Project1 report template

(2015 Fall)

Course name：J1799d Instructor：Ming Li TA：Wenbo Liu

ID number	15213796/ 15213782	Name	WANG LUTING LI DAJIAN
Tel	15622777988/ 1357030 0549	Email	364706267@qq.com 394549293@qq.com
Starting date	2015.9.6	Finished date	2015 9.16

1、 Project requirement

The project is composed of two parts. First part is data capture and the second part is Mel frequency computation. For data capture, we are supposed to capture the speech. The signals must have 16-bits resolution, mono channel and 16000 sample rate. The recording should begin at a hit-to-talk and stop automatically if detected as the end of speech. At last, the captured audio should be saved as wav format.

And for the Mel frequency computation, we should use the sample we just capture to derive their features. We visualize the spectrogram through Mel cepstrum and log Mel spectrum. After that, we should change the number of filter banks to compare the differences. More ever, the figures of log Mel energy before DCT and after IDCT should be similar.

2、 Design your program

For this project, our group use different languages to implement it. Luting Wang used C++ and Dajian Li used Python. Both of us have got reasonable answers. In this section, I will talked about the program design briefly. Although

we use different languages, the idea and process flow are similar. So I will talk about it mainly based on C++. In the following sections, we will discuss the implementation separately.

For the first part, I use PortAudio for the audio capture. I added the package to my project and it can automatically invoke callback function to capture the sound. Because the sample rate is 16000Hz, and I set the value of frame per buffer 512, so every time the PortAudio capture 512 samples, it would invoke its callback function, and we can implement the endpoint detection in the callback function. The main idea of endpoint implementation will be discussed in the next Section.

After I have captured the speech signal, I would like to calculate its features. Before that, the speech signal should be pre-emphasis. Because most of the speech signals' energy centers on low frequency. It may lead to intolerable low SNR in high frequency. So I apply high-pass filters to the signal to aggravate its high frequency weight. And later through de-emphasis we can recover its power spectrum with noise reduced. Now we get a pre-emphasis signal. In order to do the signal processing more conveniently, I would like to separate the entire signal to small segments. It is called frame blocking. We already know speech signal has Short Time Stationary characteristics (20-30ms), take the value of 25ms, the window length is 400 samples. And adjacent segments typically overlap half of each segment, so the window overlap is 200 samples. Then Hamming window is introduced to each segment to reduce Gibbs phenomenon. Then I add zeros to the end of signal to make it a desired length because the FFT

requires signals of a specified length. After that, FFT is applied to the windowed signal to change the time domain to frequency domain.

As for human beings, we have non-uniform resolution to different speech frequency. That's to say, we can detect small changes in low frequency but larger changes in higher frequency. Because information in speech signal is distributed in the manner matched to human perception, we should warp the frequency axis to fix it. That's to say, we give more importance to lower frequency and less importance to higher frequency. The standard warping transformation is Mel function. We use the same triangle filter banks in Mel frequency domain and transform the size of these filter banks in our actual frequency domain. In our project, 40 filter banks are used. For each filter banks, the spectrum values are added together and finally we get 40 values, called Mel spectrum. To reduce imbalance, we apply logarithm operation to compress the Mel energy valuesthem. And we also do Discrete Cosine Transform (DCT) to reduce signal correlation and choose the first thirteen coefficients to present the features. They are Mel Cepstrum. We also do IDCT to compare the log Mel energy after IDCT with log Mel energy before DCT to test our program's validity. The total process can be shown in Figure 1[1].

[1] Class 3: Feature extraction, Design and Implementation of Speech Recognition Systems, J1799d.

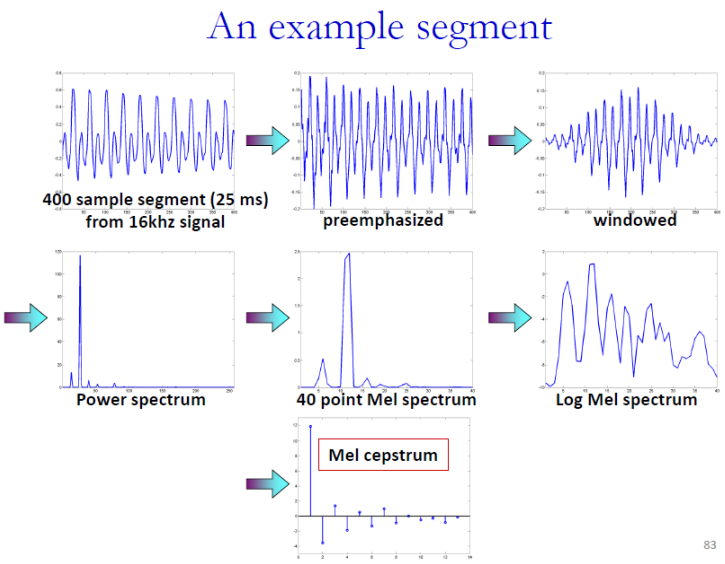

Figure 1 The Flow Diagram

3、 Program implementation and testing

In this section, we will give more details about program implementation.

For C++ implementation:

A. Endpoint detection

The algorithm I used is an adaptive endpoint algorithm. I first used the average energy of first five buffers (150ms) to set the background. And the background is adaptive during the audio recording. And I used one threshold to test the end of speech. If the average of one buffer minus background is below the threshold, the counter will be plus one. And if the value is above the threshold, the counter will be set to zero. If the counter is up to 10 (about 320ms), it will be regarded as the end of speech and stop recording. The implementation is shown below in main.cpp:

```
current = EnergyPerFrameInDecibel(in, framesperbuffer);
data->level = ((data->level * 1) + current) / (1 + 1);
if (current < data->background) data->background = current;
```

```
else data->background += (current - data->background)*0.05;
if (data->level < data->background) data->level = data->background;
if (data->level - data->background < 20) data->nospeak++;
else data->nospeak = 0;
```

B. Pre-emphasis

I used Difference Function to describe high-pass filter bank. Because after Z transformation, the frequency spectrum is like a high-pass filter. The algorithm implementation is shown here:

```
void preEmphasize(float* preSample, short *sample, int numSamples, int factor)
{
    preSample[0] = sample[0];
    for (int i = 1; i < numSamples; i++)
    {
    preSample[i] = sample[i] - factor*sample[i - 1];
    }
}
```

At first, there exists a bug. Because preSample[0] is not assigned a value at first and its default value is very big. And after pre-emphasis, the first sample will have a very high value which would affect the results.

C. Windowing and Zero Padding

I used Hamming Window with length of 400 samples, and add 112 zeros at the end of the signal. The process of adding window is shown below:

```
for (i = 0; i < numFrame && (i+n*notOverLap)<numSamples; i++)
{
        windowSample[i] = preSample[i + n*notOverLap] * (hamingwindow[i]);
}
for(int k = i; k < 512; k++)
{
        windowSample[k] = 0;
}
```

D. FFT

Here I use fftw3 library to realize this transformation.

```
void FFT(float* in, float* energySpectrum)
{
    fftwf_complex *out = (fftwf_complex *)fftwf_malloc(sizeof(fftwf_complex) * 512);
    fftwf_plan p = fftwf_plan_dft_r2c_1d(512, in, out, FFTW_ESTIMATE);
    fftwf_execute(p);
    for (int i = 0; i < 257; i++)
    {
        energySpectrum[i] = out[i][0] * out[i][0] + out[i][1] * out[i][1];
    }
    fftwf_destroy_plan(p);
    fftwf_free(out);
}
```

E. Mel energy spectrum calculation

I use 40 triangle filter banks with the same size in Mel frequency domain and transform the size of these filter banks to the speech frequency domain through the reverse Mel equation. It can be shown in Figure 2.

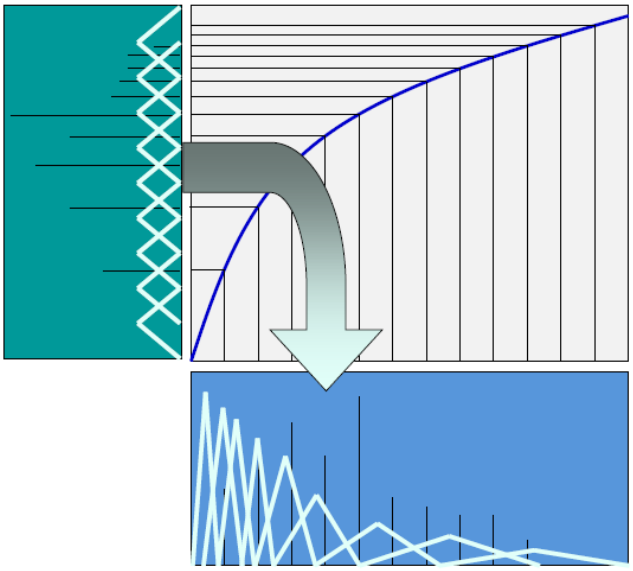

Figure 2

And the detailed implementation can be referred to the following function in mfcc.cpp:

```
void computeMel(float* mel, int sampleRate, float* energySpectrum);
```

For each filter banks, the spectrum values are added together and we get 40 values. It represents the feature of the speech based on human perception. I also try

to use 30 and 20 filter banks to compare their results, and find the more filter banks we have, the clearer spectrum I can get.

F. Testing

At last, I apply logarithm operation to the Mel spectrum and use DCT to reduce dimension, and apply IDCT to compare the Mel spectrum with the former one to test the correctness. The comparison results will be shown in Sec4.

For Python implementation:

In the program we present, we use Python as our programing language. For Python, the audio I/O library is PyAudio 0.2.8 for Python 2.7 (http://people.csail.mit.edu/hubert/pyaudio/), which can be found in the attach.

In our program, we use a callback function to detect and record an audio. The audio format is linear 16-bit PCM, single channel with a sample rate of 16kHz. After recording, the audio is saved to a wav file.

We use a same strategy as provided in the slide. Users hit a key to start recording, and we detect the endpoint to stop recording. We use an adaptive background noise level algorithm, which takes the starting 0.25s to 0.5s for background noise level initialization, to adapt to various application environment. The reason we don't use the 0 to 0.25s interval is in this interval the audio is not stable and contains more noise. Because speech always starts acutely and ends smoothly, we also use a two-threshold algorithm to detect starting point and endpoint separately. The implementation is very similar to the code given in the slide but we make several changes to advance its performance. Here is what we do:

i. 1 To avoid short, sudden noise triggering the speech switch, we have a counter to record exactly how many continuous chunks have the energy above the threshold. Only if the counter value is greater than another threshold that the speech switch will turn on.

ii. To avoid the cache getting too long, we flush the cache every 3 seconds, this could cause some problem, because we want a silence time at the beginning 0.25s. However this would not always be the case because the probability is small and we will always have enough cache to trace back.

iii. To avoid recognizing endpoint arbitrarily, we also have a counter to count how many continuous chunks have the energy below the threshold. Only if the counter value is greater than a threshold that the record process is terminated.

iv. To make start point and end point more accurate, we use a trim() function after we get the full audio. The trim() function will trace back from the end and move forward from the start in order to find the perfect cut of the audio.

A. Feature Extraction – MFCC feature

4、 Experimental results and discussion

For C++:

Now I will show you my experimental results. I speak 1, 2, 3, 4, 5 and make a recording. Figure 3 shows the waveform of my speech. As you can see, at the beginning there is some background noise. The noise is a bit louder than the end

background noise. I think it is the fault of my computer hardware. But I am not quite sure.

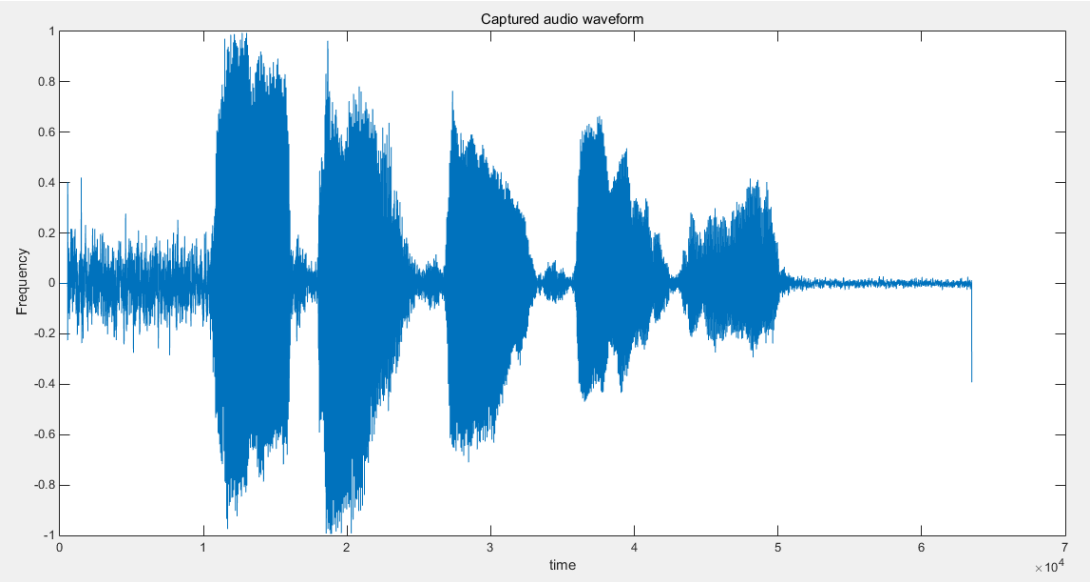

Figure 3

The power spectrum of my speech is shown in Figure 4. As you can see, the energy is centered on low frequency and different digits have different energy spectrum.

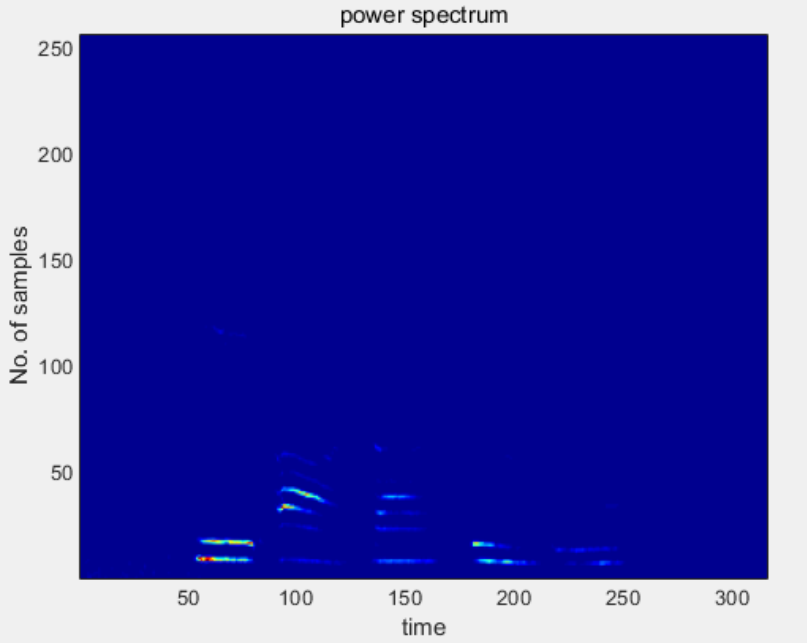

Figure 4

The log Mel energy spectrum is shown in Figure 5. These features are based on human perception and can be applied to speech recognition later.

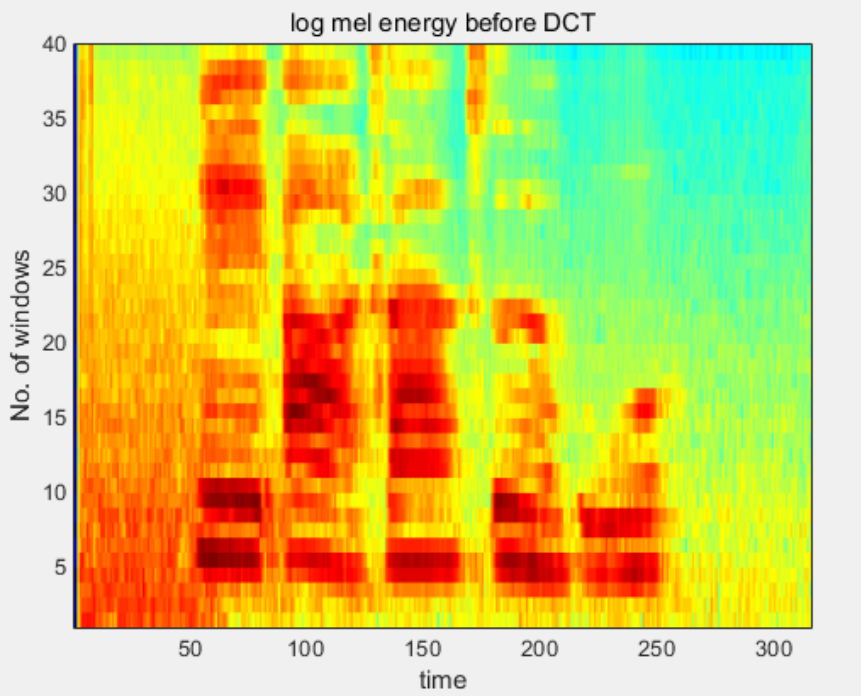

Figure 5

The log Mel energy spectrum after DCT is shown in Figure 6. It is similar to Figure 5, but is more blurry. It is because we truncate the Mel cepstrum (13 points) and add zeros to 64 points to do IDCT. Therefore some information is lost and the figure seems blurry. And if I add zeros to 128 points, it would be more blurry.

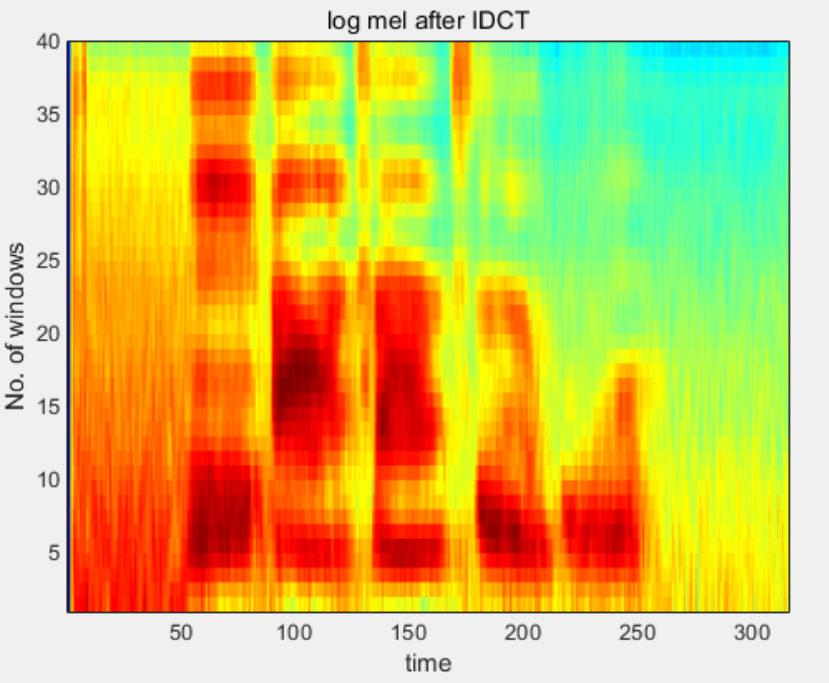

Figure 6

At last I will show you the Mel cepstrum in Figure 7. Because it is

dimensionality-reduced, it is not visually meaningful. But you can still see the difference of these digits.

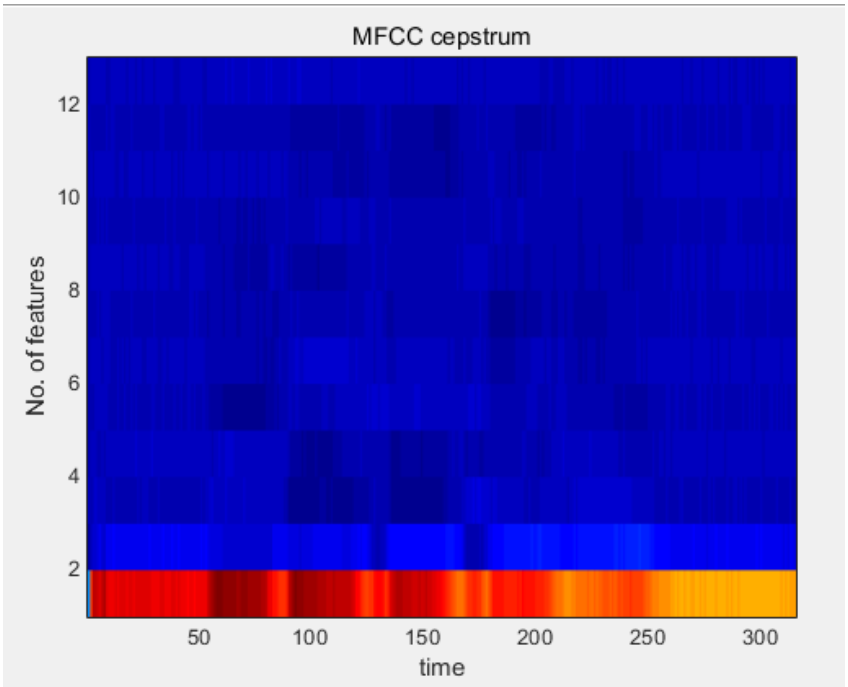

Figure 7

At the end of the report, I would like to discuss some advantages and drawbacks of my work.

First of all, the C++ program I implement can have a good endpoint detection. It can end automatically when people stop talking. The wav file can be played very naturally to my voice. As for the Mel cepstrum calculation, from the figures I show above, you can see the results are reasonable.

However, as I mentioned before, there are some louder noise at the beginning. Although it should not be blame on my program implementation, but later I will do some trim at the beginning to make the captured audio smoother. And I would like to smooth the labels, for example, minimum silence and speech segment length limits may be imposed. More ever, I want to try two threshold, one for onset and one for offset to see whether two threshold is better than one. And for the Mel cepstrum

calculation, my filter banks cover the range from 0 to half of the sample rate (8000). The range should be shortened, for example, from 133.33Hz to 6855.4976Hz. Then we can eliminate some unnecessary information and achieve more accurate and meaningful features.

For Python:

Here is my waveform (Figure 8), power spectrum (Figure 9), Log Mel energy before DCT (Figure 10 left) and after IDCT (Figure 10 right), MFCC cepstrum (Figure 11) of a sample when I said 'This is a test'.

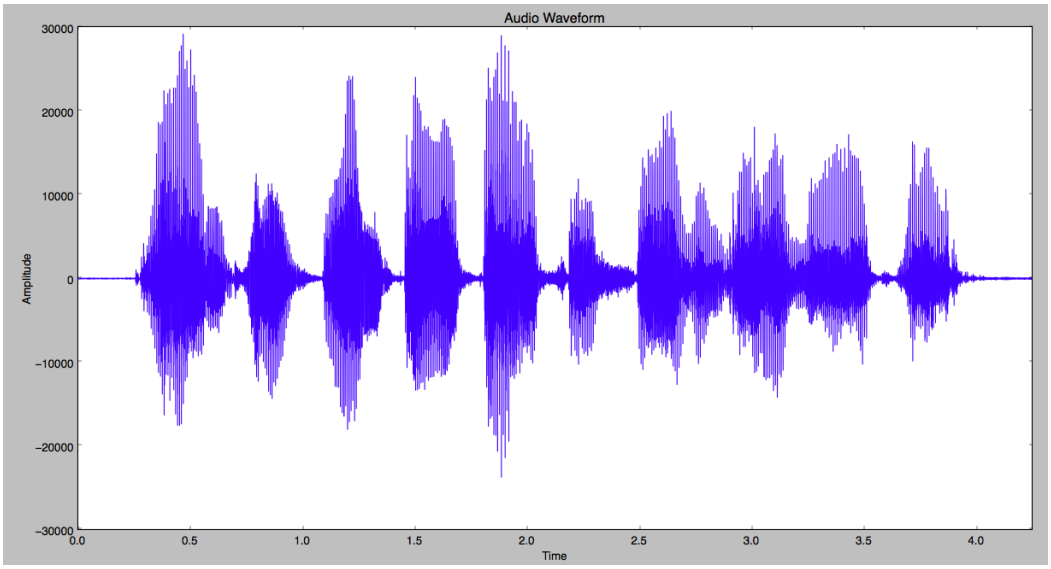

Figure 8

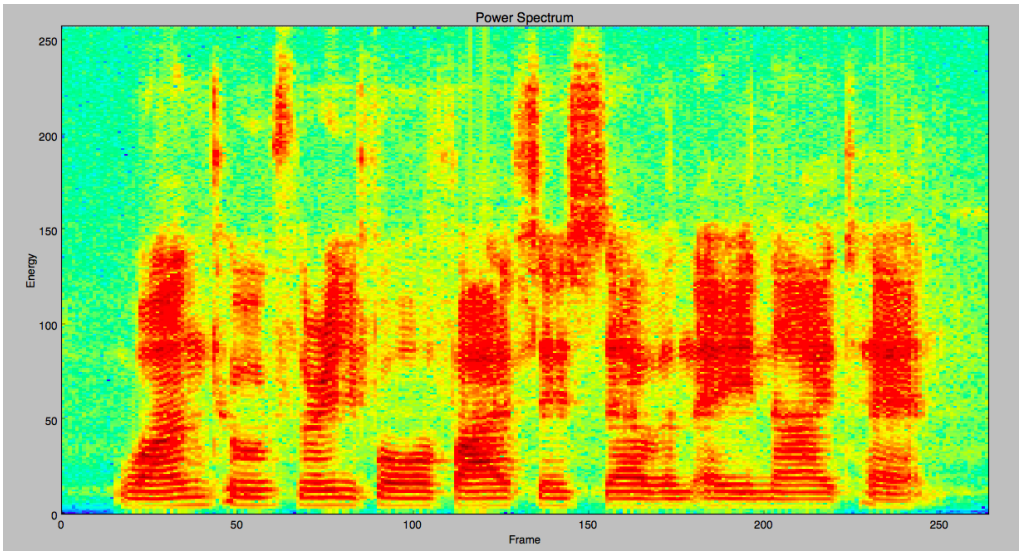

Figure 9

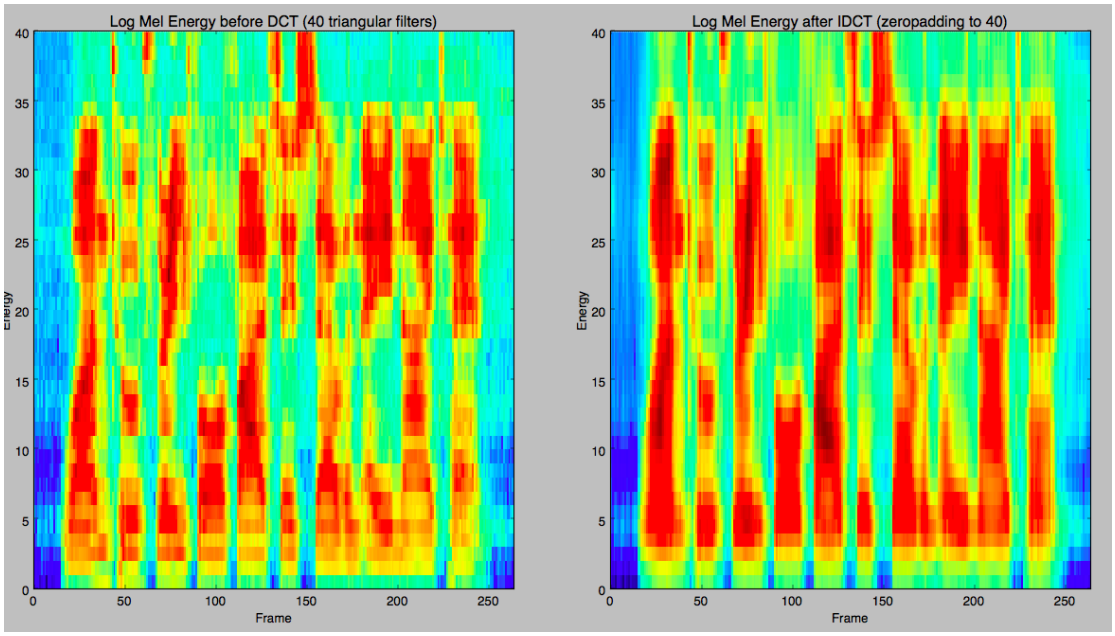

Figure 10

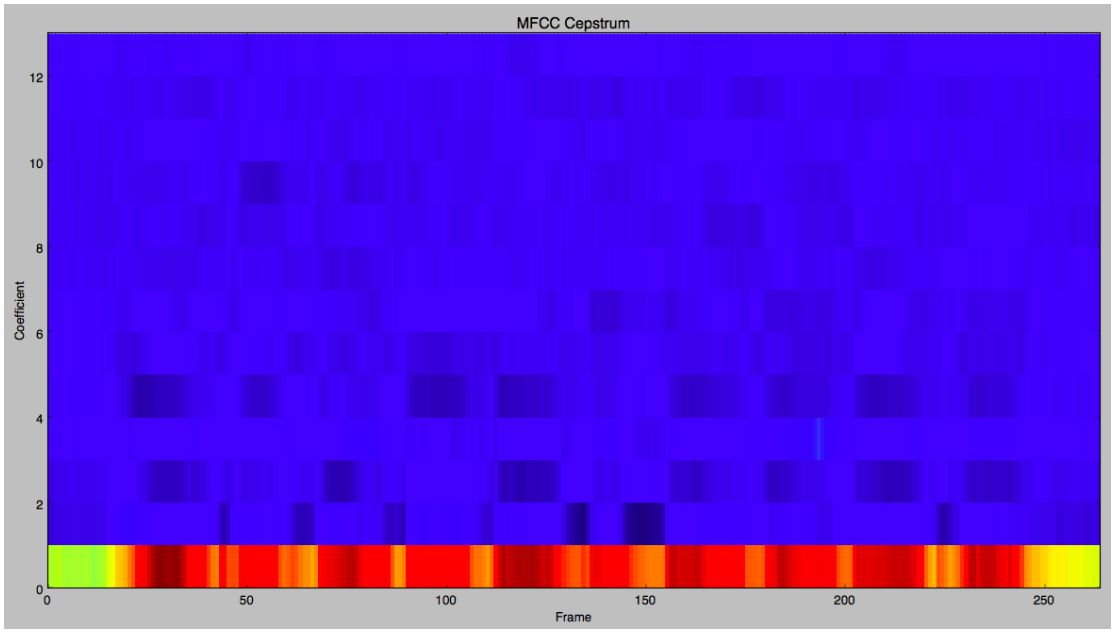

Figure 11