CS3570 Introduction to Multimedia

Homework #2 Report

103062234 張克齊

Q1. (50%) Create your own FIR filters to filter audio signal

[實作方法]

1. my_filter.m

為主要實作的函式，傳入的變數中，功能為註解中所提到的，其中要注意的是，我傳入的 fcutoff 是利用 vector 的型式傳入，裡面有兩個值，在 low-pass 和 high-pass 中，只會用到第一項的值，傳入 vector 會像是 [500 0]；而當我們在實作 band-pass 和 band-stop 時，會需要下界和上界，因此需要兩個值，這時傳入的 vector 就會像是 [200 900]，而在呼叫時就能利用 fcutoff(1)和 fcutoff(2)代表第一和第二項。

依照講義 slide #80 中的步驟，我們首先要先對傳入的 fcutoff 做 Normalization，

```
fcutoff(1) = fcutoff(1) / fsample;
fcutoff(2) = fcutoff(2) / fsample;
```

，並找出我們 FIR size 的中間點 middle，而因為 middle 的值是奇數，這邊要再取 floor()以不考慮 remainder。

第二步是我們要依照講義 slide #76 公式做出我們可以使用的四種 filter(low-pass, high-pass, bandpass, bandstop)，利用 strcmp()的方式比較字串決定要用哪種 filter，其中要小心 matlab 中 matrix 是從 1 開始所以要從 0~N-1 向右平移到 1~N(index + 1)。下圖是 low-pass 的實作，其他三種改成對應公式的值即可。

```
if strcmp(filterName,'low-pass') == 1
    for n = ceil(-N/2) : floor(N/2);
        if n == 0
            h(middle + 1) = 1;
        else
            h(n + middle + 1) = sin(wcutoff*n) / (pi*n);
        end
    end
    h(middle + 1) = 2 * fcutoff(1);
end
```

第三步中，我們要決定我們的 windowing function，在這邊用的是 Blackman，其中要將 slide #79 公式中的'+'改成'-'，原因為 ilms 討論區說的範圍是從-N/2~N/2，這樣就能透過 window 得到可以處理 input signal 的 filter。

```
for n = 1 : N;
    h(n) = h(n) * (0.42 - 0.5 * cos(2*pi*(n-1)/N) + 0.08 * cos(4*pi*(n-1)/N));
end
```

最後，利用 slide #64 中 FIR 公式對 Input 和 filter 做 1D-convolution，其中要記得處理 n-k<=0 的情況(該項會為 0)，就能得到最後 filtered 後的結果。

```
for n = 1 : length(inputSignal);
    for k = 1 : N;
        if n - k <= 0
            outputSignal(n) = outputSignal(n) + 0;
        else
            outputSignal(n) = outputSignal(n) + h(k) * inputSignal(n-k);
        end
    end
end
```

2. HW2_Q1.m

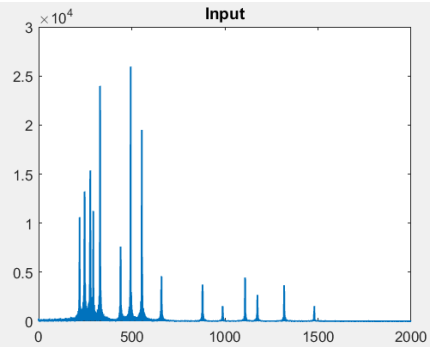
主程式中，主要是呼叫 my_filter 函式得到結果後並畫圖，傳入的變數中，我 N 設成 600，並分別利用 low-pass, band-pass, high-pass 分出三首歌，fcutoff 要記得如同前面所述傳入 vector，就能得到 output 的 Signal 和 Filter；在畫圖方面，因為我們給的 N 為 600，所以在畫 time domain filter 時我們可以把 x 軸限制在 0~600 間，方法為利用 xlim([0 600])；後面畫 frequency domain filer 和 output signal 則可以仿效畫 input 的方法得到結果，最後再利用 audiowrite()存下分離出來的三首歌。

```
[song1Signal, song1Filter] = my_filter(y_input, fs, 600, 'Blackman', 'low-pass', [200, 0]);
[song2Signal, song2Filter] = my_filter(y_input, fs, 600, 'Blackman', 'bandpass', [500, 700]);
[song3Signal, song3Filter] = my_filter(y_input, fs, 600, 'Blackman', 'high-pass', [900, 0]);
```

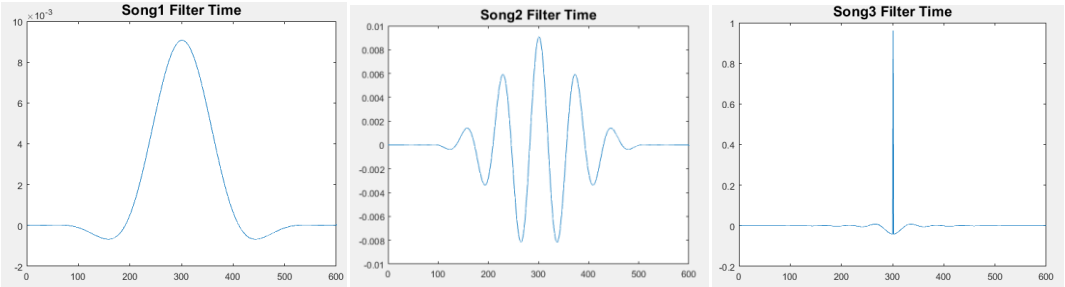
```
audiowrite('audio\low_pass_200.wav', song1Signal, fs);
audiowrite('audio\band_pass_500_700.wav', song2Signal, fs);
audiowrite('audio\high_pass_900.wav', song3Signal, fs);
```

[結果圖]

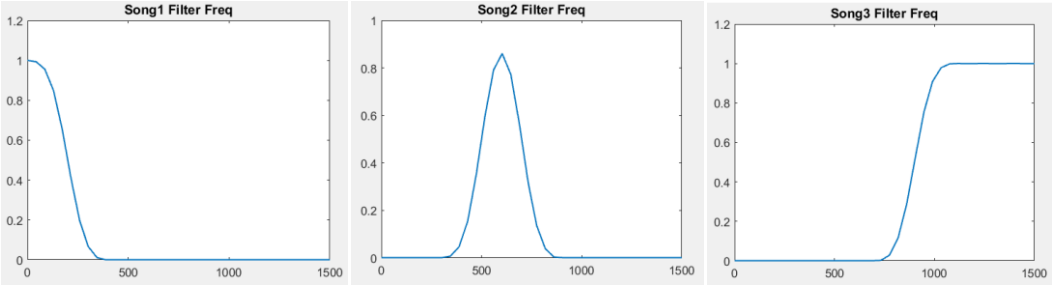
The spectrum of the input signal



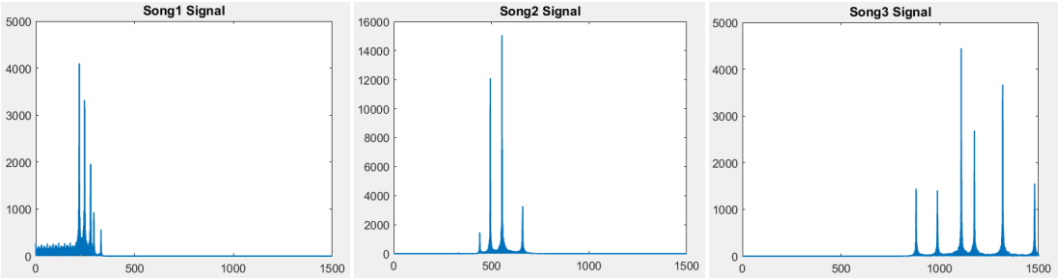
The shapes of the filters (time domain)



The spectrums of the filters



The spectrums of the output signals



[結論]

1. 決定 filter 的方法: 因為要分出三首歌，我自然就會想到可能分別在低頻、中頻、高頻的地方，所以我分別使用 low-pass, band-pass, high-pass 三種 filter 來分出三首歌，而其中的 cutoff frequency 是經過慢慢調整到沒有其他歌的介入為止。

2. filter 與 convolution 的使用: 實作方法中有提到步驟，首先算出我們要用來 convolution 的 filter，並給他我們要的 window function(這邊是用 Blackman)，因為這是一個有限的音頻，所以我們可以利用 FIR 對 input 和 filter 做 convolution，這樣就能濾掉 input 的 signal，達到分離歌曲的結果。

3. 圖形比較:

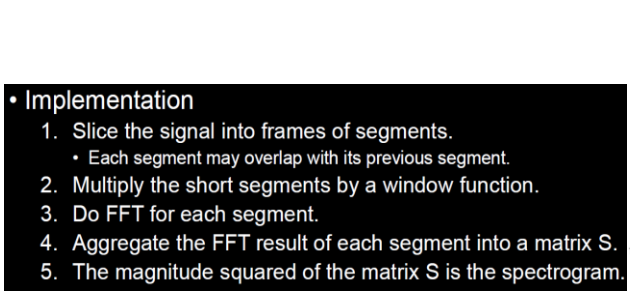
(1) Filter: 在 Time Domain 中，會得到這樣的圖是因為頻率域轉時域是利用 sinc 函數轉換，公式是 $\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ 。當我們取的頻率愈高時，會使中間的 sinc 週期增加，可以注意的是因為除上了 pi*x，所以當頻率愈高時我們可以發現 y 軸的值變小。在 Frequency Domain 的圖中，可以得到類似 slide #68 的理想圖形，但因為在計算時我們是利用 sin 函數，所以並不會很明顯的垂直向下(向上)，而是會有一些坡度的向下(向上)。

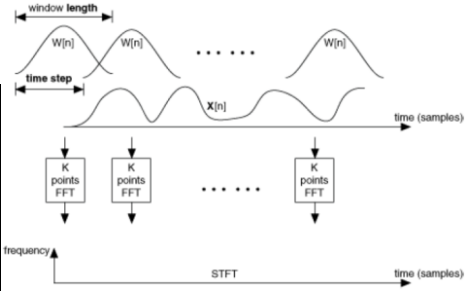
(2) Output signal: 根據不同的 filter，可以發現對應不同的 output 結果，例如如果是 low-pass，那就會發現 input 中只有低頻被保留，其他頻率的 signal 都會被濾掉。從上面 input 跟 output signal 的圖相互對照，效果是很好的。

Q2. (50%) Music classification through spectrograms and human eyes:

[實作方法]

1. my_stft.m





在實作 STFT 時，主要是參照 reference 中所提示的步驟進行，再分別得到最後的 S、T、F。我的作法是將切下的每一段做完處理並儲存後再處理下一段，也就是利用一個 for 迴圈來完成每一段的處理，但如上右圖所示知道要處理 overlap，會變成 `for i = 1 : (segment_duration - segment_overlap - 1) : (N - segment_duration);`，這樣就可以利用 `tmp = x(i : i+segment_duration-1);` 分割出每一段 segment。

第二步中，跟第一題一樣要做 window function 的規劃，而因為這題沒有限制函式，所以我是直接使用 hann()來做 Hann window，並利用矩陣點乘的方法省略 for 迴圈的計算，會變成 `tmp = tmp .* hann(segment_duration);`。接著第三步中，很簡單的可以利用 fft()得到該 slice 轉換到頻域的結果 `FFT = fft(tmp);`；第四步中，就可以儲存得到的結果 S 和 T，S 是把經 fft 轉換的每一段的結果(變數 FFT)存成 n(slice 數量) * segment_duration 的 matrix，T 則是每一段 slice 的中點對應到原本的 index，因此是個 1 * n(slice 數量)的 matrix，所以我設了一個變數 now 來決定每段 slice 的儲存位置。

算完存完每一段 slice 後，要將 T 中所有的數除上 sample rate 才是我們要的 `T = T / samplerate;`；最後，要決定我們的 F，F 代表它們的週期頻率，算法是 `F = (0:now-1)*samplerate/segment_duration;`，(now-1)會是 slice 的數量，這樣就能得到我們要的 S、T、F 了。

STF 公式參考網站

https://www.mathworks.com/matlabcentral/fileexchange/45197-short-time-fourier-transformation--stft--with-matlab-implementation

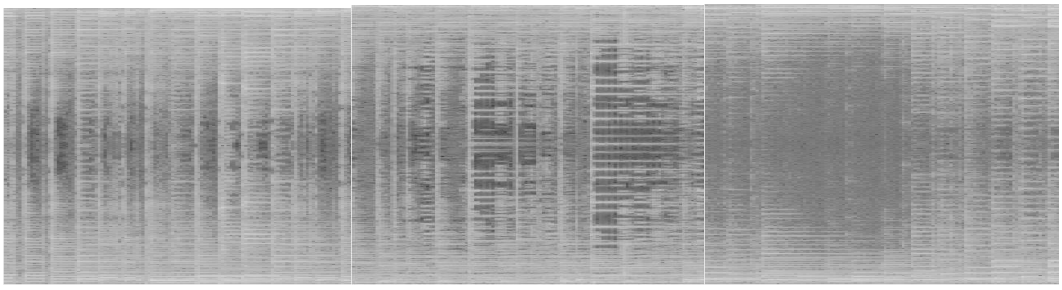
2. HW2_Q2.m

在這邊我們要做的只有把做完 STFT 後的結果的 spectrogram 存成 png 檔，用的是 imwrite()的方法，在命名我利用比較彈性的方法，用 strcat()組成我想輸出檔名的字串，這樣就能利用原本給的 class_label 來命名，如下圖。

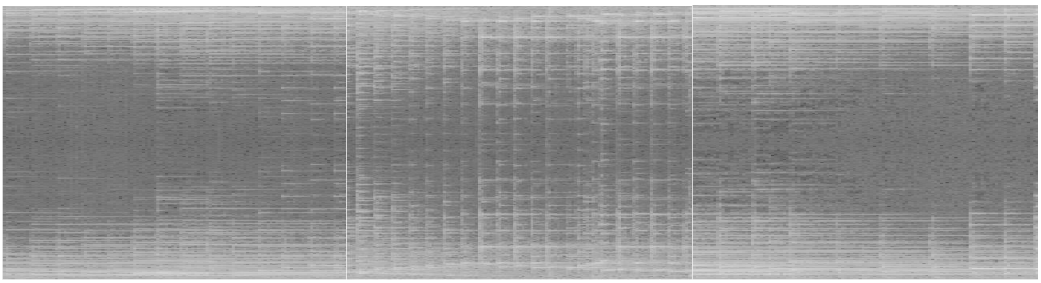
```
if i <= 3
    name = strcat('audio\', class_labels{1}, '_', num2str(i), '.png');
    imwrite(spectrogram_images{i}, name);
elseif i > 3 && i <= 6
    name = strcat('audio\', class_labels{2}, '_', num2str(i-3), '.png');
    imwrite(spectrogram_images{i}, name);
elseif i > 6 && i <= 9
    name = strcat('audio\', class_labels{3}, '_', num2str(i-6), '.png');
    imwrite(spectrogram_images{i}, name);
else
    name = strcat('audio\', class_labels{4}, '_', num2str(i-9), '.png');
    imwrite(spectrogram_images{i}, name);
end
```

[結果圖]

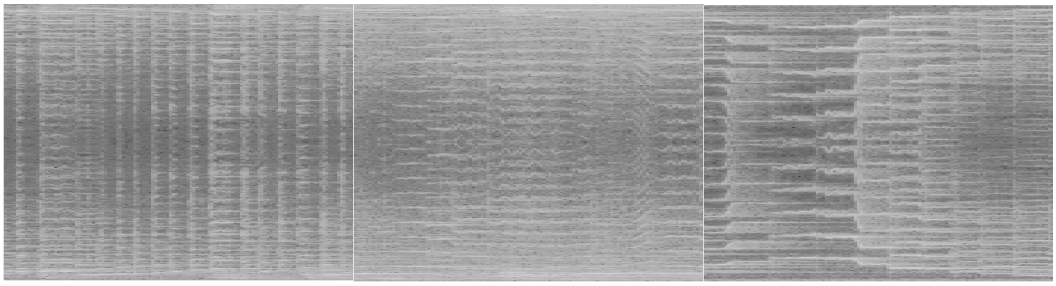
Guitar



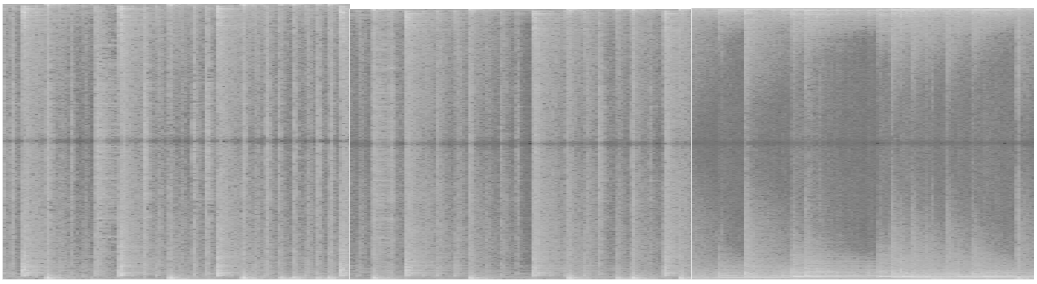
Piano



Violin



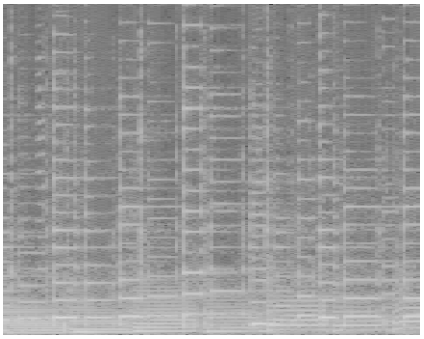
Drum



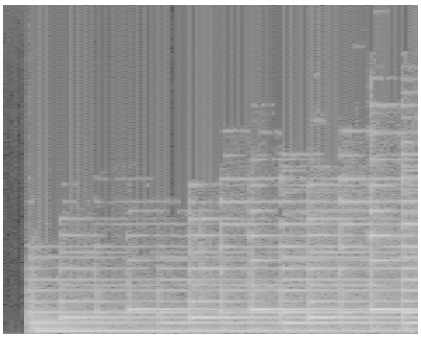
[比較]

1. 分辨不同樂器的 spectrogram:

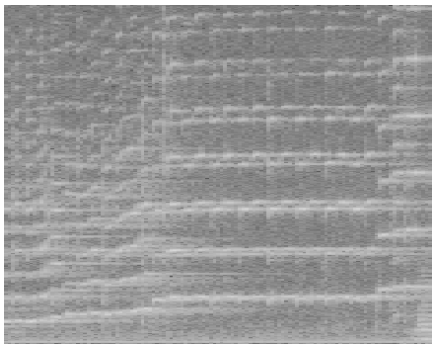
test_spectrogram_1-> Piano
排列整齊的格子狀



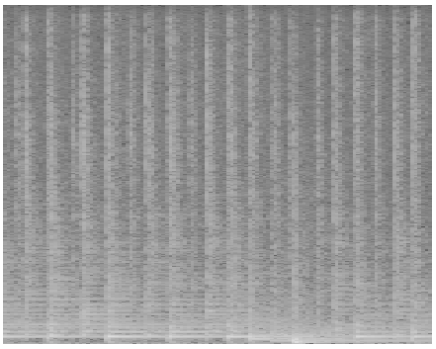
test_spectrogram_2 -> Guitar
也是格子狀，但感覺比較不整齊



test_spectrogram_3 -> Violin
右邊跟左邊的條紋規則不同但相像



test_spectrogram_4 -> Drum
很明顯的一條條直紋



2. 對 STF transform 的實作與心得: 在做 STFT 時，主要是依照 reference 中的步驟來完成，目的是將一個完整的音檔切成一段段的時間 segment 經轉換到頻域後並組合在一起，得到 STF 中的 S，並利用 T 和 F 中的資訊畫出 spectrogram，詳細實作方法在上面已說明。STFT 跟 DFT 比較來說，STFT 比 DFT 多了一個 window function，可以分析出隨著時間變化的頻率，隨著 window function 大小的不同會有不同的頻率和時間解析度，再轉成 spectrogram 也能有更好的結果來比較，因此 STFT 被常常利用在聲音的比對。