SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

50.021 Artificial Intelligence

Speech Emotion Recognition (SER) Application: Audio-based Neural Network for Affective Multiclass Analysis and Labelling using Artificial Intelligence [ANNAMALAI]

Submitted by:

Lim Sheng Xiang 1005005
Tay Wei Han Joel 1005117
Lim Fuo En 1005125
Lim Rui-Chong Anthony 1005264
Ankita Sushil Parashar 1005479

2025

Miscellaneous

Peer Review

Each member contributed equally and participated actively in discussions throughout the project. This collective effort was instrumental in achieving our objectives and ensuring the successful completion of the project.

Lim Sheng Xiang - 100%

Tay Wei Han Joel - 100%

Lim Fuo En - 100%

Lim Rui-Chong Anthony - 100%

Ankita Sushil Parashar - 100%

Github Repository

The full source code detailing our model training and supplementary materials can be accessed via the GitHub repository here. Due to file size limitations on GitHub, the trained model files are not included in the repository.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Understanding emotions in speech is a fundamental yet complex aspect of human communication. Variations in tone, pitch, intensity, accent, and context make it challenging to accurately detect emotions, and misinterpretation of these emotional cues can lead to misunderstandings—particularly in high-stakes environments such as law enforcement, customer service, and mental health support.

To address this, our project explores Speech Emotion Recognition (SER), which leverages artificial intelligence to analyze vocal patterns and automatically classify emotional states. By doing so, SER minimizes human bias, improves emotional awareness, and supports decision-making through consistent and scalable assessments of emotional intent.

This project supports the "AI for Good" initiative and aligns with two key United Nations (UN) Sustainable Development Goals (SDGs):

- SDG 3 – Good Health and Well-Being (United Nations, 2025b): SER enables early detection of emotional distress, allowing for timely mental health support and intervention.
- SDG 16 – Peace, Justice, and Strong Institutions (United Nations, 2025a): SER can aid in investigative and credibility assessment contexts, providing fairer evaluations based on emotional cues in speech.

Our project, titled "Audio-based Neural Network for Affective Multiclass Analysis and Labelling using Artificial Intelligence [ANNAMALAI]" aims to systematically evaluate different SER pipelines. These include classical machine learning approaches (Support Vector Machines, K-Nearest Neighbors, Random Forest) using handcrafted features from openSMILE, and modern deep learning architectures (Feed-Forward Neural Networks, Convolutional Neural Networks, Wav2Vec2, etc.) trained on MFCCs, Mel spectrograms and raw audio inputs. Through comparative analysis, we aim to determine the most effective approach for speech-based emotion recognition that can be used in real-world applications, including a frontend inference system.

Chapter 2

Literature Review

The goal of SER is to analyze tone, pitch, energy, and rhythm to determine the emotions included in speech signals. By allowing machines to react with empathy, it significantly improves the user experience in applications like virtual assistants, healthcare, and customer service. Nevertheless, speaker variability, differences in gender, language, accent, and emotion ambiguity, since emotions frequently reside on a continuum rather than as distinct categories, make SER a challenging task. SER systems have historically used traditional machine learning classifiers and hand-engineered features. To categorize emotions, acoustic variables like spectral (like MFCCs) and prosodic (like pitch, intensity, and duration) were extracted and fed into models such as Support Vector Machines (SVMs) or Hidden Markov Models (HMMs).

Following their extraction, these features were entered into classifiers such as decision trees, SVMs, and HMMs. Strong generalization in high-dimensional spaces was provided by SVMs, which were especially useful for small to medium-sized datasets. HMMs were a perfect fit for sequential data since they could simulate the temporal dependencies in speech. Despite their simplicity, decision trees provided quick inference and interpretability. However, these conventional approaches face drawbacks. They need a great deal of domain knowledge and are largely reliant on handcrafted features. Their scalability to real-world applications was further limited by their inability to generalize across various speakers or recording conditions and their lack of resistance to noise (Batliner and Schuller, 2011).

By allowing models to directly learn features from unprocessed audio, deep learning revolutionized SER. Convolutional Neural Networks (CNNs) effectively extracted spatial features from spectrograms to identify patterns significant to emotion, whereas Deep Neural Networks (DNNs) functioned as early deep learning baselines. Temporal modeling was introduced by Recurrent Neural Networks (RNNs) and Long-Short Term Memory (LSTMs), which captured the dynamics of emotions over time. Notwithstanding their advantages, these models still needed big datasets and were susceptible to speaker fluctuation and noise. Compared to arousal or dominance detection, valence prediction in particular continued to be difficult.

Transformers provide better sequence modeling than RNNs by capturing long-range dependencies through self-attention processes in parallel. Using transformer models such as Wav2Vec 2.0 on datasets like MSP-Podcast and IEMOCAP, the paper "Dawn of the Transformer Era in SER" demonstrated significant gains in valence prediction. Their efficacy was validated by metrics like robustness testing, fairness assessments, and Concordance Correlation Coefficient. New benchmarks for performance and generalization in SER are now set by pretrained transformers that have been refined using emotional data (Wagner et al., 2023).

From traditional methods to deep learning and now to transformers, SER has undergone significant development. Conventional models were interpretable, but they were not scalable. Although they needed a lot of training data, deep models enhanced feature learning. The field is now led by transformer topologies because they provide superior generalization, particularly for valence detection. However, noise continues to pose a major challenge in real-world deployment. Several techniques have been proposed to address noisy speech conditions, including signal preprocessing, noise-robust feature extraction, and data augmentation strategies like adding white noise or applying time and frequency masking. These methods have shown promising improvements on SER benchmarks. Still, the domain of noise-robust speech emotion recognition remains an open research area, with more exploration needed to develop models that are both highly accurate and resilient across diverse, real-world acoustic environments.

Chapter 3

Datasets Used

3.1 Data Sources

In this work we use a compilation of eight publicly-available datasets (Oh, 2025). They are:

- CREMA-D (Crowd-Sourced Emotional Multimodal Actors Dataset) (Cao et al., 2014)
- MELD (Multimodal EmotionLines Dataset) (Poria et al., 2019) (S.-Y. Chen et al., 2018)
- MLEND Spoken Numerals (Carrión and Bajaj, 2022)
- RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song) (Livingstone and Russo, 2018)
- SAVEE (Surrey Audio-Visual Expressed Emotion) (Jackson and Haq, 2025)
- TESS (Toronto Emotional Speech Set) (Dupuis and Pichora-Fuller, 2025)
- ESD (Emotional Speech Dataset for Voice Conversion) (Zhou et al., 2021) (Zhou et al., 2022)
- JL-Corpus (James, Tian, and Watson, 2018)

Detailed information about the number of samples and the types of emotional classes present in each dataset can be found in Table 3.1. Using multiple source datasets increases our dataset diversity for the model to be trained on a wider range and different data sources.

3.1.1 Combining the Datasets

These datasets were selected because they all contain the raw audio files in the English language together with their corresponding Emotion labels. They were combined to give one combined dataset to give a total of 74,705 samples. The individual datasets each have their unique data annotations in their metadata, some of which contain the text utterance of what was spoken in the audio clip, the intensity and sentiments of the emotion, as well as the information about the speaker, such as their gender, age and race. However, the only relevant information is the raw audio file, as well as our target class which is the Emotion label, since we are classifying speech inputs based on emotion. As such, we only extracted these fields.

Dataset	Number of samples	Types of classes
CREMA-D	7,442	6 Classes: Anger, Disgust, Fear, Happy, Neutral, Sadss
MELD	9,989	7 classes: Anger, Disgust, Sadness, Joy, Neutral, Surprise and Fear
MLEnd	32,654	4 classes: Neutral, Bored, Excited and Question
RAVDESS	1,440	8 classes: Calm, Happy, Sad, Angry, Fearful, Surprise, Disgust and Neutral
SAVEE	480	7 classes: Anger, Disgust, Fear, Happiness, Sadness, Surprise and Neutral
TESS	2,800	7 classes: Anger, Disgust, Fear, Happiness, Pleasant Surprise, Sadness, and Neutral
ESD	17,500	5 classes: Neutral, Happy, Angry, Sad and Surprise
JL Corpus	2,400	10 classes: Angry, Anxious, Apologetic, Assertive, Concerned, Encouraging, Excited, Happy, Neutral and Sad

TABLE 3.1: Information of Datasets

3.1.2 Data Processing

Each of the different datasets has different ways of annotating their emotion labels. A mapping was done across the different datasets to ensure that annotations of similar meanings are mapped to the same label. For instance, in the MELD dataset, the annotation 'Joy', was mapped to the final label 'Happy'. A similar mapping was done for all our source datasets. Additionally, checks on the raw audio files using various audio processing libraries were used, and there were found to be three corrupted audio files: one from MELD and two from TESS. They were removed from our dataset, leaving 74,702 samples remaining.

3.2 Exploratory Data Analysis

3.2.1 Audio Duration Distribution

The distribution of audio duration was visualized by plotting a histogram. The long end distribution is shown in Figure 3.1, and the short end distribution is shown in Figure 3.2.

Based on our distribution plots, it is clear that there are some audios that either of very short or very long duration. Most of our audio samples have a duration between 1-4 seconds. To ensure that the models we trained are not skewed by very short or very long samples, we removed these outliers from our combined dataset. After this filtering, we are left with a total of 64,754 samples, of which they have a duration between 1-4 seconds.

3.2.2 Emotion Label Distribution

The distribution of emotion labels, our target class, was also visualized by plotting a bar graph. The distribution is shown in Figure 3.3.

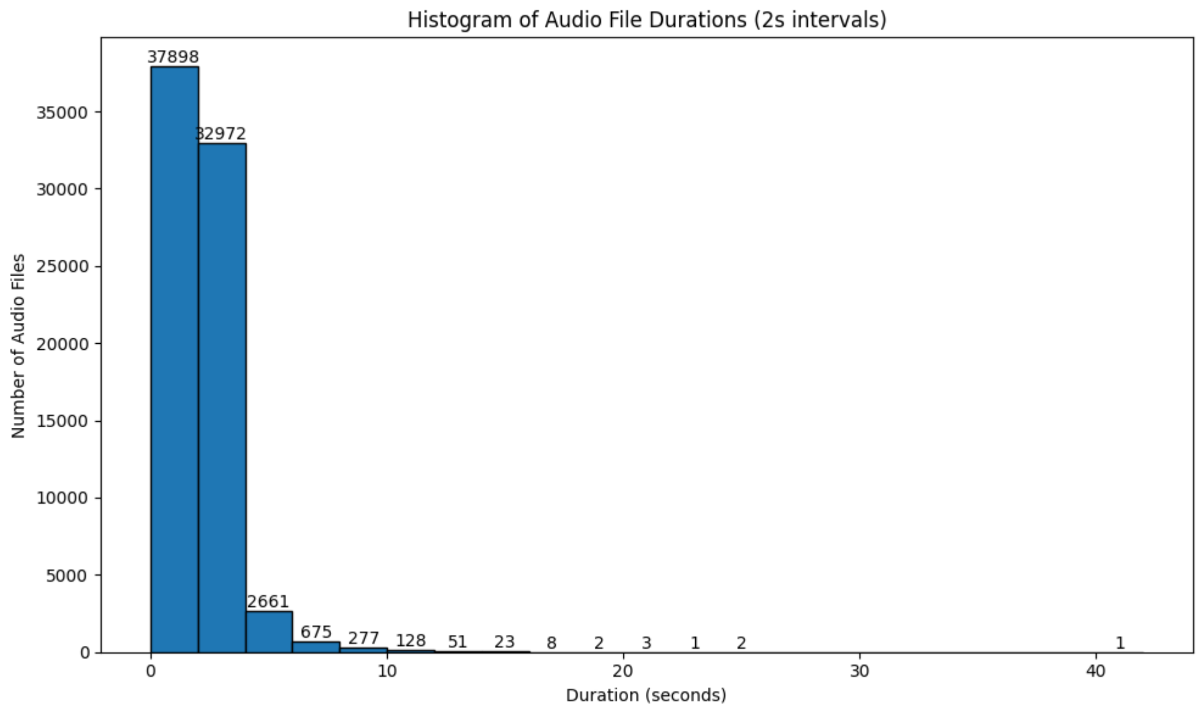


FIGURE 3.1: Long-end Distribution of Duration of Audio Samples

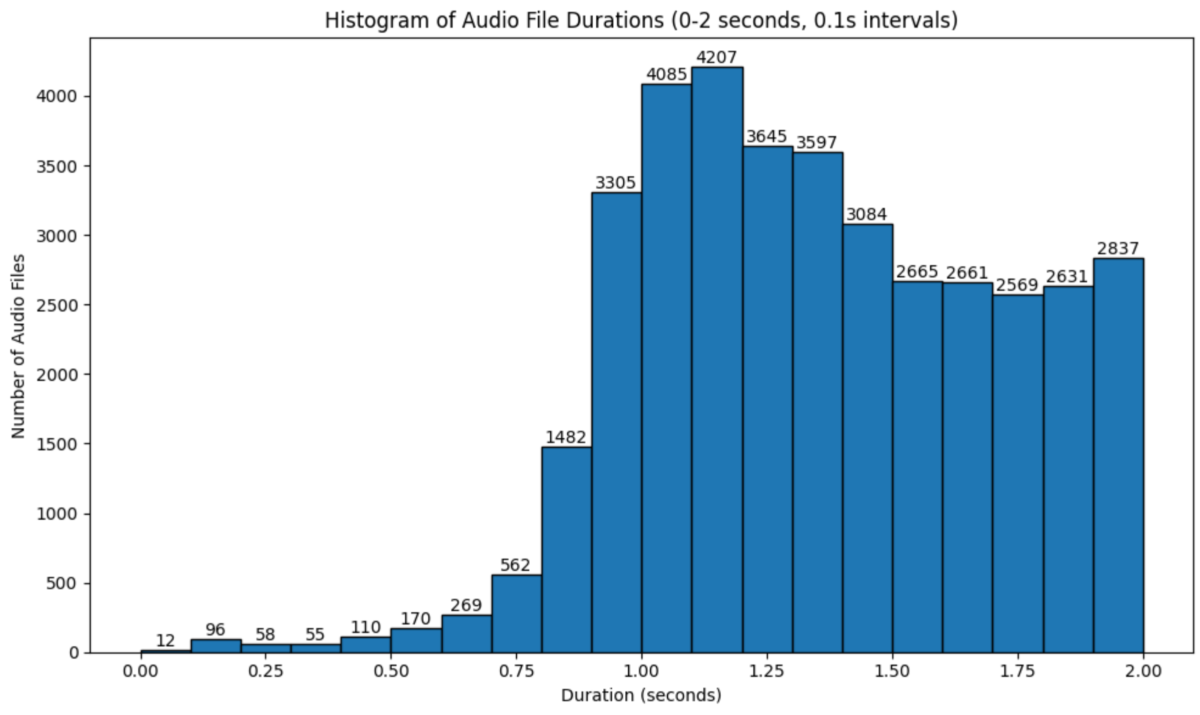


FIGURE 3.2: Short-end Distribution of Duration of Audio Samples

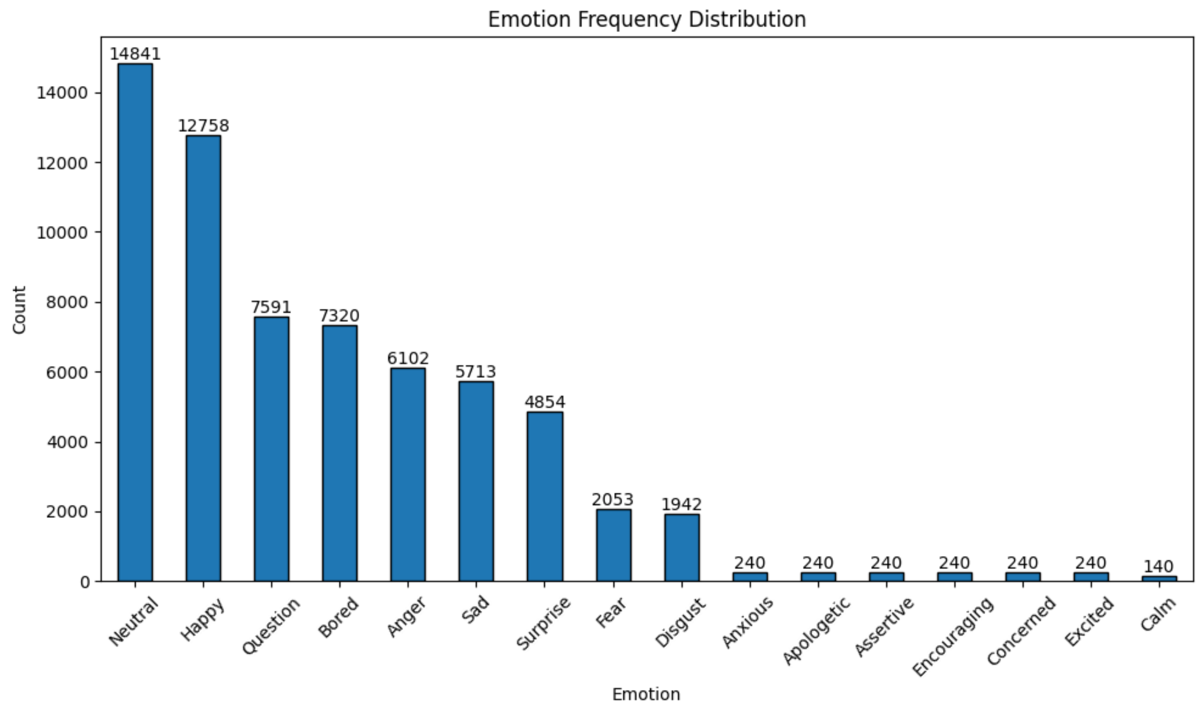


FIGURE 3.3: Initial Emotional Label Distribution

Our distribution shows there are a total of 16 unique labels. Within these unique labels, there are rare classes, namely: Anxious, Apologetic, Assertive, Encouraging, Concerned, Excited, Calm. Rare classes cause class imbalance, and the total number of all the rare classes all comprise about 2.5% of the entire dataset. This might affect the metrics used to evaluate the models we train. As such, we removed these rare classes from our dataset. After this filtering, we are left with a total of 63,174 samples.

3.3 Final Dataset

The final dataset distribution of emotion labels is shown in Figure 3.4.

3.3.1 Stratified Sampling

The distribution shown in Figure 3.4 still shows some relative class imbalance, although it is a much lesser degree than before. The classes 'Fear' and 'Disgust' are still relatively well represented by a decent number of samples. To ensure that this class distribution is kept, we will employ Stratified Sampling to ensure that each class is proportionally represented in our train, validation, and test sets with a 70-15-15 split, as seen in Table 3.2.

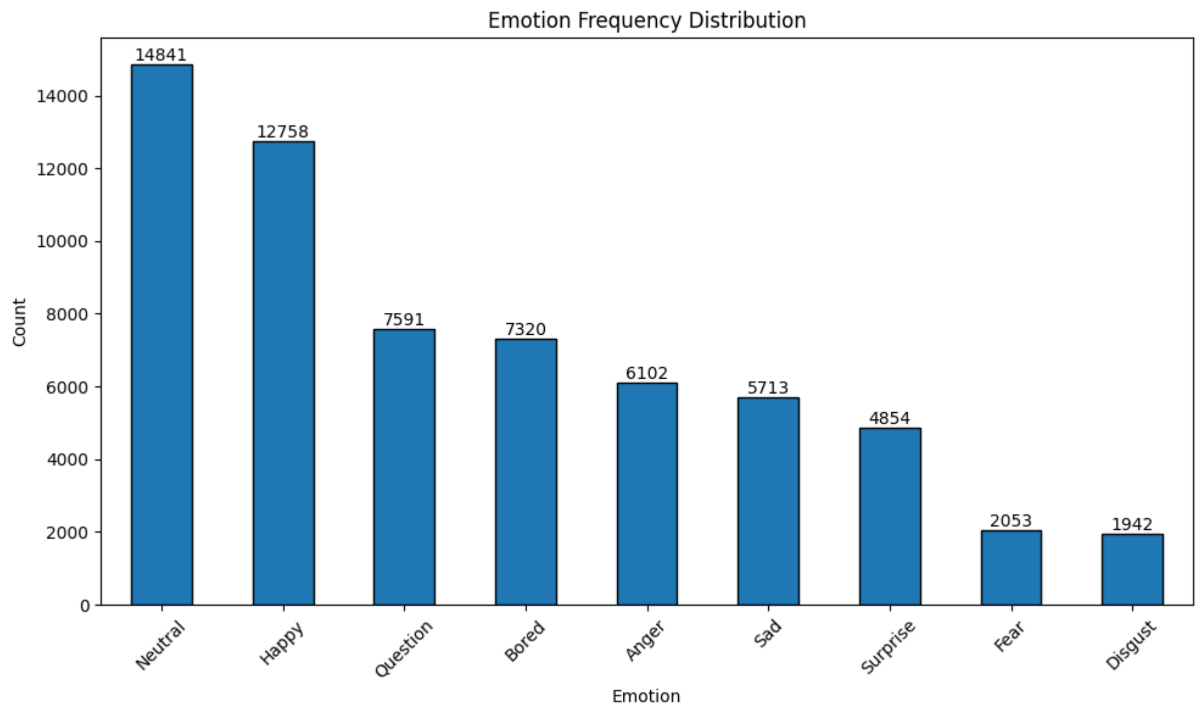


FIGURE 3.4: Final Emotional Label Distribution

Set	**Count**
Train	44,221
Test	9,476
Validation	9,477
Grand Total	63,174

TABLE 3.2: Train-Test-Validation Split

Chapter 4

Methodology

This chapter provides a broad overview of the methodology we developed and embarked on, which is then elaborated and explained in detail in the subsequent chapters.

In our workflow, we first collected source speech emotion datasets available online and performed data cleaning on them, such as dropping corrupted samples and outliers. Then, we combined them all and performed a 70-15-15 train-test-validation split, as seen in Table 3.2.

Using the same splits, we embarked on various approaches for data preprocessing and model training, which includes classical machine learning, and deep learning approaches:

- Extract features using openSMILE, an audio signal feature extractor (Eyben, Wöllmer, and Schuller, 2010), and then use these features to train:
  - Classical machine learning (ML) models, such as support vector machines (SVMs), etc.
  - Feed-forward neural networks (FFNNs).
- Extract MFCCs using librosa, an audio analysis tool (McFee et al., 2015), and then use them to train our own built convolutional neural networks (CNNs).
- Generate mel spectrogram images using librosa, and then use them to fine-tune pre-trained CNNs.
- Use the raw audio files to fine-tune pre-trained transformer models.

During model training, within each approach, we ran several experiments and iterated to maximize model performance. We then selected the best-performing model based on the test F1-score on the test set from the five approaches. Finally, we used the best-performing model on our frontend for inference. A high-level visualization of our methodology can be seen in Figure 4.1.

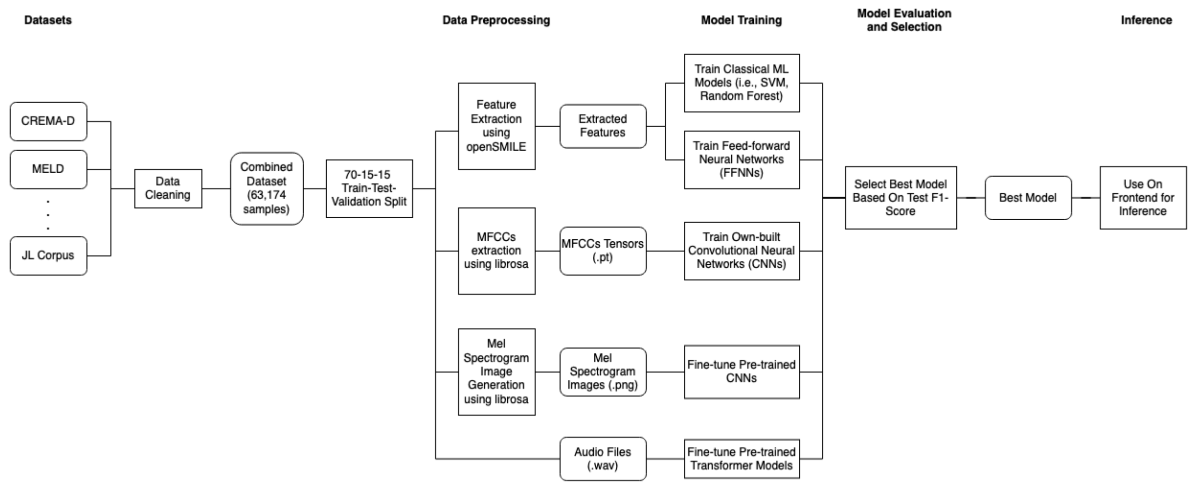

Datasets
Data Preprocessing
Model Training
Model Evaluation and Selection
Inference
CREMA-D
MELD
JL Corpus
Data Cleaning
Combined Dataset (63,174 samples)
70-15-15 Train-Test-Validation Split
Feature Extraction using openSMILE
Extracted Features
Train Classical ML Models (i.e., SVM, Random Forest)
Train Feed-forward Neural Networks (FFNNs)
MFCCs extraction using librosa
MFCCs Tensors (.pt)
Train Own-built Convolutional Neural Networks (CNNs)
Mel Spectrogram Image Generation using librosa
Mel Spectrogram Images (.png)
Fine-tune Pre-trained CNNs
Audio Files (.wav)
Fine-tune Pre-trained Transformer Models
Select Best Model Based On Test F1-Score
Best Model
Use On Frontend for Inference

FIGURE 4.1: Broad Overview of Our Methodology

Chapter 5

Classical ML Approaches

5.1 Feature Extraction with openSMILE

For the classical ML approaches explored in this project, we utilized the features extracted by openSMILE (open-source Speech and Music Interpretation by Large-space Extraction) as inputs to these models. openSMILE is an open-source toolkit for audio feature extraction and classification of speech and music signals. openSMILE is widely applied in automatic expression recognition for affective computing.

openSMILE has been utilised in the SER research space. For example, in the Speech Emotion Recognition Based on SVM and ANN study (Ke et al., 2018), a classical SVM classifier to a simple feedforward multilayer perceptron for SER was compared. Using acoustic features extracted from the CASIA Mandarin emotion corpus (via openSMILE), the authors applied feature reduction techniques and trained both models. We will be using a very similar process in our approaches for this section.

openSMILE provides numerous feature sets, and some examples are the IS09, emobase, GeMAPS, and eGeMAPS feature sets. In this project, we utilised the IS13_ComParE set to extract audio features. It was introduced for the INTERSPEECH 2013 Computational Paralinguistics Challenge (ComParE). The outputs of this feature set are utterance-level feature vectors, extracted by computing statistical functionals over frame-level low-level descriptors (LLDs). Each vector has a total of 6,373 features. The various LLDs are described in Table 5.1.

Although the feature set is very extensive with 6,373 dimensions, it is the large dimensions that might pose challenges during model training, such as being prone to overfitting and general feature redundancy. Our preprocessing pipeline has three key stages: feature normalization, dimensionality reduction via Principal Component Analysis (PCA), and supervised feature selection using univariate statistical testing.

Firstly, the input features are normalized to ensure consistent scale across all dimensions via a standard scaler (zero mean, unit variance). Normalisation is applied by fitting the scaler on the training set, and subsequently transforming the training, validation and test sets using the learned parameters. Given the large number of features produced by openSMILE, PCA is employed to reduce redundancy. PCA projects the original feature space onto an orthogonal basis of principal components ranked by the amount of variance explained. We have defined the threshold for retained variance to be 95%, ensuring that the reduced representation captures the majority of variability present in the original data. Lastly, a further reduction in dimensionality is achieved through supervised feature selection. Since the class labels (emotions) are available, the top k principal components are selected based on their ANOVA F-statistics using the `SelectKBest` method from scikit-learn. This method selects principal components

4 energy related LLD
Sum of auditory spectrum (loudness) Sum of RASTA-style filtered auditory spectrum RMS Energy Zero-Crossing Rate
54 spectral LLD
RASTA-style auditory spectrum, bands 1-26 (0-8 kHz) MFCC 1-14 Spectral energy 250-650 Hz, 1 k-4kHz Spectral Roll Off Point 0.25, 0.50, 0.75, 0.90 Spectral Flux, Entropy, Variance, Skewness, Kurtosis Slope, Psychoacoustic Sharpness, Hamonicity
6 voicing related LLD
F0 by SHS + Viterbi smoothing, Probability of voicing logarithmic HNR, Jitter (local, delta), Shimmer (local)

TABLE 5.1: Low-level descriptors from openSMILE IS13_ComParE Feature Set (Schuller et al., 2013)

with the highest discriminative power. This k number of components is a hyperparameter for the FFNN which is discussed in Section 5.3.

5.2 Traditional ML Methods

Three classical ML algorithms, K-Nearest Neighbors (KNN), Random Forest (RF), and SVM were selected for their complementary strengths in high-dimensional feature spaces. KNN offers an interpretable, instance-based classification mechanism based on local distance metrics; RF constructs robust ensembles of decision trees via bootstrap aggregation and randomized feature selection to capture complex non-linear interactions; and SVM seeks maximum-margin hyperplanes (or kernelized decision surfaces) to achieve strong separation even when classes overlap.

Hyperparameter optimization was conducted using GridSearchCV on the combined training dataset derived from the openSMILE-extracted features (i.e. the union of the original training and validation sets) with internal 5-fold cross-validation. By restricting all parameter tuning to the training folds, this protocol achieves the following:

- Avoids overfitting to a single validation split.
- Prevents information "leakage" from the held-out validation set into the model.
- Maximizes the data available for both hyperparameter selection and final model fitting.

The unseen test set remains reserved for an unbiased evaluation of each classifier's generalization performance.

5.2.1 K-Nearest Neighbors (KNN)

Overview and Hyperparameter Tuning

KNN classifies each sample by a majority vote of its k closest points in feature space. We used the 129 principal-component features (PC1–PC129) and tuned three key hyperparameters using a full grid search (5-fold CV):

Hyperparameter	Tried Values	Best Value
$n_{neighbors}$	1 to 15	15
Weights	uniform, distance	distance
Metrics	euclidean, manhattan	manhattan

TABLE 5.2: Best Hyperparameter Values for KNN

Test Accuracy	Test Precision	Test Recall	Test F1-score
0.6466	0.6503	0.6466	0.6429

TABLE 5.3: Final Test Performance using Best KNN Model

- $n_{neighbors}$: 1 to 15
- Weights: uniform, distance
- Metrics: euclidean, manhattan

The best K-Nearest Neighbors configuration ($n_{neighbors}$ = 15, weights = "distance", metric = "manhattan") achieved a mean 5-fold CV accuracy of 0.6312 before final test-set evaluation, as seen in Figure 5.2.

Model Training and Performance

With the tuned hyperparameters in hand ($n_{neighbors}$ = 15, weights = "Distance", metric = "Manhattan"), we now evaluate the best KNN model on the reserved test dataset. This provides an unbiased estimate of our model's performance, shown in Table 5.3 and Figure 5.1.

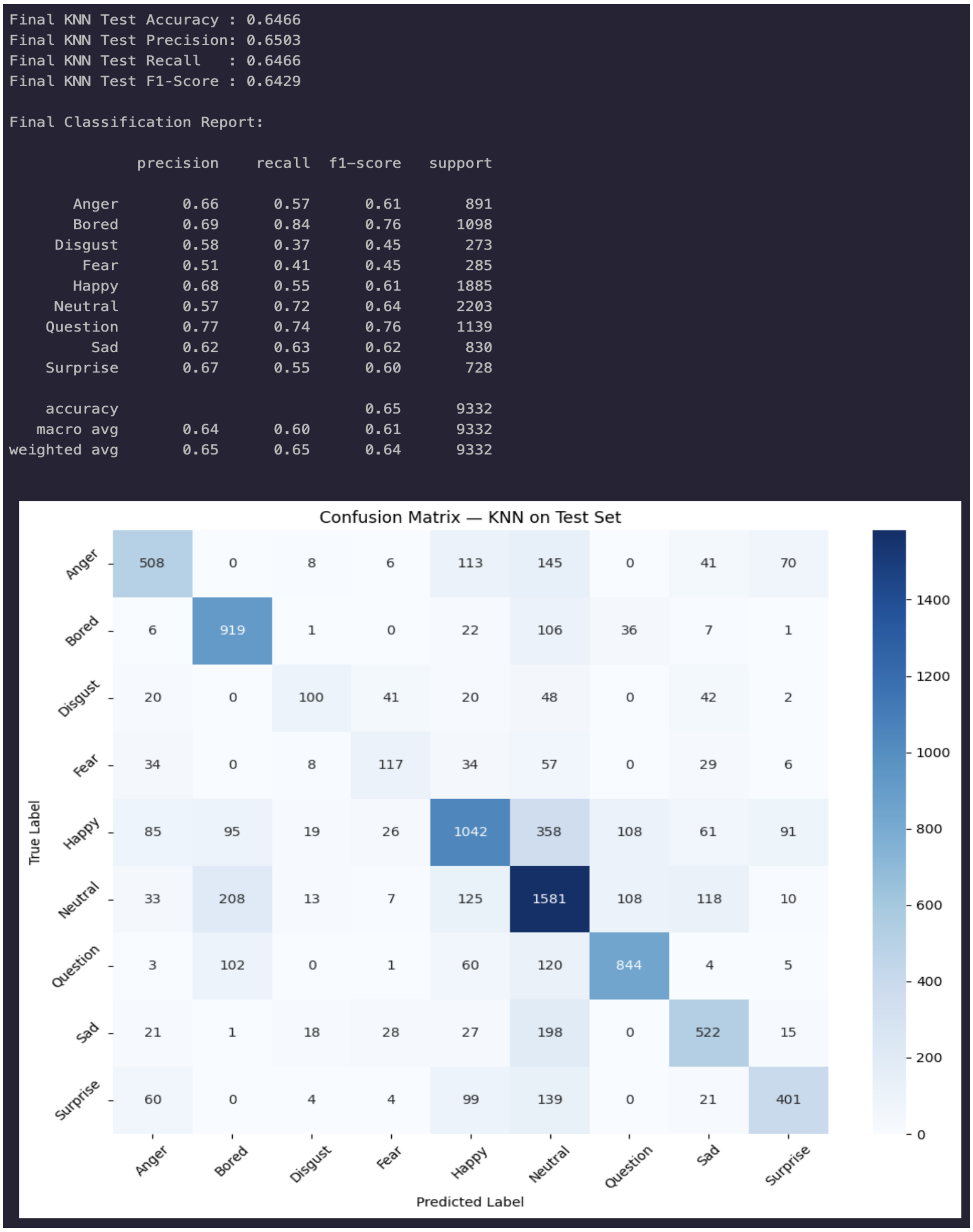
From Figure 5.1, although the tuned KNN classifier achieves only moderate performance (Test Accuracy of 0.6466, Test F1-score of 0.6429), these results suggest that pure instance-based voting on the PC feature set does not capture the full complexity of the speech-emotion boundaries. In particular, the sub-0.65 F1-score indicates that nearly one-third of samples remain misclassified, and KNN's sensitivity to class imbalance—especially for less-represented emotions—limits its practical utility in our application. Therefore, other classical models need to be explored to improve generalization and boost recognition rates across all emotion categories.

5.2.2 Random Forest (RF)

Overview and Hyperparameter Tuning

RF builds an ensemble of decision trees on bootstrapped samples, using random feature subsets to improve generalization. We used the 129 principal-component features (PC1–PC129) and tuned four key hyperparameters using a full grid search (5-fold CV):

- $n_{estimators}$: 100, 200
- max_{depth}: none, 10, 20
- $min_{samples\_split}$: 2, 5
- $max_{features}$: sqrt, log2



```
Final KNN Test Accuracy : 0.6466
Final KNN Test Precision: 0.6503
Final KNN Test Recall   : 0.6466
Final KNN Test F1-Score : 0.6429

Final Classification Report:

              precision    recall  f1-score   support

       Anger       0.66      0.57      0.61       891
       Bored       0.69      0.84      0.76      1098
     Disgust       0.58      0.37      0.45       273
        Fear       0.51      0.41      0.45       285
       Happy       0.68      0.55      0.61      1885
     Neutral       0.57      0.72      0.64      2203
    Question       0.77      0.74      0.76      1139
         Sad       0.62      0.63      0.62       830
    Surprise       0.67      0.55      0.60       728

    accuracy                           0.65      9332
   macro avg       0.64      0.60      0.61      9332
weighted avg       0.65      0.65      0.64      9332
```

Confusion Matrix — KNN on Test Set

True Label \ Predicted Label	Anger	Bored	Disgust	Fear	Happy	Neutral	Question	Sad	Surprise
Anger	508	0	8	6	113	145	0	41	70
Bored	6	919	1	0	22	106	36	7	1
Disgust	20	0	100	41	20	48	0	42	2
Fear	34	0	8	117	34	57	0	29	6
Happy	85	95	19	26	1042	358	108	61	91
Neutral	33	208	13	7	125	1581	108	118	10
Question	3	102	0	1	60	120	844	4	5
Sad	21	1	18	28	27	198	0	522	15
Surprise	60	0	4	4	99	139	0	21	401

FIGURE 5.1: Complete Model Performance Report with Classification Report and Confusion Matrix for KNN Model

Hyperparameter	**Tried Values**	**Best Value**
$n_{estimators}$	100, 200	200
max_{depth}:	none, 10, 20	none
$min_{samples\_split}$	2, 5	2
$max_{features}$	sqrt, log2	sqrt

TABLE 5.4: Best Hyperparameter Values for RF

Test Accuracy	**Test Precision**	**Test Recall**	**Test F1-score**
0.6443	0.6491	0.6443	0.6402

TABLE 5.5: Final Test Performance using Best RF Model

The best Random Forest configuration ($n_{estimators}$ = 200, max_{depth} = none, $min_{samples\_split}$ = 2, $max_{features}$ = “sqrt”), as seen in Table 5.4 achieved a mean 5-fold CV accuracy of 0.6381 before final test-set evaluation.

Model Training and Performance

With the tuned hyperparameters in hand ($n_{estimators}$ = 200, max_{depth} = None, $min_{samples\_split}$ = 2, $max_{features}$ = “sqrt”), we now evaluate the best RF model on the reserved test dataset. This provides an unbiased estimate of our model's performance, as shown in Table 5.5 and Figure 5.2.

From Figure 5.2, despite its ensemble strength, the tuned RF slightly trails the KNN baseline—achieving a test accuracy of 0.6443 (vs. 0.6466 for KNN) and an F1-score of 0.6402 (vs. 0.6429 for KNN). Both classifiers remain under the 0.65 F1-score threshold and continue to misclassify under-represented emotions. To seek further gains in generalization and per-class recognition, we evaluate the final shortlisted classical method: the SVM.

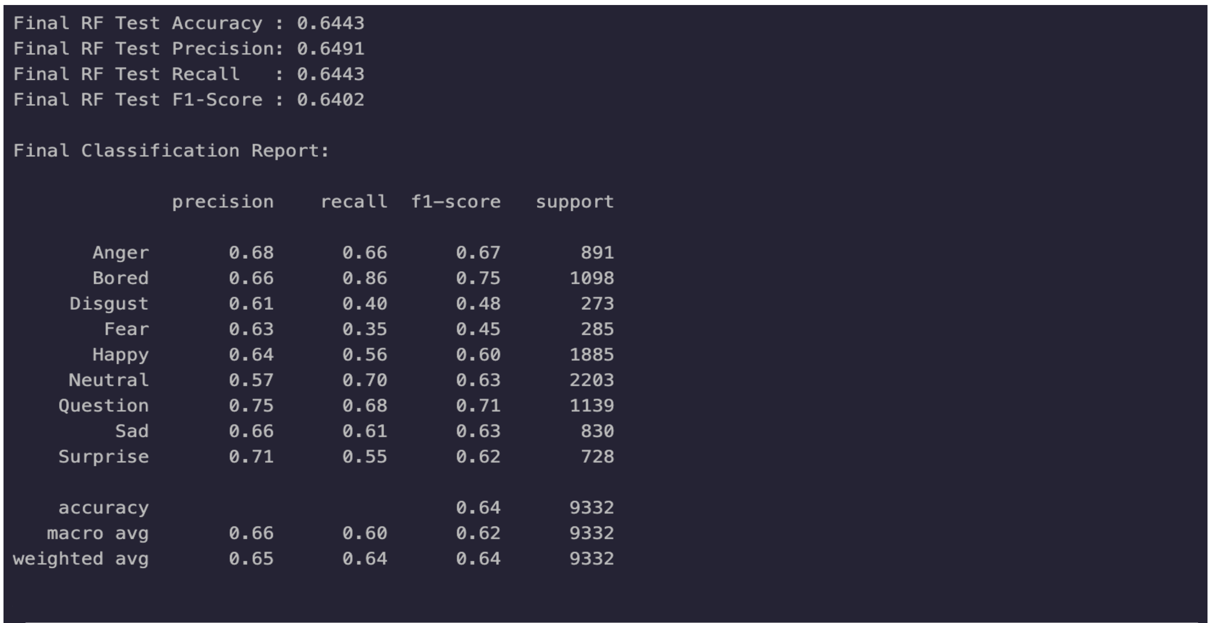
5.2.3 SVMs (Support Vector Machines)

Overview and Hyperparameter Tuning

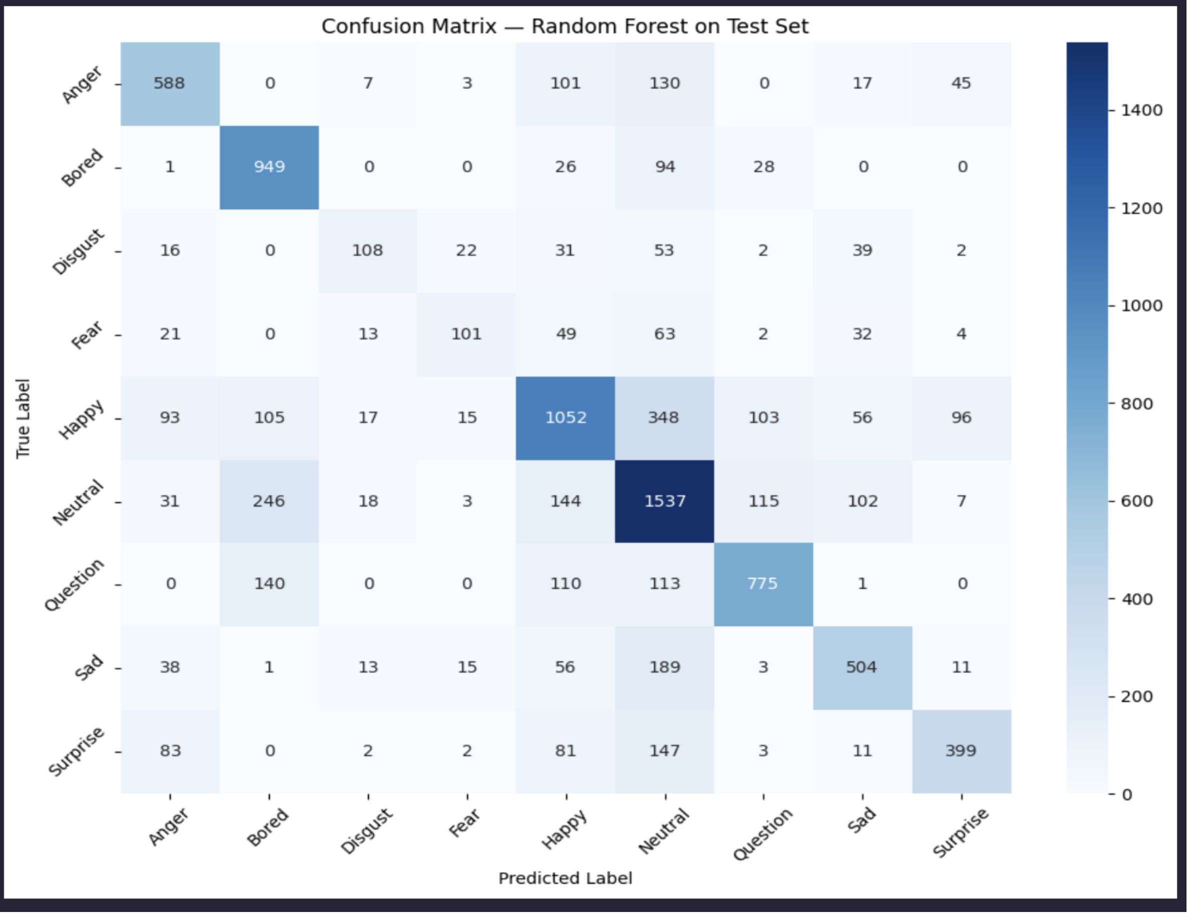
SVM constructs a maximum-margin hyperplane (or kernel-mapped decision boundary) to separate classes. We used the 129 principal-component features (PC1–PC129) and tuned three key hyperparameters using a full grid search (5-fold CV):

- C: 0.1, 1, 10
- Kernel: linear, rbf
- Gamma: scale, auto , 0.01, 0.1

The best SVM configuration (C = 10, kernel = “rbf”, gamma = “scale”), as seen in Table 5.6 achieved a mean 5-fold CV accuracy of 0.7069 before final test-set evaluation.



```
Final RF Test Accuracy : 0.6443
Final RF Test Precision: 0.6491
Final RF Test Recall   : 0.6443
Final RF Test F1-Score : 0.6402

Final Classification Report:

              precision    recall  f1-score   support

       Anger       0.68      0.66      0.67       891
       Bored       0.66      0.86      0.75      1098
     Disgust       0.61      0.40      0.48       273
        Fear       0.63      0.35      0.45       285
       Happy       0.64      0.56      0.60      1885
     Neutral       0.57      0.70      0.63      2203
    Question       0.75      0.68      0.71      1139
         Sad       0.66      0.61      0.63       830
    Surprise       0.71      0.55      0.62       728

    accuracy                           0.64      9332
   macro avg       0.66      0.60      0.62      9332
weighted avg       0.65      0.64      0.64      9332
```



True \ Predicted	Anger	Bored	Disgust	Fear	Happy	Neutral	Question	Sad	Surprise
Anger	588	0	7	3	101	130	0	17	45
Bored	1	949	0	0	26	94	28	0	0
Disgust	16	0	108	22	31	53	2	39	2
Fear	21	0	13	101	49	63	2	32	4
Happy	93	105	17	15	1052	348	103	56	96
Neutral	31	246	18	3	144	1537	115	102	7
Question	0	140	0	0	110	113	775	1	0
Sad	38	1	13	15	56	189	3	504	11
Surprise	83	0	2	2	81	147	3	11	399

FIGURE 5.2: Complete Model Performance Report with Classification Report and Confusion Matrix for RF Model

Hyperparameter	Tried Values	Best Value
C	0.1, 1, 10	10
Kernel	linear, rbf	rbf
Gamma	scale, auto, 0.01, 0.1	scale

TABLE 5.6: Best Hyperparameter Values for SVM

Test Accuracy	Test Precision	Test Recall	Test F1-score
0.7166	0.7175	0.7166	0.7156

TABLE 5.7: Final Test Performance using Best SVM Model

Model Training and Performance

With the tuned hyperparameters in hand (C = 10, kernel = “rbf”, gamma = “scale”), we now evaluate the best SVM model on the reserved test dataset. This provides an unbiased estimate of our model’s performance, as seen in Table 5.7 and Figure 5.3.

From Figure 5.3, the tuned SVM delivers a clear step-change in performance—achieving 0.7166 accuracy and a 0.7156 test F1-score on the test set, compared with roughly 0.64 for both KNN (test F1-score = 0.6429) and RF (test F1-score = 0.6402). This improvement demonstrates that the RBF-kernel’s ability to carve non-linear boundaries in PC space yields better generalization across most emotion classes. Nevertheless, a test F1-score in the low 0.70s still leaves substantial room for error, particularly on minority emotions. To push beyond this plateau, we explored FFNNs next, which—by learning hierarchical feature representations—may further increase recognition rates across all emotion classes.

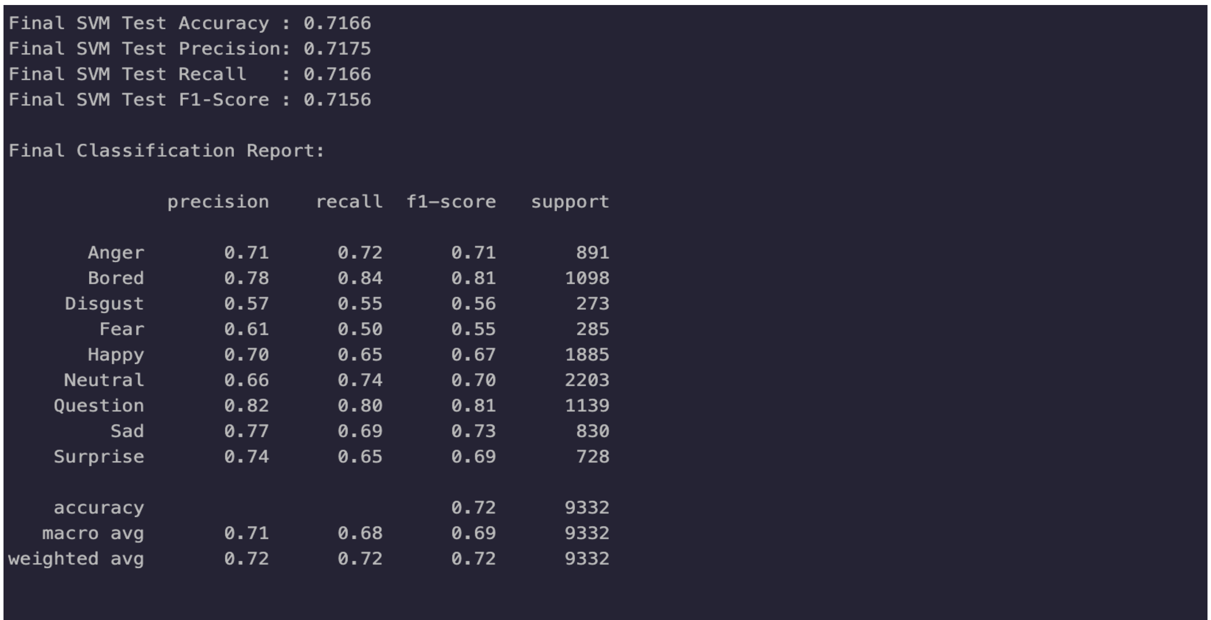
5.3 Feed-forward Neural Networks (FFNNs)

This section explores the use of basic FFNNs, where the core concept is employing fully-connected layer(s) to map the input features to output labels. Since the concept is relatively simple, our approach to training these FFNNs was to tune a number of hyperparameters. Table 5.8 lists what hyperparameters there are, and what kind of values we have tuned them with. Across all these models, the common output layer outputs nine labels, each corresponding to an emotional class.

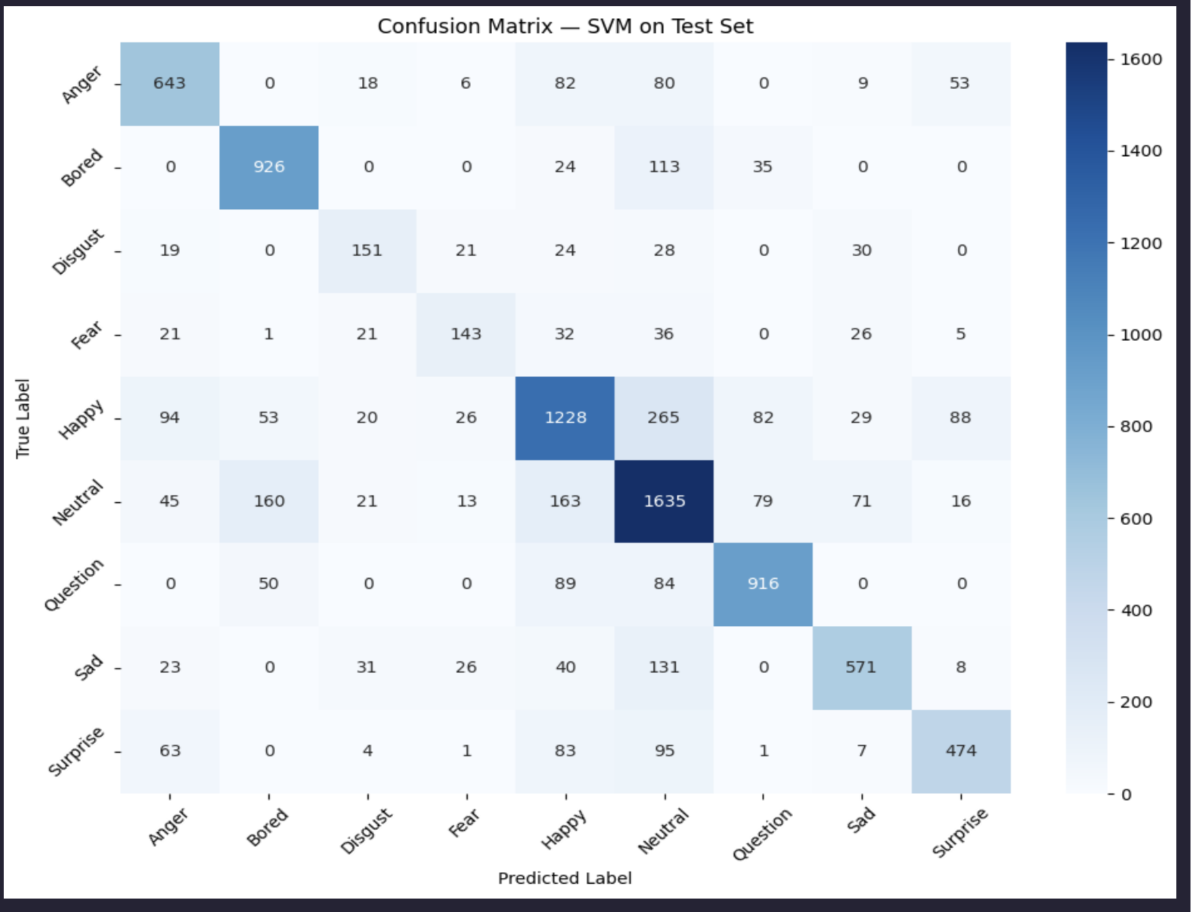
The subsequent sections describes some ablation studies we did on each of these hyperparameters, with all others staying constant. The tuning was done with Optuna, which was chosen due to its ability to intelligently pick what hyperparameters to tune via methods such

Hyperparameter	Values
Number of Principal Components	[100, 1000], in steps of 100
Hidden Dimension Size	2^x, where x is an integer between [4, 9]
Dropout	[0.1, 0.5], in steps of 0.1
Number of Layers	1, 2, or 3
Learning rate	0.00001, 0.0001 or 0.01

TABLE 5.8: The Hyperparameters we Experimented with for FFNNs



```
Final SVM Test Accuracy : 0.7166
Final SVM Test Precision: 0.7175
Final SVM Test Recall   : 0.7166
Final SVM Test F1-Score : 0.7156

Final Classification Report:

              precision    recall  f1-score   support

       Anger       0.71      0.72      0.71       891
       Bored       0.78      0.84      0.81      1098
     Disgust       0.57      0.55      0.56       273
        Fear       0.61      0.50      0.55       285
       Happy       0.70      0.65      0.67      1885
     Neutral       0.66      0.74      0.70      2203
    Question       0.82      0.80      0.81      1139
         Sad       0.77      0.69      0.73       830
    Surprise       0.74      0.65      0.69       728

    accuracy                           0.72      9332
   macro avg       0.71      0.68      0.69      9332
weighted avg       0.72      0.72      0.72      9332
```



	Anger	Bored	Disgust	Fear	Happy	Neutral	Question	Sad	Surprise
Anger	643	0	18	6	82	80	0	9	53
Bored	0	926	0	0	24	113	35	0	0
Disgust	19	0	151	21	24	28	0	30	0
Fear	21	1	21	143	32	36	0	26	5
Happy	94	53	20	26	1228	265	82	29	88
Neutral	45	160	21	13	163	1635	79	71	16
Question	0	50	0	0	89	84	916	0	0
Sad	23	0	31	26	40	131	0	571	8
Surprise	63	0	4	1	83	95	1	7	474

FIGURE 5.3: Complete Model Performance Report with Classification Report and Confusion Matrix for SVM Model

Hyperparameter	**Values**
Hidden Dimension Size	512
Dropout	0.3
Number of Layers	1
Learning Rate	0.0001

TABLE 5.9: Values used for Tuning Number of Principal Components

	Number of Principal Components			
	500	**600**	**700**	**800**
Test Accuracy	0.722	0.727	0.715	0.712

TABLE 5.10: Test Accuracy of FFNN with Varying Number of Principal Components

as pruning. Our model was trained on a separate training set, and validated on a validation set. The validation set is used to evaluate the models during training and the accuracy obtained on the validation set is the criteria for early stopping. The test set was reserved to evaluate the model performance in subsequent sections.

5.3.1 Hyperparameter: Number of Principal Components

This hyperparameter controls the number of principal components used as the model input during the preprocessing step, as mentioned in Section 5.1. A larger number means selecting more principal components which could be more descriptive. Table 5.9 shows the values that were used for tuning.

The results shown in Table 5.10 were rather similar to each other, with a slightly higher test accuracy when the number of principal components is closer to the hidden dimension size (512). The test accuracy also slightly tapers off as the number increases and gets further away from the hidden dimension size. Hence, the key finding for this hyperparameter is: keep the number of principal components closer to hidden dimension size.

5.3.2 Hyperparameter: Hidden Dimension Size

This hyperparameter controls the size of the input to the FFNN layer(s). For this study, we kept the number of layers to just 1. Table 5.11 shows the values that were used for tuning.

The results shown in Table 5.12 were rather similar to each other, with a slightly higher test accuracy when the hidden dimension size is closer to the number of principal components

Hyperparameter	**Values**
Number of Principal Components	700
Dropout	0.3
Number of Layers	1
Learning Rate	0.0001

TABLE 5.11: Values used for Tuning Hidden Dimension Size

	Hidden Dimension Size	
	256	**512**
Test Accuracy	0.706	0.715

TABLE 5.12: Test Accuracy of FFNN with Varying Hidden Dimension Size

Hyperparameter	**Values**
Number of Principal Components	700
Hidden Dimension Size	256
Number of Layers	1
Learning Rate	0.0001

TABLE 5.13: Values used for Tuning Dropout

(700). This corroborates the previous study. Hence, the key finding for this hyperparameter is: keep both hidden dimension size (assuming 1 layer) and number of principal components close.

5.3.3 Hyperparameter: Dropout

This hyperparameter controls the probability of dropout in each of the FFNN layer(s). For this study, we kept the number of layers to just 1. Table 5.13 shows the values that were used for tuning.

The results shown in Table 5.14 were rather similar to each other, and the test accuracies implied that a lower dropout improves model performance. Hence, the key finding for this hyperparameter is to select a slightly lower dropout.

5.3.4 Hyperparameter: Number of Layers

This hyperparameter controls the number of FFNN layer(s). We did not configure this hyperparameter too much, as it was empirically found that a single layer generally worked better and had more stable training. For this study, we focus on the number of layers, although their hidden dimensions were important as well. Table 5.15 shows the values that were used for tuning.

As seen in the results in Table 5.16, the model with three layers was about 4 percentage points lower in test accuracy, and in general, these multi-layer networks perform worse than single-layer FFNNs.

	Dropout Probability		
	0.1	**0.3**	**0.4**
Test Accuracy	0.710	0.706	0.705

TABLE 5.14: Test Accuracy of FFNN with Varying Dropout

Hyperparameter	**Values**
Number of Principal Components	700
Dropout	0.4
Learning Rate	0.0001

TABLE 5.15: Values used for Tuning Number of Layers

	Layer Configuration	
	2 Layers [128,64]	**3 Layers [128,32,16]**
Test Accuracy	0.663	0.620

TABLE 5.16: Test Accuracy of FFNN with Varying Number of Layers

5.3.5 Hyperparameter: Learning Rate

This hyperparameter determines the step size taken during the optimization process for model training. Table 5.17 shows the values that were used for tuning.

This study shows the effect of learning rate on the model. As seen in the results in Table 5.18, while a small learning rate might help the model in preventing overfitting, it converges slowly and also might lead the model to optimizing to a local minimum. The best learning rate for our experiments was 0.0001, which is sufficiently small but prevents the aforementioned problems.

5.3.6 Best FFNN Model

The best FFNN model that we have achieved has the parameters as shown in Table 5.19.

These values were empirically discovered via the Optuna hyperparameter tuning regime, yet it combines the key findings of the separate ablation studies detailed above such as:

- Hidden dimension size and number of principal components being similar.
- Favouring slightly lower dropout.
- Optimal learning rate of 0.0001.

Figure 5.4 details the classification report, while Figure 5.5 shows the confusion matrix, breaking down by each emotion label.

Table 5.20 summarizes the performance of our best FFNN model, where the test F1-score is 0.7218.

Hyperparameter	**Values**
Number of Principal Components	200
Hidden Dimension Size	256
Number of Layers	1
Dropout	0.1

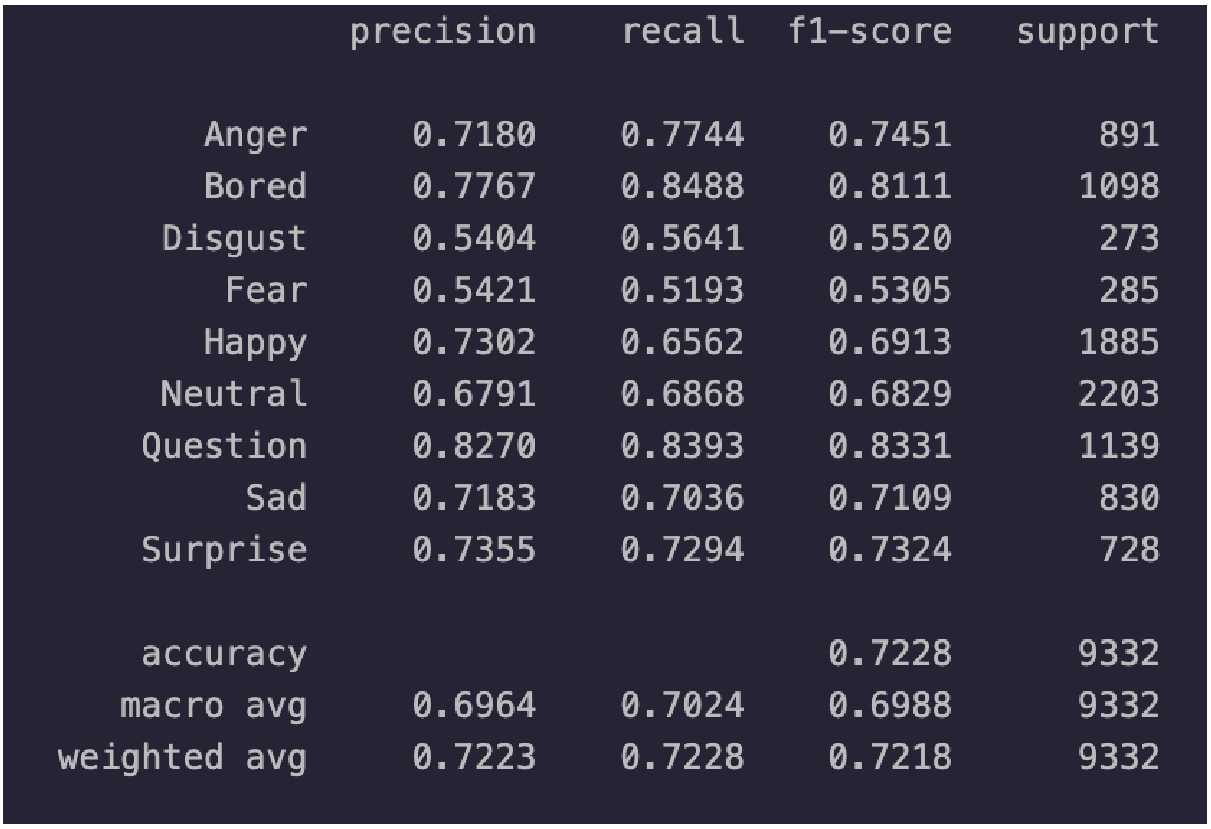
TABLE 5.17: Values used for Tuning Learning Rate

	Learning Rate		
	0.00001	**0.0001**	**0.01**
Test Accuracy	0.604	0.699	0.692

TABLE 5.18: Test Accuracy of FFNN with Varying Learning Rate

Hyperparameter	**Values**
Number of Principal Components	600
Hidden Dimension Size	512
Dropout	0.2
Number of Layers	1
Learning Rate	0.0001

TABLE 5.19: Parameters used in our Best FFNN Model

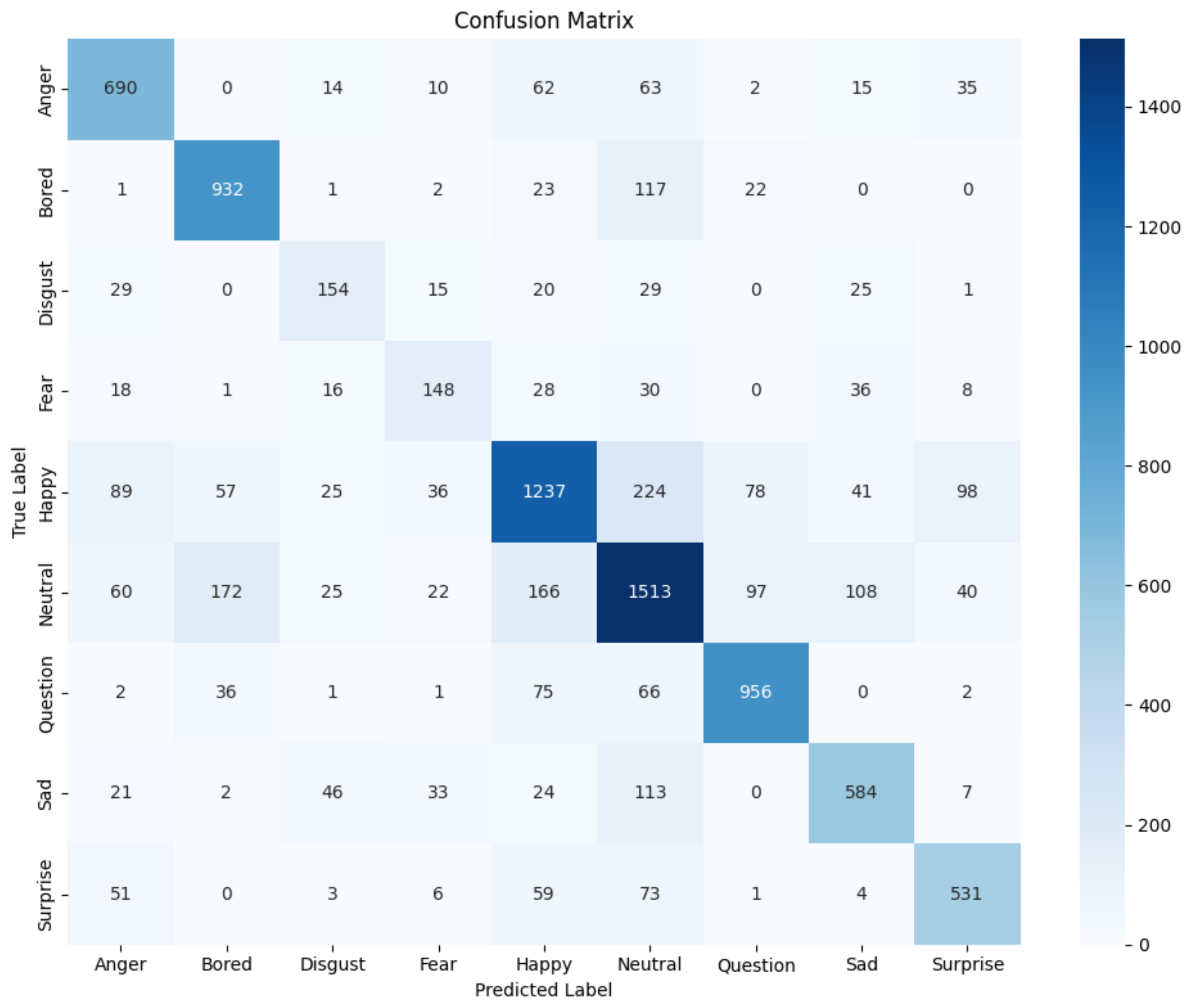


```
              precision    recall  f1-score   support

       Anger     0.7180    0.7744    0.7451       891
       Bored     0.7767    0.8488    0.8111      1098
     Disgust     0.5404    0.5641    0.5520       273
        Fear     0.5421    0.5193    0.5305       285
       Happy     0.7302    0.6562    0.6913      1885
     Neutral     0.6791    0.6868    0.6829      2203
    Question     0.8270    0.8393    0.8331      1139
         Sad     0.7183    0.7036    0.7109       830
    Surprise     0.7355    0.7294    0.7324       728

    accuracy                         0.7228      9332
   macro avg     0.6964    0.7024    0.6988      9332
weighted avg     0.7223    0.7228    0.7218      9332
```

FIGURE 5.4: Classification Report of Best FFNN Model

	Test Accuracy	**Test Precision**	**Test Recall**	**Test F1-score**
Best FFNN Model	0.7228	0.7223	0.7228	0.7218

TABLE 5.20: Model Performance of our Best FFNN Model



True Label \ Predicted Label	Anger	Bored	Disgust	Fear	Happy	Neutral	Question	Sad	Surprise
Anger	690	0	14	10	62	63	2	15	35
Bored	1	932	1	2	23	117	22	0	0
Disgust	29	0	154	15	20	29	0	25	1
Fear	18	1	16	148	28	30	0	36	8
Happy	89	57	25	36	1237	224	78	41	98
Neutral	60	172	25	22	166	1513	97	108	40
Question	2	36	1	1	75	66	956	0	2
Sad	21	2	46	33	24	113	0	584	7
Surprise	51	0	3	6	59	73	1	4	531

FIGURE 5.5: Confusion Matrix of Best FFNN Model

Chapter 6

Deep Learning Approaches

6.1 Own-built CNNs

6.1.1 MFCCs

Architecture Design

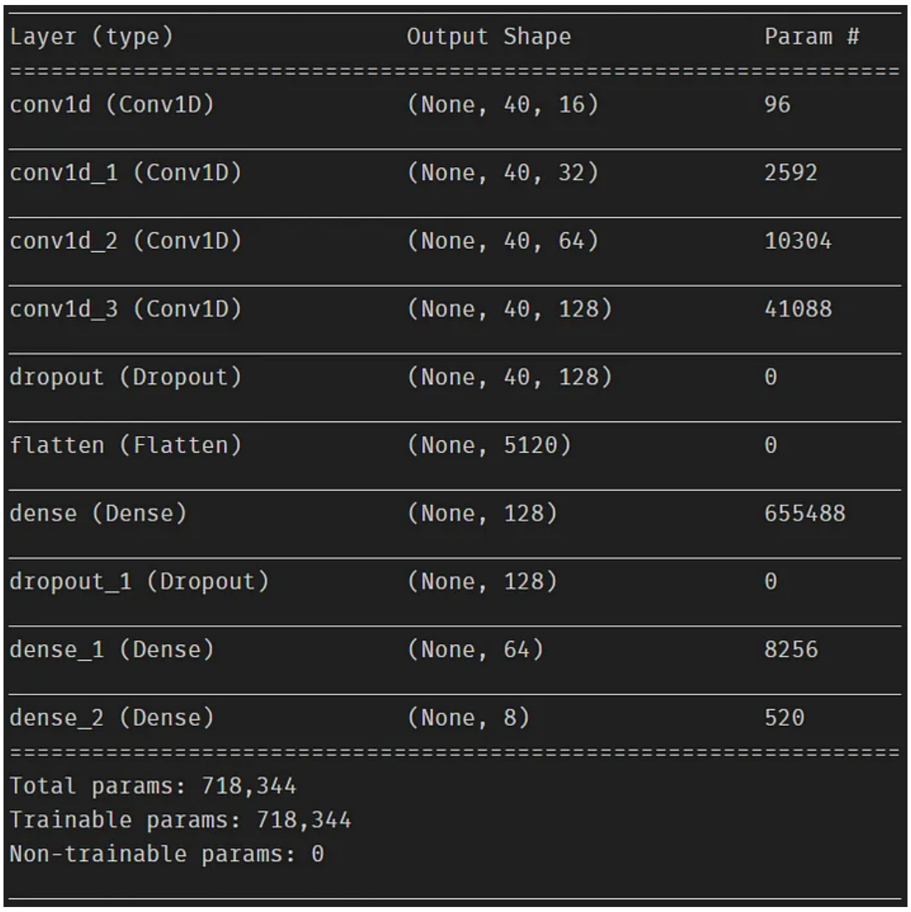
This approach was inspired by Diego's work on SER using CNNs. In this work, using librosa, Diego extracted 40 Mel Frequency Cepstral Coefficients (MFCCs) from each audio frame of each audio sample at a sampling rate of 22,050 Hz. These MFCCs were then averaged across all audio frames to get 40 MFCCs that represent the entire duration of each audio sample. This means that the resulting MFCC data is one-dimensional. MFCCs capture audio features similar to how humans perceive sound and frequencies. Diego then built a 1-dimensional CNN, whose architecture is shown in Figure 6.1 (Diego, 2020).

We replicated Diego's work by extracting 40 MFCCs from all audio samples. However, instead of averaging them across all audio frames, we kept them as they were. Additionally, the audio samples were of varying durations from 1 to 4 seconds, and since CNNs required fixed-sized inputs, we zero-padded these tensors to 4 seconds. This resulted in tensors with the same shape across all samples, which we saved as tensor files (.pt). We then adapted Diego's CNN architecture (Figure 6.1) to the MFCCs tensors we extracted. The architecture, which we named CNNMFCCs, has a dropout of 0.1 and the ReLU activation function, and is illustrated in Figure 6.2.

We tried changing the CNNMFCCs architecture in hopes of increasing model performance, and created the CNNMFCCs2 architecture, which has batch normalization, max pooling, global average pooling, a greater number of filters in the convolutional layers, higher dropout of 0.3, and fewer fully-connected layers than CNNMFCCs. The CNNMFCCs3 architecture was also created, but instead of the ReLU activation function, the SiLU activation function was used.

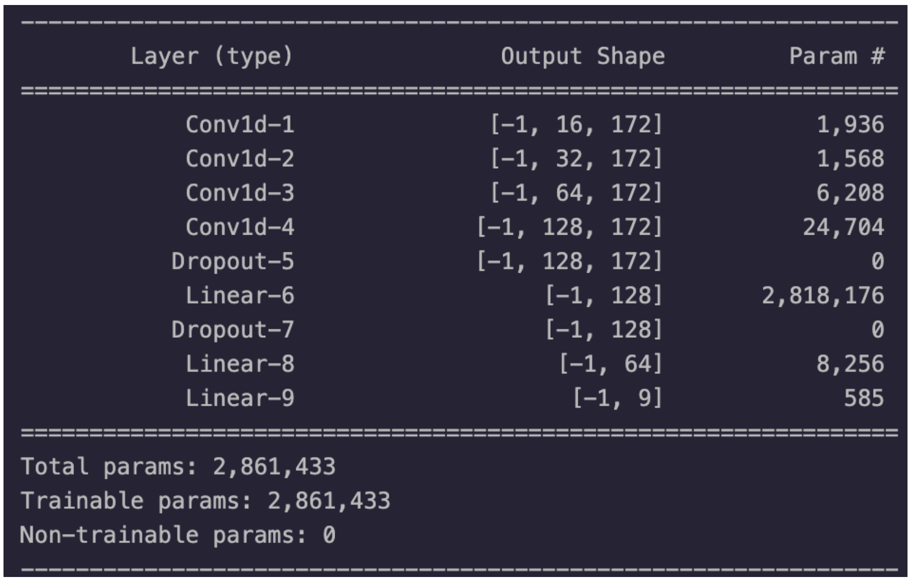
Model Training and Performance

Using the Adam optimizer, a learning rate of 0.001, weighted cross-entropy loss as the criterion, and early stopping with a patience of 7, we trained the CNN architectures. For each training epoch, we trained the network using the training set and validated using the validation set. Additionally, through model checkpointing, we saved the model weights as a .pt file if the model of this epoch exceeded the maximum validation accuracy so far. We then evaluated the test set, which was unseen during training, on the saved best model weights, and obtained the results in Table 6.1.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 40, 16)            96
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 40, 32)            2592
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 40, 64)            10304
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 40, 128)           41088
_________________________________________________________________
dropout (Dropout)            (None, 40, 128)           0
_________________________________________________________________
flatten (Flatten)            (None, 5120)              0
_________________________________________________________________
dense (Dense)                (None, 128)               655488
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 520
=================================================================
Total params: 718,344
Trainable params: 718,344
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 6.1: Diego's CNN Architecture (Diego, 2020)



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv1d-1              [-1, 16, 172]           1,936
            Conv1d-2              [-1, 32, 172]           1,568
            Conv1d-3              [-1, 64, 172]           6,208
            Conv1d-4             [-1, 128, 172]          24,704
           Dropout-5             [-1, 128, 172]               0
            Linear-6                  [-1, 128]       2,818,176
           Dropout-7                  [-1, 128]               0
            Linear-8                   [-1, 64]           8,256
            Linear-9                    [-1, 9]             585
================================================================
Total params: 2,861,433
Trainable params: 2,861,433
Non-trainable params: 0
----------------------------------------------------------------
```

FIGURE 6.2: Our CNNMFCCs Architecture

CNN Architecture	Test Accuracy	Test Precision	Test Recall	Test F1-score	Remarks
CNNMFCCs	0.6229	0.6434	0.6229	0.6223	Base architecture, with ReLU activation function and dropout of 0.1
CNNMFCCs2	0.7328	0.7468	0.7328	0.7352	With batch normalization, max pooling, global average pooling, more filters in convolutional layers, higher dropout of 0.3
CNNMFCCs3	0.7334	0.7475	0.7334	0.7325	With SiLU activation function

TABLE 6.1: Model Performance of our Own-built CNNs for MFCCs

6.1.2 MFCCs + Delta + Delta-Delta

Architecture Design

Additionally, MFCCs + Delta + Delta-Delta were extracted from the audio samples in a separate work by Venkataramanan and Rajamohan. These delta coefficients describe the rate of change of MFCC features, while the delta-delta coefficients provide insights for speech tasks, and hence, they were stacked with the base MFCCs in hopes of better model performance (Venkataramanan and Rajamohan, 2019). Thus, we replicated this work by extracting MFCCs + Delta + Delta-Delta using librosa, and zero-padded them to the maximum duration of the audio samples, and saved them as tensors (.pt) files. We then adapted our CNNMFCCs architecture to fit these MFCCs + Delta + Delta-Delta tensors, where we built the CNNMFCCsDelta architecture, a two-dimensional CNN with a dropout of 0.1, as illustrated in Figure 6.3.

Just like what we did previously for the CNNMFCCs architecture, we also created the CNNMFCCsDelta2 architecture, which had 2-dimensional batch normalization, global average pooling, and a higher dropout of 0.3.

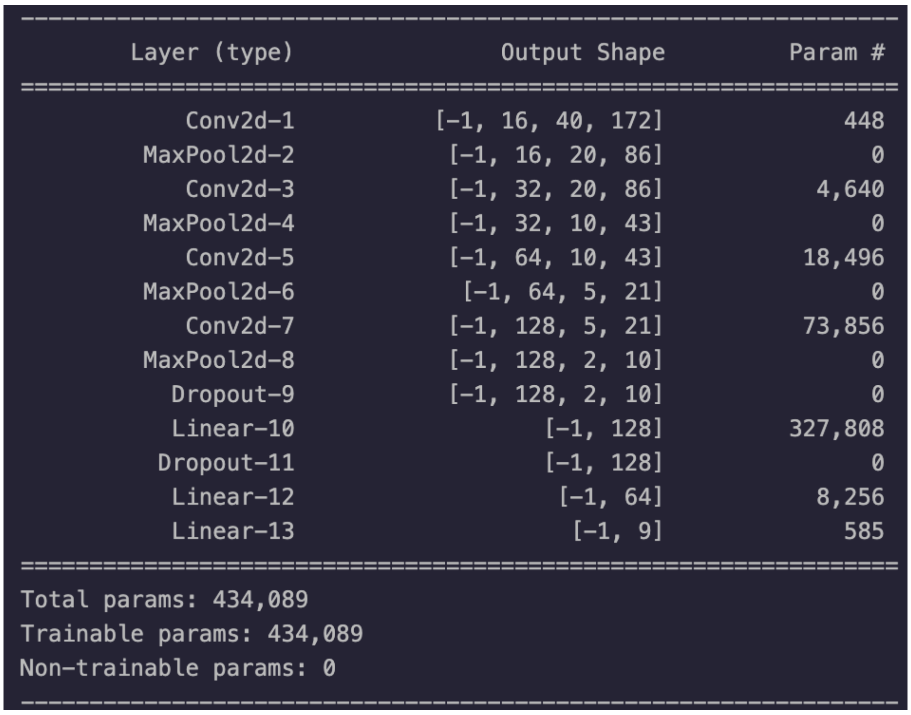
Model Training and Performance

Using the same training parameters and pipeline from before, we trained these CNN architectures and obtained the results in Table 6.2.

From the results, extracting MFCCs + Delta + Delta-Delta and using them as inputs resulted in slightly better model performance over using just MFCCs, based on the Test F1-score of 0.7402. The results of the training can be seen in Figure 6.4.

6.2 Fine-tuned Pre-trained CNNs

Instead of building and training our CNNs from scratch, we leveraged transfer learning and fine-tuned pre-trained CNNs in hopes of bringing model performance up another level. Several pre-trained CNNs have been used for SER, such as ResNet50 (Ayadi and Lachiri, 2022)

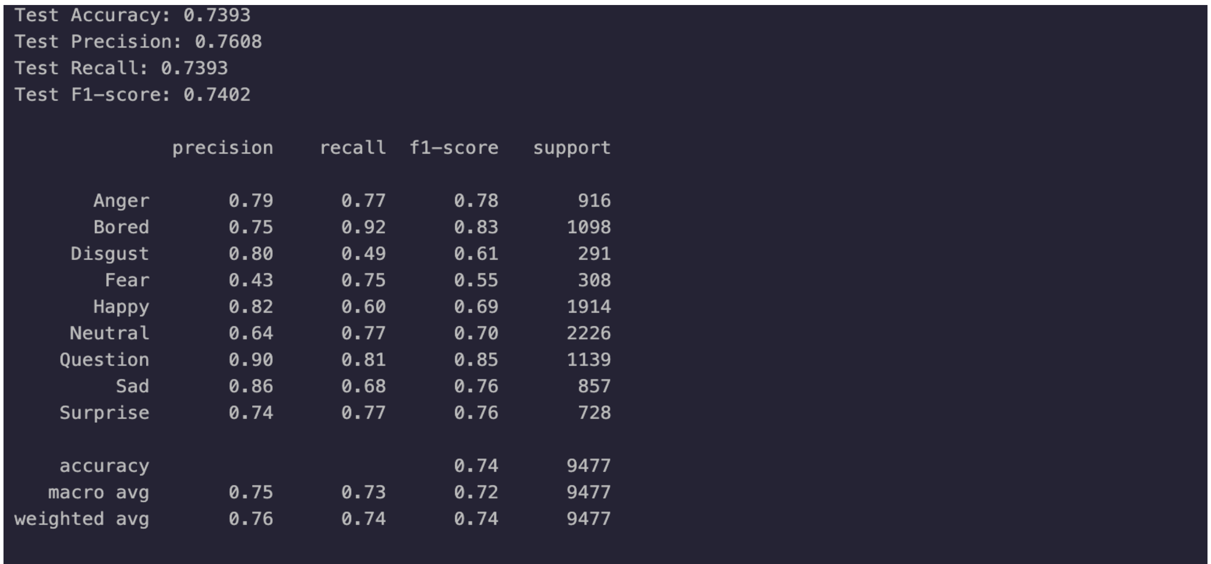


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 40, 172]             448
         MaxPool2d-2          [-1, 16, 20, 86]               0
            Conv2d-3          [-1, 32, 20, 86]           4,640
         MaxPool2d-4          [-1, 32, 10, 43]               0
            Conv2d-5          [-1, 64, 10, 43]          18,496
         MaxPool2d-6           [-1, 64, 5, 21]               0
            Conv2d-7          [-1, 128, 5, 21]          73,856
         MaxPool2d-8          [-1, 128, 2, 10]               0
           Dropout-9          [-1, 128, 2, 10]               0
           Linear-10                 [-1, 128]         327,808
          Dropout-11                 [-1, 128]               0
           Linear-12                  [-1, 64]           8,256
           Linear-13                   [-1, 9]             585
================================================================
Total params: 434,089
Trainable params: 434,089
Non-trainable params: 0
----------------------------------------------------------------
```

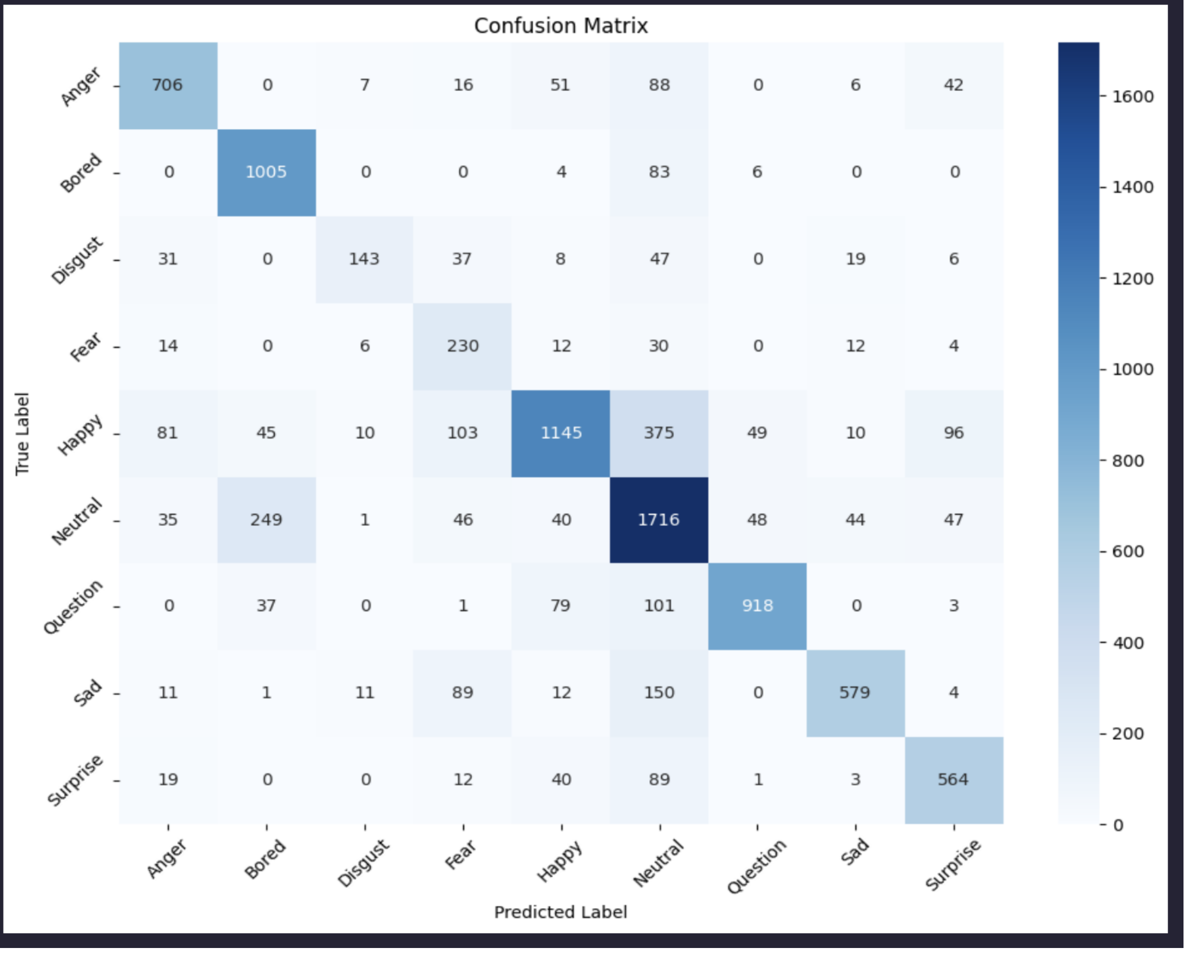
FIGURE 6.3: Our CNNMFCCsDelta Architecture

CNN Architecture	**Test Accuracy**	**Test Precision**	**Test Recall**	**Test F1-score**	**Remarks**
CNNMFCCsDelta	0.7155	0.7346	0.6155	0.7154	Base architecture, with ReLU activation function and dropout of 0.1
CNNMFCCsDelta2	0.7393	0.7608	0.7393	0.7402	With 2-dimensional batch normalization, global average pooling, and higher dropout of 0.3

TABLE 6.2: Model Performance of our Own-built CNNs for MFCCs + Delta + Delta-Delta



```
Test Accuracy: 0.7393
Test Precision: 0.7608
Test Recall: 0.7393
Test F1-score: 0.7402

              precision    recall  f1-score   support

       Anger       0.79      0.77      0.78       916
       Bored       0.75      0.92      0.83      1098
     Disgust       0.80      0.49      0.61       291
        Fear       0.43      0.75      0.55       308
       Happy       0.82      0.60      0.69      1914
     Neutral       0.64      0.77      0.70      2226
    Question       0.90      0.81      0.85      1139
         Sad       0.86      0.68      0.76       857
    Surprise       0.74      0.77      0.76       728

    accuracy                           0.74      9477
   macro avg       0.75      0.73      0.72      9477
weighted avg       0.76      0.74      0.74      9477
```



True \ Predicted	Anger	Bored	Disgust	Fear	Happy	Neutral	Question	Sad	Surprise
Anger	706	0	7	16	51	88	0	6	42
Bored	0	1005	0	0	4	83	6	0	0
Disgust	31	0	143	37	8	47	0	19	6
Fear	14	0	6	230	12	30	0	12	4
Happy	81	45	10	103	1145	375	49	10	96
Neutral	35	249	1	46	40	1716	48	44	47
Question	0	37	0	1	79	101	918	0	3
Sad	11	1	11	89	12	150	0	579	4
Surprise	19	0	0	12	40	89	1	3	564

FIGURE 6.4: Complete Model Performance Report with Classification Report and Confusion Matrix for Own-built CNN

and EfficientNet (Chakhtouna, Sekkate, and Adib, 2023), which are popular for image classification tasks (Yan, 2023). Mel spectrogram images have been used in SER tasks (Begazo et al., 2024), and hence, for this approach, we generated these images from the audio samples and inputted them into the pre-trained CNNs for fine-tuning and evaluation. However, there are endless ways to pre-process the audio samples, and many pre-trained models we could fine-tune. Therefore, we developed a methodical workflow for this approach to simplify the process, and limited the pre-trained CNNs we wanted to fine-tune to just ResNet50 and EfficientNetB0. The following sections detail the stages of this workflow.

6.2.1 Finding the Optimal Data Preprocessing Technique

Data Preprocessing Techniques

Pre-trained CNNs require image inputs of a fixed size. For instance, ResNet50 (MATLAB Help Center, 2025) and EfficientNetB0 (Fu, 2023) require an image input size of 224×224. Therefore, we generated mel 224×224 spectrogram images from the audio samples at a sampling rate of 22,050 Hz, with $n_{mels} = 224$, and a hop length based on the sampling rate and the maximum duration of the audio samples, as calculated in the equation:

$$hop\ length = \frac{sampling\ rate \times maximum\ duration}{n_{mels}}$$

Additionally, in a work by Begazo et al., 1920×1080 mel spectrogram images were generated, with the rationale of capturing the maximum possible detail, before being resized to 64×64 for input into CNNs for training (Begazo et al., 2024). Therefore, we generated 1920×1080 mel spectrogram images from the audio samples at the same sampling rate, $n_{mels} = 224$, and hop length as before.

As previously mentioned, the audio samples were not of equal durations, with plenty being shorter than the maximum duration of 4s. In such situations, zero-padding and repetitive padding have been used for CNNs in SER to fill the entire image (Yang et al., 2024), ensuring they are all generated as equal sizes. Additionally, Begazo et al. employed silence removal as a preprocessing technique, where any silences at the start and end of the audio samples were removed (Begazo et al., 2024). As such, we employed zero and repetitive padding, as well as silence removal, in generating the mel spectrogram images. For silence removal, we considered any signal at the start and end of the sample softer than 20 dB to be silent and removed them from all samples.

Model Fine-tuning and Performance

To find the optimal data preprocessing technique, we conducted 12 experiments with different combinations of pre-trained models (two models), padding techniques/silence removal (three techniques), and image sizes (two sizes). For each experiment, we normalized the images to ImageNet stats, since ResNet50 (He et al., 2015) and EfficientNet (Tan and Le, 2020) were both trained on ImageNet. We used the Adam optimizer, with a learning rate of 0.001, with weighted cross-entropy loss as the criterion, and early stopping with a patience of 5. We unfroze the last three layers of the models for fine-tuning. Additionally, for inputs using the 1920×1080 mel spectrogram images, we resized them to 224×224 in the dataloader in the fine-tuning pipeline.

Pre-trained Model	Pre-processing Technique	Image Size	Test Accuracy	Test Precision	Test Recall	Test F1-score
ResNet50	Zero Padding	224×224	0.7413	0.7471	0.7413	0.7414
		1920×1080	0.7509	0.7635	0.7509	0.7510
	Repetitive Padding	224×224	0.7596	0.7798	0.7596	0.7510
		1920×1080	0.7630	0.7684	0.7630	0.7611
	Silence Removal	224×224	0.7290	0.7367	0.7290	0.7279
		1920×1080	0.7327	0.7450	0.7327	0.7333
EfficientNetB0	Zero Padding	224×224	0.7499	0.7611	0.7499	0.7498
		1920×1080	0.7520	0.7623	0.7520	0.7533
	Repetitive Padding	224×224	0.7681	0.7789	0.7681	0.7705
		1920×1080	0.7711	0.7821	0.7711	0.7739
	Silence Removal	224×224	0.7666	0.7722	0.7666	0.7671
		1920×1080	0.7517	0.7636	0.7517	0.7517

TABLE 6.3: Model Performance of Fine-tuned Pre-trained CNNs With Various Data Preprocessing Techniques

In each epoch, we trained the model using the training set and validated using the validation set. Additionally, through model checkpointing, we saved the model weights as a .pt file if the model of this epoch exceeded the maximum validation accuracy so far. We then evaluated the test set, which was unseen during training, on the saved best model weights, and obtained the results in Table 6.3. Fine-tuning EfficientNetB0 with repetitively-padded 1920×1080 mel spectrogram images resulted in the best model performance, with a test F1-score of 0.7739. Therefore, from this set of experiments, we concluded that the optimal data preprocessing technique is to generate 1920×1080 images that are repetitively padded. Additionally, EfficientNetB0 has consistently shown better performance, in terms of test F1-score, than ResNet50 across all preprocessing techniques. Hence, we decided to focus on fine-tuning EfficientNet.

6.2.2 Finding the Optimal Pre-trained Model Architecture

EfficientNet Architectures

EfficientNet is a family of models, with several architectures of varying complexities, with the B0 architecture being the simplest and least performant, and B7 being the most complex and most performant (Tan and Le, 2020). At this point, we only fine-tuned the EfficientNet-B0 model, which is the simplest model.

Model Fine-tuning and Performance

To find the best-performing EfficientNet architecture, we fine-tuned the rest of the EfficientNet models (B1 to B7 architectures) on the 1920×1080 repetitively-padded mel spectrogram images, generated with the optimal data preprocessing technique. We kept the same fine-tuning parameters as previously used, but since the different EfficientNet architectures required different image input sizes (e.g., the B5 architecture requires 456×456 (Fu, 2023)), we resized the

Pre-trained Model	**Pre-processing Technique**	**Image Size**	**Test Accuracy**	**Test Precision**	**Test Recall**	**Test F1-score**
EfficientNetB1	Repetitive Padding, $n_{\text{mels}} = 224$	1920×1080	0.7598	0.7747	0.7598	0.7625
EfficientNetB2			0.7719	0.7821	0.7719	0.7725
EfficientNetB3			0.7802	0.7888	0.7802	0.7815
EfficientNetB4			0.7861	0.7917	0.7861	0.7870
EfficientNetB5			0.7960	0.8025	0.7960	0.7969
EfficientNetB6			0.7922	0.7978	0.7922	0.7928
EfficientNetB7			0.7808	0.7955	0.7808	0.7837

TABLE 6.4: Model Performance of Various Fine-tuned Pre-trained EfficientNet Models

images for each architecture accordingly, and obtained the results in Table 6.4. Fine-tuning EfficientNetB5 resulted in the best model performance, with a test F1-score of 0.7969, which was better than that of EfficientNetB0 on the same set of data with the same preprocessing techniques. Hence, we decided to focus on fine-tuning EfficientNetB5. This better performance on the EfficientNetB5 model is likely due to the greater complexity of the model that could have learned more from the complex patterns in the mel spectrogram images, as well as learned more robust representations from the images.

6.2.3 Optimizing the Performance of the Pre-trained Model Architecture

EfficientNetB5 Layers

Previously, we only unfroze the last three layers of the EfficientNet models for fine-tuning. Unfreezing more layers of EfficientNetB5, the best-performing EfficientNet architecture, might result in greater model performance, as there will be more learnable parameters to train on our dataset.

Data Preprocessing Technique

Since EfficientNetB5 requires an input size of 456×456 (Fu, 2023), we decided to also generate repetitively-padded mel spectrogram images of the audio samples, with the same parameters as before. We wanted see if it resulted in greater model performance, since it eliminated the need for resizing, which we hypothesized could eliminate the problem of losing important details when resizing. The only differences this time are that they were generated with a size of 456×456, $n_{mels} = 456$, and hop length adapted to $n_{mels} = 456$, to be consistent with the image size. A greater n_{mels} (456, compared to 224 previously) might result in a greater level of detail in the images, which could result in greater model performance.

Model Fine-tuning and Performance

We then fine-tuned EfficientNetB5, by iteratively unfreezing more layers using the 1920x1080 repetitively-padded mel spectrogram images, and also using the 456x456 images as input. We kept the same fine-tuning parameters as previously used, while unfreezing the layers accordingly, and obtained the results in Table 6.5. Fine-tuning EfficientNetB5 on 1920×1080

Pre-processing Technique	Image Size	Number of Final Layers Unfrozen	Test Accuracy	Test Precision	Test Recall	Test F1-score
Repetitive Padding, $n_{\text{mels}} = 224$	1920×1080	4	0.7998	0.8044	0.7998	0.7995
		5	0.8030	0.8078	0.8030	0.8024
		6	0.7993	0.8090	0.7993	0.8012
Repetitive Padding, $n_{\text{mels}} = 456$	456×456	5	0.7904	0.7940	0.7904	0.7905

TABLE 6.5: Model Performance of Various Fine-tuned Pre-trained EfficientNetB5 Models

repetitively-padded mel spectrogram images, with the final five layers unfrozen, resulted in the best model performance, with a test F1-score of 0.8024, which was better than that of EfficientNetB5 with only the final three layers unfrozen. We also observed that fine-tuning on 456×456 images did not improve model performance, showing the robustness of 1920×1080 images.

6.2.4 Fine-tuning the Optimal Model with Augmented Data

Data Augmentation

To ensure the best model (EfficientNetB5 with the final five layers unfrozen, fine-tuned on 1920×1080 repetitively-padded images) generalizes well and is robust to noise, we augmented the train set offline. Specifically, we applied three augmentation techniques, namely, Gaussian noise addition, time stretching, and pitch shifting, which are commonly used in SER (Galić, Šajić, and Marković, 2024), to the training audio samples. Each augmentation technique has a factor associated with it. For instance, noise addition with a factor of 0.5 will introduce more noise to the sample than with a factor of 0.1. Therefore, to ensure greater robustness, we introduced a range of factor values to each technique, where, for each sample, a factor value was randomly selected from this range, and the sample was augmented using this value. This ensured that the model is trained on varying degrees of noise, time-stretch, and pitch-shift, and not memorize the patterns of a specific amount of augmentation (i.e., if there was no range of factor values provided, there will only be a single factor value for each augmentation technique, and the model will only be robust to that specific amount of augmentation in the data).

Then, we generated 1920×1080 repetitively-padded mel spectrogram images from them, with a hop length adapted to the new maximum duration of the audio samples (due to time stretching). This resulted in a total of 195,837 audio samples, which are broken down in Table 6.6.

Model Fine-tuning and Performance

We then fine-tuned EfficientNetB5, with the final five layers unfrozen, using the 1920×1080 repetitively-padded mel spectrogram images of the augmented dataset. We kept the same fine-tuning parameters as previously used. The results of the fine-tuning can be seen in Figure 6.5. From the results, we see that the model performance improved with the offline data augmentation, from a test F1-score of 0.8024 on the un-augmented dataset to a test F1-score

Set	Data Augmentation	Count	Remarks
Train	None	44,221	70-15-15 train-test-validation split on the overall dataset with 63,174 samples
Train	Gaussian Noise Addition	44,221	
Train	Time Stretching	44,221	
Train	Pitch Shifting	44,221	
Test	None	9,476	
Validation	None	9,477	
Grand Total		195,837	

TABLE 6.6: A Breakdown of the Augmented Dataset

of 0.8260 on the augmented dataset. It shows that the data augmentation was successful in helping the model generalize well to the unseen test set. However, it could also be the sheer amount of data (195,837 samples in the augmented set) that contributed to this increase in model performance.

6.3 Fine-tuned Pre-trained Transformer-based Models

6.3.1 Model Architectures

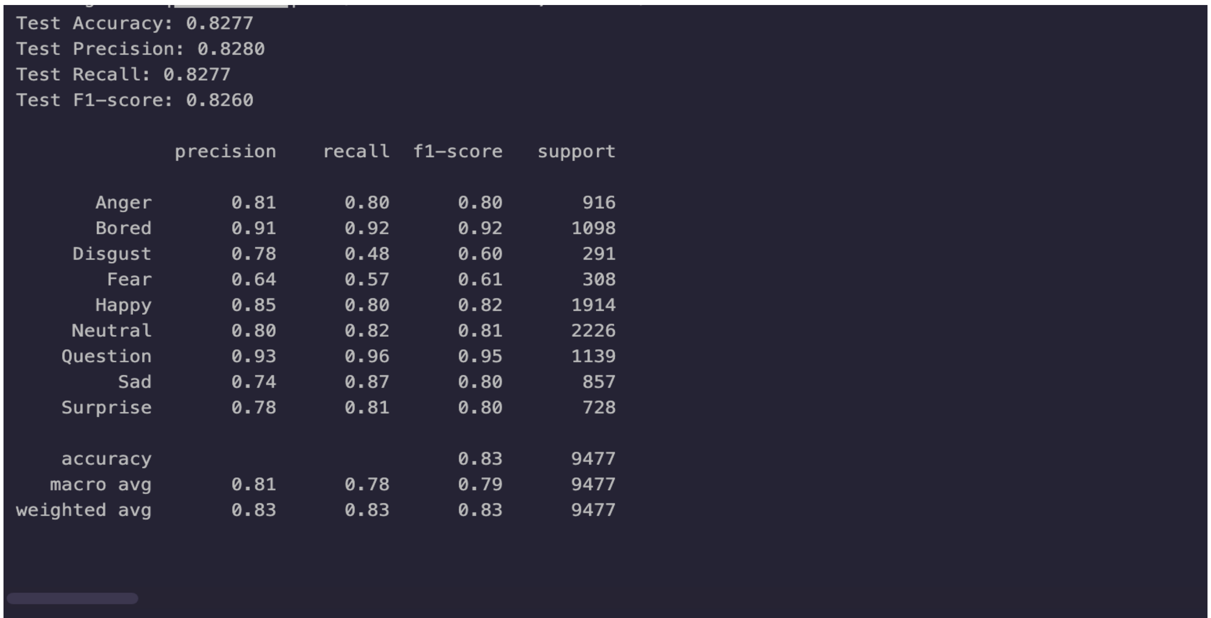
In addition to pre-trained CNN models, we experimented with more advanced pre-trained model architectures involving transformers by using transfer learning on these models in hopes of bringing model performance up another level. In this work, we decided to fine-tune two of these such models: the Wav2Vec2 (Baevski et al., 2020) base model by Facebook and the WavLM (S. Chen et al., 2022) base model by Microsoft. Several past works (Grosman, 2021)(Field, 2025) have used such models in SER tasks, and this method shows promise in improving upon our best fine-tuned EfficientNet model (pre-trained CNN).

Wav2Vec2

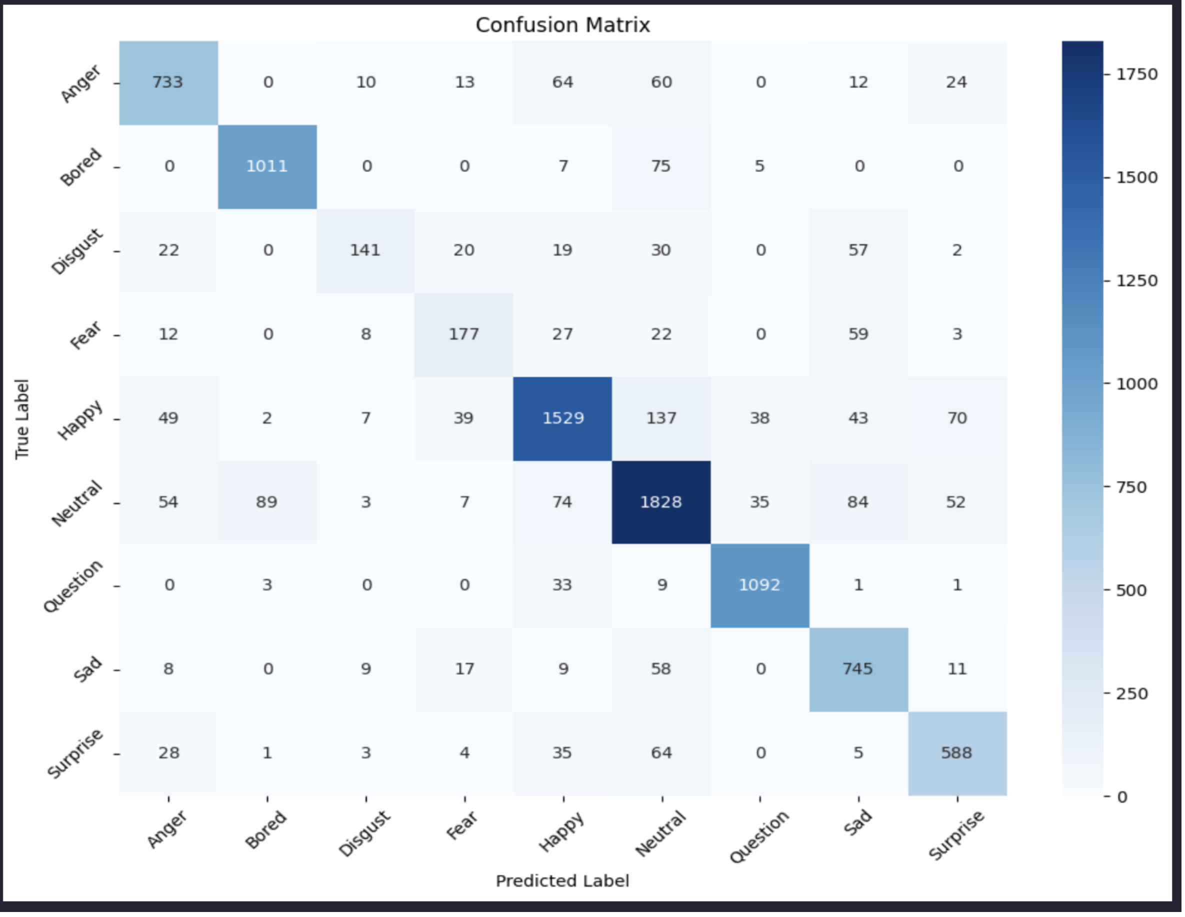
The Wav2Vec 2.0 model (Figure 6.6) is a framework for self-supervised learning of representations from raw audio data. It begins by encoding the input waveform using a multi-layer convolutional neural network to produce latent speech representations, which are then partially masked and passed through a Transformer network. The model is trained using a contrastive task, where it learns to identify the true latent representation from the distractors. During the training procedure, discrete speech units are learnt via a Gumbel softmax to represent the latent representations in the contrastive task. After pre-training on unlabelled speech, the model is fine-tuned on labelled data using Connectionist Temporal Classification (CTC) loss, making it highly suitable for downstream speech recognition tasks (Baevski et al., 2020).

WavLM

The WavLM model (Figure 6.7) is a pretrained model designed to solve full-stack downstream speech tasks. It is built upon the HuBERT and Wav2Vec 2.0 models, by optimizing their model structure and training data. It introduces masked speech prediction and denoising, using simulated noise and overlapped speech during pre-training to predict the original masked content. A novel innovation is the introduction of a gated relative position bias to the Transformer



```
Test Accuracy: 0.8277
Test Precision: 0.8280
Test Recall: 0.8277
Test F1-score: 0.8260

              precision    recall  f1-score   support

       Anger       0.81      0.80      0.80       916
       Bored       0.91      0.92      0.92      1098
     Disgust       0.78      0.48      0.60       291
        Fear       0.64      0.57      0.61       308
       Happy       0.85      0.80      0.82      1914
     Neutral       0.80      0.82      0.81      2226
    Question       0.93      0.96      0.95      1139
         Sad       0.74      0.87      0.80       857
    Surprise       0.78      0.81      0.80       728

    accuracy                           0.83      9477
   macro avg       0.81      0.78      0.79      9477
weighted avg       0.83      0.83      0.83      9477
```



	Anger	Bored	Disgust	Fear	Happy	Neutral	Question	Sad	Surprise
Anger	733	0	10	13	64	60	0	12	24
Bored	0	1011	0	0	7	75	5	0	0
Disgust	22	0	141	20	19	30	0	57	2
Fear	12	0	8	177	27	22	0	59	3
Happy	49	2	7	39	1529	137	38	43	70
Neutral	54	89	3	7	74	1828	35	84	52
Question	0	3	0	0	33	9	1092	1	1
Sad	8	0	9	17	9	58	0	745	11
Surprise	28	1	3	4	35	64	0	5	588

FIGURE 6.5: Complete Model Performance Report with Classification Report and Confusion Matrix for Best Fine-tuned Pre-trained CNN

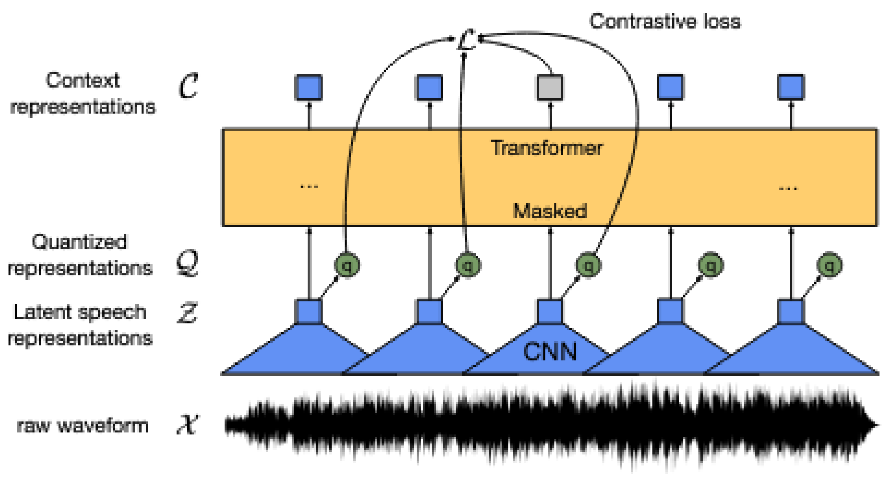

Contrastive loss
$\mathcal{L}$
Context representations $\mathcal{C}$
Transformer
Masked
Quantized representations $\mathcal{Q}$
q
Latent speech representations $\mathcal{Z}$
CNN
raw waveform $\mathcal{X}$

FIGURE 6.6: Architecture of Wav2Vec2 Model (Baevski et al., 2020)

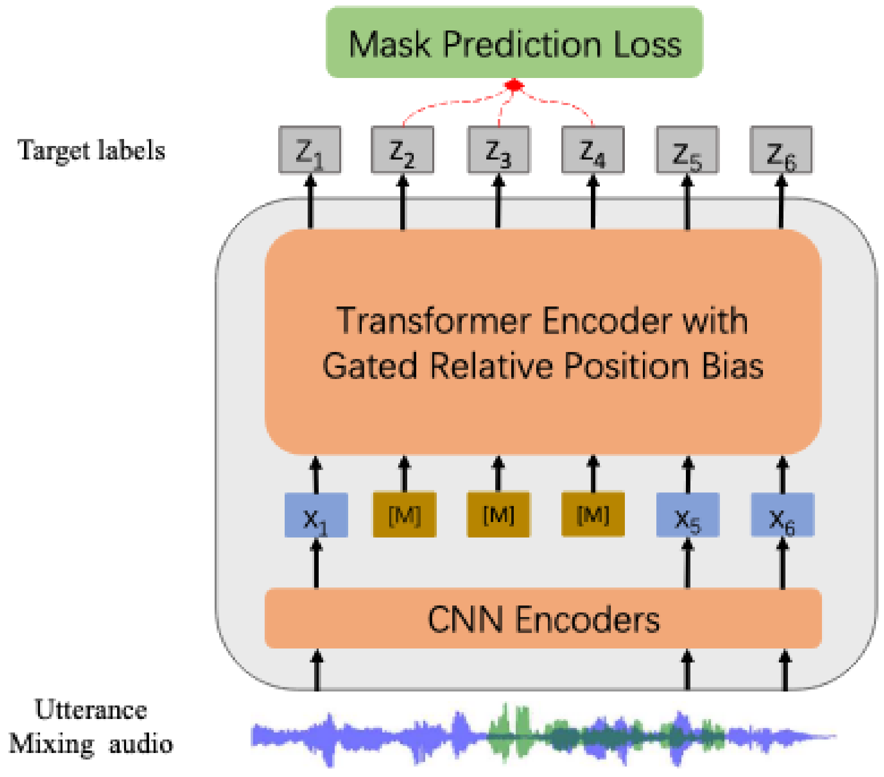

Mask Prediction Loss
Target labels
Z_1 Z_2 Z_3 Z_4 Z_5 Z_6
Transformer Encoder with Gated Relative Position Bias
X_1 [M] [M] [M] X_5 X_6
CNN Encoders
Utterance Mixing audio

FIGURE 6.7: Architecture of WavLM Model (S. Chen et al., 2022)

Pre-trained Model	Test Accuracy	Test Precision	Test Recall	Test F1-score
Wav2Vec2	0.8538	0.8578	0.8538	0.8540
WavLM	0.8451	0.8492	0.8451	0.8449

TABLE 6.7: Model Performance of Fine-Tuning the Wav2Vec2 and WavLM Models

Pre-trained Model	Test Accuracy	Test Precision	Test Recall	Test F1-score
Wav2Vec2	0.8400	0.8469	0.8400	0.8407
WavLM	0.8566	0.8572	0.8566	0.8555

TABLE 6.8: Model Performance of Fine-Tuning The Wav2Vec2 and WavLM Models after Data Augmentation

structure as the backbone. The gates adaptively adjust to the current speech content, enhancing performance on both automatic speech recognition (ASR) and generalizing well to non-ASR tasks (S. Chen et al., 2022).

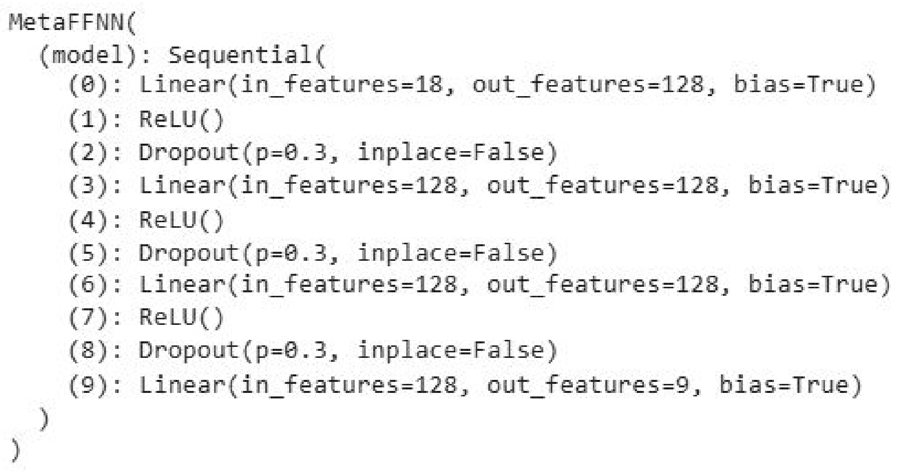
6.3.2 Fine-tuning Models with Raw Audio Samples

The raw audio samples were used as input to fine-tuning the Wav2Vec2 and WavLM models. As both these models use CNN based encoders, the samples had to be of the same duration, and hence we zero-padded the audio samples to a constant length of 4 seconds, the maximum duration we retained from Section 3.2. In addition, the audios were sampled at a sampling rate of 16,000 Hz as these models were pre-trained on 16,000 Hz sampled speech data.

Using the AdamW optimizer, a learning rate of 0.00002, weighted cross-entropy loss as the criterion, and early stopping with a patience of 3, we fine-tuned both the Wav2Vec2 and WavLM models. For each training epoch, we trained the network using the training set and validated using the validation set. Additionally, through model checkpointing on each epoch, we saved the model weights as checkpoint files. We defined the best model to be the one that achieved the best F1-score. We then evaluated the test set, which was unseen during training, on the saved best model weights, and obtained the results in Table 6.7. Fine-tuning both these models produced decent results and were comparable to each other. The metrics are also an improvement across the board compared to the fine-tuned EfficientNet models. Fine-tuning the Wav2Vec2 model resulted in a test F1-score of 0.8540, and the WavLM model resulted in a test F1-score of 0.8449, an improvement over the best EfficientNet model that has a test F1-score of 0.8260.

6.3.3 Fine-tuning the Models with Augmented Samples

As shown in Section 6.2.4, we observed that applying offline data augmentation had potential to increase the test F1-score. In addition, the model would also be more robust to other conditions that are more representative of the real-world environment. Applying the same data augmentation techniques from Section 6.2.4, the same augmented data as in Table 6.6 was used to fine-tune the models this time. However, as Time Stretching was used as an augmentation technique, some original raw audio samples had an eventual duration of longer than the initial 4 seconds. Hence, the audio samples were zero-padded to a constant length of 5 seconds.



```
MetaFFNN(
  (model): Sequential(
    (0): Linear(in_features=18, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=128, out_features=128, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.3, inplace=False)
    (6): Linear(in_features=128, out_features=128, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.3, inplace=False)
    (9): Linear(in_features=128, out_features=9, bias=True)
  )
)
```

FIGURE 6.8: Architecture of our Meta Learner

Applying the same methodology as Section 6.3.2, we obtained the results in Table 6.8. We observe that fine-tuning both models produced decent results and were comparable to each other. Applying data augmentation improved the performance of the WavLM model, while the performance of Wav2Vec2 decreased slightly. This might be due to the fact that WavLM was trained using simulated noise, which meant that the WavLM model was able to better learn from the augmented data. On the other hand, there may be various reasons why there was no improvement for the Wav2Vec2 model. We hypothesize that the augmented dataset might have distorted the extracted features crucial for emotion recognition by too much, resulting in a mismatch between the augmented inputs and the representations Wav2Vec2 was pretrained on. This could have led to suboptimal fine-tuning and ultimately a slight drop in performance. Nonetheless, its performance is still relatively high and is comparable to without any data augmentation applied.

As we aim to construct a model where it is trained to recognize emotions from real-world speech patterns, we decided that we would still apply data augmentation techniques so that we can create a model that is more robust.

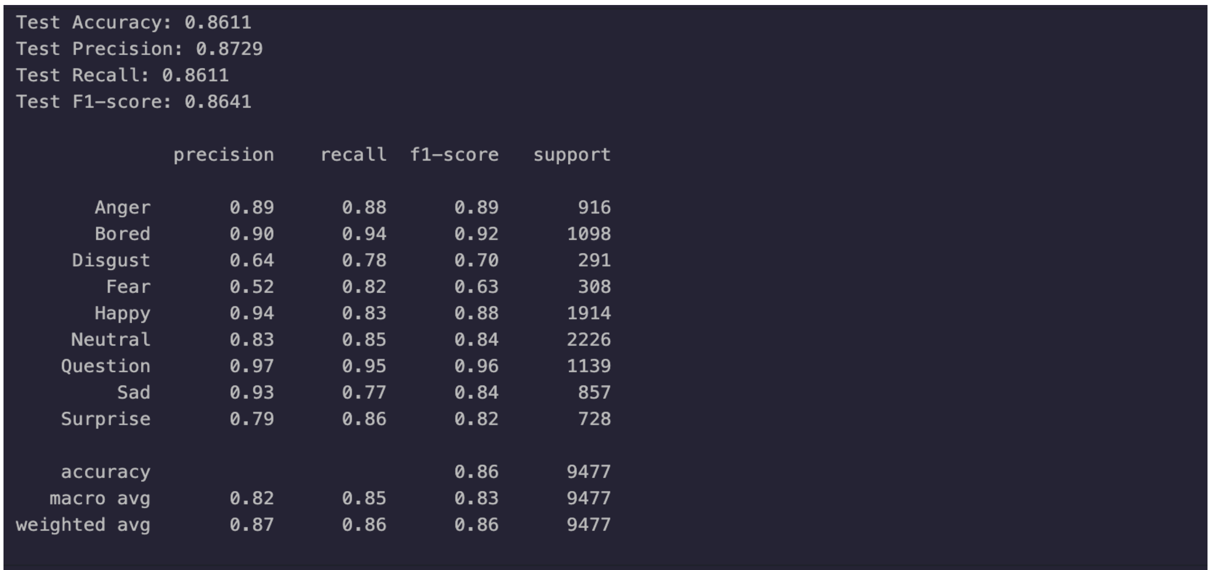
6.3.4 Fusion of Transformer-based Models: Stacked Ensemble Approach

It was found that ensemble learning, whereby the predictions of multiple base models are combined, could potentially lead to better performance over the individual deep learning models. Theoretically, ensemble learning allows leveraging on the individual strengths of the base models to make better predictions. This approach was inspired by Ghasemieh's team, where they applied a stacking ensemble learner (Ghasemieh et al., 2023). A stacking ensemble classifier is a machine learning model that combines multiple base models to improve prediction accuracy. They used the outputs of the base models as inputs to a higher-level model, also known as a Meta-Model, which then learns patterns based on the inputs to make a final prediction. In our work, we replicated the above procedure and built a simple FFNN which will serve as the Meta Learner. It has a simple architecture, which is shown in Figure 6.8.

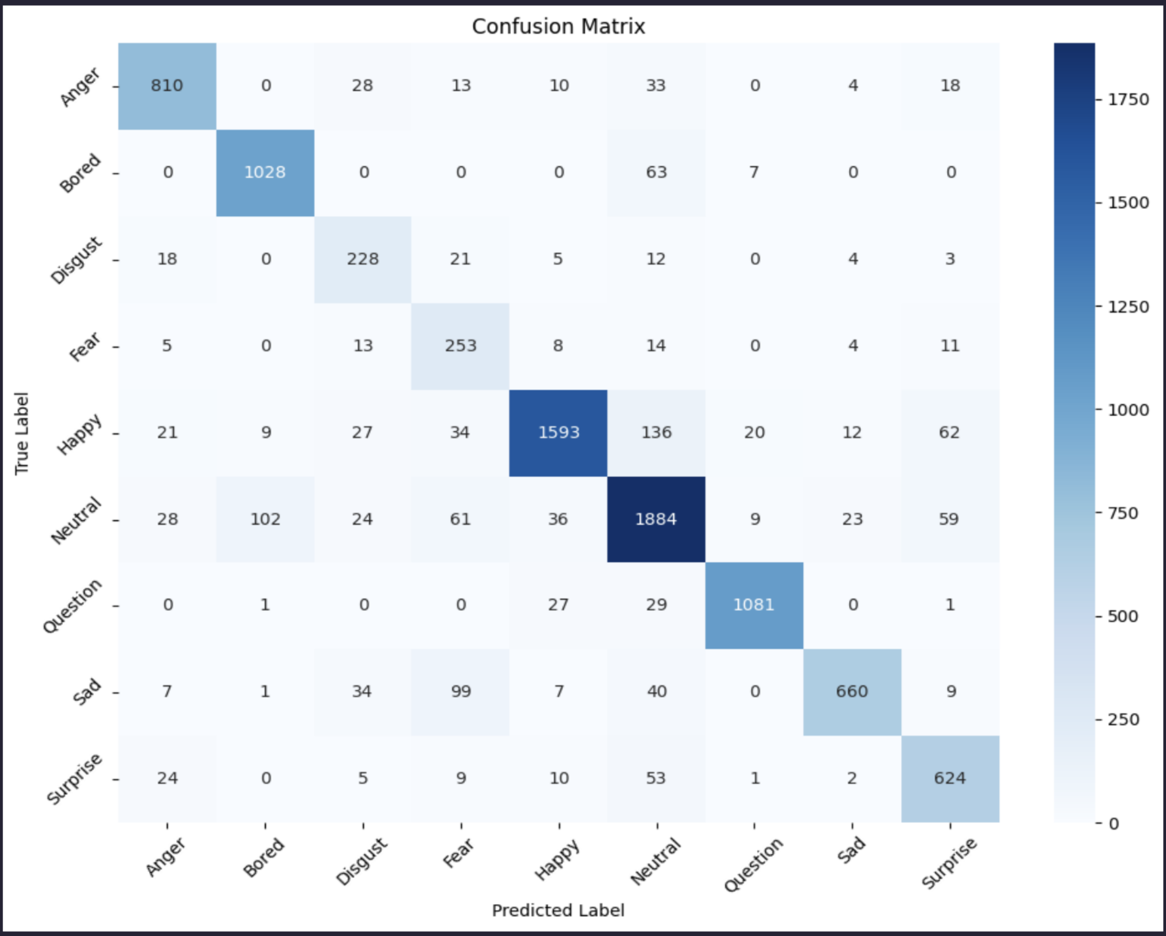
To train the Meta Learner, we used the 2 base models which we have fine-tuned in Section 6.3.3, the augmented Wav2Vec2 and WavLM models, to evaluate on the validation set. The logits from the two base models are stacked together and used to train the Meta Learner, which explains why we have 18 in_features in Figure 6.8 since there are two models and nine emotional classes to predict. After stacking the logits, the Meta Learner was trained using K-Fold cross validation using 5 folds and weighted cross-entropy loss. We saved the model with the best fold that achieved the best test F1-score which we use later to evaluate whether the stacking ensemble classifier that we trained worked.

After training the Meta Learner, we set up a pipeline which uses the trained Meta Learner to make a final prediction based on the two base fine-tuned models, Wav2Vec2 and WavLM. The data we used to evaluate this is the original test dataset which we have been using to evaluate all our trained models thus far to ensure consistency and fairness. The results of using this architecture can be seen in Figure 6.9, where we achieved our highest test F1-score of 0.8641.

Based on the results of the ensemble approach, the test accuracy, test precision, test recall and test F1-scores all being above 0.86, which outperforms both the individual base models. This shows that the ensemble approach was able to leverage the strengths of the individual base models to obtain better results than if the base models were used individually. We thus employ this model architecture to be used in our Frontend Application.



```
Test Accuracy: 0.8611
Test Precision: 0.8729
Test Recall: 0.8611
Test F1-score: 0.8641

              precision    recall  f1-score   support

       Anger       0.89      0.88      0.89       916
       Bored       0.90      0.94      0.92      1098
     Disgust       0.64      0.78      0.70       291
        Fear       0.52      0.82      0.63       308
       Happy       0.94      0.83      0.88      1914
     Neutral       0.83      0.85      0.84      2226
    Question       0.97      0.95      0.96      1139
         Sad       0.93      0.77      0.84       857
    Surprise       0.79      0.86      0.82       728

    accuracy                           0.86      9477
   macro avg       0.82      0.85      0.83      9477
weighted avg       0.87      0.86      0.86      9477
```



True Label \ Predicted Label	Anger	Bored	Disgust	Fear	Happy	Neutral	Question	Sad	Surprise
Anger	810	0	28	13	10	33	0	4	18
Bored	0	1028	0	0	0	63	7	0	0
Disgust	18	0	228	21	5	12	0	4	3
Fear	5	0	13	253	8	14	0	4	11
Happy	21	9	27	34	1593	136	20	12	62
Neutral	28	102	24	61	36	1884	9	23	59
Question	0	1	0	0	27	29	1081	0	1
Sad	7	1	34	99	7	40	0	660	9
Surprise	24	0	5	9	10	53	1	2	624

FIGURE 6.9: Complete Model Performance Report with Classification Report and Confusion Matrix for Meta Learner on top of the two Base Models, Wav2Vec2 and WavLM

Chapter 7

Application

7.1 Overview

This chapter details the graphical user interface (GUI) application developed for SER. The application integrates a Flask backend with an HTML/JavaScript frontend. Users can upload speech audio files, which are processed on the server using an ensemble of deep learning models. The frontend provides an interactive waveform player that displays predicted emotions for every five-second segment of the speech, enhanced to show the top three emotions live during playback along with its corresponding confidence level.

7.2 Frontend

The frontend of the application is responsible for user interaction and visualization. The key components include:

7.2.1 User Interface

The user interface (UI) consists of HTML, Bootstrap and JavaScript that allows users to upload speech audio files in WAV or MP3 format. Upon submission, the file is sent to the backend for processing.

7.2.2 Visualization

Once processing and predictions are completed, the frontend displays:

- The uploaded speech audio as a waveform.
- A playback feature for users to listen to the speech.
- For each five-second segment, the top three predicted emotions with confidence scores are displayed in real-time as the speech audio plays.
- Emotions are color-coded by confidence: green (>80%), orange (60–80%), and red (<60%).
- A list of emotions predicted in the entire speech audio for each segment.

JavaScript is used to handle file selection, AJAX requests to send the file to the backend, and dynamic updating of the UI once results are received.

7.3 Backend

Implemented using Flask, which deals with the server logic and API endpoints.

7.3.1 Audio Handling

- Receives the uploaded speech audio files via POST requests from the frontend.
- Temporarily saves the files and loads it using librosa.
- The speech audio is segmented into fixed five-second chunks for consistent processing.

7.3.2 Model Processing

Based on our best overall model, as described in Section 6.3.4. Each of the speech audio segment is passed through the two fine-tuned pre-trained models:

- `facebook/wav2vec2-base`
- `microsoft/wavlm-base`

Both models output logits that represent emotion predictions across nine emotional classes.

Ensemble Prediction

- Concatenate logits from both models.
- Feed them into a Meta-FFNN to get the final prediction scores.

Output Formatting

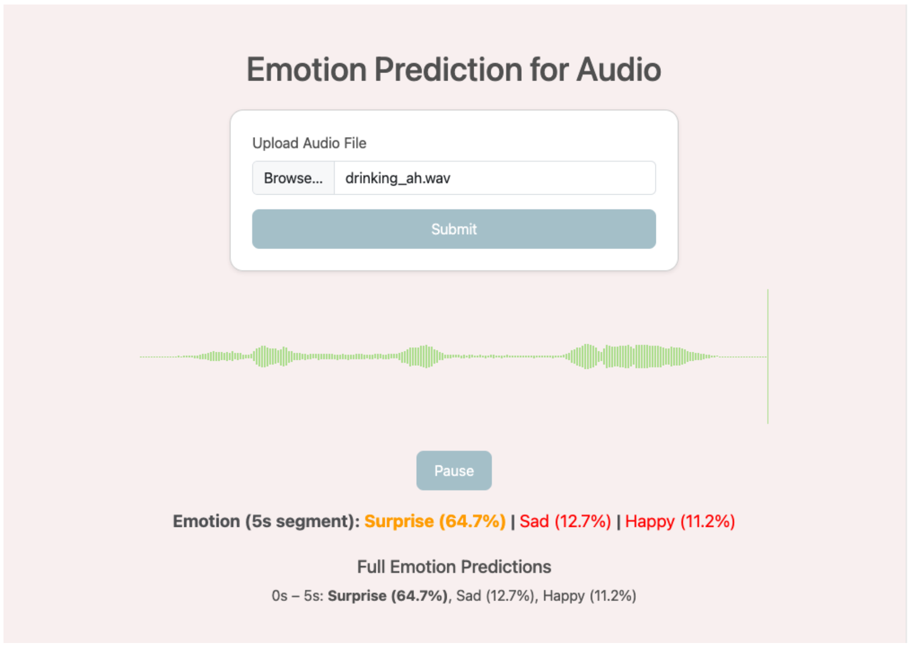
- Apply softmax activation function to get probabilities.
- Select the top three predicted emotions with the confidence for each speech audio segment.
- Return the results as a JSON response with:
  - Segmented start and end timestamps.
  - The top three emotions and their confidence percentages.

7.4 Results Display and Interaction

The frontend processes the received predictions and updates the UI dynamically. Users can view:

- The segmented speech audio waveform.
- The top three predicted emotions for each segment along with its confidence level as a percentage.

This interactive approach enhances user engagement by providing a clear and intuitive visualization of the models' outputs.



Emotion (5s segment): Surprise (64.7%) | Sad (12.7%) | Happy (11.2%)

Full Emotion Predictions

0s – 5s: **Surprise (64.7%)**, Sad (12.7%), Happy (11.2%)

FIGURE 7.1: Our Application/GUI

7.5 Setting up the Project

7.5.1 Creating Virtual Environment

1. Clone our project repository from here
2. Install `virtualenv` if not already installed: `pip install virtualenv`
3. Create a new virtual environment named 'venv': `virtualenv venv`
4. Activate the virtual environment: `source venv/bin/activate`
5. Install the project dependencies: `pip install -r requirements.txt`

7.5.2 Downloading Our Best Model

1. Download `final_model.zip` from our Microsoft Teams here
2. Move `final_model.zip` to the `gui/` directory, if it is not downloaded and saved there already
3. Extract the contents of `final_model.zip` into the current `gui/` directory
4. Once the extraction is complete, there should be `best_meta_ffnn_model.pt`, `checkpoint-22112/` and `checkpoint-55280/`. These files and folders contain the model weights.
5. You may delete `final_model.zip`

7.6 Running the Application

7.6.1 Backend

1. Ensure that you are inside the virtual environment that you just set up. If not, activate it using `source venv/bin/activate`
2. From the project root directory, navigate to the GUI folder: `cd gui`
3. Start the application/backend server using `python app.py`
4. On the first run, wait up to 1 minute or so for the backend to initialize
5. It is crucial that any saved data and cookies from `localhost` in the browser settings is cleared

7.6.2 Frontend

1. Open the `index.html` file in a browser
2. This will launch the GUI for users to see the emotion predictions for the uploaded speech audio file
3. Upload a .mp3 or .wav speech audio file, and click on 'Submit' to process and view the predicted emotions for five-second segments.
4. To analyze another file, refresh the page, and repeat the previous step.

7.7 Conclusion

The developed GUI application successfully integrates Flask with a JavaScript-powered frontend to enable SER. It allows users to upload a speech audio file, processes it efficiently, and visually represents the top three predicted emotions dynamically for each five-second segment, ensuring an interactive user experience fitting of our use case.

Chapter 8

Discussion

8.1 Comparison with State-of-the-Art (SOTA) Models

We compared our best-performing models/architecture trained and fine-tuned in each of our approaches with the known SOTA models. It is critical to note that these models are the best-performing models we can find and are aware of, which are evaluated on metrics that our models have been evaluated on for a fairer comparison. The comparison is shown in Table 8.1

Work	**Model**	**Dataset**	**Model Performance**	**Our Best Model's Performance**
(Paul et al., 2024)	KNN	RAVDESS	95% (Accuracy)	0.6466 or 64.66% (Test Accuracy)
(Agrawal et al., 2023)	RF	SAVEE	93% (Accuracy)	0.6443 or 64.43% (Test Accuracy)
(Al Zoubi, Turky, and Foufou, 2024)	SVM	RAVDESS	85% (Accuracy)	0.7166 or 71.66% (Test Accuracy)
(Agrawal et al., 2023)	FFNN	RAVDESS	77% (Accuracy)	0.7228 or 72.28% (Test Accuracy)
(Nfissi et al., 2025)	CNN	RAVDESS	81.3% (Test Accuracy)	0.7393 or 73.93% (Test Accuracy)
(Chakhtouna, Sekkate, and Adib, 2023)	Fine-tuned Efficient-Net	DEMoS	0.67 (F1-score), EfficientNetB0	0.8260 (Test F1-score), Efficient-NetB5
(Ali, Arymurthy, and Prasojo, 2023)	Fine-tuned Wav2Vec2	RAVDESS	78.91% (Unweighted Accuracy)	0.8538 or 85.38% (Test Accuracy)
(Ali, Arymurthy, and Prasojo, 2023)	Fine-tuned WavLM	RAVDESS	80.47% (Unweighted Accuracy)	0.8566 or 85.66% (Test Accuracy)
None	Ensemble Model of Wav2Vec2 and WavLM	None	None (Unable to find)	0.8641 (Test F1-Score)

TABLE 8.1: Comparison of Our Models with Known SOTA Models

It is important to note that our model is trained on a combination of eight datasets and is capable of predicting nine emotional classes. The existing SOTA models were mostly evaluated on other datasets or a subset of the combined dataset we used. These models may also be trained to classify inputs with a smaller number of emotional classes, rendering a direct comparison pointless and unfair. Nonetheless, it can still provide a rough idea on how our models fare against the SOTA results.

8.2 Limitations

Despite some of our models being comparable to some of the SOTA models that we were aware of, there still exists several limitations in our work.

8.2.1 Acted Datasets

At least four datasets (SAVEE, CREMA-D, TESS, RAVDESS) in our combined dataset were acted, where professional actors spoke in different emotions, and were recorded in controlled environments. However, they may not be representative of real-world speech inputs, since real-world speech may not always be recorded in controlled environments, and the emotion expressed in such audio samples may be exaggerated (Hashem, Arif, and Alghamdi, 2023).

8.2.2 Limited Compute

A portion of our experiments was conducted on a gaming laptop with a RTX 4090 GPU, specifically the experiments involving fine-tuning pre-trained models. Despite the compute power of the laptop RTX 4090 GPU, it is still rather limited given the amount of data we had (63, 471 in the raw and combined dataset, 195,837 in the augmented dataset). In some experiments, even after optimizations to minimize training time by reducing the batch size, one training epoch still took close to 60 minutes to complete due to the size and complexity of some of the pre-trained models. This limited the breadth of experiments we could conduct, and also the batch size we could set in the dataloader without overly extending epoch time. The batch size we chose which could have impacted the generalizibility of the models (Devansh, 2022).

8.2.3 Class Imbalance

Despite our efforts to minimize class imbalance, there still exists some class imbalance. Certain emotional classes, like neutral and happy, have significantly more samples than the rarer classes, like fear and disgust (refer to Figure 3.4). This likely limited the performance of our models in classifying inputs into these rare emotions, and the overall model performance.

8.3 Future Work

While this study explored many approaches, several directions remain for future work. Such work include the following:

- For the models developed in this study, further experimentation with hyperparameters could potentially have led to improved performance. However, we observed that performance gains are often more significantly influenced by the choice of model architecture and the preprocessing techniques used to generate input data to the models. Given the scope and time constraints of this project, extensive hyperparameter tuning was not prioritized, in favor of focusing on model selection and input representation.

- The collection of more samples, particularly those annotated with the rare emotional classes could be done. This is to address the emotional class imbalance we faced in our work. We believe that doing so will improve model performance without having to completely drop samples annotated with rare labels.

- For computation of loss, we used weighted cross-entropy loss in our work. Focal loss is a loss function that acts as a more effective alternative to weighted cross-entropy loss. It applies a modulating term to the cross entropy loss in order to focus learning on hard negative samples (Lin et al., 2018), allowing the model to focus its learning on the rare classes. Due to the emotional class imbalance in our original dataset, using focal loss instead might have resulted in better performance, given that our models did not predict the rare classes as well as the more common classes.

- For Data Augmentation techniques, we also wanted to augment the audio samples with realistic noise from noise datasets, instead of using Gaussian noise addition for our offline data augmentation. After all, audio inputs from the real-world are not always recorded in controlled environments, and contain noise to a certain degree (George and Ilyas, 2024). Augmented audio samples with realistic noise from noise datasets could unlock greater generalizability and robustness in our models compared to Gaussian noise, which may not always be realistic and representative of real-world noise.

- The exploration of more advanced deep learning architectures, like CNN+LSTMs, which have achieved great accuracies exceeding 99% for several common SER datasets (George and Ilyas, 2024).

Chapter 9

Conclusion

In this project, we developed ANNAMALAI — an end-to-end SER system designed to classify human speech audio into one of nine emotional categories. We conducted a comprehensive study encompassing a wide spectrum of techniques, from traditional machine learning models to modern deep learning and transformer-based architectures. Each model was trained and evaluated using a large and diverse dataset constructed from eight publicly available SER datasets, cleaned and standardized to a consistent format. Our methodology incorporated robust data preprocessing, rigorous experimentation, and modelling practices commonly used in SER.

We began with classical machine learning models such as KNN, RF, and SVM trained on openSMILE-extracted features. While these models provided baseline insights, we further advanced performance by implementing FFNNs trained on these openSMILE-extracted features. We also custom-designed CNNs trained on MFCCs and MFCCs with delta features.

Subsequently, we explored transfer learning using pre-trained CNNs, in particular the EfficientNetB5, and optimized their performance with a carefully tuned data preprocessing pipeline, which included generating high-resolution 1920×1080 mel spectrograms with repetitive padding. Offline data augmentation (Gaussian noise addition, time stretching, pitch shifting) was applied to improve robustness, leading to further improvements in model performance.

Building on this, we next fine-tuned transformer-based models, which include Wav2Vec2 and WavLM, which demonstrated significant performance gains. Our final and most performant model was a stacked ensemble that fused the logits of the two transformer models via a meta-learner FFNN, achieving an F1-score of 0.8641 on the test set, which is the highest among all of our models.

To demonstrate practical applicability, we developed a Flask-based GUI application that allows users to upload speech audio and receive real-time, segment-based emotion predictions. The frontend visualizes predictions using waveform plots and confidence-based emotion indicators, offering an intuitive and engaging user experience.

9.1 Contribution and Significance

This project offers several key contributions to the field of SER:

- **Comprehensive Exploration**: We undertook a methodical and empirical exploration of a wide range of SER techniques, covering classical ML, deep learning, transfer learning with CNNs, and transformer-based architectures — all evaluated on a unified dataset of over 63,000 samples and consistently benchmarked.

- **Multi-class Capability**: All models developed in this work are capable of predicting 9 distinct emotional classes, going beyond binary or small-class SER setups commonly found in prior studies.
- **Novel Ensemble Model**: We proposed and implemented a novel ensemble strategy, fusing Wav2Vec2 and WavLM models through a stacked ensemble FFNN. This approach synergizes the strengths of both base models and yielded our best results to date, demonstrating the effectiveness of ensemble learning in SER.
- **Practical Deployment**: We built a fully functional SER application, bridging research and deployment. The system is accessible, efficient, and scalable, with potential applications in mental health, customer support, and affective computing interfaces.

9.2 Closing Remarks

While our work has yielded promising results, there remain areas for future improvement, including the integration of focal loss to better address class imbalance, exploration of realistic noise datasets for augmentation, and the investigation of hybrid architectures like CNN-LSTMs. Nevertheless, this study marks a significant step forward in building robust, real-world-ready SER systems. Through this project, we demonstrate the feasibility of deploying AI-powered emotional intelligence systems that are both technically sound and socially impactful, aligned with global initiatives such as AI for Good and the UN SDGs.

Bibliography

Agrawal, Aditi et al. (Apr. 2023). “Comparative Analysis of Speech Emotion Recognition Models and Technique”. In: *2023 International Conference on Computational Intelligence, Communication Technology and Networking (CICTN)*, pp. 499–505. DOI: 10.1109/CICTN57981.2023.10141044. URL: https://ieeexplore.ieee.org/document/10141044 (visited on 04/17/2025).

Al Zoubi, Rouaa, Ayad Turky, and Sebti Foufou (2024). “Speech Emotion Recognition Using Support Vector Machine”. en. In: *Proceedings of Ninth International Congress on Information and Communication Technology*. Ed. by Xin-She Yang et al. Singapore: Springer Nature, pp. 519–532. ISBN: 978-981-9733-02-6. DOI: 10.1007/978-981-97-3302-6_42.

Ali, Fadel, Aniati Murni Arymurthy, and Radityo Eko Prasojo (Dec. 2023). “A Comprehensive Exploration of Fine-Tuning WavLM for Enhancing Speech Emotion Recognition”. In: *2023 6th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*. ISSN: 2832-1456, pp. 295–300. DOI: 10.1109/ISRITI60336.2023.10467733. URL: https://ieeexplore.ieee.org/document/10467733 (visited on 04/17/2025).

Ayadi, Souha and Zied Lachiri (Mar. 2022). “Deep Neural Network for visual Emotion Recognition based on ResNet50 using Song-Speech characteristics”. In: *2022 5th International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, pp. 363–368. DOI: 10.1109/IC_ASET53395.2022.9765898. URL: https://ieeexplore.ieee.org/document/9765898 (visited on 04/15/2025).

Baevski, Alexei et al. (Oct. 2020). *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. arXiv:2006.11477 [cs]. DOI: 10.48550/arXiv.2006.11477. URL: http://arxiv.org/abs/2006.11477 (visited on 04/21/2025).

Batliner, Anton and Björn Schuller (Jan. 2011). “The Automatic Recognition of Emotions in Speech”. en. In: *ResearchGate*. DOI: 10.1007/978-3-642-15184-2_6. URL: https://www.researchgate.net/publication/224929599_The_Automatic_Recognition_of_Emotions_in_Speech (visited on 04/21/2025).

Begazo, Rolinson et al. (Jan. 2024). “A Combined CNN Architecture for Speech Emotion Recognition”. en. In: *Sensors* 24.17. Number: 17 Publisher: Multidisciplinary Digital Publishing Institute, p. 5797. ISSN: 1424-8220. DOI: 10.3390/s24175797. URL: https://www.mdpi.com/1424-8220/24/17/5797 (visited on 04/16/2025).

Cao, Houwei et al. (2014). “CREMA-D: Crowd-sourced Emotional Multimodal Actors Dataset”. In: *IEEE transactions on affective computing* 5.4, pp. 377–390. ISSN: 1949-3045. DOI: 10.1109/TAFFC.2014.2336244. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4313618/ (visited on 04/21/2025).

Carrión, Jesús Requena and Nikesh Bajaj (2022). *MLEnd Spoken Numerals*. en. URL: https://www.kaggle.com/datasets/jesusrequena/mlend-spoken-numerals (visited on 04/21/2025).

Chakhtouna, Adil, Sara Sekkate, and Abdellah Adib (2023). “Speech Emotion Recognition Using Pre-trained and Fine-Tuned Transfer Learning Approaches”. en. In: *Innovations in Smart*

Cities Applications Volume 6. Ed. by Mohamed Ben Ahmed et al. Cham: Springer International Publishing, pp. 365–374. ISBN: 978-3-031-26852-6. DOI: 10.1007/978-3-031-26852-6_35.

Chen, Sanyuan et al. (Oct. 2022). "WavLM: Large-Scale Self-Supervised Pre-Training for Full Stack Speech Processing". In: *IEEE Journal of Selected Topics in Signal Processing* 16.6. arXiv:2110.13900 [cs], pp. 1505–1518. ISSN: 1932-4553, 1941-0484. DOI: 10.1109/JSTSP.2022.3188113. URL: http://arxiv.org/abs/2110.13900 (visited on 04/21/2025).

Chen, Sheng-Yeh et al. (May 2018). *EmotionLines: An Emotion Corpus of Multi-Party Conversations*. arXiv:1802.08379 [cs]. DOI: 10.48550/arXiv.1802.08379. URL: http://arxiv.org/abs/1802.08379 (visited on 04/21/2025).

Devansh (Jan. 2022). *How does Batch Size impact your model learning*. en. URL: https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa (visited on 04/18/2025).

Diego (June 2020). *Speech Emotion Recognition with Convolutional Neural Network*. en. URL: https://medium.com/@diego-rios/speech-emotion-recognition-with-convolutional-neural-network-ae5406a1c0f7 (visited on 04/15/2025).

Dupuis, Kate and M. Kathleen Pichora-Fuller (2025). "Toronto emotional speech set (TESS)". en. In: (). URL: https://utoronto.scholaris.ca/collections/036db644-9790-4ed0-90cc-be1dfb8a4b66 (visited on 04/21/2025).

Eyben, Florian, Martin Wöllmer, and Björn Schuller (Oct. 2010). "Opensmile: the munich versatile and fast open-source audio feature extractor". In: *Proceedings of the 18th ACM international conference on Multimedia*. MM '10. New York, NY, USA: Association for Computing Machinery, pp. 1459–1462. ISBN: 978-1-60558-933-6. DOI: 10.1145/1873951.1874246. URL: https://doi.org/10.1145/1873951.1874246 (visited on 04/21/2025).

Field, Rob (2025). *Speech Emotion Recognition By Fine-Tuning Wav2Vec 2.0*. URL: https://huggingface.co/r-f/wav2vec-english-speech-emotion-recognition (visited on 04/21/2025).

Fu, Yixing (July 2023). *Keras documentation: Image classification via fine-tuning with EfficientNet*. en. URL: https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/ (visited on 04/16/2025).

Galić, Jovan, Slavko Šajić, and Branko Marković (Nov. 2024). "Exploring the Impact of Data Augmentation Techniques on Emotional Speech Recognition". In: *2024 32nd Telecommunications Forum (FOR)*. ISSN: 2994-5828, pp. 1–4. DOI: 10.1109/âĎąTELEFOR63250.2024.10819112. URL: https://ieeexplore.ieee.org/document/10819112 (visited on 04/16/2025).

George, Swapna Mol and P. Muhamed Ilyas (Feb. 2024). "A review on speech emotion recognition: A survey, recent advances, challenges, and the influence of noise". In: *Neurocomputing* 568, p. 127015. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2023.127015. URL: https://www.sciencedirect.com/science/article/pii/S0925231223011384 (visited on 04/18/2025).

Ghasemieh, Alireza et al. (June 2023). "A novel machine learning model with Stacking Ensemble Learner for predicting emergency readmission of heart-disease patients". In: *Decision Analytics Journal* 7, p. 100242. ISSN: 2772-6622. DOI: 10.1016/j.dajour.2023.100242. URL: https://www.sciencedirect.com/science/article/pii/S2772662223000826 (visited on 04/21/2025).

Grosman, Jonatas (2021). *Fine-tuned XLSR-53 large model for speech recognition in English*. URL: https://huggingface.co/jonatasgrosman/wav2vec2-large-xlsr-53-english (visited on 04/21/2025).

Hashem, Ahlam, Muhammad Arif, and Manal Alghamdi (Oct. 2023). "Speech emotion recognition approaches: A systematic review". In: *Speech Communication* 154, p. 102974. ISSN: 0167-6393. DOI: 10.1016/j.specom.2023.102974. URL: https://www.sciencedirect.com/science/article/pii/S0167639323001085 (visited on 04/18/2025).

He, Kaiming et al. (Dec. 2015). *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs]. DOI: 10.48550/arXiv.1512.03385. URL: http://arxiv.org/abs/1512.03385 (visited on 04/21/2025).

Jackson, Philip and Sanaul Haq (2025). *Surrey Audio-Visual Expressed Emotion (SAVEE) Database*. URL: http://kahlan.eps.surrey.ac.uk/savee/ (visited on 04/21/2025).

James, Jesin, Li Tian, and Catherine Inez Watson (2018). "An Open Source Emotional Speech Corpus for Human Robot Interaction Applications". In: pp. 2768–2772. DOI: 10.21437/Interspeech.2018-1349. URL: https://www.isca-archive.org/interspeech_2018/james18_interspeech.html (visited on 04/18/2025).

Ke, Xianxin et al. (June 2018). "Speech Emotion Recognition Based on SVM and ANN". In: *International Journal of Machine Learning and Computing* 8.3, pp. 198–202. ISSN: 20103700. DOI: 10.18178/ijmlc.2018.8.3.687. URL: http://www.ijmlc.org/index.php?m=content&c=index&a=show&catid=78&id=799 (visited on 04/21/2025).

Lin, Tsung-Yi et al. (Feb. 2018). *Focal Loss for Dense Object Detection*. arXiv:1708.02002 [cs]. DOI: 10.48550/arXiv.1708.02002. URL: http://arxiv.org/abs/1708.02002 (visited on 04/21/2025).

Livingstone, Steven R. and Frank A. Russo (May 2018). "The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English". en. In: *PLOS ONE* 13.5. Publisher: Public Library of Science, e0196391. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0196391. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0196391 (visited on 04/21/2025).

MATLAB Help Center (2025). *resnet50*. en. URL: https://www.mathworks.com/help/deeplearning/ref/resnet50.html (visited on 04/16/2025).

McFee, Brian et al. (June 2015). "librosa: Audio and Music Signal Analysis in Python". en. In: *scipy*. DOI: 10.25080/Majora-7b98e3ed-003. URL: https://proceedings.scipy.org/articles/Majora-7b98e3ed-003 (visited on 04/21/2025).

Nfissi, Alaa et al. (2025). *Unveiling Hidden Factors: Explainable AI for Feature Boosting in Speech Emotion Recognition*. URL: https://arxiv.org/html/2406.01624v2 (visited on 04/17/2025).

Oh, Seoyoung (2025). *standing-o/Combined_Dataset_for_Speech_Emotion_Recognition*. original-date: 2023-11-12T10:44:14Z. URL: https://github.com/standing-o/Combined_Dataset_for_Speech_Emotion_Recognition (visited on 04/18/2025).

Paul, Bachchu et al. (Jan. 2024). "Machine learning approach of speech emotions recognition using feature fusion technique". en. In: *Multimedia Tools and Applications* 83.3, pp. 8663–8688. ISSN: 1573-7721. DOI: 10.1007/s11042-023-16036-y. URL: https://doi.org/10.1007/s11042-023-16036-y (visited on 04/17/2025).

Poria, Soujanya et al. (June 2019). *MELD: A Multimodal Multi-Party Dataset for Emotion Recognition in Conversations*. arXiv:1810.02508 [cs]. DOI: 10.48550/arXiv.1810.02508. URL: http://arxiv.org/abs/1810.02508 (visited on 04/21/2025).

Schuller, Björn et al. (2013). “The INTERSPEECH 2013 computational paralinguistics challenge: social signals, conflict, emotion, autism”. In: pp. 148–152. DOI: 10.21437/Interspeech.2013-56. URL: https://www.isca-archive.org/interspeech_2013/schuller13_interspeech.html (visited on 04/21/2025).

Tan, Mingxing and Quoc V. Le (Sept. 2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv:1905.11946 [cs]. DOI: 10.48550/arXiv.1905.11946. URL: http://arxiv.org/abs/1905.11946 (visited on 04/21/2025).

United Nations (2025a). *Goal 16 | Department of Economic and Social Affairs*. URL: https://sdgs.un.org/goals/goal16 (visited on 04/21/2025).

United Nations (2025b). *Goal 3 | Department of Economic and Social Affairs*. URL: https://sdgs.un.org/goals/goal3 (visited on 04/21/2025).

Venkataramanan, Kannan and Haresh Rengaraj Rajamohan (Dec. 2019). *Emotion Recognition from Speech*. arXiv:1912.10458 [cs]. DOI: 10.48550/arXiv.1912.10458. URL: http://arxiv.org/abs/1912.10458 (visited on 04/15/2025).

Wagner, Johannes et al. (Sept. 2023). “Dawn of the transformer era in speech emotion recognition: closing the valence gap”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.9. arXiv:2203.07378 [eess], pp. 10745–10759. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2023.3263585. URL: http://arxiv.org/abs/2203.07378 (visited on 04/21/2025).

Yan, Kevin (July 2023). *These are the 5 best pre-trained neural networks*. en. URL: https://medium.com/@kyan7472/these-are-the-5-best-pre-trained-neural-networks-23798e61a043 (visited on 04/15/2025).

Yang, Zijun et al. (Jan. 2024). “Optimizing Speech Emotion Recognition with Hilbert Curve and convolutional neural network”. In: *Cognitive Robotics* 4, pp. 30–41. ISSN: 2667-2413. DOI: 10.1016/j.cogr.2023.12.001. URL: https://www.sciencedirect.com/science/article/pii/S2667241323000411 (visited on 04/16/2025).

Zhou, Kun et al. (Feb. 2021). *Seen and Unseen emotional style transfer for voice conversion with a new emotional speech dataset*. arXiv:2010.14794 [cs]. DOI: 10.48550/arXiv.2010.14794. URL: http://arxiv.org/abs/2010.14794 (visited on 04/21/2025).

Zhou, Kun et al. (Jan. 2022). *Emotional Voice Conversion: Theory, Databases and ESD*. arXiv:2105.14762 [cs]. DOI: 10.48550/arXiv.2105.14762. URL: http://arxiv.org/abs/2105.14762 (visited on 04/21/2025).