Accent Classification: Identifying native- and foreign-accented english using convolutional neural network

CMSC 263 Final project report

Jiangxue Han, Ruiming Li, Shi Jie Samuel Tan, Xiaorong Wang

1. Introduction

In recent years the development of voice recognition has been successful, with the prevalence of mobile assistants such as Alexa and Siri, and also home security and service systems. However, because the way training data is collected for most popular systems, only the "standard accent" is picked often, resulting in poor performance for classification for users with different accents. As international students, we, the team members, have experienced certain difficulties talking to our own mobile assistants with non-standard American accents. Since we have already covered a lot of other traditional machine learning methods in the labs in this class, we decided to explore this new area using convolutional neural networks, which is a method popular in the field of voice recognition. We have used Leon Mak An Sheng and Mok Wei Xiong Edmund's paper Deep Learning Approach to Accent Classification (2017) as guidance to our project. Our goal is to classify a participant by his or her accent, with input as a voice segment of this person pronouncing certain words.

2. Dataset

We used the Wildcat Corpus of Native and Foreign-Accented English, a free data source containing a scripted reading scenario in which participants clearly enunciated scripted words one at a time. This dataset is also what the authors of Deep Learning Approach to Accent Classification (2017) chose to use. The data source has multiple categories based on participants native languages, and we only chose those conversations in which both speakers have the same native language. The reason why we did this is that the datasets are huge and our computers are not powerful enough to run all of them in a reasonable time, and those with speakers with the same native language save us the trouble of separating the dialogue to find out which segments are uttered by who. This might have some limitations due to potential influence of a person's accent by the person he or she is conversing with.

3. Method

3.1 Overview

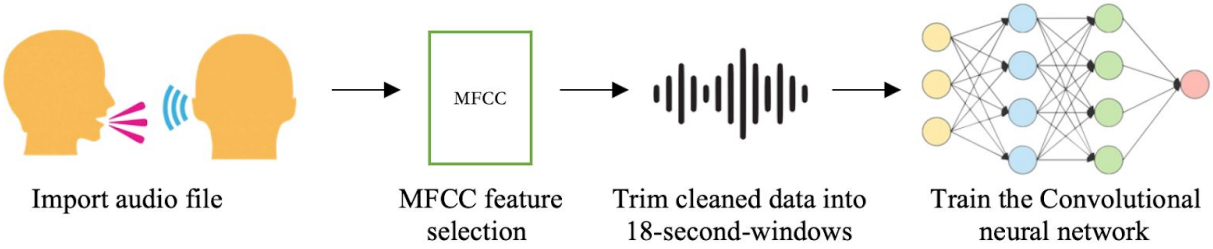


Figure 1 Overview

First, we read and import audio from Wildcat Corpus of Native- and Foreign-Accented English in extract.py. We then removed outliers and extracted features of the audio file by their MFCCs (mel-frequency cepstral coefficients). The data is then segmented into windows of 18 seconds' uniform length. In preprocess.py we sorted the data into training and testing dictionaries, and then loaded them into torch in the form of dataloaders. To save time, instead of running data extraction and preprocess everytime we debug or test, we put the preprocessed data into json files to be used directly in training. We then built the 2-layer convolutional neural network and implemented training and testing. We used data for English, Chinese and Korean native speakers.

3.2 Preprocess

We used librosa library to extract mfcc coefficients from recording files. We chose to use mfcc coefficients because it is said to be appropriate and used frequently. We found the start and end time of each word in the data file and segment out each word's audio. And then further divide it into segments of 0.18 seconds. We also removed outliers with energy density lower than 4.8% of average energy density, as suggested in the paper

Deep Learning Approach to Accent Classification (2017).We went through the data and put them into a dictionary with the key representing the speaker's native language and value the data segment. Then data is divided by a ratio of 4:1 of training and testing data. For training data we added Gaussian noise to avoid overfitting problems. We have a Dataset class to turn our dictionary into dataloaders that can be used by pytorch. The Dataset class makes a map-like object that has a list of IDs for each data segment and label, and turns the dictionary into tensor. Then dataloaders of training data and testing data are both created. We have 57576 data segments for training and 7207 segments for testing.

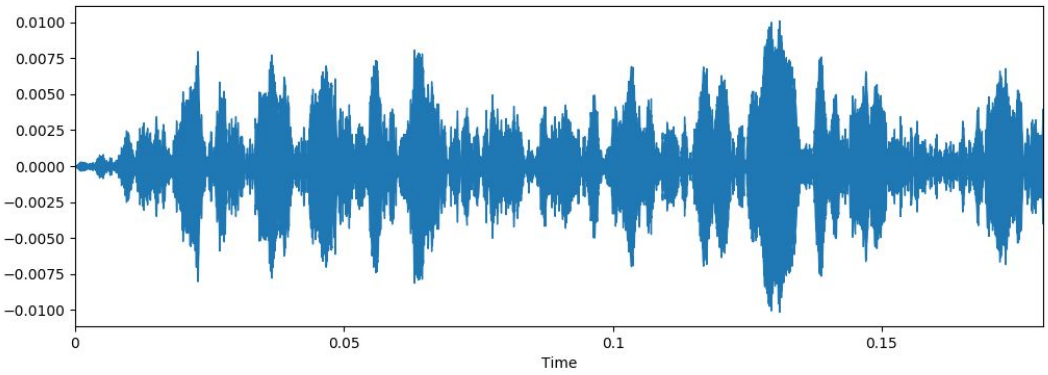


Figure 2 Waveform Plot for a window of 0.18 seconds

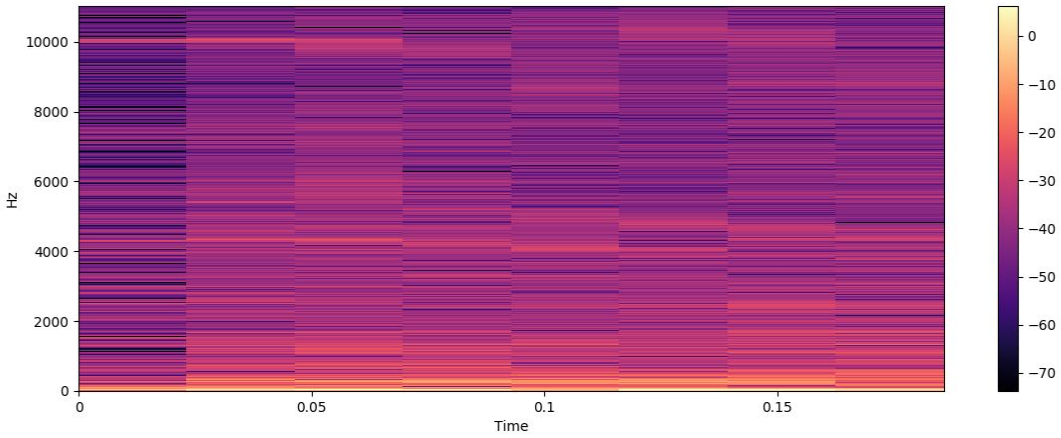


Figure 3 MFCC Frequency Plot for a window of 0.18 seconds

3.3 Network and training

The MFCC arrays we extracted are used to train the convolutional neural network (CNN) we created.

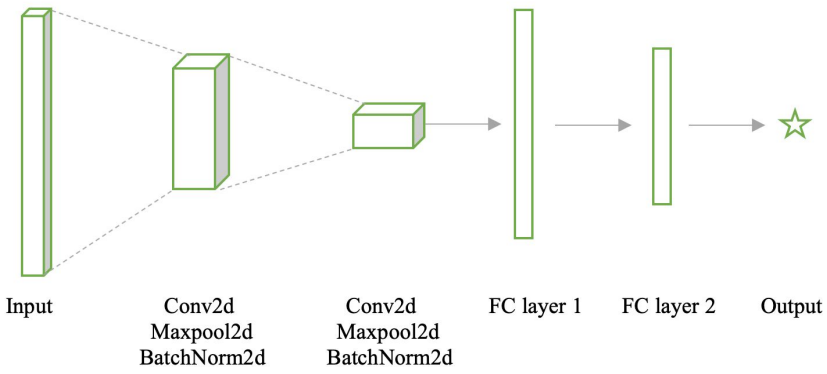


Figure 4 Network overview

Our model is implemented with Pytorch.

We have initially tried a convolutional network with 5 convolutional layers, however we found it too complicated for this model and it takes too long to run, so for the finalized model we use a 2-layer CNN design. Except for the CNN layers, the model also contains two max pooling layers that aims to reduce the dimensionality of the matrix. It also contains two batch normalization layers which are designed to speed up learning.

Specifically, after inputting the data with batch size 32, which first goes through a convolutional network with kernel size 3. The ReLu activation function is used after the network to extract more features. Then, it goes through the max pooling layer with size 2*2 and stride 2. After the max pooling layer, the matrix is processed with the batch normalization layer. The same functional layers (convolutional layer, activation function, max pooling and batch normalization) are applied again.

The final layers are densely connected, also known as the fully connected layers with a dropout layer of 0.2 to prevent overfitting of the model. Another linear-fully-connected layer is applied after the dropout layer. The final layer is applied with a LogSoftmax function to output the confidence of the prediction of each class.

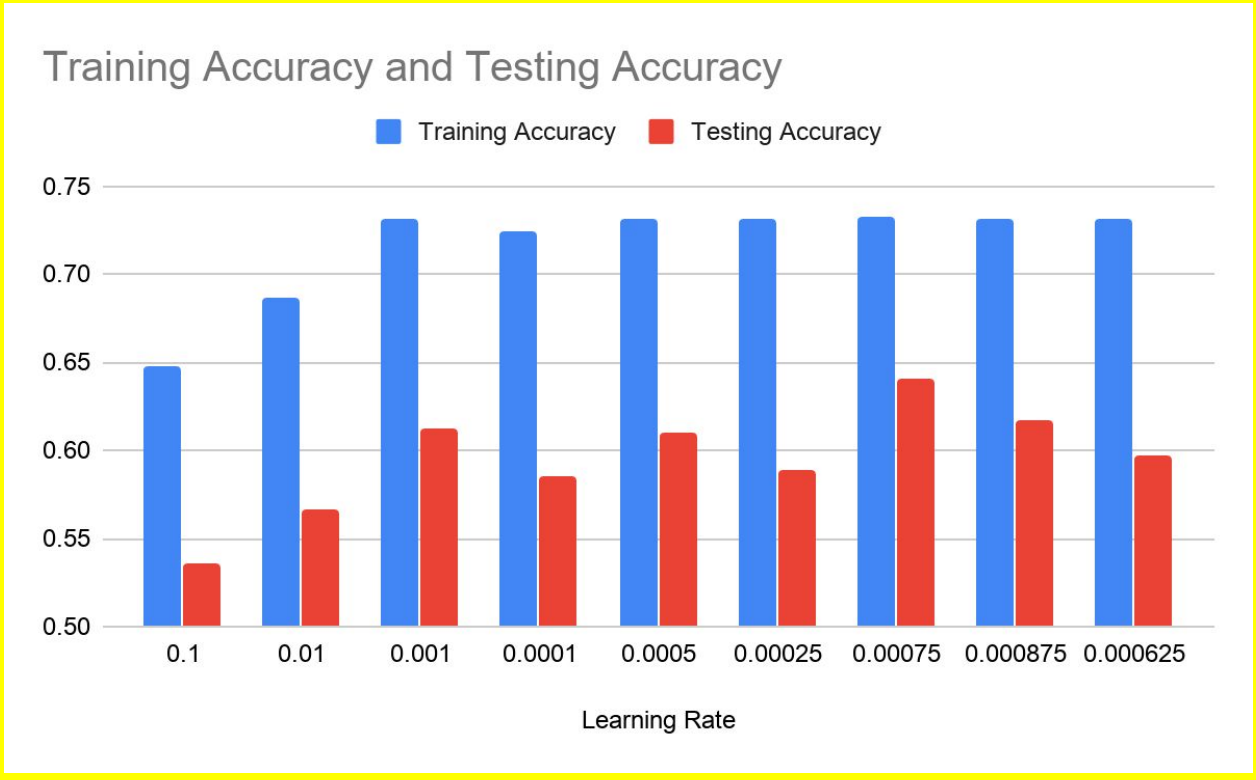
We tried to train the model with a learning rate of 0.0001, a batch size of 32 and a controlled number of epochs.

4. Results

After successfully building our neural network, we proceeded to tune the different parameters—batch size, epoch, and learning rate—to improve both our training and testing accuracy. Even though we planned to use GridSearchCV to tune the parameters, we ran into some technical difficulties with integrating it with PyTorch's model. Hence, we decided to try different values to figure out which values are the best for the different parameters. We also took a "binary search" approach for the learning rate parameter to find the optimal learning rate for our model.

4.1 Calibrating the Learning Rate

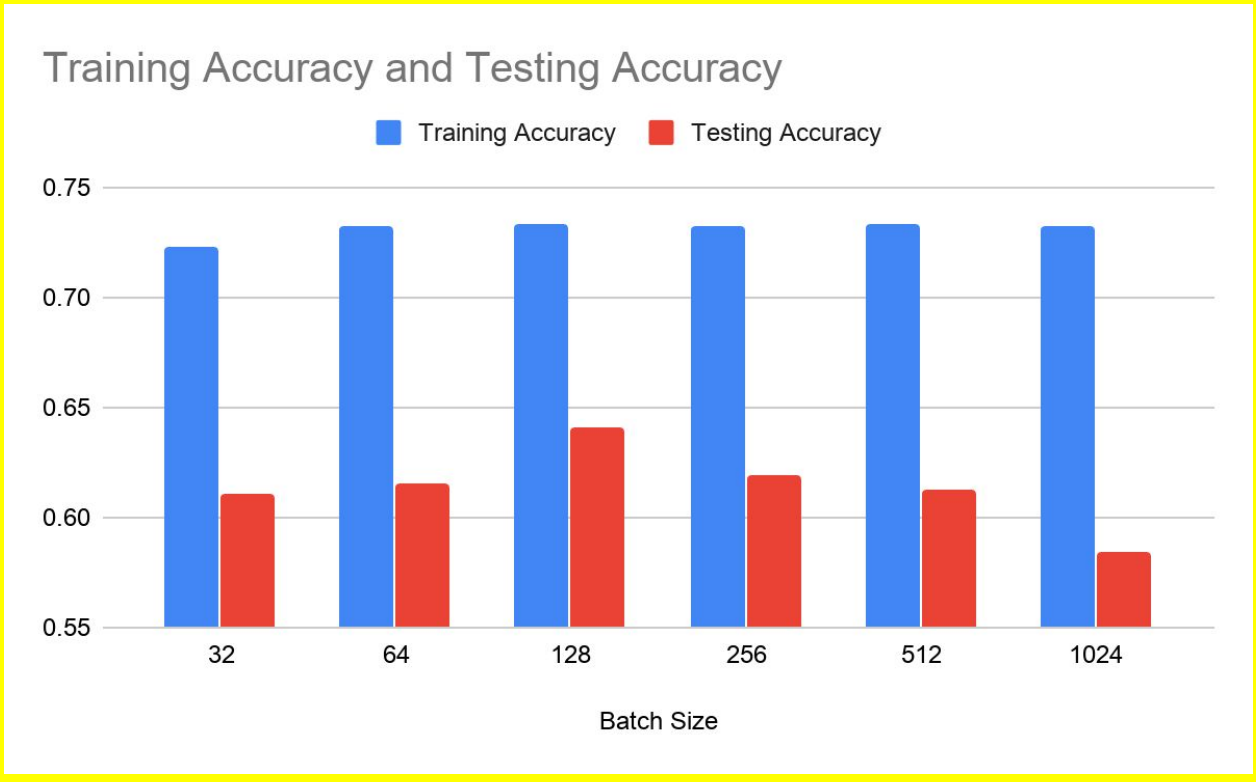
Batch Size	Epoch	Learning Rate	Training Accuracy	Testing Accuracy
128	50	0.1	0.648	0.536
128	50	0.01	0.687	0.567
128	50	0.001	0.732	0.612
128	50	0.0001	0.724	0.585
128	50	0.0005	0.731	0.61
128	50	0.00025	0.732	0.589
128	50	0.00075	0.733	0.641
128	50	0.000875	0.732	0.617
128	50	0.000625	0.731	0.597



We took a binary search approach to find the best learning rate value for our model. Based on our results, it is clear that the learning rate value of 0.00075 gives us the best testing accuracy. It is also noteworthy that the training accuracies are more or less similar between 0.001 and 0.0001. It is probably limited by the values for the other parameters.

4.2 Calibrating the Batch Size

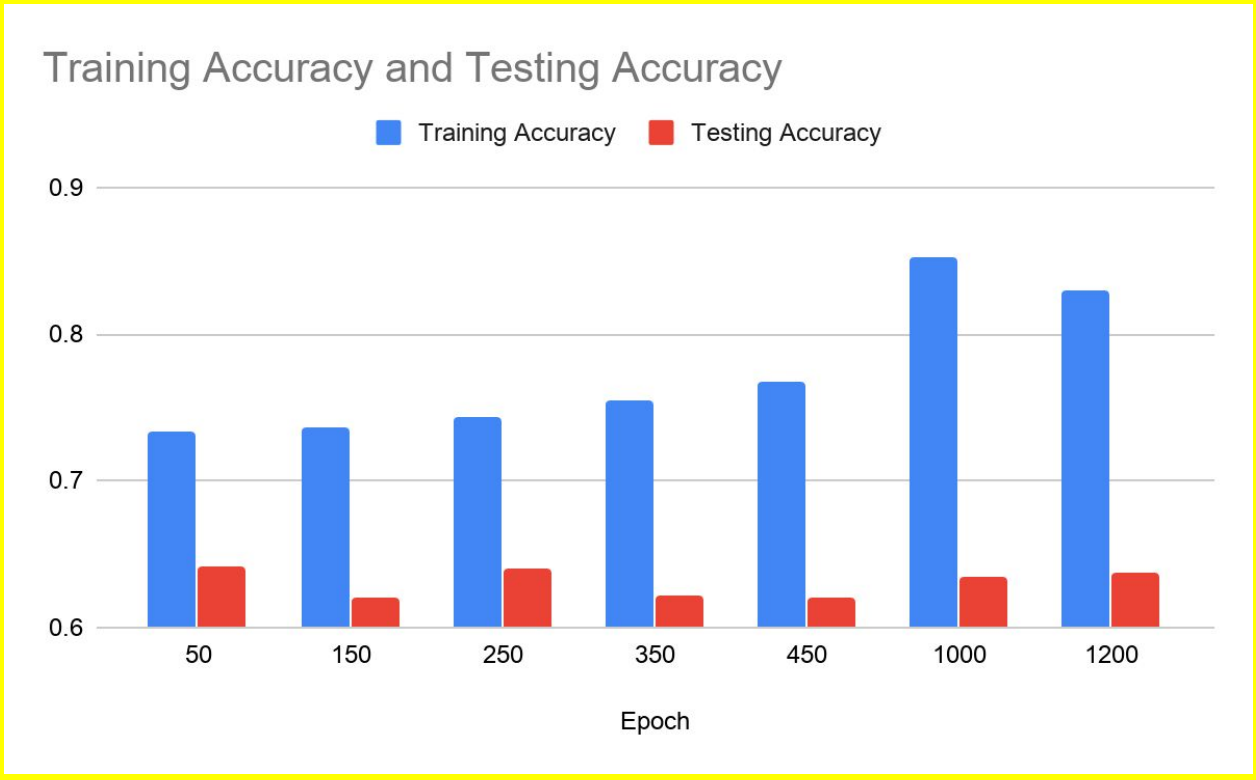
Epoch	Learning Rate	Batch Size	Training Accuracy	Testing Accuracy
50	0.00075	32	0.723	0.611
50	0.00075	64	0.732	0.615
50	0.00075	128	0.733	0.641
50	0.00075	256	0.732	0.619
50	0.00075	512	0.733	0.613
50	0.00075	1024	0.732	0.584



The batch size of the model is limited by the shape of the tensor that we pass into the model. Hence, the smallest and biggest batch sizes are 32 and 1024 respectively. After iterating through the different batch sizes, we found that the batch size value 128 gives us the best testing accuracy. It is clear that a small batch size would result in lack of information that could result in high bias and an overly large batch size would be harder to “learn”.

4.3 Calibrating the Epoch

Batch Size	Learning Rate	Epoch	Training Accuracy	Testing Accuracy
128	0.00075	50	0.733	0.641
128	0.00075	150	0.736	0.621
128	0.00075	250	0.744	0.64
128	0.00075	350	0.755	0.622
128	0.00075	450	0.768	0.621
128	0.00075	1000	0.852	0.634
128	0.00075	1200	0.829	0.637



Our research has informed us that a batch size of 128 and a 2 layer CNN would require about 1000 epoch to get good training and testing accuracies. Hence, we tried out the different epoch values above. Our approach was very much limited by how time-consuming training the model (more than 12 hours) with large epoch values was. It is clear that a larger epoch value equates to a higher training accuracy. However, it is also clear that training for longer periods of time would result in overfitting. Hence, our testing accuracy actually worsened. Based on our trials, we believe that the optimal epoch value would lie somewhere between 50 and 250.

5. Conclusion

After figuring out the optimal parameters, we tested our model and found our best testing accuracy to be approximately 66%. The corresponding training accuracy value for this testing accuracy is 74%.

This result is considerably lower compared to what other papers have published. Other papers have shown testing accuracy that range from 75% to 98% depending on the kinds of accent that they worked with. Because we are dealing with two East Asian accents that are very similar, our testing accuracy would naturally be lower than comparing accents from different continents and cultures. However, our results are still not close enough to what other papers have released. We have reflected on some of the possible reasons and have listed them in the challenges that we faced and future work that we can take on.

6. Challenges

We have taken some detours in the start of this project.

(a) We extracted data by word instead of standard time windows, which led to problems in preprocessing data into equal size chunks.
(b) We also preprocessed data in the way that normal machine learning methods such as svm or knn needed-- 2-d arrays, instead of suitable form for dataloaders that pytorch needs.
(c) The large data size also creates difficulties in terms of team collaborating, since git pull does not work for large files, and we had to spend a lot of time waiting for the results.
(d) We spent a lot of time figuring out how dataloaders work, modifying network.py, and tuning hyperparameters.

We also faced several challenges during the hyperparameter tuning process and the improvement of accuracy of the model.

(a) While training the model, it takes much longer than we thought to tune the data, because none of our computers have GPU. Thus, while tuning the hyperparameters, it takes a very long time to test how each parameter fits the model.
(b) Due to the limitation of technology, after extracting the features using 50 MFCC and normalizing them, which is much more precise, the processed data contains too much information and is too large (more than 10GB in size) to run on all of our laptops. Due to the virus, our accessible computers do not allow us to train the better dataset, which is one reason that limits our model's accuracy.
(c) We also planned to use L2 Regularization to prevent the overfitting problem and apply a customized Loss Function to optimize our training process. However, due to time limitations, we were unable to implement both of these elements that would have helped to improve our testing accuracy.

7. Future Work

(a) If possible, we should have used more extensive data, including conversations with speakers with different native languages. Analyzing if one speaker's accent would affect another is also intriguing and worth researching on.

(b) We did not normalize our data segments. The paper we referred to did so, and the reason why we didn't is that mfcc coefficients are the only feature we use, and I'm not sure how much different it would be after normalization. However, if we are looking at the relative pattern for frequencies instead of absolute frequencies themselves, normalization is worth trying. After failing to get a good enough testing accuracy, we redesigned the data extraction process and tried to work with the normalized 50 MFCC values but we could not overcome the lack of memory problem.

(c) If we have time, we could have tried other neural networks, such as RNN or MC.

(d) We could have included more non native speakers in our training data outside of Chinese and Korean native speakers.

(e) In the future, when we have access to lab computers or just laptops with GPUs, we can try the 50 MFCC feature extraction and thus getting more detailed data information. In this way, I believe we can achieve a much higher accuracy than what we get now.