CS583 Final Project Report

Team: Ji Zhang (Anna) and Wei-heng Lin (Wayne)

Topic: Speech Accent Classification and Recognition

Context

English is the most spoken second language in the world, and people speak the language with various accents. Even native English speakers do not have a unified accent when speaking the language. This project aims to classify a person's native language when he/she gives a recording of himself/herself speaking a given text. We believe that this project is significant because it might be useful for voice recognition. For instance, a voice assistant could first identify the accent of the user and apply relevant models to improve its accuracy.

Dataset

For this project, we are going to use George Mason University's Speech Accent Archive (http://accent.gmu.edu). This dataset contains 2140 speech samples, each is recorded by a speaker reading the same text in English. Speakers come from 177 countries and have 214 different native languages. We also have several other supporting datasets for testing or training purposes. It includes Arabic, Spanish, German, French, Chinese, and native English speakers speaking English speech data.

Our Approach

We started our project with simple feature analysis and found that the mel-spectrum has some patterns across different accents. Thereby, we tried to build a Convolutional Neural Network based on mel-spectrum images.

To clear the white noise in the audio file, we used the white noise cancelation function from homework 2. Then we applied a normalized function to make sure all the audio files have the same largest amplitude and export all the mel-spectrum images of each audio file. Then, we used these png files as the primary source of data for our CNN.

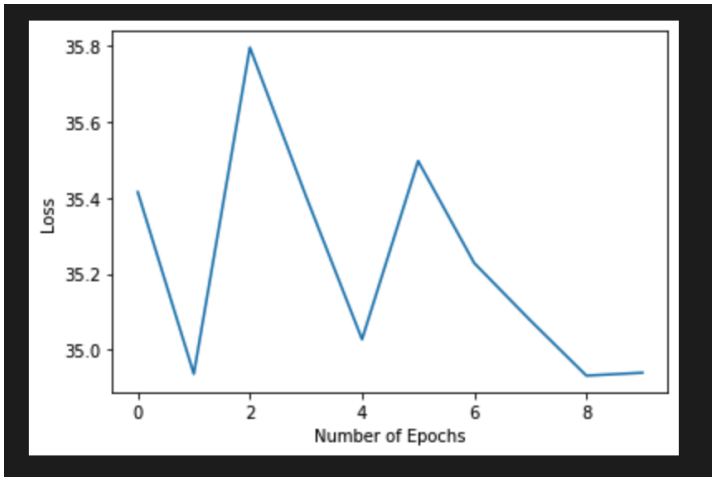
The way that we split up the training and testing dataset is based on the 80-20 rule (i.e. 80% of the data is used for training and 20% of the data is used for testing)

The CNN algorithm we are building is based on Wayne's previous project made in BUMIC and the Instrument Detection code provided by Professor Snyder. The major difference is that we will be using PyTorch as our main framework for CNN, instead of Tensorflow.

Training Process

Since CNN finds features automatically, our job is to find the correct hyperparameters that generate the minimum loss and the highest correct prediction rate. The followings are snippets of our attempts at hyper-tuning the network:

Attempt #1

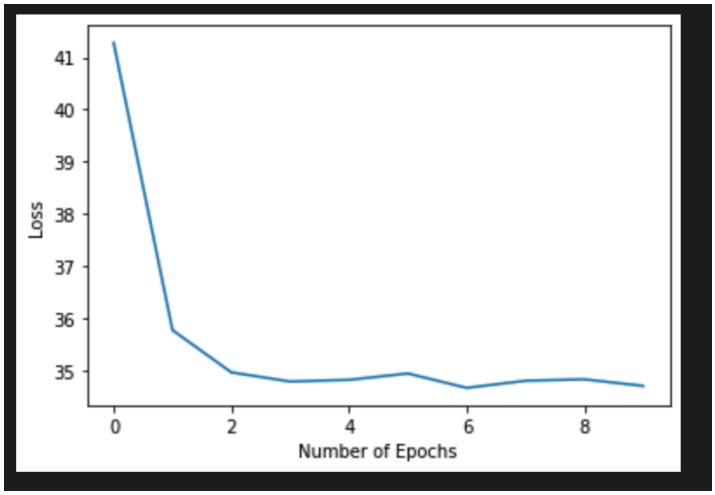


Hyperparameters: num_epochs = 10, batch_size = 16, learning_rate = 0.005

Result: loss =34.94, correct prediction rate = 68.77%

Analysis: We can see a volatile loss function produced by the given hyperparameters. This model is certainly not desirable as we want to have a loss function that gradually steps downhill. We thought that this might be caused by overshooting (i.e. the learning rate is too large that it goes over the optimum loss).

Attempt #2

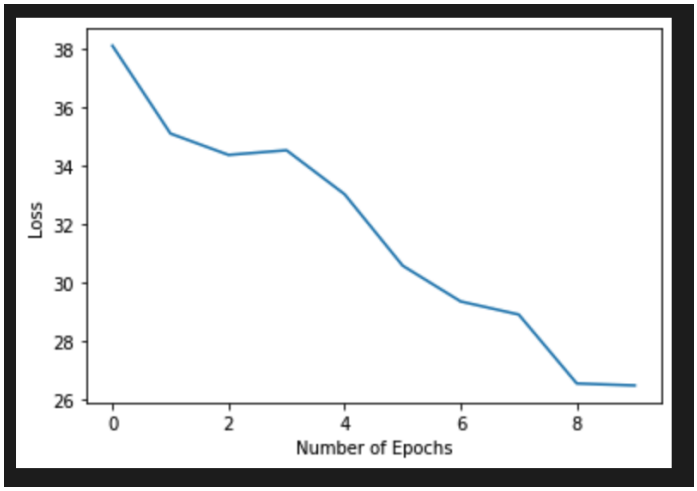


Hyperparameters: num_epochs = 10, batch_size = 8, learning_rate = 0.0001

Result: loss = 34.71, correct prediction rate = 68.77%

Analysis: This loss function looks a lot more desirable as it shows a constant decrease of loss values. However, the correct prediction rate did not increase. The reason is perhaps that the final loss value is very close to the one in the previous attempt. Therefore, in this case, the learning rate might be too small for the model to make further improvements at the end.

Attempt #3



Hyperparameters: num_epochs = 10, batch_size = 8, learning_rate = 0.001

Result: loss =26.47, correct prediction rate = 71.32%

Analysis: In this attempt, the final loss value has improved compared to the previous attempt (from 34.71 to 26.47). Correspondingly, the prediction rate also improves by roughly 3%. After multiple other attempts of hyper-tuning, attempt#3 has been the one that we managed to generate the lowest loss value and highest correct prediction rate. Therefore, we decided to go with these hyperparameters in the actual validation process.

Validation (Testing) Process

We ran the model with the hyperparameters found in attempt#3 on the testing dataset. The final correct prediction percentage is 67.63%, which we believe is reasonable as the percentage error is around 5%.

Future

The final result is reasonable but definitely has room for improvement. Due to the limited time and resources, we didn't get to finish our second approach, which is separating each word and building the CNN based on some specific words. We have tried some existing speech separation python packages, but the accuracies are even worse than our first approach. If we have time in the future, we could work on this approach and potentially use a model where we can identify the features that we want to capture (instead of relying on auto-feature mapping in CNN).