Hanze Zhang

Mr. Jason Bramburger

AMATH 482

Assignment 2 Write-up

02/10/21

Abstract:

Given a music piece consisted of plays of different instruments, is it possible for us to only filter out the play of a certain instrument? In this assignment, we will be using Gabor Transform to analyze a portion of two of the greatest rock and roll songs --- *Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd. We will try to filter out the guitar and bass from them and reproduce the music score for each instrument by creating and observing their time-frequency spectrograms.

Introduction and Overview:

The Fast Fourier Transform is the most basic tool we use for most signal processing, as it can transform our data from the spatial domain to the frequency domain. However, the result produced by Fast Fourier Transform does not retain any useful information from the spatial domain.

In order to obtain both time and frequency information, we can apply Gabor Transform on our original data. The process of Gabor Transform is to create a window at each timestep and slides it through our whole dataset. Inside each window, we will apply Fast Fourier Transform so that we can obtain the frequency information of that specific point of time. After we slides the window created by Gabor Transform through the whole data, we will be able to create a time-frequency spectrogram which allows us to observe the frequency information at each point of time.

In this assignment, we will try to use Fast Fourier Transform along with Gabor Transform to try to reproduce the music score for guitar and bass in two different songs.

With Gabor Transform, we can plot the spectrogram of each song and match each point on the spectrogram to a music note with the same frequency. In order to get a clean spectrogram, we can also apply a Gaussian filter on the maximum frequency at each timestep to filter overtones. In this way, we can reproduce the music score for *Sweet Child O' Mine* as its clip only contains guitar frequencies.

However, to get the music score for bass and guitar in *Comfortably Numb*, we have to apply another filter, as the song contains both bass and guitar. We know that the frequency range of a bass generally is from 60 to $250\ Hz$. Thus, to only get the bass, we can apply a filter in the frequency domain such that it will make any frequency higher than $250\ Hz$ and lower than $60\ Hz$ to zero. After we get bass frequencies, we can subtract them from complete frequencies to get guitar frequencies.

Theoretical Background:

In order to obtain both time and frequency information from a dataset, we have to use Fast Fourier Transform along with Gabor Transform. Gabor Transform is essentially a filter that applies on the original data so that we can only look at a specific part of the data based on timestep. Below is its equation:

$$\tilde{f}_g(\tau, k) = \int_{-\infty}^{\infty} f(t) g(t-\tau) e^{-ikt} dt \qquad (1)$$

Here, $g(t-\tau)$ is a sliding filter function and τ is the timestep we set. In this assignment, we use Gaussian as our filter function:

$$g(t-\tau) = e^{-a(t-\tau)^2} \qquad (2)$$

In this equation, $a > 0$ represents the width of the window and τ is the center of the window. By gradually increasing τ, we are sliding our filter through the whole dataset. In this way, we can capture both time and frequency information from a signal.

However, Gabor Transform also has limitations. With Gabor Transform, it is still impossible for us to get perfect resolution in time and frequency domain. An increase in the width of the filter function can increase the amount of information we can obtain from the frequency domain, but at the same time it decreases the amount of information we can obtain from the time domain and vice versa. As we increase the window width a, we are focusing more on frequencies, but we cannot really localize the signal in time. Thus, the answer to the question that how to balance the amount of information we want to get from the time domain and the frequency domain varies from tasks to tasks.

Algorithm Implementation and Development:

Before implementing Fast Fourier Transform and Gabor Transform, we need to load our data first. We use the MATLAB command "$audioread$" to help us extract the signal and its sample rate from two songs. Once we have our sample data, we can define variables to store the length of each signal in time, which can be found by dividing the number of data points by the sample rate. Meanwhile, we have to define our frequency components. Since the frequency unit of songs is Hz, we need to scale our frequency components by $\frac{1}{length\ of\ the\ sigal}$ to convert the unit from s to Hz. Then, we can start defining variables for Gabor Transform.

Since the first song clip, *Sweet Child O' Mine*, only contains guitar solo and only lasts for about 10 seconds, to get as accurate guitar notes as possible, we can set the timestep of our sliding window to 0.1 and a to 1000. At each timestep, we multiply the original signal with the Gaussian filter to perform Gabor Transform. Then, we take the Fast Fourier Transform to transform the data from the spatial domain to the frequency domain. Now we can obtain the maximum frequency at this specific time point using the MATLAB command "max". To filter overtones, I apply another Gaussian filter centered at the maximum frequency with a very thin window. Then, we can store this filtered frequency information into the corresponding column of a pre-created matrix, where each column represents a timestep. Finally, we can plot the spectrogram in which its *x-axis* is time in seconds and its *y-axis* is frequency in Hz.

As for the second song, *Comfortably Numb*, consists of both plays of guitar and bass and lasts for about 1 minutes. If we use a timestep of 0.1 for a 1-minute song clip, there would be too many windows and MATLAB cannot process that large amount of data. Thus, we adjust the timestep from 0.1 to 1 so that MATLAB will not crush. Meanwhile, in this song, the tempo is relatively faster than the first one, indicating that there are more notes being played in each timestep. We want a thinner window so we can focus more on time. We set $a = 6000$ to keep the window as thinner as possible. The process of Gabor Transform is as same as what we did for last song. The only different part is that in this song we want to separate bass frequencies from guitar frequencies. Through searching on the Internet, we know that a typical bass has frequencies ranging from 60 Hz to 250 Hz. Based on this information, we can consider keeping frequencies in this range as bass frequencies and making all other frequencies zero.

After we obtain guitar frequencies, we want to obtain guitar frequencies now. The problem now is that in this song there are too many guitar notes being played in each timestep. In order to get as many guitar notes as possible, I only use the first 10 seconds of the song for analysis. Since we only have a 10-second clip now, we can then make our timestep 0.1 and $a = 6000$. In this setting we can get as many notes as possible. The process of Gabor Transform and obtaining bass frequencies are as same as what we did above. To obtain guitar frequencies, we can subtract guitar frequencies from our original signal frequencies. However, this approach has a drawback. It is possible that a guitar and a bass play the same note, creating the same frequency. If we use this method to filter guitar and bass, it is possible that we also filter some guitar notes out.

Computational Results:

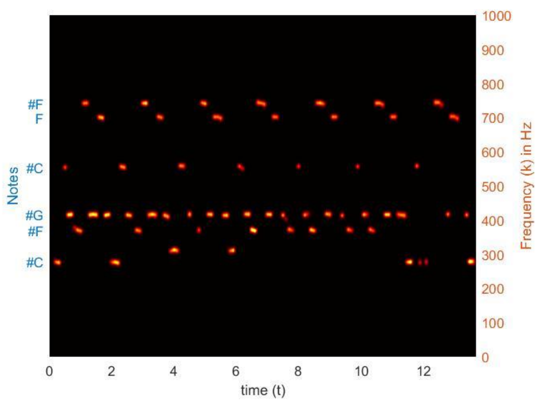

Figure 1 - Spectrogram of Sweet Child O' Mine

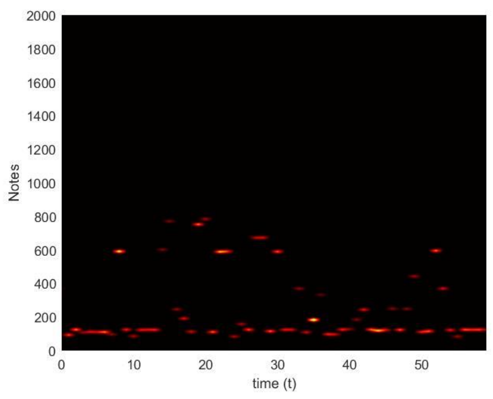

Figure 2 - Spectrogram of Comfortably Numb

From Figure 1 above, we can clearly read out the note being played at each timestep. Notes being played are (in Hz):

$$[\#C(277.18), \#G(415.30), \#F(369.99), \#C(554.37), F(698.46), \#F(739.99)]$$

From Figure 2 above, we can observe that there are a lot of notes below or around 250 Hz. After setting any frequencies higher than 250 Hz or lower than 60 Hz to 0 in order to obtain bass frequencies, we can obtain Figure 3.

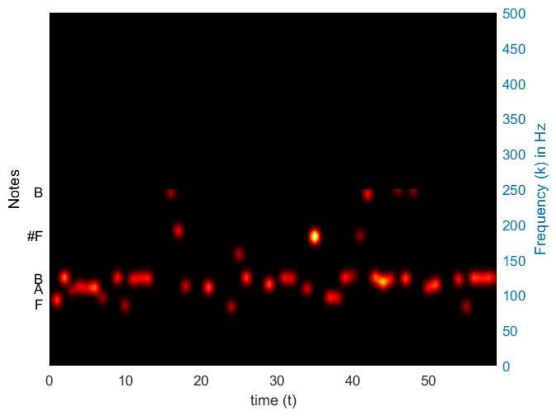

Figure 3 – Spectrogram of Comfortably Numb (bass)

From Figure 3, we can clearly read out notes being played by a bass (frequency in Hz):

$$[B(123.47), A(110.00), F(87.307), \#F(185.00), B(246.94)]$$

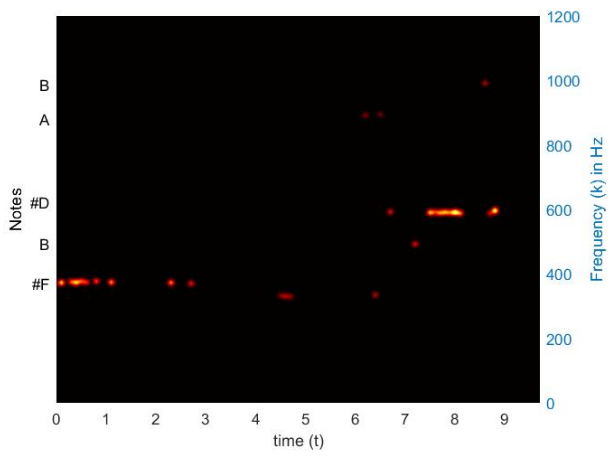

Figure 4 – Spectrogram of first 10 seconds of Comfortably Numb (guitar)

From Figure 4, we can clearly read out notes being played by guitar (frequency in Hz):

$$[B(987.77), A(880.00), \#D(622.25), B(493.88), \#F(369.99)]$$

Summary and Conclusion:

With Fast Fourier Transform and Gabor Transform, we can visualize both time and frequency information of a given signal in a spectrogram. If we are given music signals, it is possible for us to reproduce the music score of each instrument being played in the music. In this assignment, we have successfully reproduced the guitar score for Sweet Child O' Mine by Guns N' Roses, and reproduced the approximate guitar and bass score for Comfortably Numb by Pink Floyd.

Appendix A (MATLAB functions):

- $[y, Fs] = audioread(filename)$: reads data from the file named $filename$, and returns sampled data, y, and a sample rate for that data, Fs.
- $fftn$: N-D fast Fourier transform.
- $fftshift$: Shift zero-frequency component to center of spectrum.
- $max(A)$: returns the maximum elements of an array.
- $zeros$: Create array of all zeros.
- $yyaxis\ right$: activates the side of the current axes associated with the right y-axis. Subsequent graphics commands target the right side.
- $colormap(map)$: sets the colormap for the current figure to the colormap

specified by map.

- $yticks(ticks)$: sets the y-axis tick values, which are the locations along the y-axis where the tick marks appear. Specify $ticks$ as a vector of increasing values; for example, [0 2 4 6]. This command affects the current axes.
- $yticklabels(labels)$: sets the y-axis tick labels for the current axes. Specify $labels$ as a string array or a cell array of character vectors. If you specify the labels, then the y-axis tick values, and tick labels no longer update automatically based on changes to the axes.

Appendix B (MATLAB Code):

```
%%
clear all; close all; clc

% Reproduce music score for GNR
% figure(1)
[y, Fs] = audioread('GNR.m4a');
n = length(y); % Fourier modes
trgnr = n / Fs; % record time in seconds
t = (1:n) / Fs;
% plot(t, y);
% xlabel('Time [sec]'); ylabel('Amplitude');
% title('Sweet Child O'' Mine');
% p8 = audioplayer(y,Fs); playblocking(p8);

k = (1 / trgnr) * [0:(n/2 - 1) (-n/2):-1]; % frequency component
ks = fftshift(k);

tau = 0:0.1:trgnr;
a = 1000;
S = y';
Sgt_spec = zeros(n, length(tau));

for j = 1:length(tau)
        g = exp(-a * (t - tau(j)).^2); % Window function
        Sg = g .* S;
        Sgt = fft(Sg);
        [maximum, index] = max(abs(Sgt));
        Sgtf = Sgt .* exp(-0.01 * (k - k(index)).^2); % filter overtone
        Sgt_spec(:, j) = fftshift(abs(Sgtf)); % We don't want to scale it
end

figure(2)
yyaxis left
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0 1000],'Fontsize',10)
yticks([277.18, 369.99, 415.30, 554.37, 698.46, 739.99])
yticklabels({'#C', '#F', '#G', '#C', 'F', '#F'})
colormap(hot)
xlabel('time (t)'), ylabel('Notes')

yyaxis right
set(gca, 'ylim', [0, 1000], 'Fontsize', 10);
ylabel('Frequency (k) in Hz')
```

```
%%
% Reproduce music score for Floyd
clear all; close all; clc
[y, Fs] = audioread('Floyd.m4a');
n = length(y); % Fourier modes
trgnr = n / Fs; % record time in seconds
t = (1:n) / Fs;

k = (1 / trgnr) * [0:(n/2 - 1) (-n/2):-1]; % frequency component
ks = fftshift(k);

tau = 0:1:trgnr;
a = 6000;
S = y';
Sgt_spec = zeros(n - 1, length(tau));

for j = 1:length(tau)
        g = exp(-a * (t - tau(j)).^2); % Window function
        Sg = g .* S;
        Sgt = fft(Sg);

        Sgt = Sgt(1:n-1);
        [maximum, index] = max(abs(Sgt));
        filter = exp(-0.01 * (k - k(index)).^2);
        Sgtf = Sgt .* filter; % filter overtone with a Gaussian filter
        Sgtf(k > 250) = 0; % filter out any frequency higher than 250
        Sgtf(k < 60) = 0; % filter out any frequency lower than 60
        Sgt_spec(:, j) = fftshift(abs(Sgtf)); % We don't want to scale it
end

figure(3)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0 500],'Fontsize',10)
yticks([87.307, 110.00, 123.47, 185.00, 246.94])
yticklabels({'F','A', 'B', '#F', 'B'})
colormap(hot)
xlabel('time (t)'), ylabel('Notes')

yyaxis right
set(gca, 'ylim', [0 500], 'Fontsize', 10);
ylabel('Frequency (k) in Hz')
```

```
clear all; close all; clc
[y, Fs] = audioread('Floyd.m4a');
n = length(y); % Fourier modes
samples=[1,n - (50 * Fs)];
[y,Fs] = audioread('Floyd.m4a',samples);
n = length(y); % Fourier modes
trgnr = n / Fs; % record time in seconds
t = (1:n) / Fs;

k = (1 / trgnr) * [0:(n/2 - 1) -n/2:-1]; % frequency component
ks = fftshift(k);

tau = 0:0.1:trgnr;
a = 6000;
S = y';
Sgt_spec = zeros(n-1, length(tau));

for j = 1:length(tau)
    g = exp(-a * (t - tau(j)).^2); % Window function
    Sg = g .* S;
    % Sg_lowpass = lowpass(Sg, 250, Fs);
    Sgt = fft(Sg);

    Sgt = Sgt(1:n-1);
    [maximum, index] = max(abs(Sgt));
    filter = exp(-0.01 * (k - k(index)).^2);
    Sgtf = Sgt .* filter; % filter overtone with a Gaussian filter
    bass_notes = Sgtf;
    bass_notes(k > 250) = 0; % filter out any frequency higher than 250
    bass_notes(k < 60) = 0; % filter out any frequency lower than 60
    % subtract bass from the music to get the guitar
    guitar_notes = Sgtf - bass_notes;
    Sgt_spec(:, j) = fftshift(abs(guitar_notes)); % We don't want to scale it
end
```

```
figure(4)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0 1200],'Fontsize',10)
yticks([369.99, 493.88, 622.25, 880.00, 987.77])
yticklabels({'#F','B', '#D', 'A', 'B'})
colormap(hot)
xlabel('time (t)'), ylabel('Notes')

yyaxis right
set(gca, 'ylim', [0 1200], 'Fontsize', 10);
ylabel('Frequency (k) in Hz')
```