Final Project Report

Results:

82.14% On All Imported Audio
100% On Original Data Set + Member Voices
87.5% On Original Data Set Notch Filter 1
75% On Original Data Set Notch Filter 2

The program outperformed the human benchmark of 75% accuracy on the original dataset.

Contributions:

Anson:
LBG Algorithm
Code to Compare Test Audio to Codebooks
Generated plots for Tests 2-6
Created Notched Audio Files for Test 8

***Joseph*:**
MFCC and filterbanks
Imported audio files
Created extra dataset
Code to display results

Approaches:

Feature Extraction:

Our first step in speaker recognition is importing the training data into the program. This is done by the function called getAudioFiles.

In order to extract the features from each audio signal, of which there is one per speaker, the program extracts the Mel Frequency Spectrum Coefficients (MFCC) in a process which will be described below.

The first step in the extraction process is to break each audio signal into a number of overlapping frames of the same length. We decided to go with 256 long frames which overlap by 100 samples. After calculating the beginning and ending indices of each frame, each speaker's training data is broken up into frames.

In order to remove the distortions which come with abruptly cutting off the data at the beginning and end of each frame, the program windows each frame using a Hamming window.

Once the distortion is removed, the absolute value of the fast Fourier transform is taken of each frame. The result is the short time Fourier transform of the original audio signal.

The next step is to use Mel-Spectrum filterbank to extract unique coefficients for each speaker. At this point, each speaker has a certain amount of frames and each frame contains 20 coefficients.

The final step in the MFCC is to take the discrete cosine transform of the log of each coefficient. After this, the first coefficient is ignored, leaving each frame with 19 coefficients.

Feature Matching:
Then the LBG algorithm is run, first the code converts the cell array for a given speaker to a matrix, so that it will be easier to deal with using Matlab functions.

Then the code sums up each MFC coefficient across all frames and divides by the number of frames to find the centroid for all the data.

Next, the code splits the centroid in two by creating one new one with each coordinate of the original centroid multiplied by 1+epsilon, and the other new one having each coordinate of the original centroid multiplied by 1-epsilon.

The next step is to find the distance between each frame and each centroid in MFCC space. The centroid with the minimum distance to a given frame is then stored in the variable ‘I’, and the frame is assigned to that centroid in a new variable called centroid_frames.

Then, the average of each mel coefficient across the frames assigned to a given centroid is computed. This is assigned as the new location of that centroid. This is done for all the centroids in the codebook.

Then an error term is calculated to determine if the frames should be reassigned and the centroids recomputed again. If the new error is less than a certain percentage different from the previous error, the code will exit the loop, and it will split the centroids again using the epsilon parameter above. This seemed to be the best way to calculate the error and determine if the loop should run again, since an absolute error threshold would not work because error will decrease as centroids are added. After splitting centroids again, the code will reassign frames, recompute the centroids and continue this loop until the centroid limit is reached.

The code will then loop through all speakers in the training audio to create a set of centroids for each one.

Next, the code to compute the MFCC is run on the test audio.

Finally, the code compares the test audio to the codebooks for each speaker in the training audio. For example, for speaker 1 of the test audio, the code does this by calculating the distance in MFCC space between each frame of the test audio and each centroid of the codebook for speaker 1 in the training audio. Then it finds the nearest centroid to each frame, and sums up that distance for each frame. This represents the total VQ 'distortion' for test audio 1 compared to training audio 1. The code then calculates this distortion for the codebooks of each of the rest of the speakers in the training audio, it finds the codebook with the minimum distortion, and decides that the closest match to the test speaker is the training speaker that generated that codebook.

Imported Dataset:
We imported audio from a training set in order to test the robustness of our speaker recognition. This dataset included speeches from various political leaders. We decided to use some of the samples from Nelson Mandela, Margaret Thatcher, and Jens Stoltenberg.

Inside of the getAudioFiles function, we included "subfunctions" to import extra data from the other speakers. We also allowed for a way to select whether to use the notched audio files or the unaltered audio files for the original dataset.

Due to many of the audio files in the dataset not being useful, such as moments of pause or applause, we hand selected 6 files per speaker to be used as a test set. Also, one file per speaker was selected to be used as training data.

Displaying Results:
Our program also automatically calculated the accuracy of the speaker recognition. This is mostly hardcoded, since the audio files are using three different naming schemes.

Tests:

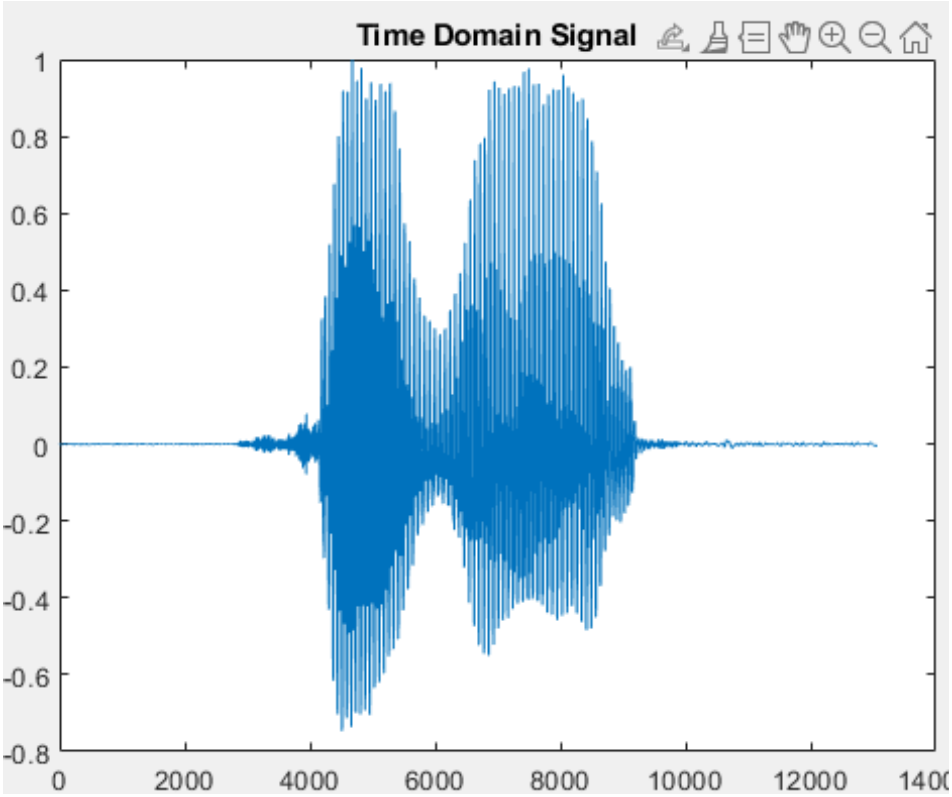
Test 1:

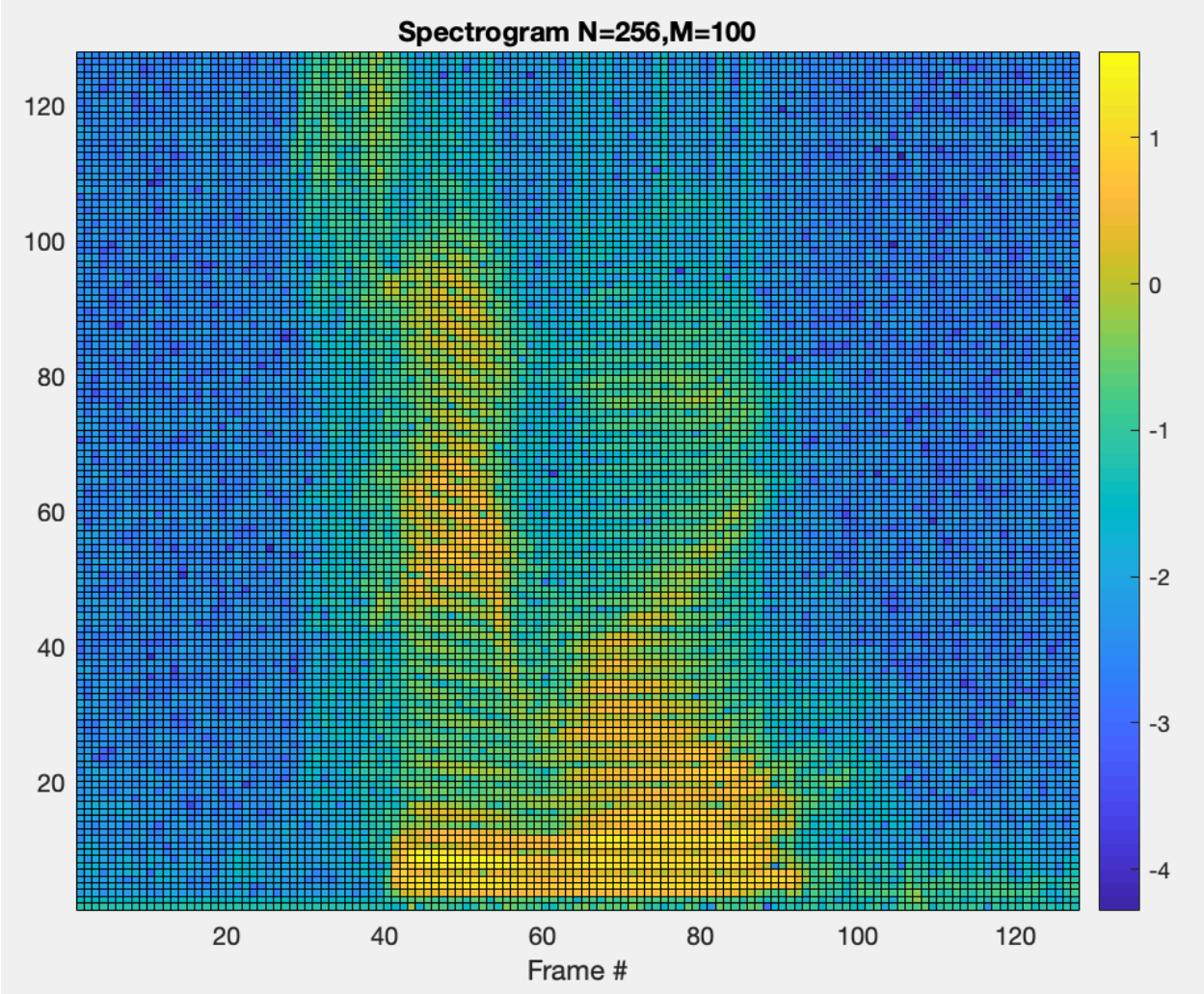
Human accuracy = 75% on original data.

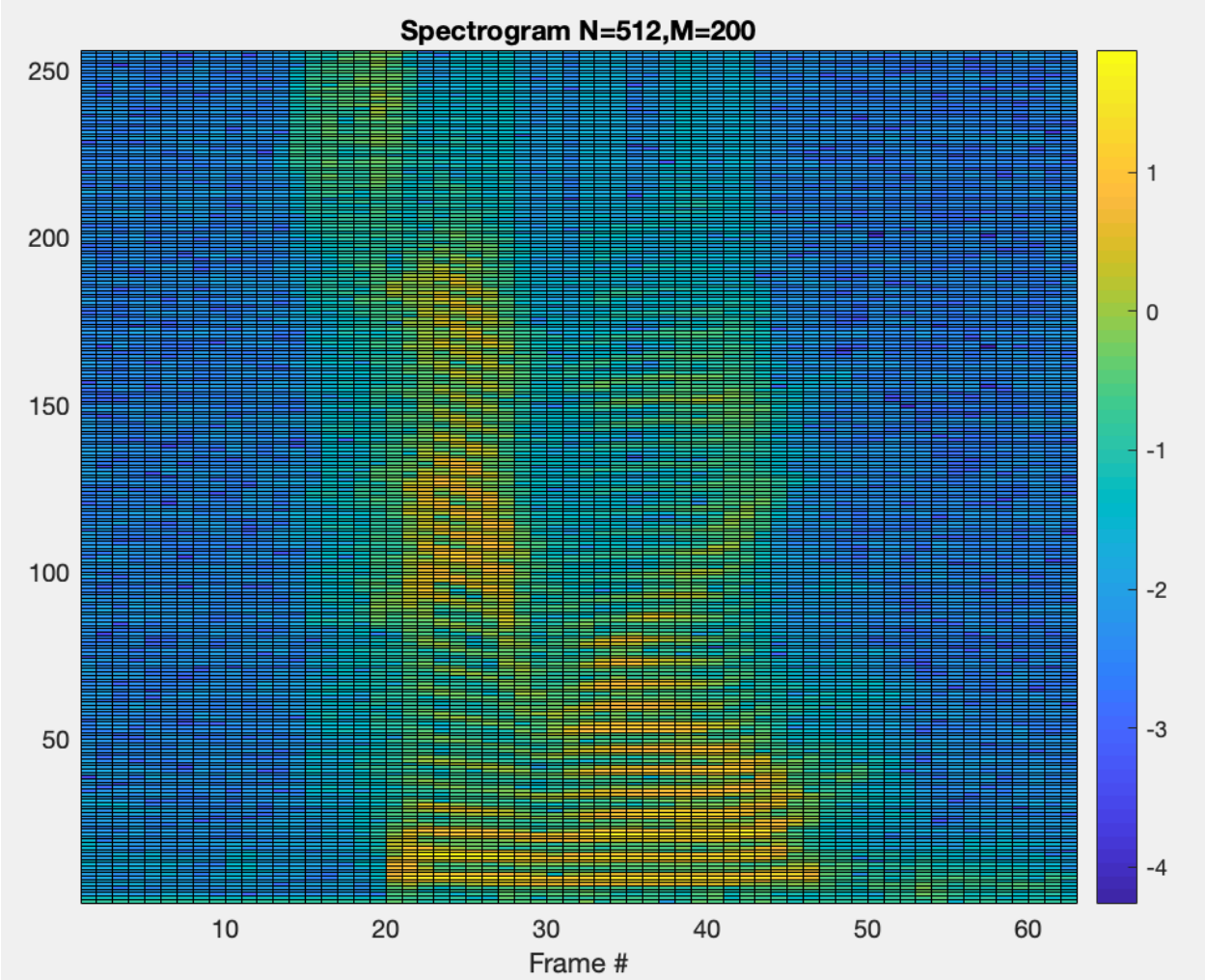
Test 2:

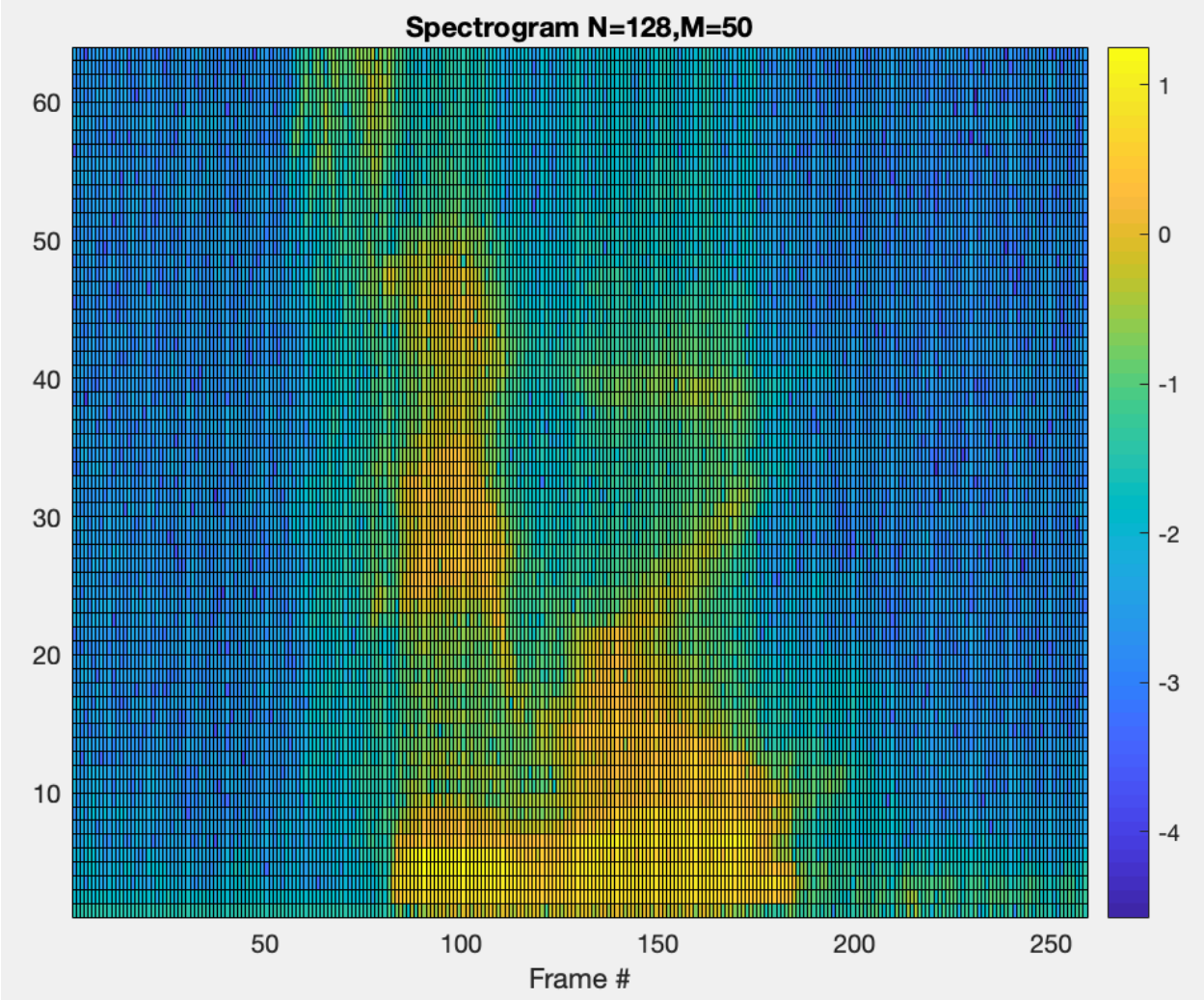
Sampling rate = 12.5 KHz

Time = 20.48 ms per frame



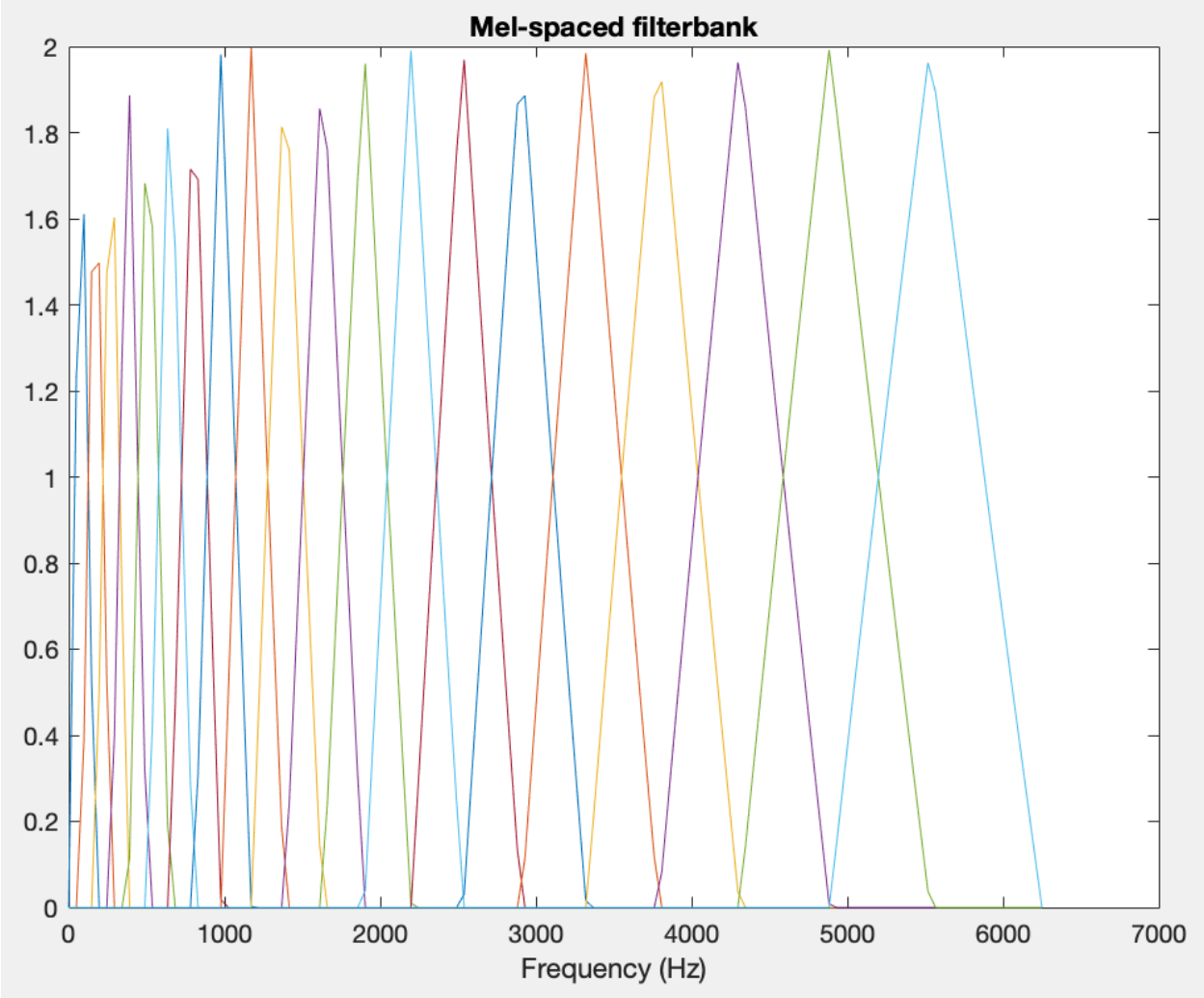

Spectrogram N=256,M=100
120
100
80
60
40
20
20
40
60
80
100
120
Frame #
1
0
-1
-2
-3
-4

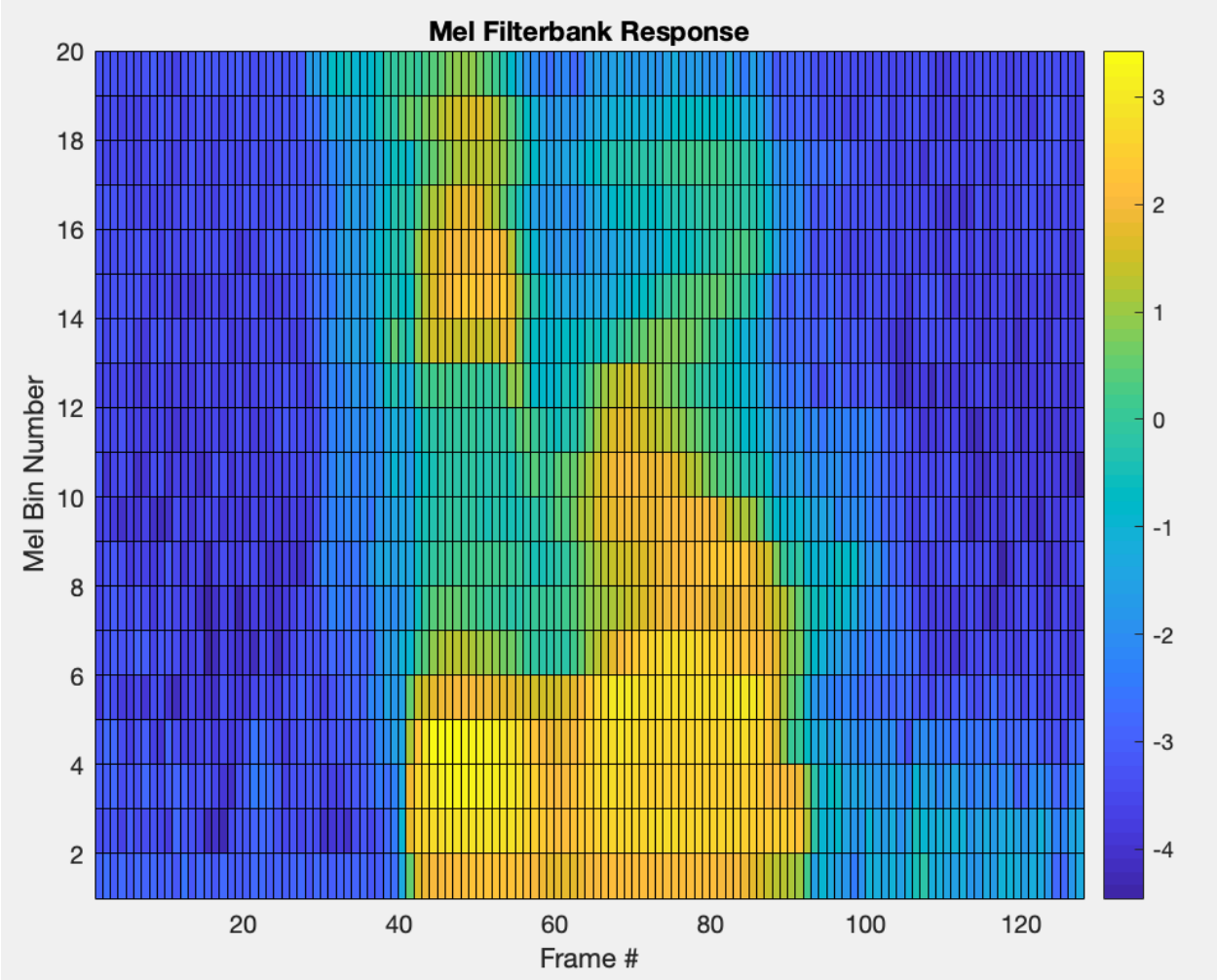

Spectrogram N=512,M=200
250
200
150
100
50
10
20
30
40
50
60
Frame #
1
0
-1
-2
-3
-4


Spectrogram N=128,M=50
60
50
40
30
20
10
50
100
150
200
250
Frame #
1
0
-1
-2
-3
-4

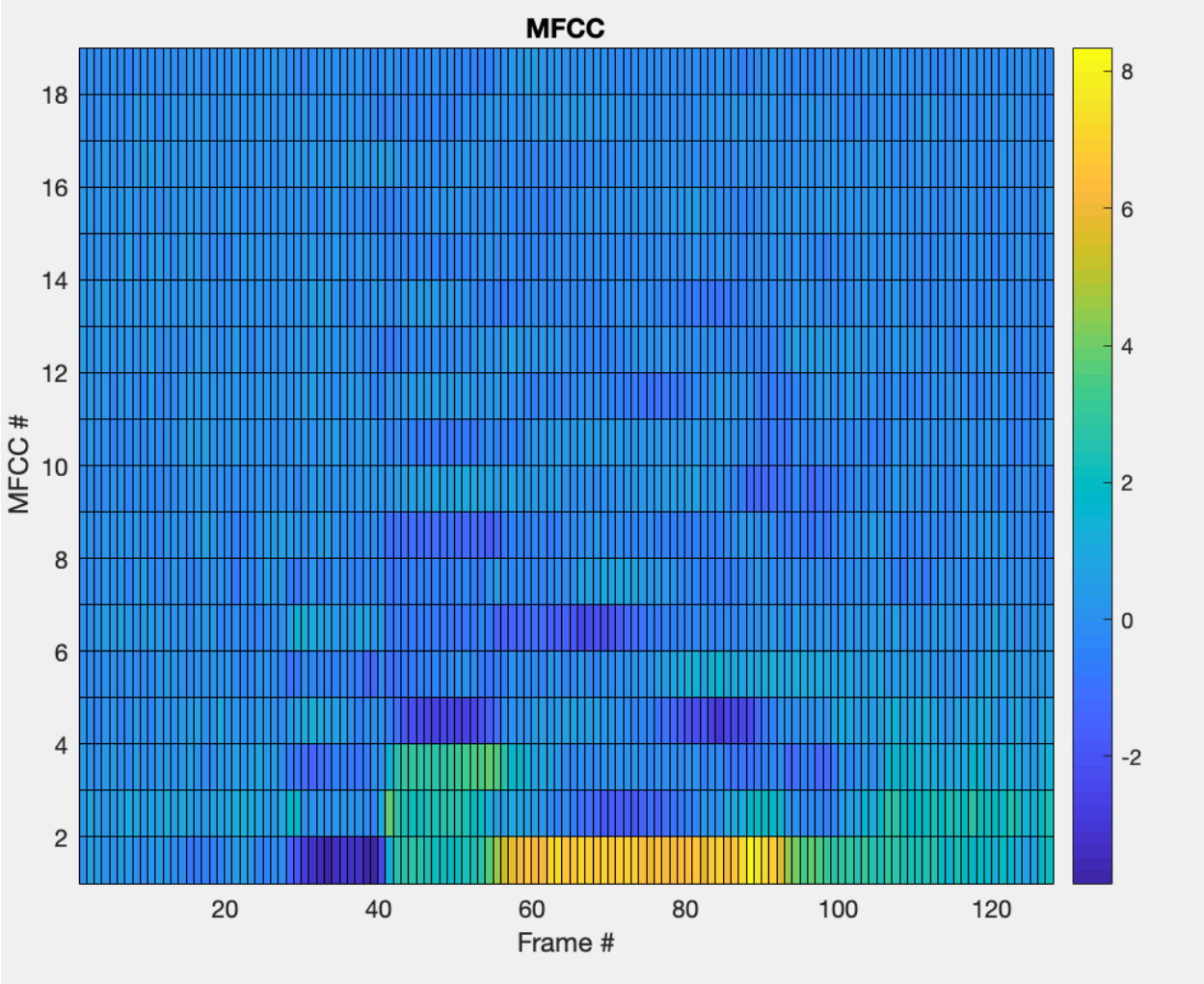
Test 3:

Mel filterbank plot looks somewhat jagged due to the discretization error of having only N = 256 points. They are not all perfectly triangular and they don't peak at the same amplitude, with more points they would look more smooth.



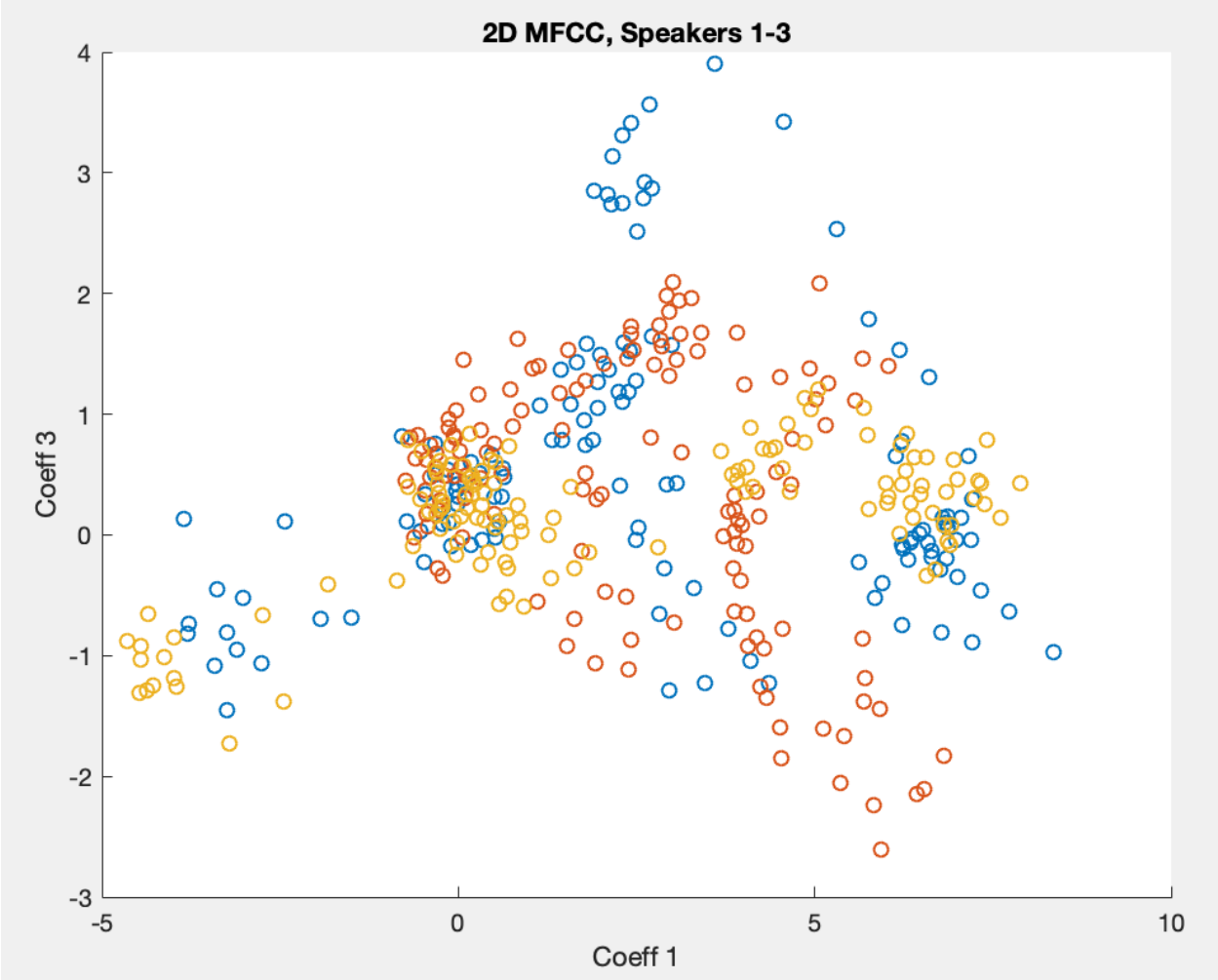

Mel Filterbank Response
Mel Bin Number
Frame #
20
18
16
14
12
10
8
6
4
2
20
40
60
80
100
120
3
2
1
0
-1
-2
-3
-4

Test 4:



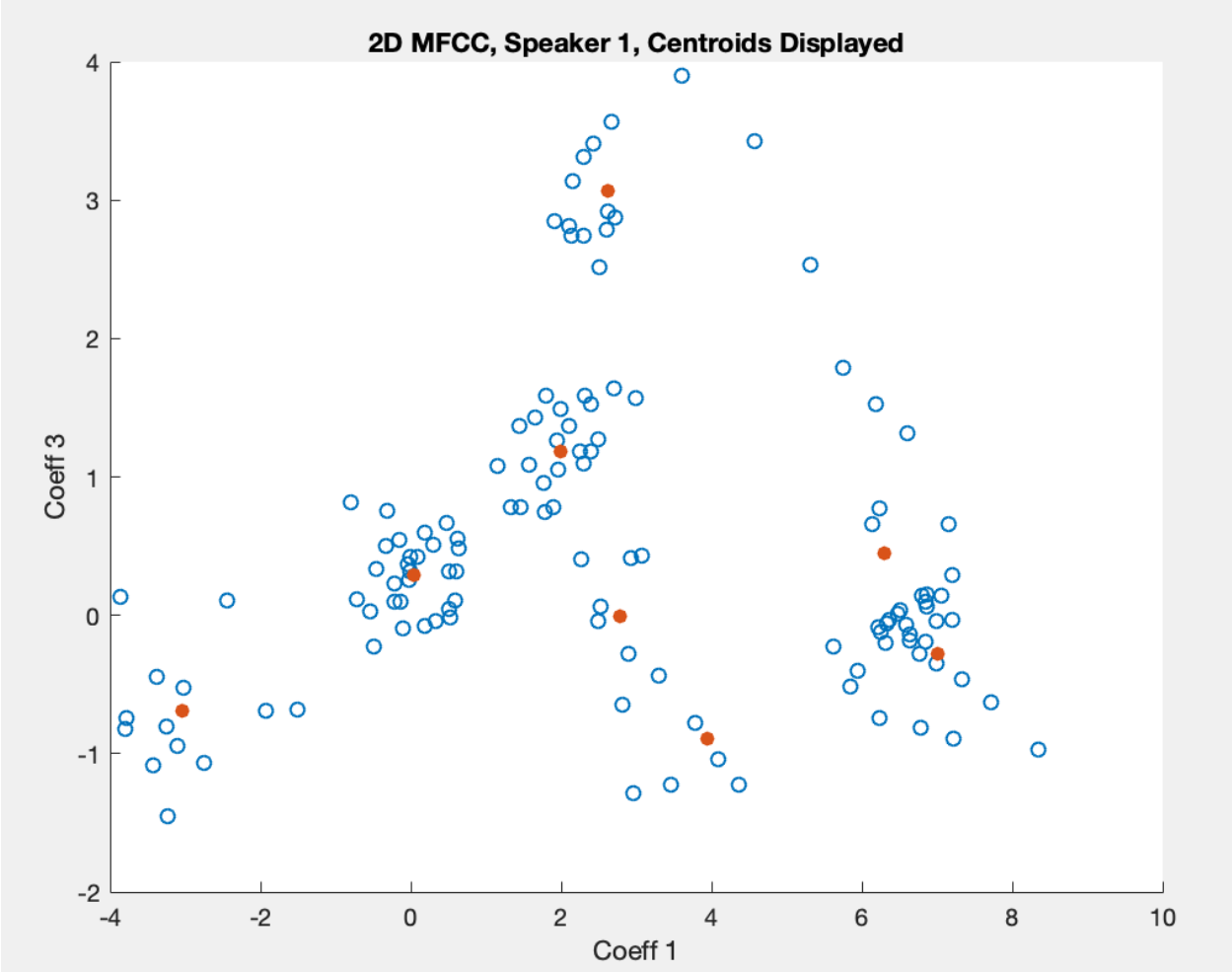
Test 5:

It can be seen that although there is some overlap, each speaker's frames tend to form different clusters in MFCC space.



Test 6:

There are 8 centroids in this image, as can be seen, they tend to be at the center of clusters of locations of frames as expected, even though this is only plotted in 2 of the 20 MFCC space dimensions.



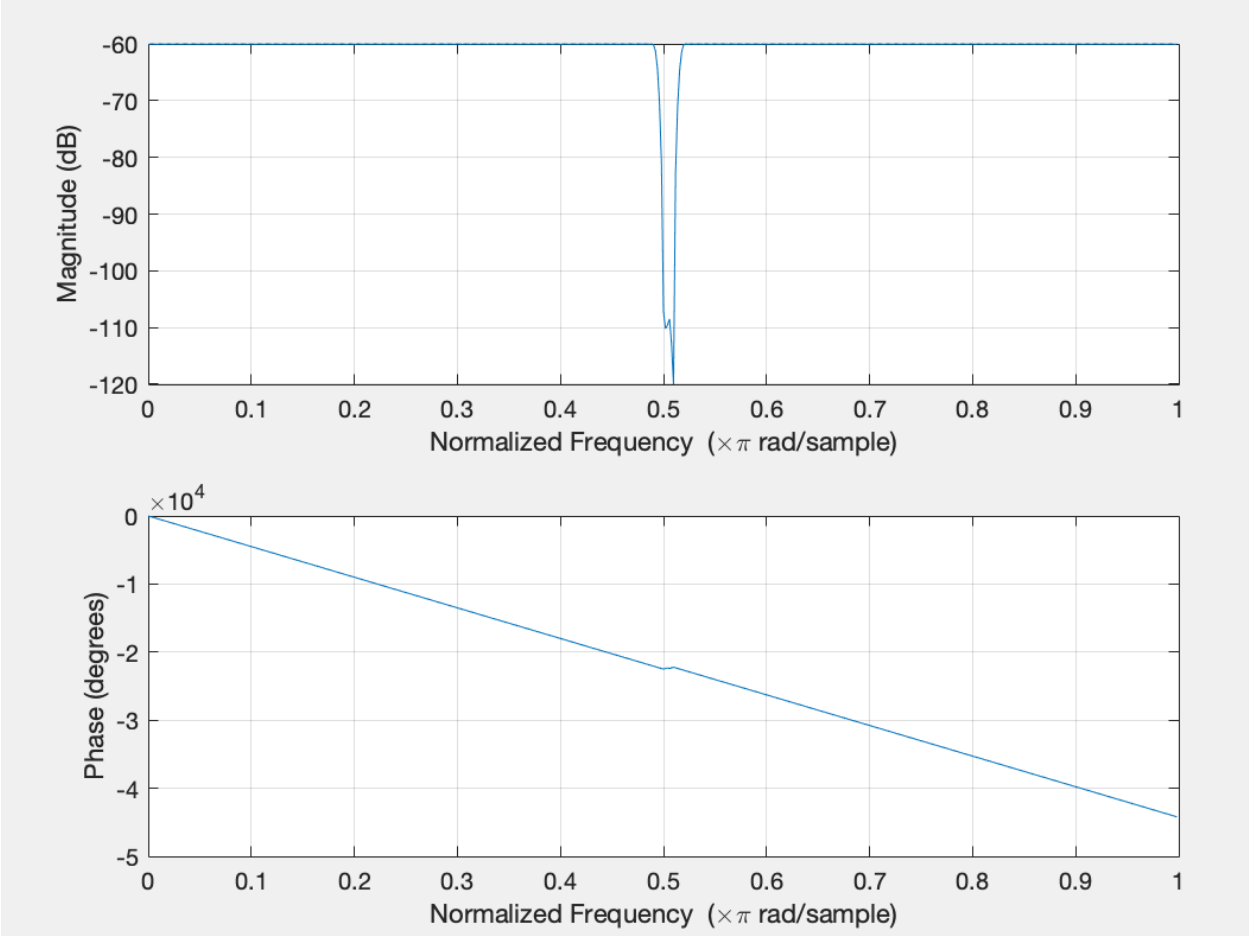
Test 7:

With parameters, 256,100,20,.001,.01, and 8 for N,M,P, thresh, epsilon and codeword_lim respectively, code gets 100% correct on original test audio, much better than human 75% correct

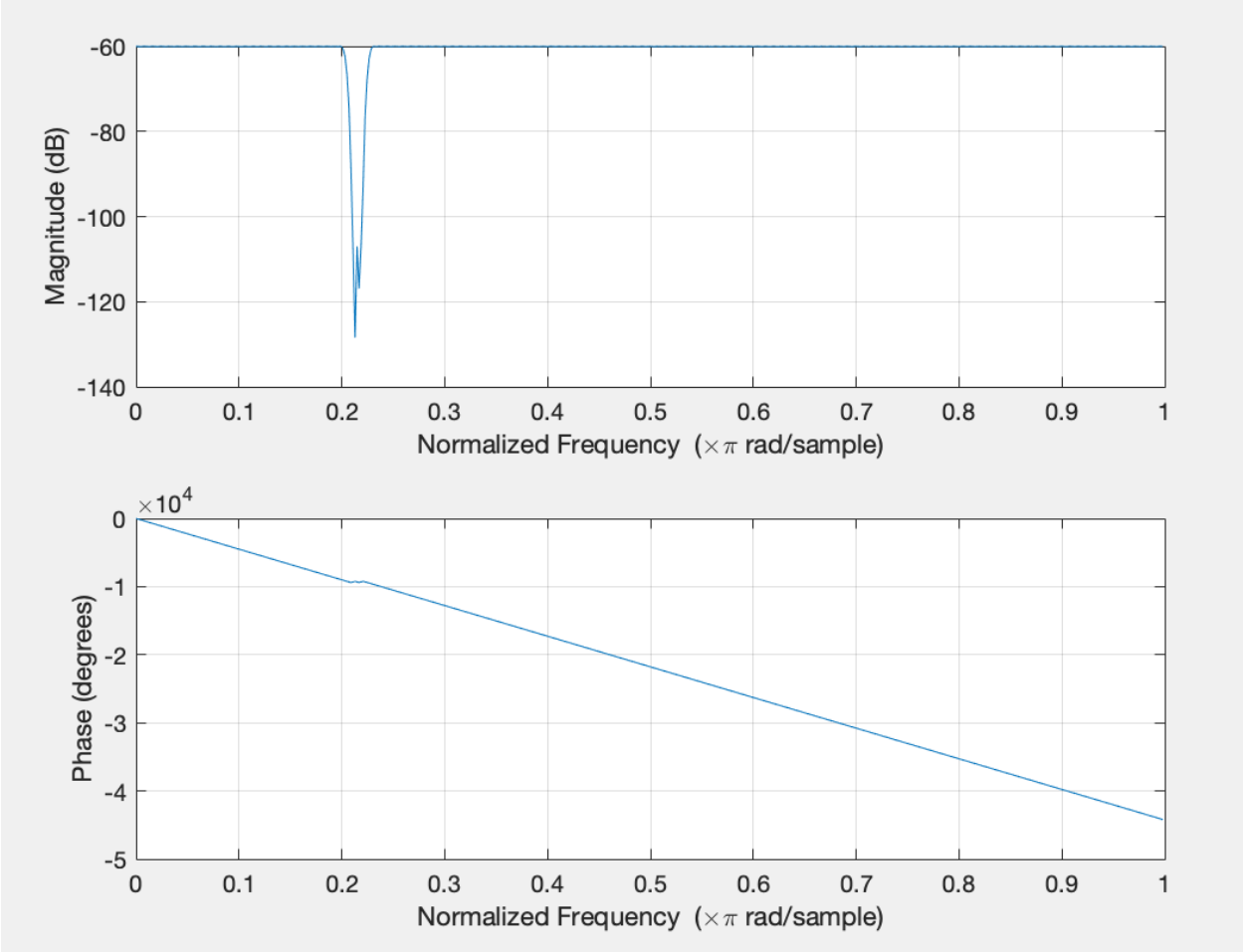
Test 8:

Reduction in accuracy due to notch filtering is highly dependent on which frequency the notch filters out, the lower frequencies where most of the speech energy is have a much larger effect. With notch at .5*pi, accuracy is reduced to 87.5% on original audio.

Notch Filter 1: 87.5% Accuracy



Notch Filter 2: 75% Accuracy



Test 9:
100% accuracy on group member's voices

Test 10:
82.14% overall accuracy on 28 test samples, and 16 different training voices