HW2 Report

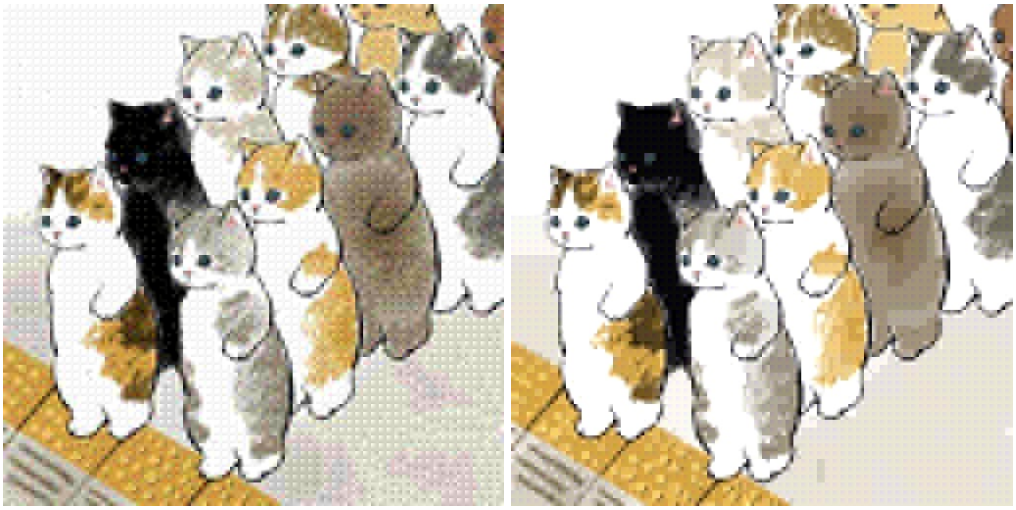
108062119 鄭幃謙

1. DCT image compression

Compare the differences between the results with DCT compression performed in two color spaces in (a) and (b) and discuss the results.

理論上，(b)的做法有先將圖片轉至YCbCr空間，圖片的亮度和色度先被分離，讓我們在做2D-DCT壓縮時高頻分量集中在色度，而低頻集中在亮度，壓縮的效果會相比於(a)的作法更好。

1. cat_n2m4 (a) / (b)

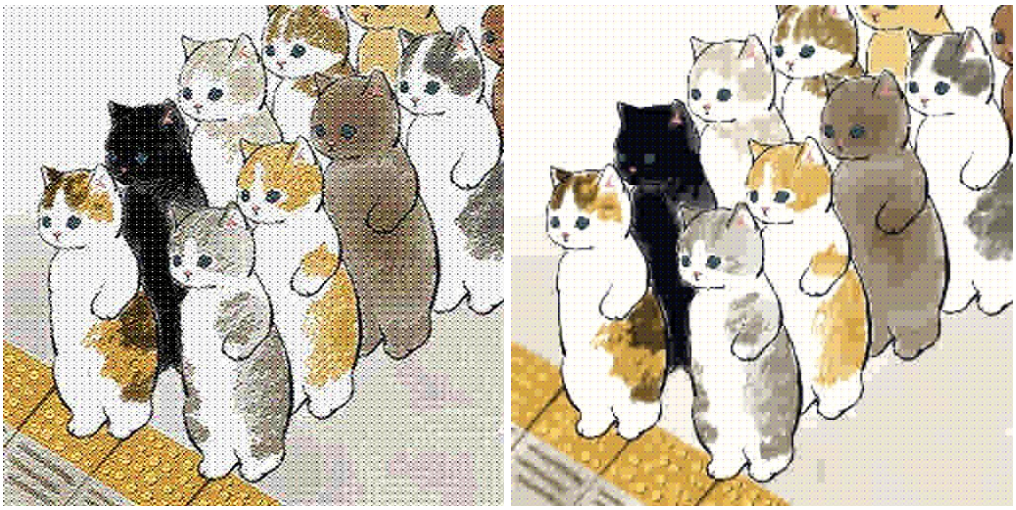


這裡很明顯的可以看到(a)作法使得圖片中每個8*8的方格明顯的左上較亮，我想這是在2D-DCT過程中產生的亮度失真，而在(b)的結果中這個問題並不嚴重，可知透過先把圖片轉至YCbCr空間確實能夠在壓縮過程中有更好的結果。

2. bar_n4m8 (a) / (b)

因為壓縮過程中的失真主要是由quantization造成，在m提高後可以有很好的改善，這兩個結果對比下肉眼就看不出什麼區別了。

3. cat_n4m4 (a) / (b)



在m=4的時候明顯看得出圖片中的方格感，但是(a)的作法在亮度的失真上明顯比(b)嚴重。

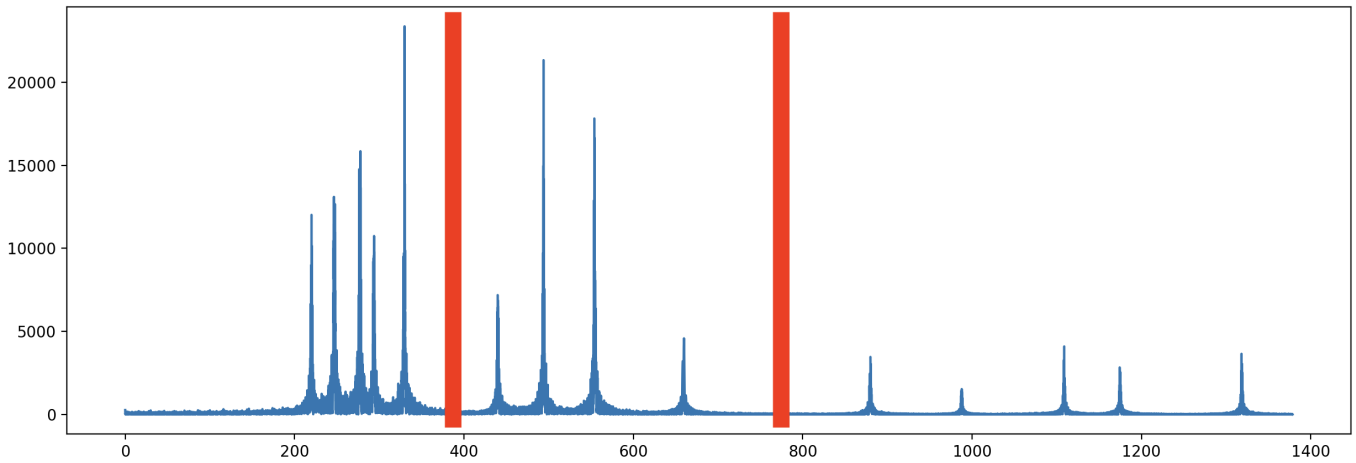
Compression rate and PSNR values

```
cat_N2_M4_A -- compression ratio: 32, PSNR: 29.9718
cat_N2_M8_A -- compression ratio: 16, PSNR: 32.9167
cat_N4_M4_A -- compression ratio: 08, PSNR: 29.0216
cat_N4_M8_A -- compression ratio: 04, PSNR: 33.9120
bar_N2_M4_A -- compression ratio: 32, PSNR: 30.2938
bar_N2_M8_A -- compression ratio: 16, PSNR: 31.3996
bar_N4_M4_A -- compression ratio: 08, PSNR: 29.7751
bar_N4_M8_A -- compression ratio: 04, PSNR: 33.0103
cat_N2_M4_B -- compression ratio: 64, PSNR: 29.9664
cat_N2_M8_B -- compression ratio: 32, PSNR: 29.6289
cat_N4_M4_B -- compression ratio: 16, PSNR: 29.7525
cat_N4_M8_B -- compression ratio: 08, PSNR: 29.7852
bar_N2_M4_B -- compression ratio: 64, PSNR: 29.1305
bar_N2_M8_B -- compression ratio: 32, PSNR: 29.5969
bar_N4_M4_B -- compression ratio: 16, PSNR: 28.3580
bar_N4_M8_B -- compression ratio: 08, PSNR: 29.7928
```

在PSNR的部分全部的圖片都差不多是在30左右，應該算是還ok。

2. Create your own FIR filters to filter audio signal

1. Discuss how you determine the filters.




```
# build filter
filter1 = constructFilter(freq_sample, 10001, 'low-pass', 400)
filter2 = constructFilter(freq_sample, 10001, 'high-pass', 800)
filter3 = constructFilter(freq_sample, 10001, 'band-pass', 400, 800)
```

我在觀察input的spectrum後分別使用low pass, high pass, band pass分離出頻率中的三個區間，由上圖的spectrum可以看到三個區間的分隔點。

2. How you implement the filter and convolutions to separate the mixed song and one/multiple fold echo?

filter

基本上是參考講義中的作法:

`constructFilter(freq_sample, windowSize, mode, *freq_cut)`

```
if mode == 'band-pass':
    (freq_cut1, freq_cut2) = freq_cut
    freq_cut1, freq_cut2 = freq_cut1 / freq_sample, freq_cut2 /
freq_sample
else:
    (freq_cut,) = freq_cut
    freq_cut = freq_cut / freq_sample
middle = windowSize // 2
fltr = np.zeros(windowSize)
for n in range(- windowSize // 2, windowSize // 2 + 1):
    if n == 0:
        fltr[middle] = 1
    else:
        if mode == 'low-pass':
            fltr[n + middle] = np.sin(2 * np.pi * freq_cut * n) / (np.pi *
n)
        elif mode == 'high-pass':
            fltr[n + middle] = -np.sin(2 * np.pi * freq_cut * n) / (np.pi
* n)
        else:
            fltr[n + middle] = np.sin(2 * np.pi * freq_cut2 * n) / (np.pi
```

```
* n) - np.sin(2 * np.pi * freq_cut1 * n) / (np.pi * n)
if mode == 'low-pass':
    fltr[middle] = 2 * freq_cut
elif mode == 'high-pass':
    fltr[middle] = 1 - 2 * freq_cut
else:
    fltr[middle] = 2 * (freq_cut2 - freq_cut1)
for i in range(windowSize):
    fltr[i] *= (0.42 - 0.5 * np.cos((2 * np.pi * i) / (windowSize - 1)) +
0.08 * np.cos((4 * np.pi * i) / (windowSize - 1)))
return fltr
```

這裡的windowSize我都是使用10001，再用更高的order也聽不太出區別了。

convolution

我使用fft後相乘再ifft的做法，算的比較快

```
def convolution(audio, filter)
```

```
filter_pad = np.zeros(len(audio))
filter_pad[:len(filter)] = filter[::-1]
filtered_audio = np.fft.ifft(np.fft.fft(audio) *
np.fft.fft(filter_pad)).real
return filtered_audio
```

其中必須先將filter加上padding使他和音頻的長度相同。

echo

同樣是講義的方法

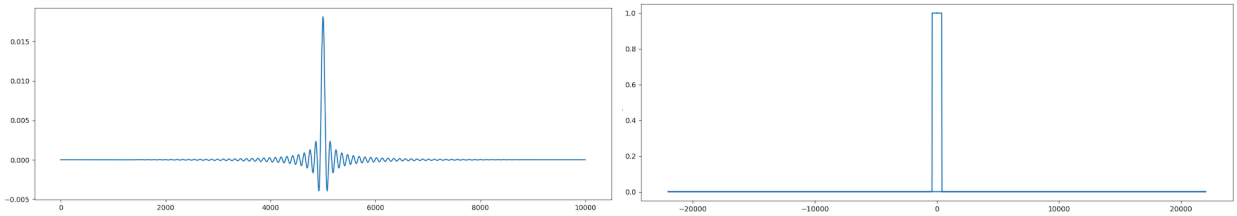
```
def echo(signal, mode='one')
```

```
delay = 12800
newSignal = np.zeros_like(signal)
newSignal[:delay] = signal[:delay]
for i in range(delay, len(signal)):
    if mode == 'one':
        newSignal[i] = signal[i] + 0.8 * signal[i - delay]
    else:
        newSignal[i] = signal[i] + 0.8 * newSignal[i - delay]
return newSignal
```

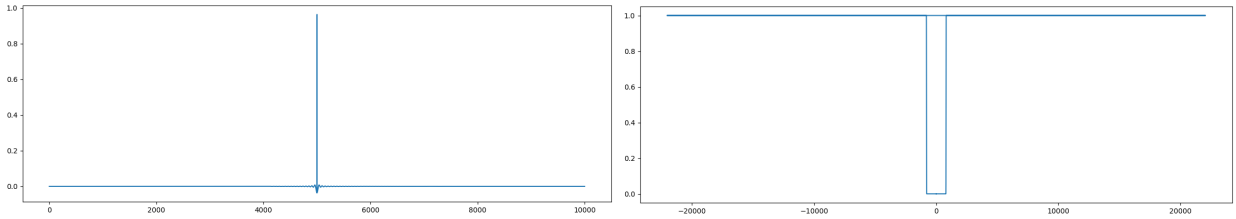
就是將音頻加上去疊起來，這裡delay我用12800比較聽得出回音。

3. Compare spectrum and shape of the filters.

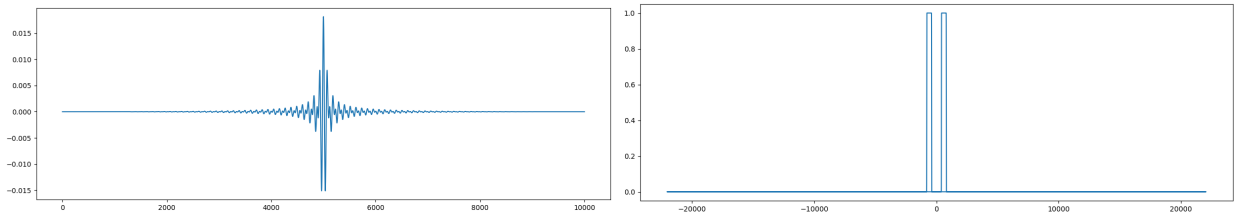
Low pass shape / spectrum



High pass shape / spectrum



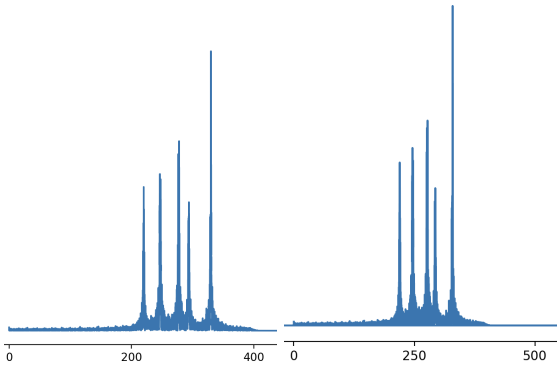
Band pass shape / spectrum



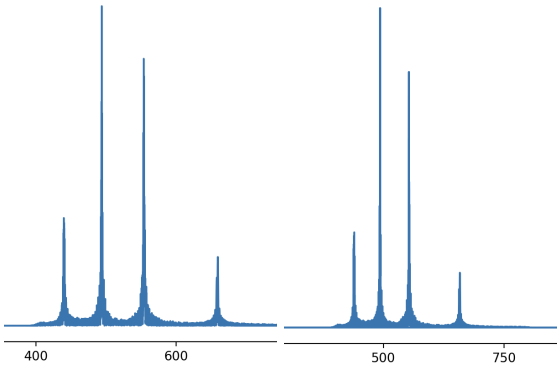
可以看到這些filter的shape和spectrum符合各自的特徵，spectrum更可以清楚看出是讓哪部分的頻率通過，如low pass只讓threshold以下的頻率通過，high pass則相反，band pass則是只讓一個區間的通過。

4. Briefly compare the difference between signals before and after reducing the sampling rates.

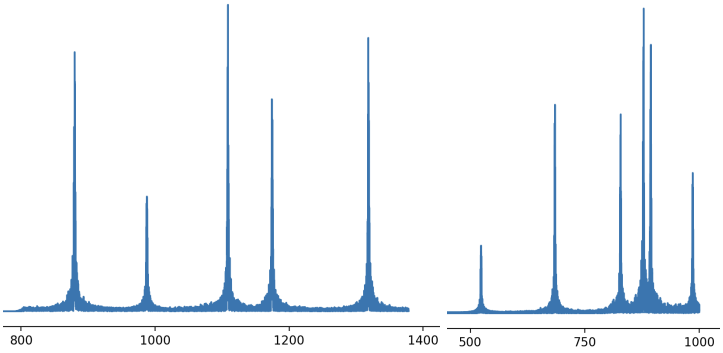
Low pass before / after



Band pass before / after



High pass before / after



根據Nyquist Theorem，用於sample的頻率至少要為取樣音頻中最高頻率的兩倍，而在low pass和band pass中取出的音頻最高頻率分別是350Hz和670Hz左右，因此Nyquist frequency分別是700Hz和1340Hz，以題目要求的2000Hz降取樣出的音頻跟原本的並無太大差別。但是在high pass取出的音頻中最高頻率約為1330Hz，Nyquist frequency為2660Hz，因此無法以2000Hz取樣出和原本相同的音頻，聽起來就和原本的音頻差非常多，整體的頻率降低非常多，這是sample rate太低造成的aliasing。