Speech Emotion Recognition

CS460G - Final Project - Dr. Siddique

Authors:

Harrison Jacob

Braxton Goble

Kevin Dawson-Fischer

Alexander Dimayuga

Role of Each Member

GitHub Link

Kevin Dawson-Fischer
ID: 12691744

- Designed and implemented the Mel-spectrogram preprocessing pipeline from raw WAV files, including fixed sample rate/duration and export to PNG for model training and evaluation.
- Implemented the PyTorch CNN pipeline, including the custom spectrogram dataset class, train/validation/test data loaders, CNN architecture, training loop with validation monitoring, and final test-set evaluation and metric generation (classification report and confusion matrix).
- Helped define the reduced 3-class problem (anger, happy, neutral) and ensured the code is easily extendable to the full 6-class dataset.

Alexander Dimayuga
ID: 12495872

- Developed the script for extracting features for our baseline models from our Voice Emotion Dataset. Features include MFCC, RMS Energy, and Spectral Centroid.
- Implemented the pipelines for KNN and SVM, included Grid Search CV (Cross-Validation) to find the best hyperparameters of the models, and evaluated said models
- Created the GitHub repository for collaborative development, sourced the dataset used from Kaggle, organized file structure, and set comment headers/sections

Braxton Goble
ID:12508489

- Developed the realtime.py program and GUI to allow a user to use our KNN implementation to recognize the emotion of the user speaking into their microphone, in real-time.
- Added functionality to save the CNN model to a new file after each round of testing with new hyperparameters
- Created a real-time audio capture pipeline with energy and silence detection for the real-time recognition application.

Harrison Jacob
ID: 12447355

- Created all visuals to display different metrics on KNN, SVM, and CNN to help the user understand the training model learning parameters & results.
- Worked on researching and adding new features to improve scores of baseline algorithms, such as PCA, label encoding, and additional distance metrics (such as cosine) to the KNN & SVM algorithms to increase scores of the models.
- Tested, bug-fixed, and cleaned up the workflow of CNN & SVM algorithms to ensure simple and bug-free execution.

1.1 Project Objective

The objective of our project is to build a series of machine learning models capable of recognizing human voice tones, specifically anger, happiness, and neutrality, from short WAV recordings. We train three different classifiers: a spectrogram-based Convolutional Neural Network (CNN), and two classical baselines, k-Nearest Neighbors (KNN) and a Support Vector Machine (SVM), which use features extracted directly from the audio.

Voice tone carries a huge amount of information in everyday communication. Emotion-aware systems can enhance accessibility tools, improve virtual assistants, and make interacting with computers more natural. Unlike text-based emotion analysis, vocal emotion recognition requires pitch, frequency energy, rhythm, and acoustic cues, which are far more challenging to model. The rise of Artificial Intelligence (AI) agents has created new possibilities for tone-sensitive models, making vocal emotion recognition more relevant than ever before. As AI systems begin to increasingly interact with people in a conversational setting, the ability to interpret not just what was said but how it was said is essential for creating natural human-centered experiences.

One major challenge we faced was turning thousands of variable-length audio clips from the dataset into a consistent representation suitable for a convolutional neural network. The raw WAV files vary in duration and recording conditions, which makes them difficult to use directly. We addressed this by building a preprocessing pipeline that resamples every clip to 16 kHz, enforces a fixed 3-second window via padding or truncation, and then converts the audio into a 128-band Mel-spectrogram. This gave us a uniform time-frequency grid for all clips and made it possible to train image-based models on the audio.

Another challenge was balancing model complexity with the risk of overfitting on a relatively small subset of the data. Early CNN designs were either too shallow to capture useful patterns or began to memorize the training set. We iterated toward a compact VGG-style architecture with three convolution-pooling blocks followed by a small fully connected head with dropout, and we selected the final model based on validation accuracy instead of just the last epoch. This combination helped us control overfitting while still learning meaningful spectrogram features.

Finally, training a single model on every emotion in the dataset would have been difficult to manage within our time and computing budget. To keep the project feasible, we restricted our initial CNN experiments to three representative emotions (anger, happy, neutral), while designing the code and directory structure so that extending to additional emotions later would mainly require regenerating spectrograms and updating the class list.

2.1 Datasets

The dataset we used is the Voice Emotion Classification dataset we sourced from kaggle.com. The data subset that we used for experiments includes a total of 26,545 audio files, each representing a short clip of a person expressing a specific categorical emotion. For our project, we focused on three emotion classes: happiness (9315 samples), anger (9315 samples), and neutral (7915 samples). We chose these three particularly because they felt like the most distinct options, and would facilitate better learning for the models. Each audio file is a WAV recording with varying durations and sampling rates.

Before training our CNN, we standardized the audio to a 16,000 Hz sample rate, a 3-second fixed duration, and a 128-band mel-spectrogram representation. These processed

spectrograms served as the primary feature inputs for our CNN model. The baseline models use a separate feature-extraction pipeline directly on the audio. The features this pipeline extracted directly from the WAV recordings are:

- MFCC
    - Mel-Frequency Cepstral Coefficients capture the timbral and spectral characteristics of audio signals by modeling how humans perceive sound. Informs us how the energy in different frequency bands evolves over time. Summarizes the overall shape of the sound spectrum.
- Delta MFCC
    - First-order derivatives of MFCCs over time, capturing how the spectral features of an audio signal change frame to frame. Reveals the dynamics of speech and how it evolves. Measures how the shape changes, like tracking pitch, energy, or articulation shifts
- Zero-Crossing Rate
    - Measure of how frequently an audio signal changes sign (crosses the zero amplitude axis) within a given time frame. Reflects the noisiness, sharpness, or percussiveness of a sound
- RMS Energy
    - Measure of the average power of an audio signal over time. Reflects how loud or intense a sound is. A key feature for analyzing speech dynamics, emotion, and musical texture
- Spectral Centroid
    - Indicates the "center of mass" of the spectrum. Tells you where the majority of the energy in a sound is concentrated along the frequency axis. Often used to describe the brightness or sharpness of a sound
- Spectral Bandwidth
    - Measures the spread of frequencies around the spectral centroid in an audio signal. Tells you how wide or narrow the frequency content is. Essentially how much of the spectrum is active at a given moment
- Spectral Roll-Off
    - Indicates the frequency below a specified percentage (in this case 85%) of the total spectral energy is contained. It is a measure of how quickly the energy in a signal "rolls off" toward higher frequencies. Essentially tells you how much high-frequency content is present
- Spectral Contrast
    - Measures the difference in energy between peaks and valleys in the frequency spectrum across multiple sub-bands. Captures how tonal vs. noisy a sound is

2.2 Samples from the Dataset

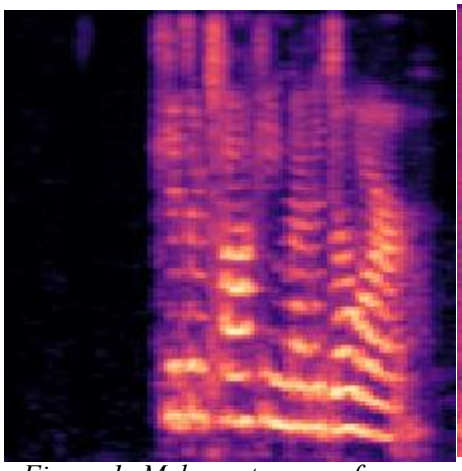


Figure 1: Mel-spectrogram from a 3-second "angry" utterance showing higher energy in low and mid freauency bands.

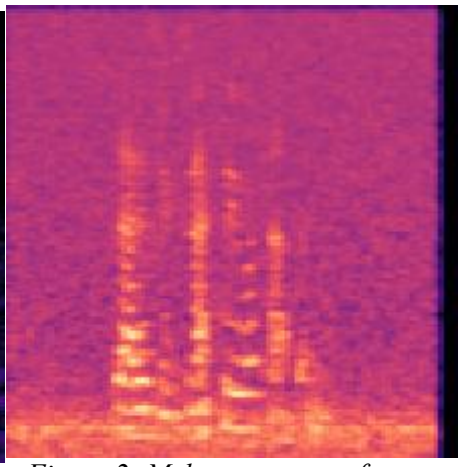


Figure 2: Mel-spectrogram from a 3-second "happy" utterance showing higher energy in low and mid frequency bands.

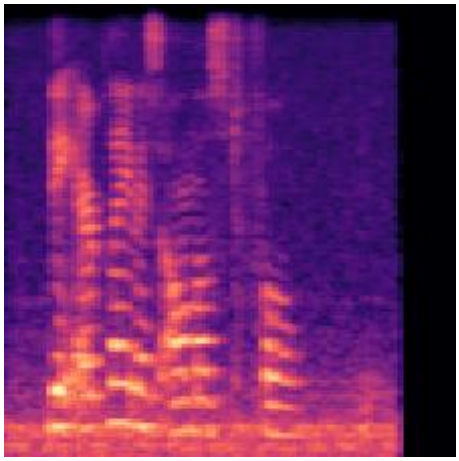


Figure 3: Mel-spectrogram from a 3-second "neutral" utterance showing higher energy in low and mid freauency bands.

3. Evaluation Metric

For the CNN, we evaluate model performance primarily using accuracy on a held-out test set. Accuracy is a natural global metric for our three-class classification problem, especially because the selected emotions are reasonably balanced in our subset of the dataset.

In addition to accuracy, we generate a detailed classification report that includes precision, recall, and F1-score for each emotion, along with macro-averaged and weighted-averaged scores. These metrics help us understand whether the model is biased toward or against particular emotions. We also compute and save a confusion matrix, which visualizes how often each true emotion is predicted as each other emotion. The confusion matrix is particularly useful for inspecting mistakes, such as angry samples being misclassified as neutral. For the CNN, both the classification report and the confusion matrix plot are automatically saved for use in the analysis.

For the KNN model, we evaluate performance using accuracy on an 80/20 stratified train-test split. Accuracy serves as the main metric for comparing KNN to the other approaches in this three-class classification task. We also generate a classification report that includes precision, recall, and F1-score for each emotion, as well as macro and weighted averages, to better understand class-by-class performance. A confusion matrix visualization is included to highlight where the model tends to make mistakes and whether certain emotions (such as happy and angry) are more easily confused.

For the SVM model, we use the same evaluation procedure: accuracy on an 80/20 stratified split as the primary metric, supported by precision, recall, and F1-scores reported per class. We test both linear and RBF kernels using grid search with cross-validation and select the best-performing configuration for final testing. As with KNN, we generate a confusion matrix to analyze error patterns and compare how well each kernel separates the emotion classes.

4. Main and Baseline Models

To establish performance benchmarks for our recognition system, we implemented three models that operate on different feature representations of the audio data.

K-Nearest Neighbors (KNN)

Our first baseline employs a K-Nearest Neighbors classifier trained on extracted audio features including Mel-Frequency Cepstral Coefficients (MFCCs), spectral features (spectral centroid, rolloff, bandwidth, and contrast), zero-crossing rate, chroma features, and root-mean-square energy (RMS). These hand-crafted features capture various acoustic properties, including spectral, temporal, and prosodic characteristics of the speech signal, and are stored in features.csv.

The pipeline includes an 80/20 train-test split with stratification to maintain class balance, a feature scaling standard scaler normalization to ensure all features contribute equally to distance calculations, and hyperparameter tuning with a GridSearchCV with 5-fold cross-validation. We iterated through different distance metrics like Euclidean and Manhattan, different n-neighbor amounts, etc. The KNN model serves as a simple baseline that requires no training phase and makes predictions based purely on the similarity of features.

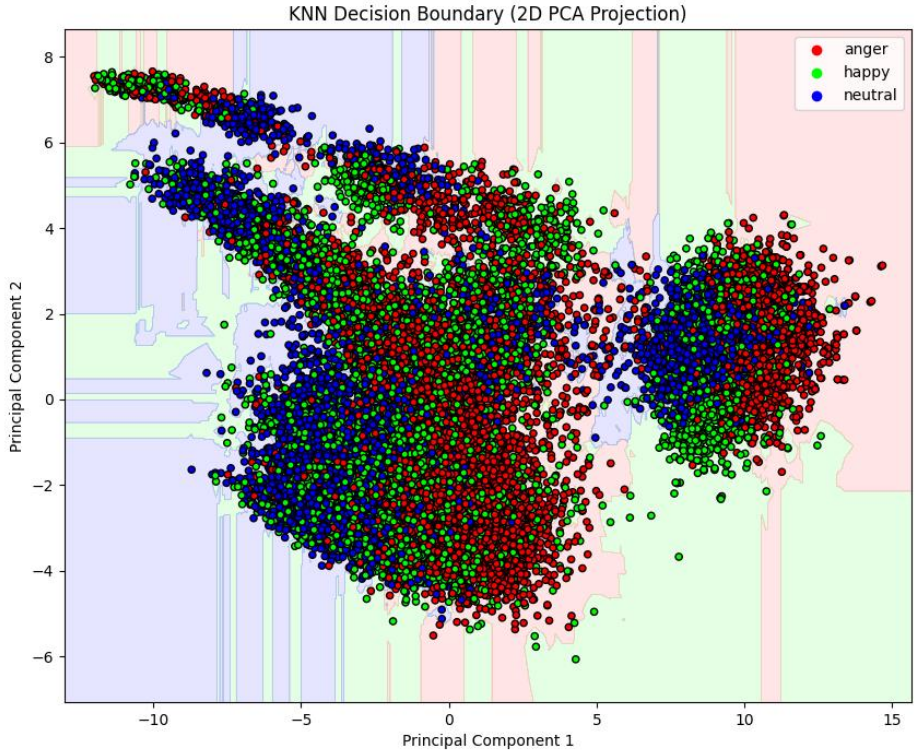
Support Vector Machine (SVM)

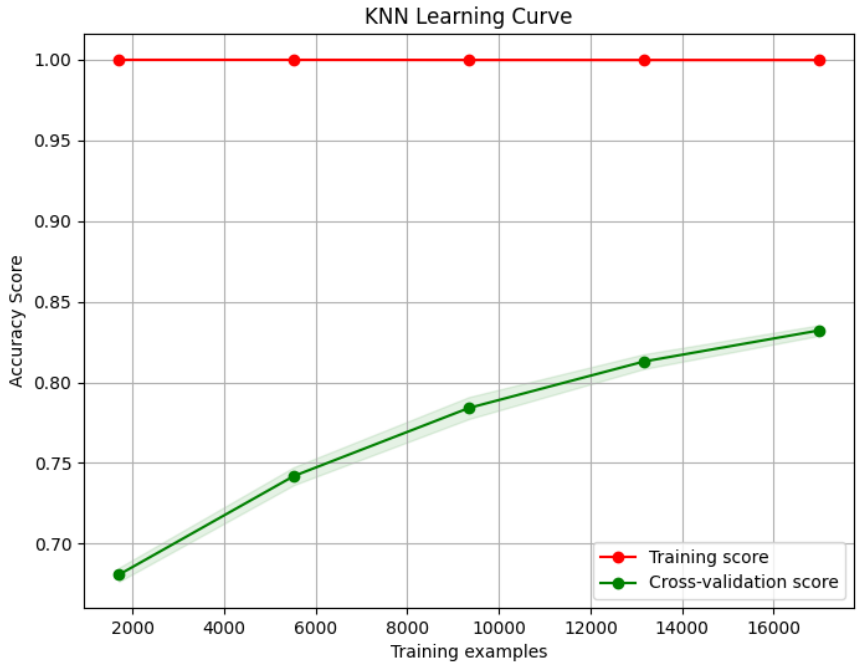
Our second baseline uses an SVM classifier on the same audio feature set. We explored both a linear and an RBF kernel through grid search. Linear SVM tuned over regularization parameters (C: [0.1, 1, 10]) and class weighting strategies. RBF SVM tuned the gamma parameter (['scale', 0.01]) to control the kernel's influence radius. The kernel configuration that achieved the highest 5-fold cross-validation accuracy was selected for final evaluation. The SVMs provide a robust baseline that can capture non-linear decision boundaries while maintaining good generalization through the margin maximization principle.

Convolutional Neural Network (CNN)

Our CNN baseline processes mel-spectrogram images (128x128 pixels) rather than hand-crafted features, allowing the model to learn acoustic patterns directly from the time-frequency representation. It is a three-layer convolutional network with progressive channel expansion, followed by fully connected layers with dropout (p=0.3) for regularization, trained using a 70/15/15 train/validation/test split with a fixed random seed for reproducibility. Adam optimizer (learning rate $1x10^{-3}$) for 20 epochs, with final model selection based on validation accuracy. Grayscale spectrograms were resized to 128x128 and normalized to the [-1,1] range. This CNN baseline approach learns hierarchical feature representations automatically, providing a strong comparison point for evaluating whether hand-crafted features (KNN/SVM) or learned features (CNN) better capture fluctuations associated with certain emotions in speech.

4.1 Main and Baseline Model Visualization Output Examples





5.1 Preprocessing: WAV --> Mel Spectrogram PNG

We implemented a dedicated preprocessing script to convert the Kaggle WAV files into fixed-shape mel-spectrogram PNGs. The script walks a root directory (by default data/Voice Emotion Dataset/) recursively, treats the immediate parent folder of each .wav file as its label (for example, …/happy/*.wav is labeled "happy"), and records the (path, label) pairs. For each file, it (1) loads the audio in mono at 16 kHz, (2) enforces a fixed 3-second duration by padding or truncation, (3) computes a 128-band Mel-spectrogram with n_fft = 2048, hop_length = 512, fmin = 20, and fmax = 8000, and (4) converts the Mel-spectrogram to dB using librosa.power_to_db. Each spectrogram is then rendered as a 224x224 PNG with a "magma" colormap, no axes, and tight cropping, and saved to data/spectrograms/<label>/<filename>.png.

The script exposes key parameters (sample rate, duration, number of Mel bands, frequency range, FFT size, and hop length) as command-line arguments. This allows us to regenerate spectrograms with different settings without modifying the core code, which is useful for experimenting with alternative feature configurations.

5.2 Data Loading & Splits for the CNN

For CNN, we implement a custom PyTorch Dataset called SpectrogramDataset. It expects a root directory like data/spectrograms/ and a list of emotion labels. The dataset walks each label folder, collects image paths, and stores them along with the corresponding label index. Each image is opened in grayscale, resized to 128x128 pixels, converted to a tensor, and normalized with a mean of 0.5 and standard deviation 0.5 so that the input values are roughly centered around 0.

We split this dataset into training, validation, and test sets using a 70/15/15 ratio with a fixed random seed (420) to ensure reproducibility. The training loader shuffles data each epoch,

while the validation and test loaders do not. All CNN experiments in this report use these same splits and data loaders.

5.3 CNN Architecture

Our main model is a two-dimensional convolutional neural network designed to operate on spectrogram images. The network takes a 1-channel 128x128 image and passes it through three convolutional blocks followed by a fully connected classifier:

1. Block 1: 3x3 convolution with 1 -> 16 channels (padding 1), ReLU, and a 2x2 max-pooling (spatial size 128x128 -> 64x64)
2. Block 2: 3x3 convolution with 16 -> 32 channels, ReLU, and 2x2 max-pooling (64x64 -> 32x32)
3. Block 3: 3x3 convolution 32 -> 64 channels, ReLU, and 2x2 max-pooling (32x32 -> 16x16)

The resulting 64x16x16 feature map is flattened and passed into a classifier head consisting of a linear layer to 128 hidden units, a ReLU activation, a dropout layer with dropout probability 0.3, and a final linear layer with three outputs corresponding to anger, happy, and neutral. This is a simplified VGG-style design that uses small 3x3 filters and pooling to gradually build higher-level features while keeping the overall parameter count modest for our dataset size.

5.4 Training Procedure & Hyperparameters

We train the CNN using the Adam optimizer with a learning rate of $1x10^{-3}$ and a batch size of 32. The loss function is cross-entropy, which is standard for multi-class classification problems. We train for 20 epochs and, at each epoch, compute both training and validation loss and accuracy. To avoid overfitting, we track the model state that achieves the highest validation accuracy and restore that "best" checkpoint at the end of training before evaluating on the held-out test set.

We set fixed random seeds for both PyTorch and NumPy to make the results repeatable and automatically choose the computation device at runtime: the code uses a GPU if one with CUDA cores is available and falls back to the CPU otherwise. After evaluation, the script writes the text classification report (cnn_classification_report.txt) and the confusion matrix image (cnn_confusion_matrix.png) into metrics/cnn_outputs/ for use in the Results section.

Results

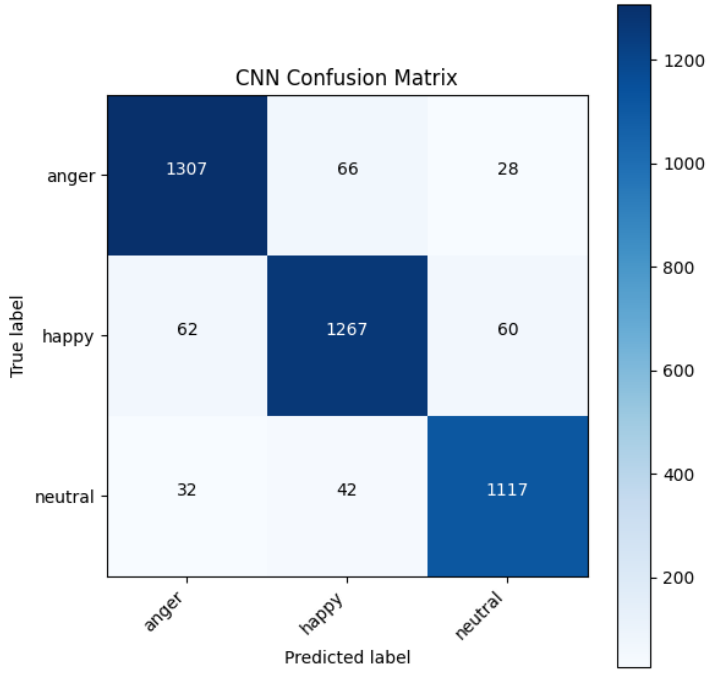
6.1a CNN vs Baselines Table

Model	**Accuracy**	**Macro F1**	**Weighted F1**	**Precision (Avg)**	**Average (Avg)**
KNN	82.4%	82.4%	82.4%	82.4%	82.4%
SVM	87.7%	87.8%	87.2%	87.8%	87.7%
CNN	93.2%	93.2%	93.2%	93.2%	93.2%

6.1b Per-Class F1 Scores:

Model	Anger	Happy	Neutral
KNN	86.9%	77.5%	82.9%
SVM	90.5%	84.3%	87.7%
CNN	93.9%	91.7%	94.1%

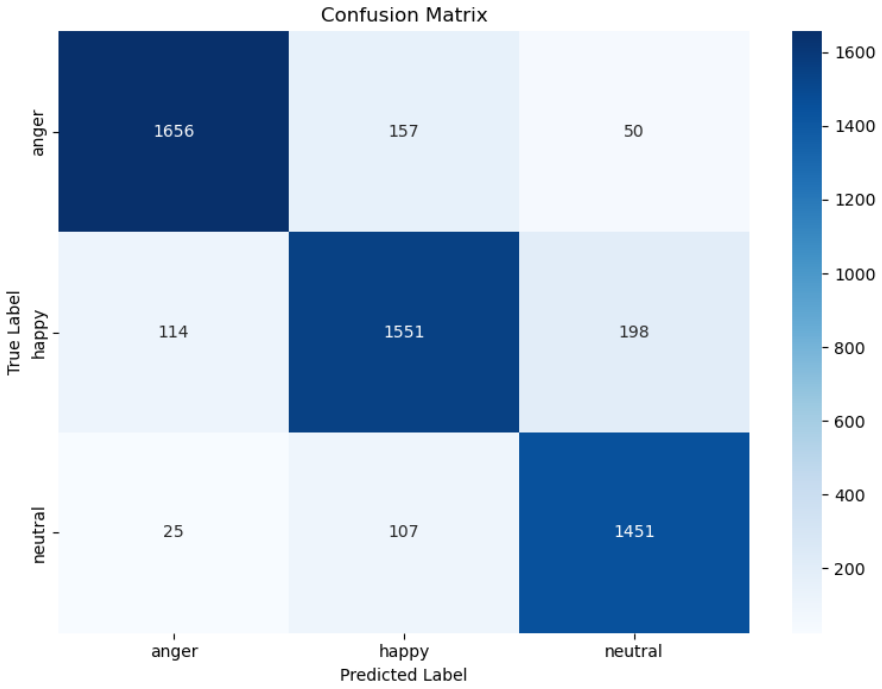
6.1c CNN Confusion Matrix



6.1d CNN Evaluation and Classification Report

Class	Precision	Recall	F1-Score	Support
Anger	0.9451	0.9336	0.9393	1401
Happy	0.9190	0.9143	0.9166	1389
Neutral	0.9317	0.9505	0.9410	1191
Accuracy	—	—	0.9319	3981
Macro Avg	0.9319	0.9328	0.9323	3981
Weighted Avg	0.9320	0.9319	0.9319	3981

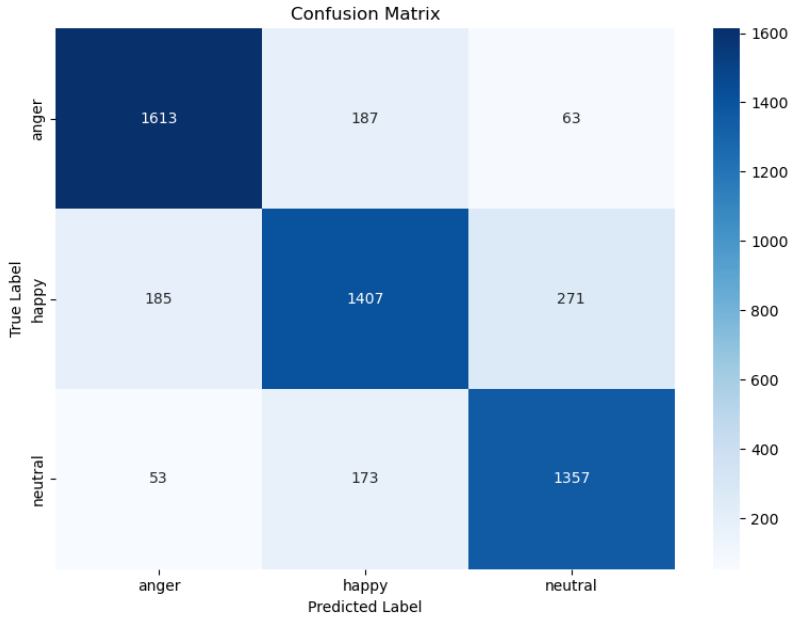
6.1e SVM Confusion Matrix



6.1f SVM Evaluation and Classification Report

```
Model Evaluation Metrics
================================================
Accuracy : 0.877
Precision: 0.878
Recall   : 0.877
F1-Score : 0.877
================================================

Classification Report:
================================================
              precision    recall  f1-score   support

       anger      0.923     0.889     0.905      1863
       happy      0.855     0.833     0.843      1863
     neutral      0.854     0.917     0.884      1583

    accuracy                          0.877      5309
   macro avg      0.877     0.879     0.878      5309
weighted avg      0.878     0.877     0.877      5309

================================================
```

6.1g KNN Confusion Matrix



6.1h KNN Evaluation and Classification Report

```
Model Evaluation Metrics
================================================
Accuracy : 0.824
Precision: 0.824
Recall   : 0.824
F1-Score : 0.824
================================================

Classification Report:
================================================
              precision    recall  f1-score   support

       anger      0.871     0.866     0.869      1863
       happy      0.796     0.755     0.775      1863
     neutral      0.802     0.857     0.829      1583

    accuracy                          0.824      5309
   macro avg      0.823     0.826     0.824      5309
weighted avg      0.824     0.824     0.824      5309

================================================
```

6.2 Outperforming Baselines

Our CNN-based approach outperformed both baseline models significantly, achieving an accuracy of 93.2% on the unseen test set for the anger, happy, and neutral classes, a 5.5 percentage point gain over SVM and a 10.8 percentage point increase over KNN. CNN's superior performance can be explained by several factors. Automatic feature learning allows the CNN to learn hierarchical representations directly from mel-spectrogram images, capturing complex temporal and spectral patterns that are more complex than manually encoded features. This is particularly evident in the “happy” class, where CNN achieved 91.7% compared to SVM’s 84.3% and KNN’s 77.5%. The convolutional layers effectively capture the local patterns

in the spectrograms, identifying emotion-specific acoustic signatures such as pitch contours, energy distributions, and formant structures that hand-crafted features may not fully represent.

The three-layer convolutional architecture progressively learns increasingly abstract features, from lower levels capturing spectral edges to higher-level emotionally formed patterns, providing richer representations than fixed feature sets of the baselines. The dropout layer (p=0.3) and validation-based early stop mechanism prevented the model from overfitting to the training data, while the Adam optimizer navigated the high-dimensional parameter space, allowing the model to generalize to unseen data.

Looking at per-class performance in Table 6.1b, all three models find the happy class more challenging than anger or neutral. KNN achieves F1-scores of 86.9% (anger), 77.5% (happy), and 82.9% (neutral), while SVM reaches 90.5%, 84.3%, and 87.7% respectively. The CNN improves these to 93.9% (anger), 91.7% (happy), and 94.1% (neutral), reducing the gap for the happy class the most. This suggests that happy utterances share acoustic similarities with the other emotions and are easier to confuse, but that the CNN's learned spectrogram features capture more of the subtle cues that distinguish them.

6.3 Additional Improvements

If given more time, we would integrate a method of combining spectrogram images with hand-crafted features used in the baseline models to create an ensemble approach that leverages both methods to get even better accuracy.

Future Work and Lessons Learned

7.1 Summary of Potential Expansions

Our real-time voice emotion recognizer currently only uses our KNN model, but in the future, we could add the ability to switch between the models, utilizing the CNN, KNN, and SVM.

Going forward, we could plan to extend our spectrogram and CNN pipeline from the current three-class subset (anger, happy, neutral) to more or all emotions available in the Kaggle dataset. Because the preprocessing and data-loading code already generalizes to arbitrary label folders, scaling up mainly requires regenerating spectrograms and updating the list of class labels in the CNN script. We could also have experimented with deeper CNN architectures, batch normalization layers, and audio or spectrogram augmentations (such as time shifts and time/frequency masking) to further improve robustness.

7.2 Key Lessons Learned

From building the CNN pipeline, we learned that decisions made at the preprocessing stage (sample rate, fixed duration, Mel filter settings, and even how spectrograms are rendered) have a direct impact on model performance. We also saw how important it is to monitor validation performance and save the best model checkpoint, especially when working with a relatively small subset of a large audio dataset. Overall, the spectrogram and CNN approach gave us a clear, end-to-end path from raw audio files to interpretable metrics for speech emotion recognition.