-01-02-01-12.wav
  - Each component represents different aspects such as modality, emotion, intensity, statement, and actor ID.



- **Filename Format (e.g., 03-01-06-01-02-01-12.wav)**:
  - **03** = Audio-only, **01** = Speech, **06** = Fearful, **01** = Normal intensity, **02** = “Dogs”, **01** = 1st repetition, **12** = Female actor.
- **Use Cases**: Emotion classification, ML/DL experiments, speech emotion analysis.

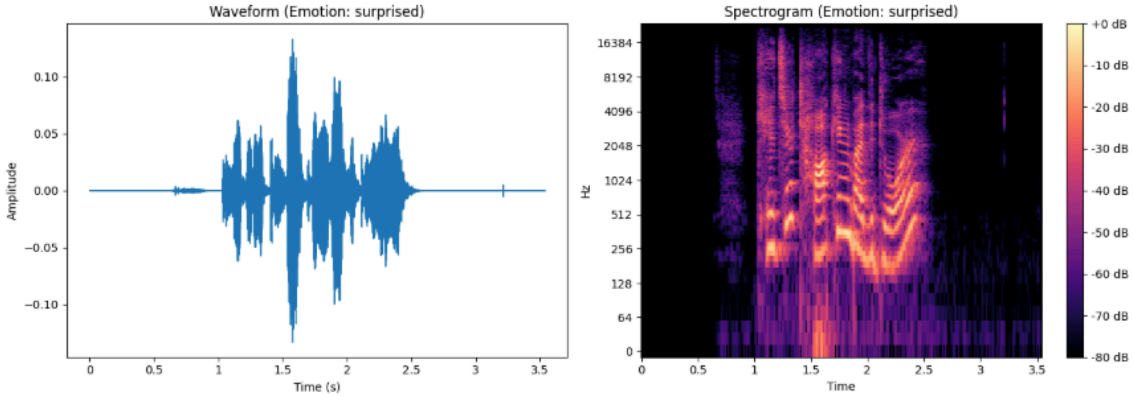
❖ Experimental Tasks

1. Data Preprocessing

To improve model performance, raw audio data must be preprocessed before feature extraction and model training. The following steps were performed:

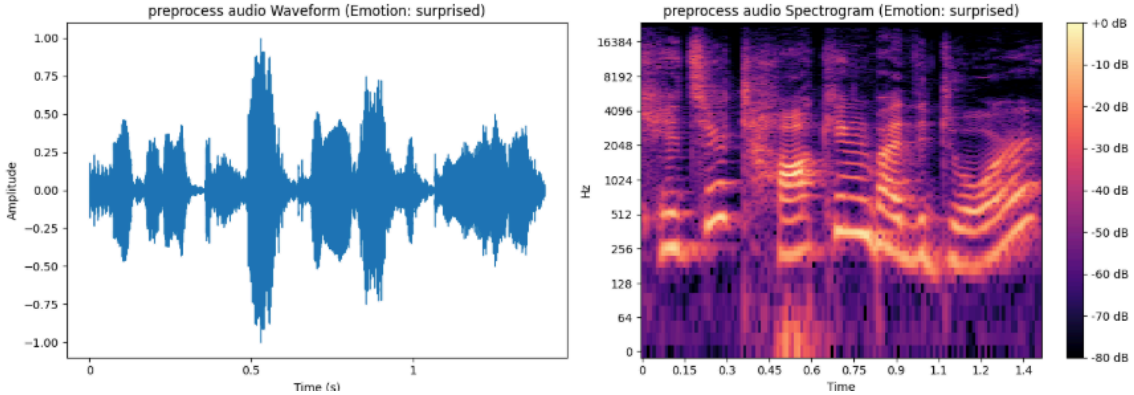
Step 1: Audio Visualization

- Plotted waveforms and spectrograms to analyze raw audio signals.
- Observed silent segments and variations in amplitude among samples.



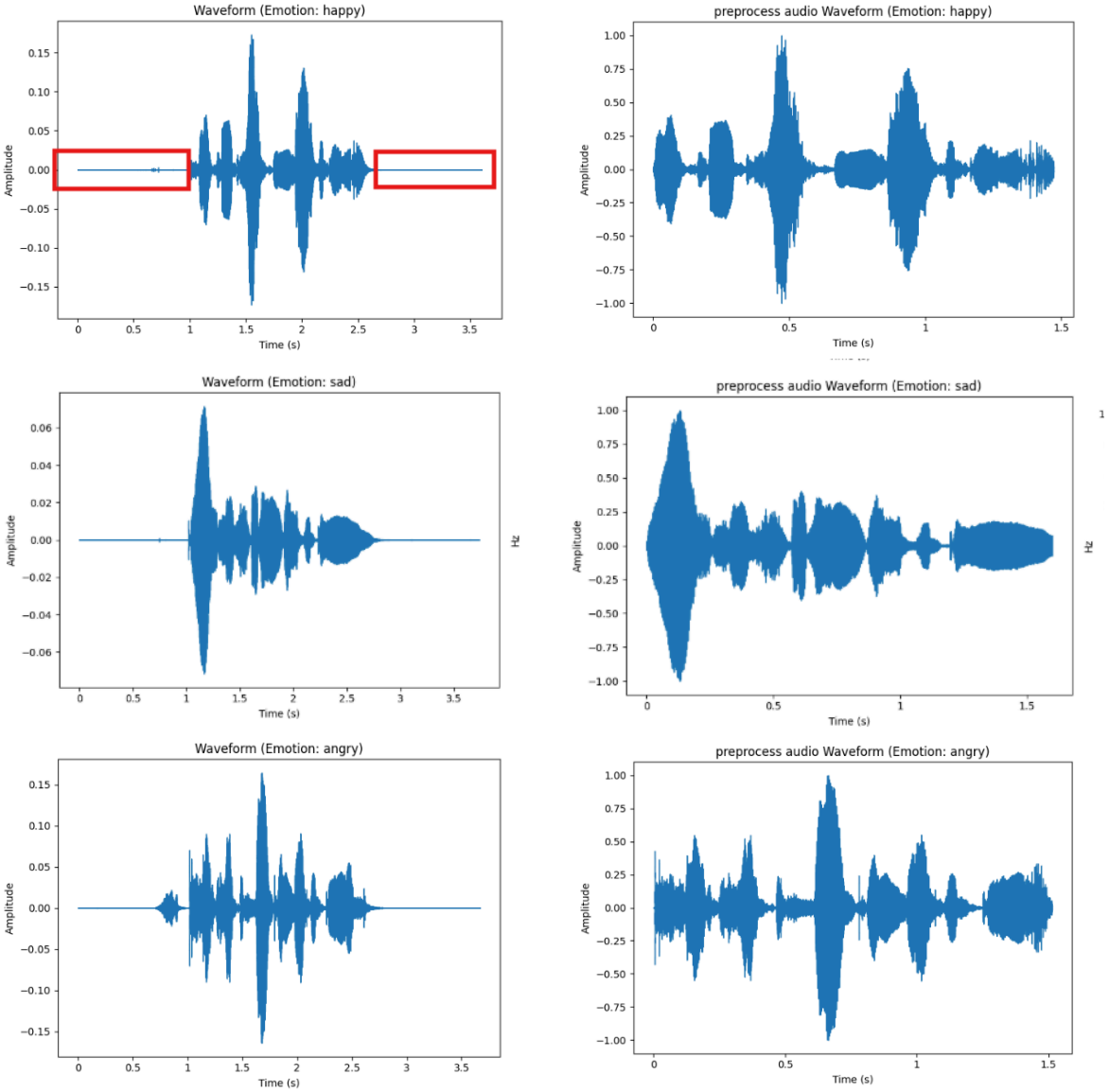
Step 2: Audio Preprocessing

- Trimming Silence: Used librosa.effects.trim() to remove silent parts of the audio, applying a 20 dB threshold.
- Normalization: Applied librosa.util.normalize() to standardize amplitude across samples, preventing bias due to loudness variations.
- Resampling: Converted all audio files to a common sampling rate of 16kHz to match deep learning model requirements.



➢ Benefits of Audio Preprocessing

- **Cleaner Waveforms**
  - Removes silent gaps, resulting in continuous speech signals that improve feature extraction accuracy.
- **More Compact Spectrograms**
  - Refines audio, highlighting key features for emotion classification, making the spectrograms more focused and relevant.
- **Reduced Data Size**
  - Minimizes data size while retaining essential speech features, boosting model efficiency without losing critical information.



2. Feature Extraction

➢ Extracted meaningful features to train models effectively:

- **MFCC (Mel-Frequency Cepstral Coefficients):** Captures the short-term spectral characteristics of speech by mimicking human auditory perception.
- **Delta & Delta-Delta MFCCs:** Represent the first and second derivatives of MFCCs, capturing speech dynamics and changes over time.
- **Mel Spectrogram:** A time-frequency representation showing speech energy distribution across frequency bands, useful for deep learning models.

```
# ------------------------------------------------------------------------------------ #
    def preprocess_audio(self, audio_path, target_sr=16000):
        # Load audio
        audio, sr = librosa.load(audio_path, sr=target_sr)
        # Trim silence
        trimmed_audio, _ = librosa.effects.trim(audio, top_db=20)
        # Normalize
        normalized_audio = librosa.util.normalize(trimmed_audio)
        return normalized_audio, sr
# ------------------------------------------------------------------------------------ #
    # MFCC Extraction
    def extract_mfcc_features(self, file_path, resampling_freq=16000, n_mfcc=13, start_time=0.5, end_time=3):
        preprocessed_audio, sr = self.preprocess_audio(file_path, resampling_freq)

        # Extract MFCCs
        mfccs = librosa.feature.mfcc(y=preprocessed_audio, sr=sr, n_mfcc=n_mfcc)

        # Calculate Delta and Delta-Delta
        delta_mfccs = librosa.feature.delta(mfccs)
        delta_delta_mfccs = librosa.feature.delta(mfccs, order=2)

        # Mel spectrogram
        mel_spec = librosa.feature.melspectrogram(y=preprocessed_audio, sr=sr, n_mels=40)
        mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)

        # Concatenate features
        combined_features = np.concatenate([np.mean(mfccs, axis=1), np.mean(delta_mfccs, axis=1),
                                            np.mean(delta_delta_mfccs, axis=1), np.mean(mel_spec_db, axis=1)])

        return combined_features
```

```
# Choose feature type ('MFCC' or 'WAV2VEC')
feature_type = "MFCC-48000-40"
MFCC_X, MFCC_y = data_prep.load_dataset(feature_type

print(f"\nMFCC_X shape: {MFCC_X.shape}")
print(f"MFCC_y shape: {MFCC_y.shape}")
```

```
Method: MFCC
Sample Rate: 48000
Number of Coefficients: 40

🔍 Scanning dataset directory: /kaggle/input/ravdess-emot

✅ Found 1440 valid audio files.

Extracting the MFCC type feature with sample_rate: 48000
```

```
✅ Dataset Extracted and Loaded Successfully!
Total audio files processed: 1440
Emotion distribution:
  surprised: 192 files
  neutral: 96 files
  disgust: 192 files
  fearful: 192 files
  sad: 192 files
  calm: 192 files
  happy: 192 files
  angry: 192 files

MFCC_X shape: (1440, 160)
MFCC_y shape: (1440,)
```

3. Focal LOSS Function:

[3]:
```
focal_loss(alpha=0.25, gamma=2.0):
    def loss_fn(y_true, y_pred):
        cce = SparseCategoricalCrossentropy(reduction=tf.keras.losses.Reduction.NONE)
        cross_entropy = cce(y_true, y_pred)
        pt = tf.exp(-cross_entropy)
        focal_weight = alpha * (1 - pt) ** gamma
        focal_loss = focal_weight * cross_entropy
        return tf.reduce_mean(focal_loss)
    return loss_fn
```

- **Defines a focal loss function** to handle class imbalance in classification tasks.
- **Uses SparseCategoricalCrossentropy loss** to compute cross-entropy for integer-labeled classes.
- **Computes pt = exp(-cross_entropy)**, which represents the model's confidence in the correct class.
- **Applies the focal weighting factor**: (1−pt)** γ to down-weight easy examples and focus more on hard examples.
- **Scales the loss by alpha**, which adjusts the importance of positive/negative classes.

4. Model Training & Optimization:

- **Optimization Strategies**
  - **ReduceLROnPlateau:** Dynamically adjusts learning rate for convergence.
  - **Early Stopping:** Stops training when validation loss plateaus to prevent overfitting.

- **Training Process**
  - Compiled with **accuracy** as the metric.
  - **Adaptive learning** using callbacks for better generalization.

```
# ------------------------------------------------------------------------------------------
 # Define ReduceLROnPlateau callback to adjust learning rate dynamically
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',    # Monitor validation loss
    factor=0.5,            # Reduce learning rate by half when triggered
    patience=5,            # Wait 3 epochs before reducing learning rate
    min_lr=1e-6,           # Set a lower bound for the learning rate
    verbose=1              # Print updates when learning rate is reduced
)

# Define EarlyStopping callback to prevent overfitting and unnecessary training
early_stopping = EarlyStopping(
    monitor='val_loss',           # Monitor validation loss
    patience=10,                   # Stop training if no improvement for 5 epochs
    restore_best_weights=True,     # Restore the best model weights
    verbose=1                      # Print a message when training stops
)

# Train the model with defined callbacks
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=epochs,
    batch_size=batch_size,
    verbose=1,                     # Show training progress
    callbacks=[reduce_lr, early_stopping]  # Include callbacks for adaptive training
)
```

5. Model Training

Implemented and trained multiple deep learning models to evaluate performance:

A. LSTM Model:

- **Architecture**

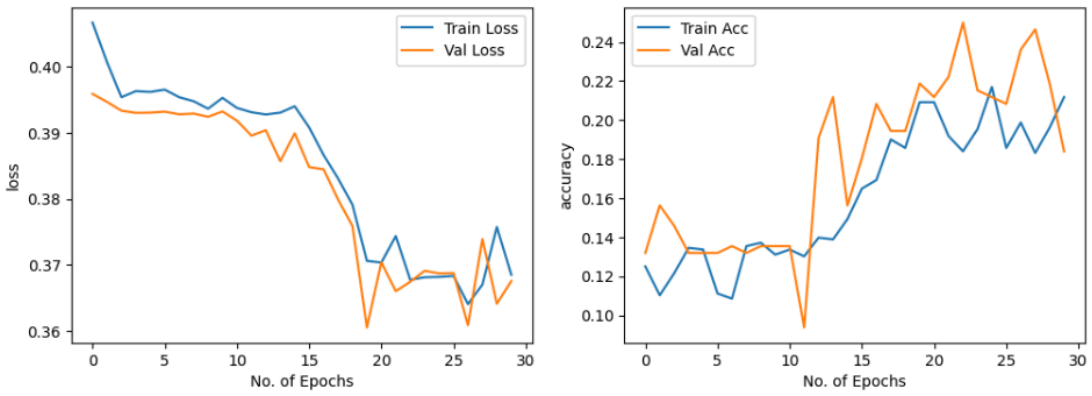
```
input_shape shape: (160, 1)
num_classes shape: 8
✅ Model compiled and this is its visualization.
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 256)	264,192
dense_3 (Dense)	(None, 128)	32,896
dropout_2 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8,256
dropout_3 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 8)	520

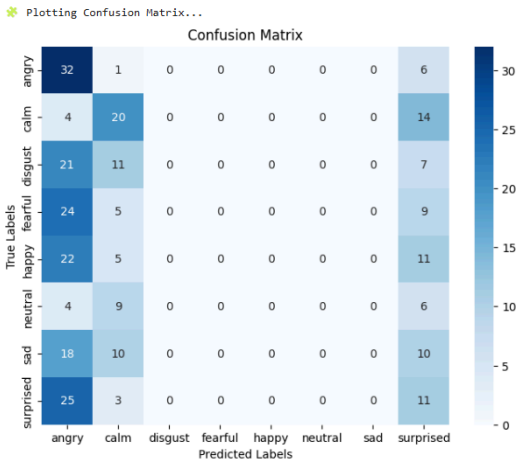
```
Total params: 305,864 (1.17 MB)
Trainable params: 305,864 (1.17 MB)
Non-trainable params: 0 (0.00 B)
```

- **Training Parameters**
  - **Parameters**: 305,864 trainable (Heavy-Weight)
  - **Optimizer**: Adam
  - **Batch Size**: 36
  - **Epochs**: 30

- **Performance**
  - **Train Accuracy**: 21.61%
  - **Test Accuracy**: 21.88%



```
🏋️ Train Accuracy: 21.61%
🔑 Test Accuracy: 21.88%

📊 Generating Classification Report...

Classification Report:
              precision    recall  f1-score   support

       angry       0.21      0.82      0.34        39
        calm       0.31      0.53      0.39        38
     disgust       0.00      0.00      0.00        39
     fearful       0.00      0.00      0.00        38
       happy       0.00      0.00      0.00        38
     neutral       0.00      0.00      0.00        19
         sad       0.00      0.00      0.00        38
   surprised       0.15      0.28      0.19        39

    accuracy                           0.22       288
   macro avg       0.08      0.20      0.12       288
weighted avg       0.09      0.22      0.12       288
```



- ## <u>Analysis of Classification Report and Confusion Matrix</u>

1. Extremely Low Overall Performance

- Train Accuracy (21.61%) vs. Test Accuracy (21.88%) → The model is not learning effectively, as both scores are close to random guessing (12.5% for 8 classes).
- Possible Causes: Poor feature extraction, model underfitting, or insufficient training time.

2. Class-Wise Performance Breakdown

- Strongest Class:
  - *Angry (82% recall, 34% F1-score)* → The model classifies most "angry" samples but with low precision.
- Weakest Classes:
  - *Disgust, Fearful, Happy, Neutral, Sad* → All have 0% precision, recall, and F1-score, meaning they are completely misclassified.

3. Macro vs. Weighted Averages

- Macro Avg (8% precision, 20% recall, 12% F1-score) → Very low performance across all classes.
- Weighted Avg (9% precision, 22% recall, 12% F1-score) → Slightly higher recall, but overall poor classification.

B. CNN Model

- **Architecture**

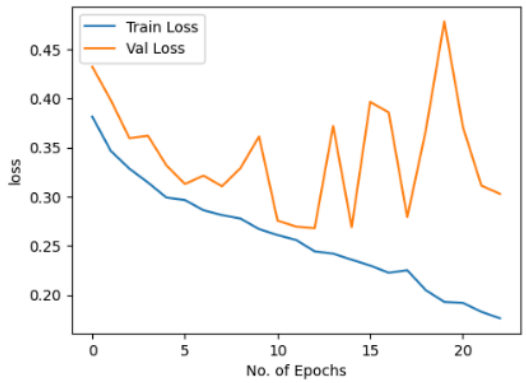
```
input_shape shape: (160, 1)
num_classes shape: 8
✅ Model compiled and this is its visualization.
Model: "sequential_58"
```

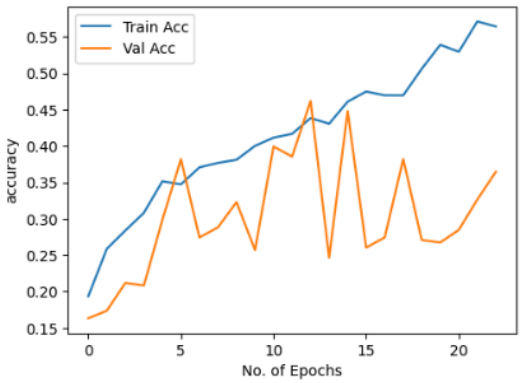
Layer (type)	Output Shape	Param #
conv1d_232 (Conv1D)	(None, 160, 64)	256
batch_normalization_232 (BatchNormalization)	(None, 160, 64)	256
max_pooling1d_174 (MaxPooling1D)	(None, 80, 64)	0
conv1d_233 (Conv1D)	(None, 80, 128)	24,704
batch_normalization_233 (BatchNormalization)	(None, 80, 128)	512
max_pooling1d_175 (MaxPooling1D)	(None, 40, 128)	0
dropout_241 (Dropout)	(None, 40, 128)	0
conv1d_234 (Conv1D)	(None, 40, 256)	98,560
batch_normalization_234 (BatchNormalization)	(None, 40, 256)	1,024
max_pooling1d_176 (MaxPooling1D)	(None, 20, 256)	0
global_average_pooling1d_25 (GlobalAveragePooling1D)	(None, 256)	0
dense_183 (Dense)	(None, 128)	32,896
dropout_242 (Dropout)	(None, 128)	0
dense_184 (Dense)	(None, 64)	8,256
dropout_243 (Dropout)	(None, 64)	0
dense_185 (Dense)	(None, 8)	520

```
Total params: 166,984 (652.28 KB)
Trainable params: 166,088 (648.78 KB)
Non-trainable params: 896 (3.50 KB)
```

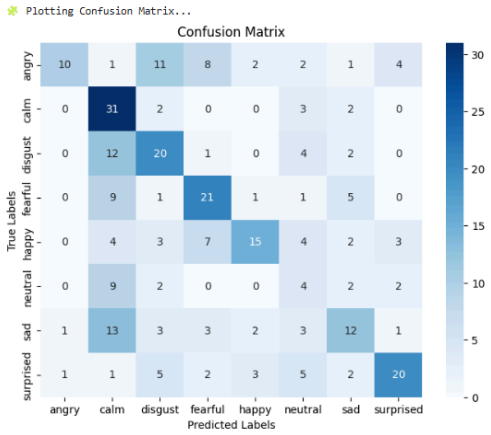
- **Training Parameters**
  - **Parameters**: 166088 trainable (Light-Weight)
  - **Optimizer**: Adam
  - **Batch Size:** 36
  - **Epochs**: 23

- **Performance**
  - **Train Accuracy:** 45.57%
  - **Test Accuracy:** 46.18%






```
Train Accuracy: 45.57%
Test Accuracy: 46.18%

Generating Classification Report...

Classification Report:
              precision    recall  f1-score   support

       angry       0.83      0.26      0.39        39
        calm       0.39      0.82      0.53        38
     disgust       0.43      0.51      0.47        39
     fearful       0.50      0.55      0.53        38
       happy       0.65      0.39      0.49        38
     neutral       0.15      0.21      0.18        19
         sad       0.43      0.32      0.36        38
   surprised       0.67      0.51      0.58        39

    accuracy                           0.46       288
   macro avg       0.51      0.45      0.44       288
weighted avg       0.53      0.46      0.46       288
```



- **Analysis of Classification Report and Confusion Matrix**

1. **Low Overall Performance & Generalization Issues**
   - Train Accuracy (45.57%) vs. Test Accuracy (46.18%) → The model is not overfitting but also not learning effectively, as both accuracies are low.

2. **Class-Wise Performance Variability**
   - **Strong Classes:**
     - *Calm (82% recall, 53% F1-score)* → The model recognizes calm emotions well but struggles with precision.
   - **Weak Classes:**
     - *Angry (83% precision, 26% recall, 39% F1-score)* → High precision but very low recall, meaning many angry samples are misclassified.
     - *Neutral (15% precision, 21% recall, 18% F1-score)* → The model struggles significantly with neutral emotions, likely due to feature overlap with other classes.

3. **Imbalanced Class Detection**
   - *Angry & Happy* have high precision but low recall, meaning the model is confident but missing many actual instances.
   - *Sad & Neutral* have low precision and low recall, indicating difficulty in distinguishing them.
   - *Calm & Fearful* have higher recall than precision, suggesting the model tends to classify multiple samples as these emotions.

4. **Macro vs. Weighted Averages**
   - Macro Avg (0.51 precision, 0.45 recall, 0.44 F1-score): Shows that, on average, the model performs poorly across all classes.
   - Weighted Avg (0.53 precision, 0.46 recall, 0.46 F1-score): Suggests that class imbalance may not be severe, but the model still lacks strong generalization.

C. Hybrid CNN+LSTM Model

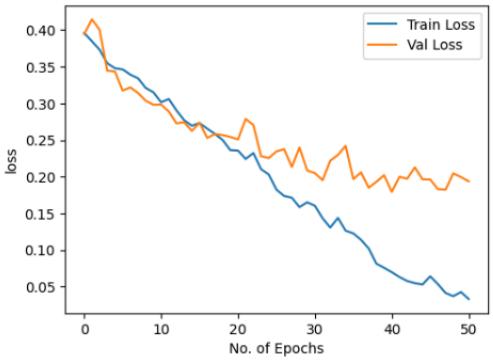
- **Architecture**
  - Combines CNN for feature extraction with LSTM for temporal learning.
  - CNN layers process spectrogram images, while LSTM layers model temporal dependencies.

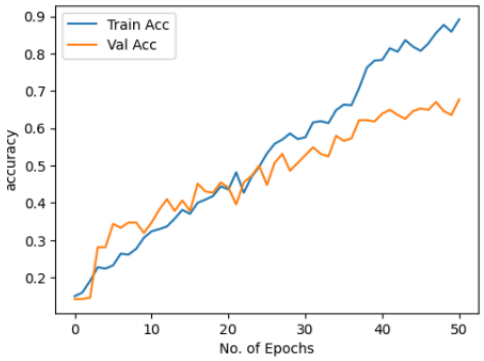
Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 160, 64)	256
batch_normalization (BatchNormalization)	(None, 160, 64)	256
max_pooling1d (MaxPooling1D)	(None, 80, 64)	0
conv1d_1 (Conv1D)	(None, 80, 128)	24,704
batch_normalization_1 (BatchNormalization)	(None, 80, 128)	512
max_pooling1d_1 (MaxPooling1D)	(None, 40, 128)	0
dropout_4 (Dropout)	(None, 40, 128)	0
conv1d_2 (Conv1D)	(None, 40, 256)	98,560
batch_normalization_2 (BatchNormalization)	(None, 40, 256)	1,024
max_pooling1d_2 (MaxPooling1D)	(None, 20, 256)	0
lstm_2 (LSTM)	(None, 20, 128)	197,120
lstm_3 (LSTM)	(None, 64)	49,408
dense_6 (Dense)	(None, 64)	4,160
dropout_5 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 32)	2,080
dropout_6 (Dropout)	(None, 32)	0
dense_8 (Dense)	(None, 8)	264

Total params: 378,344 (1.44 MB)
Trainable params: 377,448 (1.44 MB)
Non-trainable params: 896 (3.50 KB)

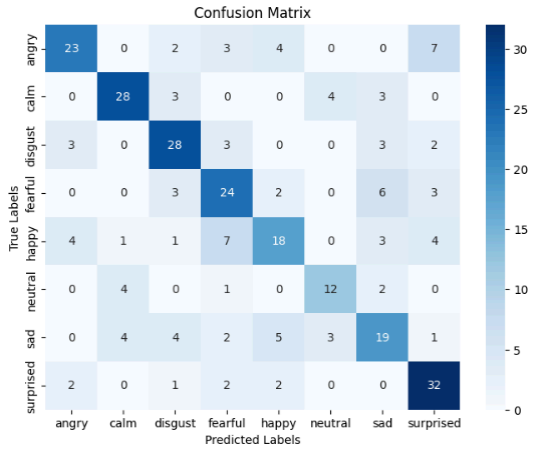
- **Training Parameters**
  - **Parameters**: 377,448 trainable
  - **Optimizer**: Adam
  - **Batch Size:** 36
  - **Epochs:** 23

- **Performance**
  - **Train Accuracy:** 88.28%
  - **Test Accuracy:** 63.89%
  - Best performing model among baseline architectures.






```
Train Accuracy: 88.28%
Test Accuracy: 63.89%

Generating Classification Report...

Classification Report:
              precision    recall  f1-score   support

       angry       0.72      0.59      0.65        39
        calm       0.76      0.74      0.75        38
     disgust       0.67      0.72      0.69        39
     fearful       0.57      0.63      0.60        38
       happy       0.58      0.47      0.52        38
     neutral       0.63      0.63      0.63        19
         sad       0.53      0.50      0.51        38
   surprised       0.65      0.82      0.73        39

    accuracy                           0.64       288
   macro avg       0.64      0.64      0.64       288
weighted avg       0.64      0.64      0.64       288
```



- **Analysis of Classification Report and Confusion Matrix**

1. **Improved Generalization but Scope for Enhancement**
   - Train Accuracy (88.28%) vs. Test Accuracy (63.89%): Overfitting is reduced compared to the previous model, but there is still a noticeable performance gap.

2. **Class-Wise Performance Variations**
   - **Strong Classes:** Surprised (82% recall, 73% F1-score) and Calm (74% recall, 75% F1-score) are well-classified, suggesting the model effectively captures their features.
   - **Weak Classes:** Happy (47% recall, 52% F1-score) and Sad (50% recall, 51% F1-score) struggle, possibly due to feature overlap or lack of distinguishing patterns.

❖ Evaluation: CNN v/s LSTM v/s LSTM+CNN

Model	Train Accuracy	Test Accuracy	Best Recognized Emotion	Weakest Recognized Emotion
LSTM	21.61%	21.88%	Angry (82% recall)	Disgust, Happy, Sad (0% recall)
CNN	45.57%	46.18%	Calm (82% recall)	Neutral (21% recall)
CNN + LSTM	88.28%	63.89%	Surprised (82% recall)	Happy (47% recall)

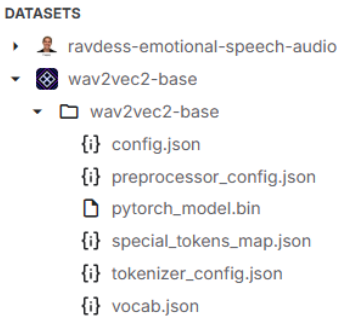
- The CNN + LSTM model outperforms both the LSTM and CNN models in terms of accuracy and recall, particularly excelling in recognizing the **Surprised** emotion.
- While the LSTM model is effective for recognizing **Angry**, it struggles with other emotions, showing poor recall for **Disgust**, **Happy**, and **Sad**.
- The CNN model performs well for **Calm**, but still has limitations, especially with **Neutral** emotions, highlighting the advantage of combining CNN and LSTM for improved emotion classification.

❖ Wav2Vec 2.0 + Random Forest Classifier:

- Transformer-based self-supervised model for raw audio (Facebook AI).
- Learn speech features without labeled data.

- **Benefits**
  - Captures rich audio features, boosting accuracy.
  - Reduces need for large labeled datasets.
  - Adapts across languages & speech variations.

- **Implementation Details**
  - Used wav2vec2-base for feature extraction.
  - Fine-tuned the model on the emotion classification task.
  - Model files used:
    - config.json, preprocessor_config.json
    - pytorch_model.bin (Pre-trained weights)
    - tokenizer_config.json, vocab.json



- **Architecture**
  - **Feature Extraction:**
    - Resamples audio (16kHz), extracts embeddings
    - Outputs feature matrix [time_steps, feature_dim]

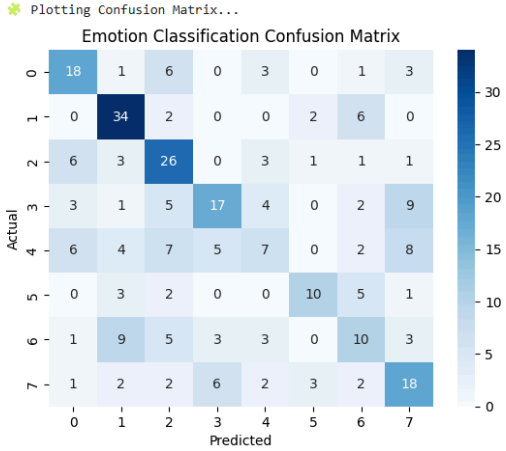
  - **Random Forest Classifier:**
    - Trains on extracted features
    - Predicts on the train & test data

```
✅ Dataset Extracted and Loaded Successfully!
Total audio files processed: 1440
Emotion distribution:
  surprised: 193 files
  neutral: 96 files
  disgust: 192 files
  fearful: 192 files
  sad: 192 files
  calm: 192 files
  happy: 192 files
  angry: 192 files

WAV2VEC_X shape: (1440, 170, 768)
WAV2VEC_y shape: (1440,)
```

- **Performance**
  - **Train Accuracy:** 100% (overfitting issue)
  - **Test Accuracy:** 48.61% (generalization problem)

```
Making predictions on the test set...
Predictions completed.

Train Accuracy: 100.00%
Test Accuracy: 48.61%

Generating Classification Report...

Classification Report:
              precision    recall  f1-score   support

           0       0.51      0.56      0.54        32
           1       0.60      0.77      0.67        44
           2       0.47      0.63      0.54        41
           3       0.55      0.41      0.47        41
           4       0.32      0.18      0.23        39
           5       0.62      0.48      0.54        21
           6       0.34      0.29      0.32        34
           7       0.42      0.50      0.46        36

    accuracy                           0.49       288
   macro avg       0.48      0.48      0.47       288
weighted avg       0.48      0.49      0.47       288
```

```
Plotting Confusion Matrix...
```



- **Analysis of Classification Report and Confusion Matrix**

1. **Overfitting & Generalization Issues**
   - **Train Accuracy (100%) vs. Test Accuracy (48.61%)** → The model memorizes training data but fails on unseen data.
   - **Possible Cause:** High variance due to an overly complex model or lack of regularization.

2. **Performance Variability Across Classes**
   - **Class 1 (0.60 precision, 0.77 recall, 0.67 F1-score)** is well classified, showing strong model confidence.
   - **Class 4 (0.32 precision, 0.18 recall, 0.23 F1-score)** is poorly classified, indicating difficulty in feature distinction.

3. **Imbalanced Class Performance**
   - **High Recall in Some Classes (e.g., 1 and 2):** The model detects these classes more accurately.
   - **Low Recall in Other Classes (e.g., 4 and 6):** The model struggles to correctly identify these categories.

4. **Macro vs. Weighted Averages**
   - **Macro Avg (0.48 precision, 0.48 recall, 0.47 F1-score):** Treats all classes equally, showing performance imbalance.
   - **Weighted Avg (0.48 precision, 0.49 recall, 0.47 F1-score):** Considers class frequency but still indicates weak overall performance.

❖ Evaluation: CNN V/s Transformer Based Model

- CNN+LSTM performed best, demonstrating the advantage of combining feature extraction and sequential learning.
- Wav2Vec 2.0 exhibited overfitting, requiring further regularization and dataset expansion.
- Emotion classification imbalance was observed in lower-represented emotions like 'disgust' and 'surprised.'

❖ Conclusion

The Speech Emotion Recognition project successfully implemented various models for emotion classification. The CNN+LSTM model achieved the highest accuracy (63.89%), demonstrating the importance of both spatial and temporal feature extraction. The Wav2Vec 2.0 model showed potential but needs better optimization. Future work will focus on uses of more fine-tuned transformer models, handling class imbalance, and exploring advanced architectures like attention mechanisms.

This project underscores the complexity of SER and the need for more robust deep learning techniques, larger datasets, and improved preprocessing strategies to develop reliable emotion recognition systems for real-world applications.