CS360 Machine Learning Final Competition Report

Jae Gao

May 2024

1 Introduction

This report documents my approach and results for the CS360 Machine Learning Final Kaggle Competition. The goal was to classify 3-second song snippets into four categories: no voices, one male-like voice, one female-like voice, or multiple voices.

2 Data Preprocessing

The provided data consisted of MP3 audio files. I extracted the files using the `tarfile` library in Python. For feature extraction, I used the `librosa` library to compute Mel-frequency cepstral coefficients (MFCCs) with 40 coefficients for each audio file. I then took the mean across time to get a single 40-dimensional feature vector per snippet. To speed up preprocessing, I used `ThreadPoolExecutor` for parallel file processing. After preprocessing, I had 40-dimensional feature vectors for both the training and test sets, with corresponding integer labels for the training set. To avoid recomputing features, I saved the preprocessed data using `pickle`. Subsequent runs loaded the saved data if available.

3 Model Architecture and Training

3.1 Simpler Model

I used a feedforward neural network implemented in PyTorch. The architecture consisted of:

- An input layer with 40 units (for the 40 MFCC features)
- A hidden layer with 128 units and ReLU activation
- 50

- A second hidden layer with 64 units and ReLU activation
- 50
- An output layer with 4 units (for the 4 classes)

I used cross entropy loss, Adam optimizer with learning rate 0.001, and trained for 500 epochs. The model was trained on an NVIDIA GPU. To track performance during training, I computed accuracy on the validation set after each epoch.

3.2 More Complex Model

In addition to the simpler feedforward model, I also experimented with a more complex model architecture. The more complex model included the following components:

- Additional audio features like Mel spectrograms, chroma, and tonnetz were extracted using the `librosa` library.
- The model architecture was expanded to include convolutional layers and more fully connected layers.
- Batch normalization layers were added after each convolutional layer to normalize the activations and improve training stability.
- Dropout layers with higher dropout rates were used to reduce overfitting.
- The model was trained using a combination of cross-entropy loss and focal loss to handle class imbalance.
- Data augmentation techniques such as SpecMix were applied to further increase the training data diversity.

The more complex model aimed to capture a wider range of audio features and improve the model's ability to handle class imbalance and generalize to unseen data.

3.3 Training Details

The models were trained using an NVIDIA RTX 4090 GPU with 24GB VRAM, an Intel Core i9-13900K CPU with 32 threads, and 64 GB DDR5 memory. A batch size of 8192 was used for both training and testing data loaders. The simpler model was trained for a total of 500 epochs, while the more complex model was trained for a longer duration. For the simpler model, the training process took approximately 30 seconds to reach 150 epochs. I experimented with longer training durations, but it was observed that the model started overfitting, with both training and validation losses worsening. This suggests that the model had learned the training data too well and was not generalizing effectively to

unseen data. The more complex model took about 10 minutes to train. However, despite the additional complexities and augmentations, the more complex model did not significantly outperform the simpler model in terms of accuracy on the test set.

4 Results

My best model achieved an accuracy of 0.6755 on the test set.

5 Additional Experiments

In addition to the simpler and more complex models, I also tried further tuning and ensembling methods:

- Stratified K-Fold cross-validation was used for model selection.
- An ensemble of the feedforward neural network, logistic regression, SVM, and decision tree models was trained using soft voting.
- Hyperparameter tuning was performed using grid search over various model and training parameters.

However, these additional techniques did not improve the accuracy beyond the simpler feedforward model. This suggests that the feedforward model was able to effectively learn relevant features from the MFCC representation, and further feature engineering and model complexity did not provide additional benefit for this task.

6 Conclusion

In summary, I developed neural network models to classify song snippets based on voice content. The key steps were computing MFCC features and training feedforward neural network architectures. My final model achieved 0.6755 test accuracy. While a more complex model with additional audio features and augmentations was explored, the simpler feedforward model proved most effective, likely due to its ability to learn relevant features from the MFCC representation. Potential areas for further improvement could include exploring more advanced audio-specific neural architectures like SampleCNN or WaveNet. The training process was conducted using an NVIDIA RTX 4090 GPU, Intel Core i9-13900K CPU, and 64 GB DDR5 memory. The simpler model took approximately 30 seconds to train for 150 epochs, while the more complex model took about 10 minutes. Longer training durations were experimented with, but it was observed that the models started overfitting, with both training and validation losses worsening. The simpler model was chosen for the final submission due to its better performance, faster training time, and better generalization.

Overall, this project provided valuable experience in applying machine learning techniques to audio classification tasks and participating in a Kaggle competition. The results demonstrate the effectiveness of neural networks in learning discriminative features from MFCC representations of audio data.