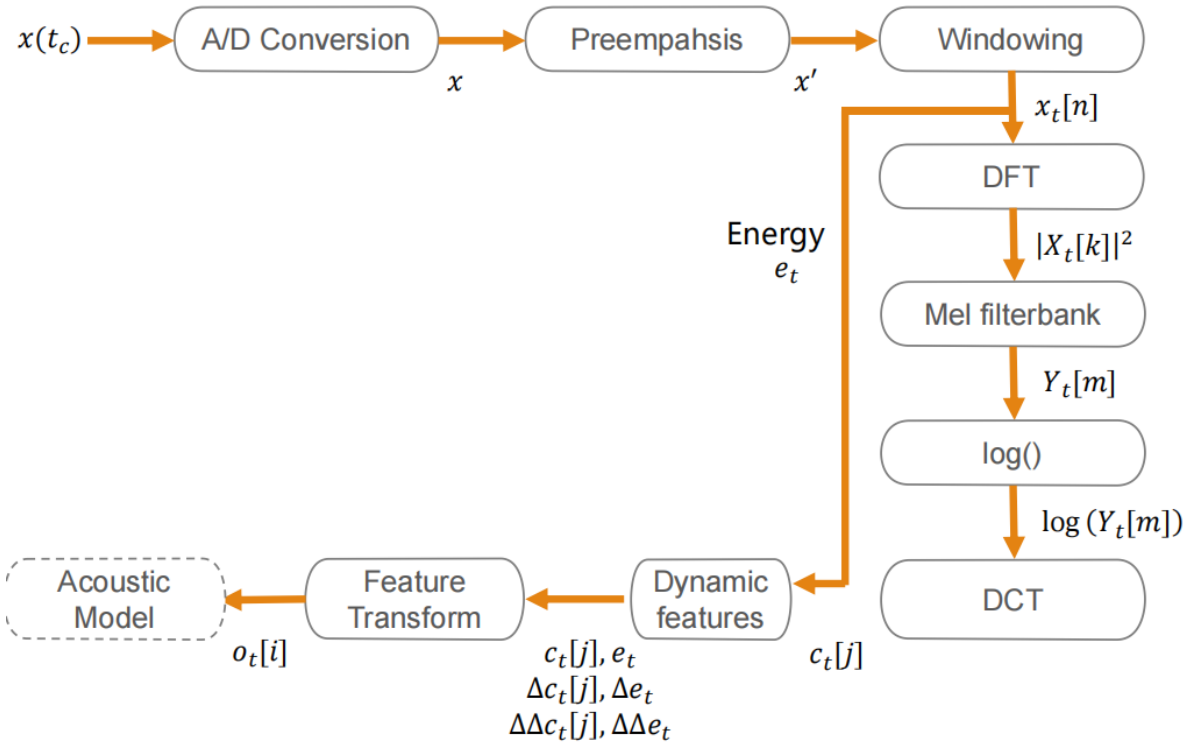
Extract acoustic features (MFCC)

ASR has some standard feartures including FBANK, MFCCs and PLP analysis. MFCC stands for Mel-Frequency Cepstral Coefficients. MFCC can describe the spectral characteristics of sound. The calculation process of MFCC typically involves the following steps:



Pre-emphasis

Pre-emphasis involves adjusting the original audio signal by applying a filtering operation to reduce the amplitude of lower-frequency components while enhancing higher-frequency components, thereby improving the signal's signal-to-noise ratio and distinctiveness.

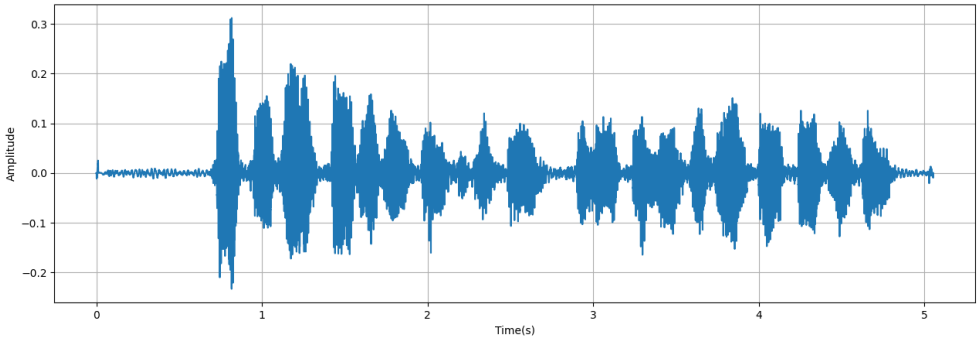
Pre-emphasis typically involves applying a first-order high-pass filter (often a first-order lag filter) with a transfer function represented as

$$x'[n] = x[n] - \alpha x[n-1]$$

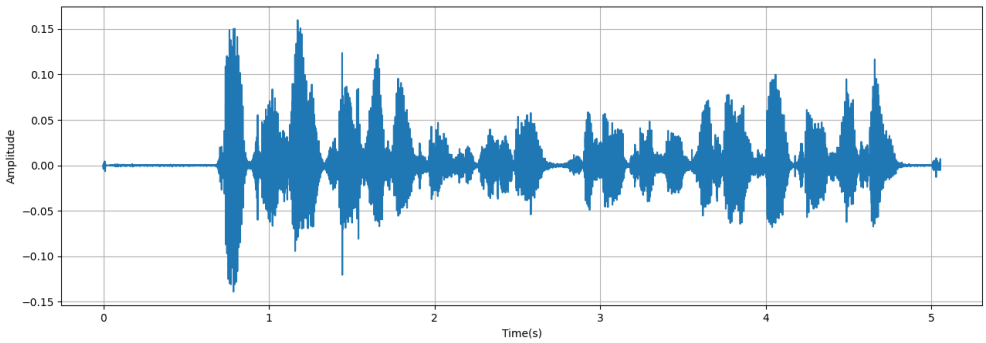
where α is the pre-emphasis coefficient, typically ranging from 0.9 to 0.97. This filter reduces the amplitude of lower-frequency components, enhancing the higher-frequency components.

The result graph is:

Origin graph



Pre-emphasis graph

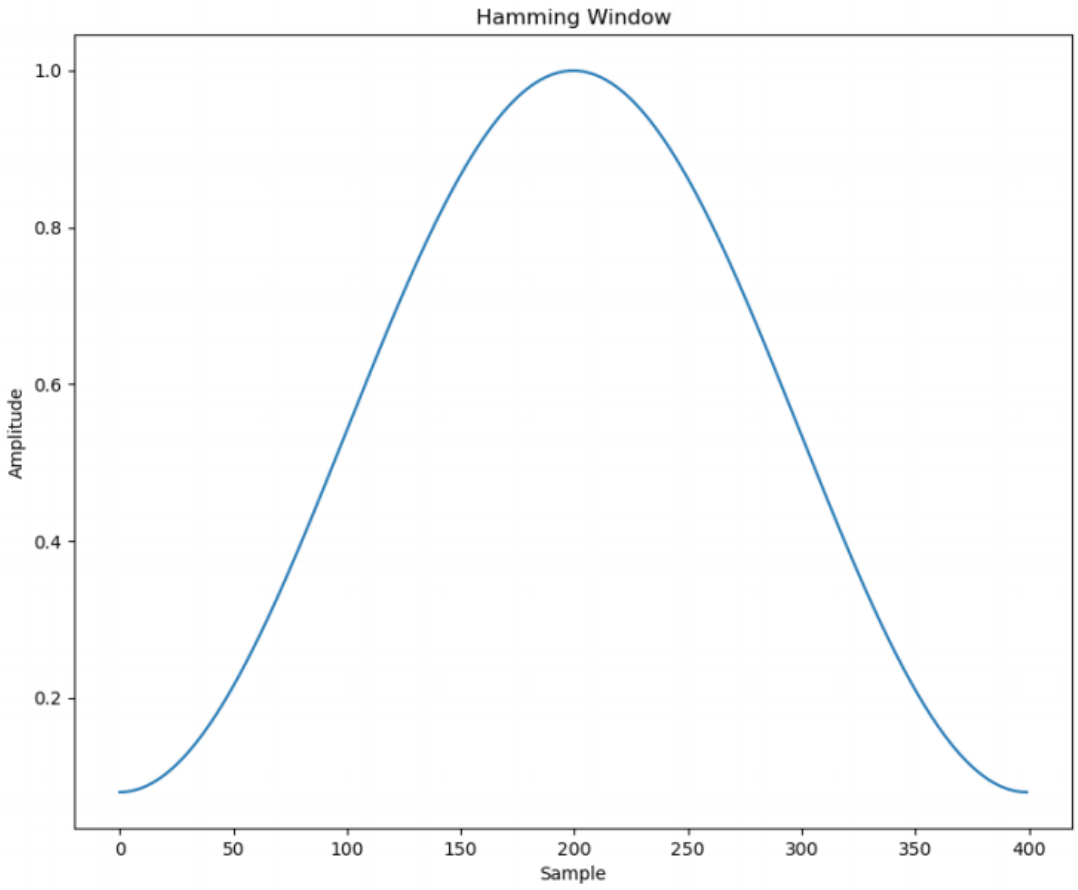


Windowing

The purpose of the window function is to limit the signal within the frame and reduce the amplitude at the frame boundaries, thus mitigating spectral leakage. There are some windowing function including rectangular window, hamming window, hanning window. There we choose to use hamming window.

Windowing need to divide the signal into plenty of frames. Each frame represents a short segment of the audio signal, typically ranging from tens of milliseconds to a few hundred milliseconds. This framing operation is done to capture the time-varying nature of the signal, as audio signals are rarely stationary.

Hamming window time domain waveform



hamming window time domain waveform

```
    divide_audio, frame_length = divide_frame(emp_audio, sr)  # frame dividing
    hamming_win = np.hamming(frame_length)  # hamming window
    windowed = divide_audio * hamming_win  # windowing
```

STFT

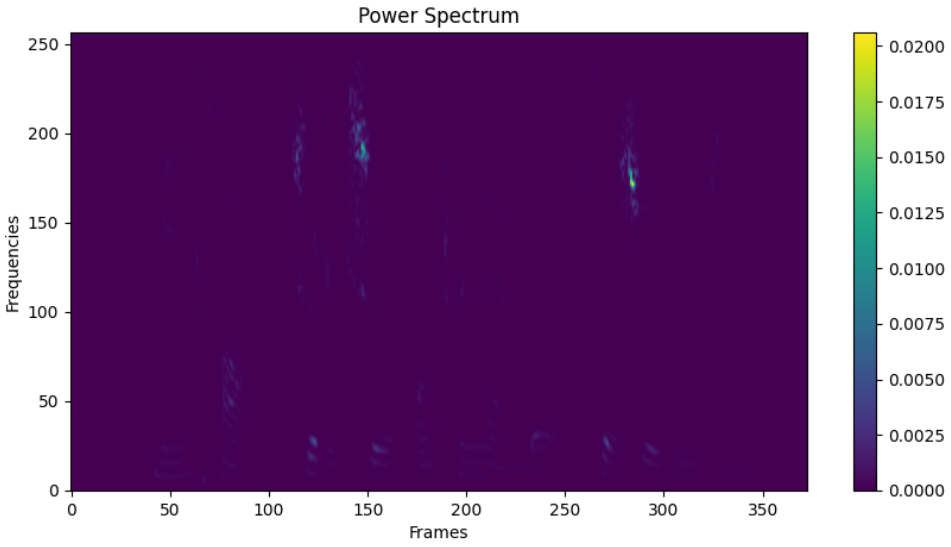
STFT can decompose a long-duration audio signal into short-time windows, each representing a spectral snapshot of the signal within a specific time frame. STFT provides a means to obtain the spectral information of the audio signal across different time and frequency components, which can capture the non-stationary nature of audio signals.

For each frame of the windowed signal, an N-point FFT transform is performed, with N usually taken to be 256 or 512, and then the energy spectrum is computed using the following equation: $P = \frac{|FFT(x_i)|^2}{N}$,

The code and the power spectrum is as follows.

```
    n_fft = 512  # points of FFT
    magnitude = np.absolute(np.fft.rfft(windowed_audio, n_fft))  # the magnitude
    power_audio = (1.0 / n_fft) * (magnitude ** 2)  # the powered audio
```

Power Spectrum graph



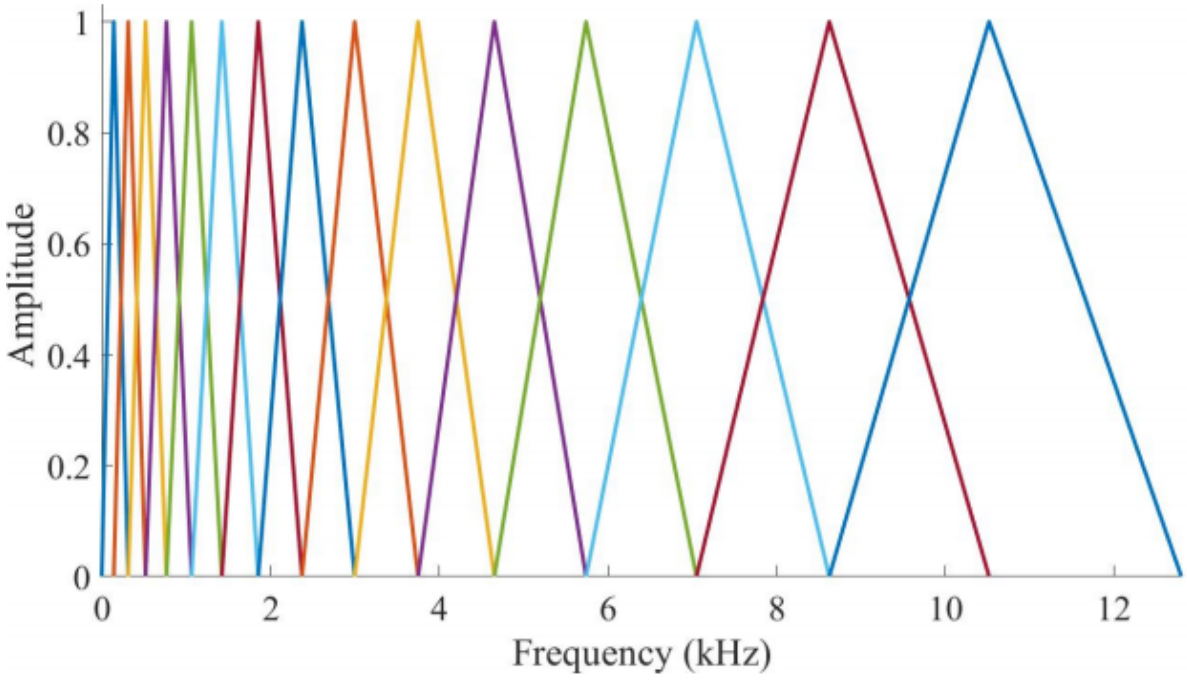
Mel-filter bank

Mel Scale can simulate receiving sound by human ear. The human ear presents a non-linear state,which is more insensitive to the high frequency, so the Mel scale has a higher resolution in the low-frequency area and a lower resolution in the high-frequency area. The conversion relationship between the frequency is as follows:

$$m = f_{mel}(f) = 1127 \ln\left(1 + \frac{f}{700}\right)$$

$$f = f_{mel}^{-1}(m) = 700\left(e^{\frac{m}{1127}} - 1\right)$$

A Mel filter bank is a series of triangular filters, usually 40 or 80, with a response value of 1 at the center frequency point, attenuated to 0 at the center of the filters on either side, as shown below:



The equation is:

$$H_m[k] = \begin{cases} 0, & k < f(m-1) \\ \frac{k-f(m-1)}{f(m)-f(m-1)}, & f(m-1) < k < f(m) \\ 1, & k = f(m) \\ \frac{f(m+1)-k}{f(m+1)-f(m)}, & f(m) < k < f(m+1) \\ 0, & k > f(m+1) \end{cases}$$

Applying the Mel filter bank on the energy spectrum with the equation:

$$Y_t[m] = \sum_{k=1}^{N} H_m[k] |X_t[k]|^2$$

where *k* is DFT bin number, m is mel-filter bank number.

Based on the above theory, the code and image results are:

```
    # generate filters
    for i in range(1, n_filters + 1):
        left = int(f[i - 1])
        center = int(f[i])
        right = int(f[i + 1])
        # f(m-1)<k<f(m)
        for j in range(left, center):
            filter_bank[i - 1, j + 1] = (j + 1 - f[i - 1]) / (f[i] - f[i - 1])
        # f(m)<k<f(m+1)
        for j in range(center, right):
            filter_bank[i - 1, j + 1] = (f[i + 1] - (j + 1)) / (f[i + 1] - f[i])
    # apply the filters to the audio
    energy = np.sum(stft_audio, 1)  # get the energy
    filtered_data = np.dot(stft_audio, filter_bank.T)  # dot product
```

Log()

The logarithm of the Mel-filtered coefficients can emulate human auditory perception, compress the dynamic range, simplify computations, and enhance noise robustness.

$$Y_t[m] = \log Y_t[m]$$

DCT

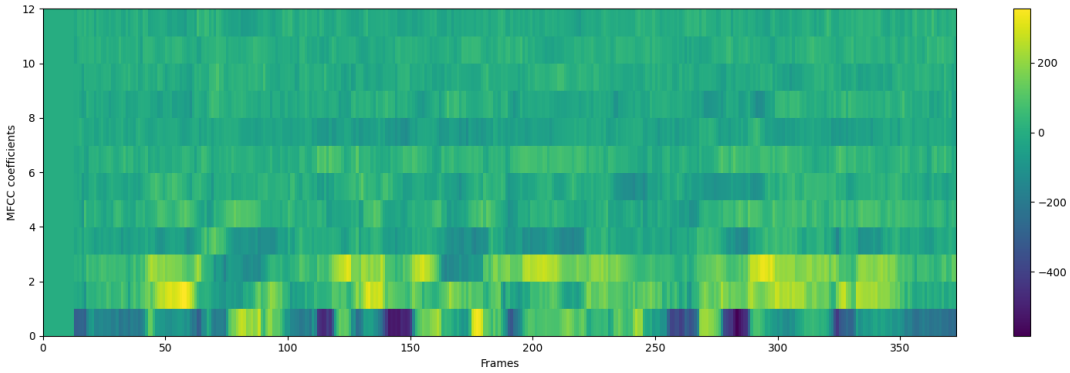
The mel-filter bank features are often highly correlated. Therefore it is possible to continue to compress these correlated filter bank coefficients with the DCT transform. Here we compress to 12 dimensions.

The code and the result are as follows:

```
def DCT(logged_audio, n_mfcc=26, n_ceps=12):
    """
    :param n_mfcc: number of MFCC
    :param n_ceps: number of cepstral coefficients
    :return: the DCTed audio(keep only 12 frames of 26)
    """
    transpose = logged_audio.T
    len_data = len(transpose)
```

```
    dct_audio = []
    for j in range(n_mfcc):
        temp = 0
        for m in range(len_data):
            temp += (transpose[m]) * np.cos(j * (m + 0.5) * np.pi / len_data)
        dct_audio.append(temp)
    mfcc = np.array(dct_audio[1:n_ceps + 1])
    plot_spectrogram(mfcc, "MFCC coefficients")
    return mfcc
```

MFCC graph



Dynamic feature extraction

Dynamic feature extraction records changes in MFCC over time, which enhances the representation of MFCC over time and also helps to minimize noise effects. Dynamic feature is calculated by Δ :

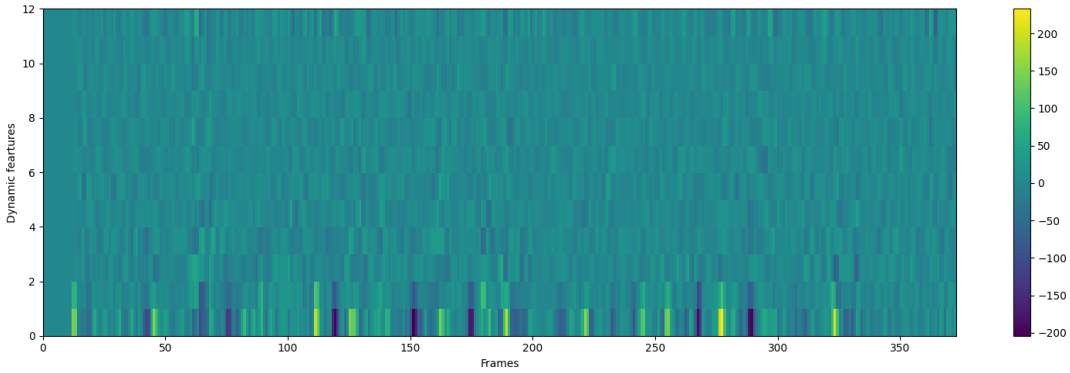
$$\Delta_t = \frac{\boldsymbol{c}_{t+1} - \boldsymbol{c}_{t-1}}{2}$$

where c is cepstral feature.

The code and the dynamic fearture is as follows:

```
def dynamic_fearture_extraction(dcted_audio):
    delta_audio= delta(dcted_audio)
    delta2_audio = delta(delta_audio)
    plot_spectrogram(delta2_audio, "Dynamic feartures")
    return delta2_audio
```

Dynamic feature graph



Feature normalization

The step is to transform the features to reduce mismatch between training and test. Calculated using the following equations:

$$\hat{y}_t[j] = \frac{y_t[j] - \mu(\boldsymbol{y}[j])}{\sigma(\boldsymbol{y}[j])}$$

where μ is the mean and the δ is the standard deviation

```
def normalization(dynamic_feature):
    mean = np.mean(dynamic_feature, axis=1)
    std = np.std(dynamic_feature, axis=1)
    transpose = dynamic_feature.T
    for i in range(len(transpose)):
        transpose[i] = (transpose[i] - mean) / std
    return transpose.T
```

Summary

Based on the course content and slides, I follow the 7 steps to complete the homework content step by step. I have learned some of the basics of speech recognition, such as time domain, frequency domain, Fourier transform, etc. And I have learned to use some python plot function. I have extracted the semantic features, the actual meaning of the information, from the digital signal, after the pre-emphasis, windowing, STFT, filtering, etc. It is really a great and magical thing, the sense of accomplishment after the code is written also came out.

黄晨冉 2050664