NTHU Music Information Retrieval
Group 10 Final Report

Comparison Between Fingerstyle Composition and its Corresponding Band Music

Team Members
林妍廷 108060017
鄭博薪 108062107
南政佑 111062422
陳冠宇 1112002S

Table of Content

Introduction

Fingerstyle guitar refers to a style of guitar playing technique in which a player plucks guitar with finger instead of pick. There are fingerstyle players who play accompaniment for a singer and players who solo without a singer. In our research, we focus on the latter, and specifically those who "cover" another song. By "cover", we mean that a player tries to perform another song with merely a guitar. It is an arduous and painstaking work to compose fingerstyle music, because one not only has to take care of the melody of the original song but also its chord and bass notes.

We want to see if there are rules to compose fingerstyle music. Here, we aim to identify the notes difference between fingerstyle music and their corresponding band music.

Dataset

Our dataset consists of 9 songs, each with a fingerstyle music audio file and its original band music audio file. The fingerstyle audio files are collected from a talented fingerstyle music composer/player Kelly Valleau. His style of playing authentically reproduces the melody and bass notes of the original song without fancy improvisation.

The final list of songs is as below with the format being "Name of Performer - Name of Song". To simplify, the only criteria of selecting the fingerstyle covers is to pick those which do not involve much advanced techniques, such as percussive, harmonics and nail attack.

- **John Lennon – Imagine**
- **Maroon 5 – Moves Like Jagger**
- **Coldplay – Yellow**
- **John Denver – Take Me Home, Country Roads**
- **Adele – Skyfall**
- **John Legend – All Of Me**
- **Coldplay – Hymn for the Weekend**
- **Jack Johnson – Better Together**
- **Bon Jovi – Always**

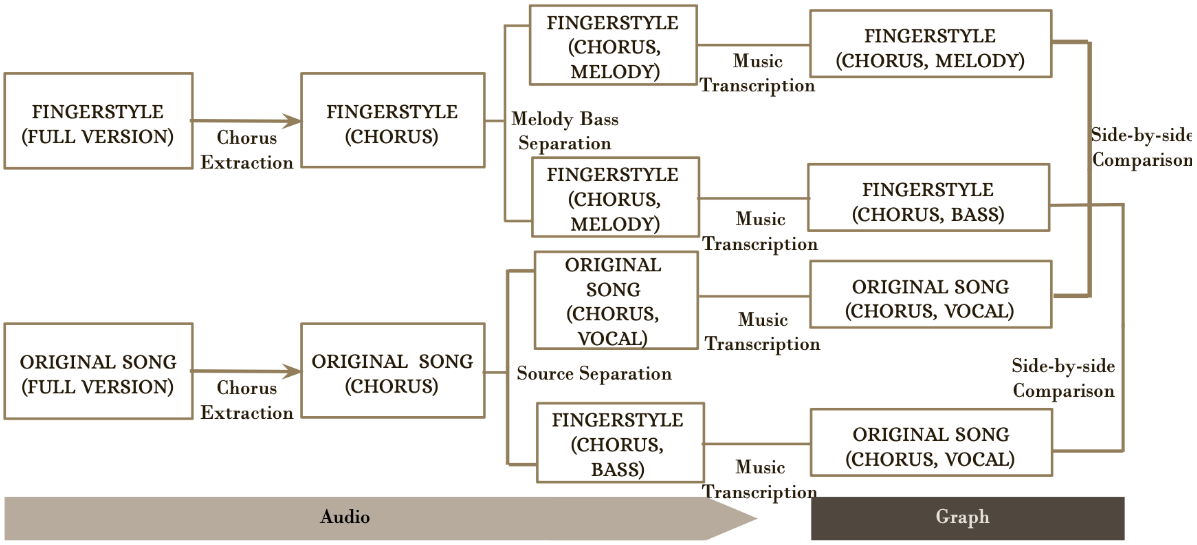
Process

We can break down the process into several nodes:

For fingerstyle audio files, we extract the chorus from the full version. Then we separate the chorus into two audio files, the melody part and the bass part. Lastly, two audio files are transcribed into time-frequency domain data respectively.

As for the original song that the fingerstyle music covers, we extract the chorus from the full version. Then we utilize source separation tools to break the audio into different tracks. Only vocal and bass tracks are selected to be transcribed into time-frequency domain data.

Finally, we do side-by-side comparison between the melody/bass audio files from fingerstyle musics and their original songs. In the next section, we will explain our implementation of these nodes in detail.


FINGERSTYLE (FULL VERSION)
Chorus Extraction
FINGERSTYLE (CHORUS)
Melody Bass Separation
FINGERSTYLE (CHORUS, MELODY)
Music Transcription
FINGERSTYLE (CHORUS, MELODY)
FINGERSTYLE (CHORUS, MELODY)
Music Transcription
FINGERSTYLE (CHORUS, BASS)
Side-by-side Comparison
ORIGINAL SONG (FULL VERSION)
Chorus Extraction
ORIGINAL SONG (CHORUS)
Source Separation
ORIGINAL SONG (CHORUS, VOCAL)
Music Transcription
ORIGINAL SONG (CHORUS, VOCAL)
FINGERSTYLE (CHORUS, BASS)
Music Transcription
ORIGINAL SONG (CHORUS, VOCAL)
Side-by-side Comparison
Audio
Graph

Explanation of Each Node

I. Chorus Extraction

- Input: Full song (audio files)
- Output: Chorus (audio files)
- Method: Pychorus
- What Is Chorus:

  A chorus is a musical section or a group of singers that repeats a specific melody or lyrics within a song. It often serves as a memorable and catchy part of a composition, typically following the verse or other sections. The chorus is usually characterized by a fuller and more harmonically rich sound, often involving multiple voices or instruments.

- Algorithm Overview:

  At a high level, since the chorus is the section that is usually repeated the most times, we try to find a part of the song that is repeated the most. The algorithm is roughly as follows:

  1) Find out what notes are playing at any moment in the song.
  2) Compare short sections of the song to every other section to see where there are repeated sections.
  3) Look for long sections that are repeated several times with a large gap between consecutive repeats.

- What Notes Are Playing:

  The first thing we need to do is turn the 1s and 0s of the song into something useful. The easiest way is to use frequencies, since frequencies tell us what notes are present (e.g. A is 440 hz).

  First it converts the raw waveform into the frequencies using a fourier transform. Then it uses knowledge about note frequencies at different octaves (A at different octaves is 440 hz, 880 hz, 1760 hz, etc) to get an idea of what notes are playing at any moment. The list of notes playing at any moment in the song is called a chromagram.

  Although looking at raw frequencies would work, we turn it into notes for several reasons. First, it provides more robustness to changes in instrumentation across different choruses. For example, if the last chorus had an extra guitar or more drums, the frequencies could be very different but the actual notes and chords would be similar. Also, working with musical notes allows us to work in a lower dimensional space (12 possible notes vs 5,000 different frequencies humans can hear).

- Finding Similarities:

Because our song is a collection of frames of 12 notes each, we need a similarity function to compare any given song frames. If v_1 and v_2 are the 12 note vectors at any two instances in the song, then we'll use the following similarity function:

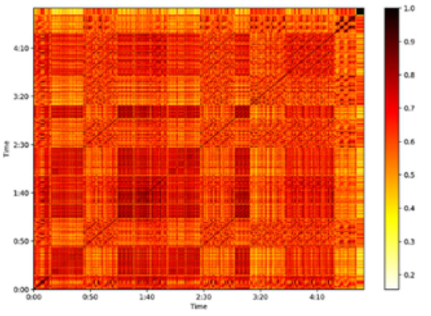
$$1 - \frac{||v_1 - v_2||}{\sqrt{12}}$$

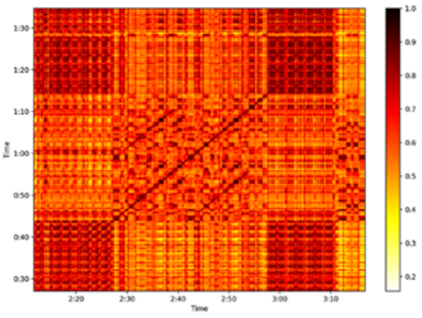
What we now need to do is iterate over all possible pairs of song frames, see which ones are similar, and look for two regions with consecutive frames that are similar. Formally we can create a matrix M where $M[x][y] = similarity(x, y)$ with the similarity function used above. This is called a time-time similarity matrix.

The full time-time matrix is pictured below. Notice a few properties:

It is symmetric about the diagonal since the similarity between frames x and y is the same as between y and x: $M[x][y] = M[y][x]$

The diagonal is all 1 because every frame sounds exactly like itself: Specifically $M[x][x] = 1$



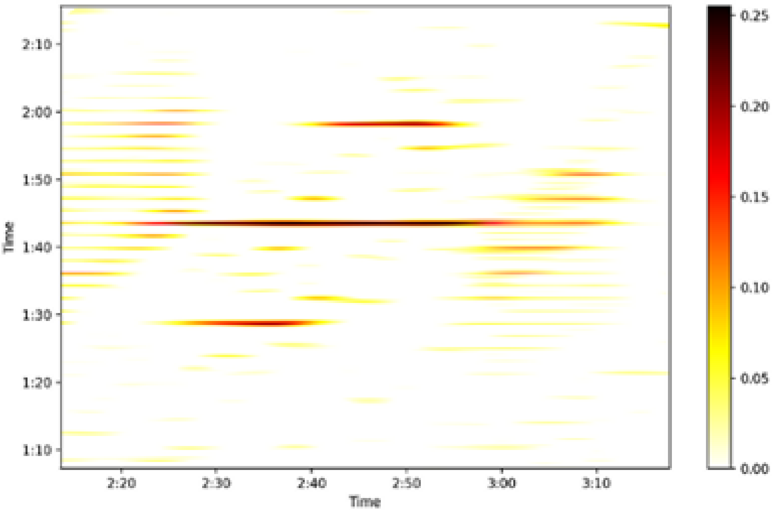


Notice the dark diagonal line that stretches from 2:30–3:00 on the x axis, and 0:45–1:15 on the y axis. This corresponds to the match between the first and second chorus that were played above.Intuitively, if the clip from 0:45–1:15 is similar to the clip from 2:30–3:00, then that means the frame at 0:45 should match the frame at 2:30, the frame 0:46 matches the frame at 2:31… until the frame at 1:15 matches the frame at 3:00. Hopefully you can see that repeated sections will always be expressed as diagonal lines.

- Time Lag Similarity Matrix:

  Unfortunately, detecting diagonal lines in a matrix is harder and slower than detecting vertical or horizontal lines. Because of this, we can apply a simple linear transform so that the repeated choruses appear as horizontal lines instead of diagonal lines, making later processing much easier. This is done by creating a time-lag similarity matrix. If the time-time matrix measures the similarity between frames x and y, then the time-lag matrix measures the similarity between frame x and the frame that happened y seconds ago. Formally, time-lag matrix $T[x][y] = M[x][x - y] = similarity(x, x - y)$.

  Since we know we are looking for horizontal lines, we can do some nice 1-D denoising and smoothing techniques that isolate these lines. This is basically computer vision now, and you can see the result of that chorus intersection after denoising.



  From this denoised matrix, we can easily identify lines (which are repeated sections) by iterating over each row and using a threshold for the minimum detection score and minimum line length.

- Counting All Repeated Sections:

  Now that we have a method to detect repeated sections in the song, we need a way to count all the times a chorus repeats. In theory, if the chorus happened n times in the song, we should have x^2 lines representing all possible intersections. We rarely detect all of them, but only a few are necessary to have an idea of where the choruses are.

Notice how all the detected lines match up either horizontally or vertically. For each line segment, we simply look in the horizontal and vertical direction for other line segments, and get a score based on how many other lines we intersect. To output the chorus, we can choose any of the repeated chorus sections, but we'll take the one with the most intersections of the strongest similarity scores. (Again, in theory they all have the same number of intersections, but in practice there is always one that has stronger similarity scores).

Source:
https://towardsdatascience.com/finding-choruses-in-songs-with-python-a925165f94a8

- Shortcomings and Future Work:

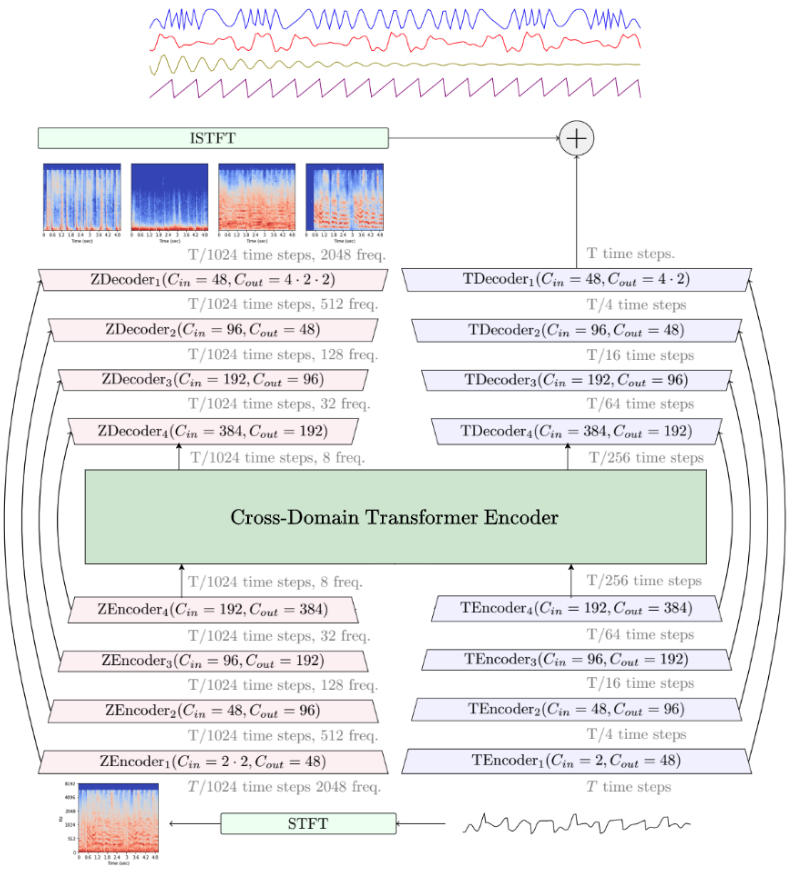
  One of the main problems is that this method does not work for music recorded at an inconsistent tempo. For example, a lot of older music such as rock was often recorded without a metronome but with the drummer keeping time. If one chorus is faster or slower than another then it often doesn't get picked up.

- Conclusion:

  One of the main advantages of this method is that it runs in a few seconds for a full length song on a cpu, as opposed to neural network methods which can require more computation or a gpu. Even though a machine learning model would almost certainly do better, this algorithm is easy to understand and a good introduction to signal processing and music pattern recognition.

II. Source Separation for Band Music

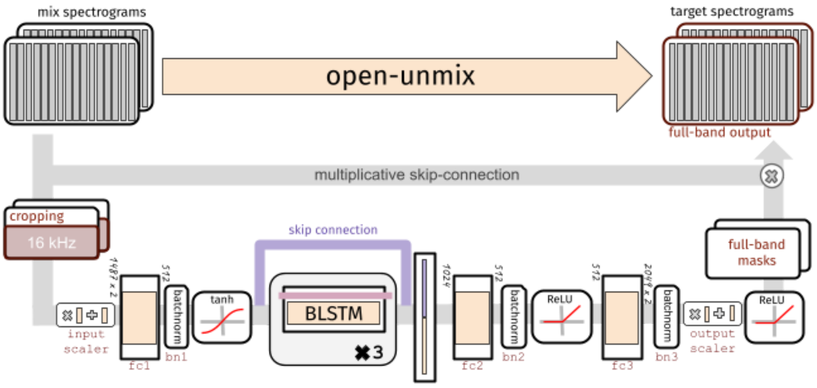
- Input: Chorus (audio files)
- Output: Four tracks, including bass, vocal, drum and others (audio files)
- Method: Demucs v4, Open-Unmix
- Algorithm Overview:
  - ○ Demucs v4: Demucs is based on a U-Net convolutional architecture inspired by Wave-U-Net. The v4 version features Hybrid Transformer Demucs, a hybrid spectrogram /waveform separation model using Transformers. It is based on Hybrid Demucs (also provided in this repo) with the innermost layers are replaced by a cross-domain Transformer Encoder. This Transformer uses self-attention within each domain, and cross-attention across domains.


ISTFT
T/1024 time steps, 2048 freq.
T time steps.
ZDecoder$_1$(C_{in} = 48, C_{out} = 4 · 2 · 2)
TDecoder$_1$(C_{in} = 48, C_{out} = 4 · 2)
T/1024 time steps, 512 freq.
T/4 time steps
ZDecoder$_2$(C_{in} = 96, C_{out} = 48)
TDecoder$_2$(C_{in} = 96, C_{out} = 48)
T/1024 time steps, 128 freq.
T/16 time steps
ZDecoder$_3$(C_{in} = 192, C_{out} = 96)
TDecoder$_3$(C_{in} = 192, C_{out} = 96)
T/1024 time steps, 32 freq.
T/64 time steps
ZDecoder$_4$(C_{in} = 384, C_{out} = 192)
TDecoder$_4$(C_{in} = 384, C_{out} = 192)
T/1024 time steps, 8 freq.
T/256 time steps
Cross-Domain Transformer Encoder
T/1024 time steps, 8 freq.
T/256 time steps
ZEncoder$_4$(C_{in} = 192, C_{out} = 384)
TEncoder$_4$(C_{in} = 192, C_{out} = 384)
T/1024 time steps, 32 freq.
T/64 time steps
ZEncoder$_3$(C_{in} = 96, C_{out} = 192)
TEncoder$_3$(C_{in} = 96, C_{out} = 192)
T/1024 time steps, 128 freq.
T/16 time steps
ZEncoder$_2$(C_{in} = 48, C_{out} = 96)
TEncoder$_2$(C_{in} = 48, C_{out} = 96)
T/1024 time steps, 512 freq.
T/4 time steps
ZEncoder$_1$(C_{in} = 2 · 2, C_{out} = 48)
TEncoder$_1$(C_{in} = 2, C_{out} = 48)
T/1024 time steps 2048 freq.
T time steps
STFT

Source: https://github.com/facebookresearch/demucs

- Open-Unmix: To perform separation into multiple sources, Open-unmix comprises multiple models that are trained for each particular target. While this makes the training less comfortable, it allows great flexibility to customize the training data for each target source.

  Each Open-Unmix source model is based on a three-layer bidirectional deep LSTM. The model learns to predict the magnitude spectrogram of a target source, like vocals, from the magnitude spectrogram of a mixture input. Internally, the prediction is obtained by applying a mask on the input. The model is optimized in the magnitude domain using mean squared error.



Source: https://github.com/sigsep/open-unmix-pytorch

- Discovery

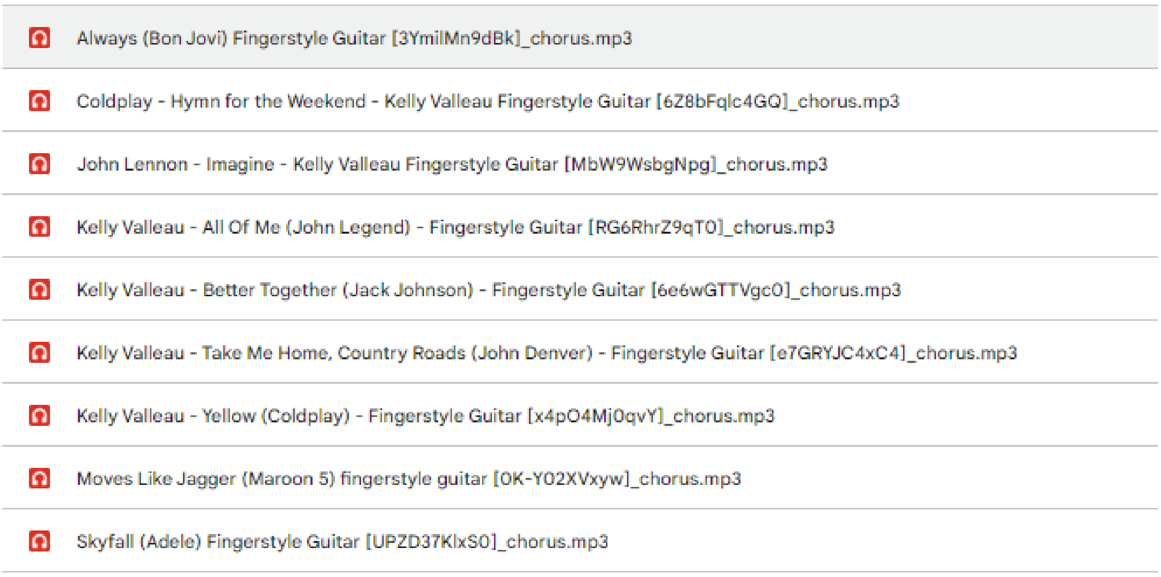
  By comparing the two models, in general Demucs model performs well in separating vocal track. The vocal track from Demucs sounds more natural and clean. On the other hand, Open-Unmix model performs better in separating bass tracks. The bass track from Open-Unmix sounds more natural without distortion and drum beats.

- Shortcomings and Future Work:

  The major shortcoming is the separation of bass track. The reason may be that bass can sound very different in different songs. Some may sound very similar to piano or guitar. So it is understandable that the model cannot separate the audio well. Secondly, bass usually follows closely with drum, most of the time on the same beats. With both of them blending together, it is even difficult for humans to hear the pitch of bass, not to mention the model. Similarly, in drum tracks we can often hear the sound of bass.

III. Bass and Melody Separation for Fingerstyle Music

- Input: Chorus (audio files)



Always (Bon Jovi) Fingerstyle Guitar [3YmilMn9dBk]_chorus.mp3

Coldplay - Hymn for the Weekend - Kelly Valleau Fingerstyle Guitar [6Z8bFqlc4GQ]_chorus.mp3

John Lennon - Imagine - Kelly Valleau Fingerstyle Guitar [MbW9WsbgNpg]_chorus.mp3

Kelly Valleau - All Of Me (John Legend) - Fingerstyle Guitar [RG6RhrZ9qT0]_chorus.mp3

Kelly Valleau - Better Together (Jack Johnson) - Fingerstyle Guitar [6e6wGTTVgc0]_chorus.mp3

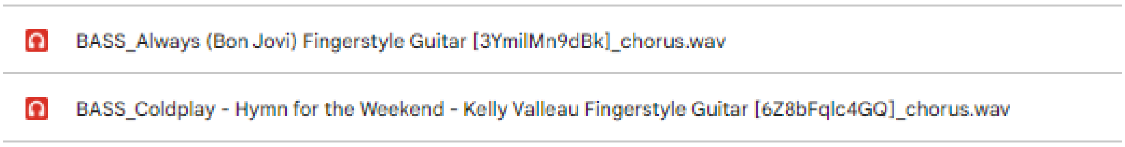
Kelly Valleau - Take Me Home, Country Roads (John Denver) - Fingerstyle Guitar [e7GRYJC4xC4]_chorus.mp3

Kelly Valleau - Yellow (Coldplay) - Fingerstyle Guitar [x4pO4Mj0qvY]_chorus.mp3

Moves Like Jagger (Maroon 5) fingerstyle guitar [0K-YO2XVxyw]_chorus.mp3

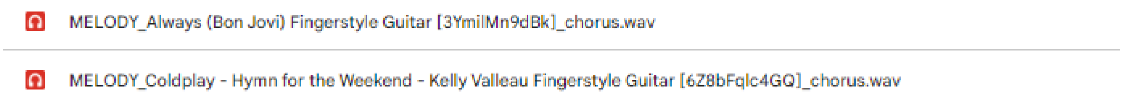
Skyfall (Adele) Fingerstyle Guitar [UPZD37KlxS0]_chorus.mp3

- Output: separation bass and melody two audio file
  - STFT- based approach : output file name BASS and MELODY



BASS_Always (Bon Jovi) Fingerstyle Guitar [3YmilMn9dBk]_chorus.wav

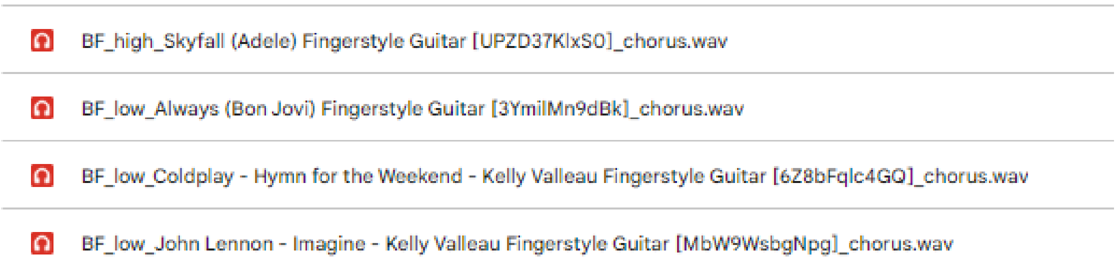
BASS_Coldplay - Hymn for the Weekend - Kelly Valleau Fingerstyle Guitar [6Z8bFqlc4GQ]_chorus.wav



MELODY_Always (Bon Jovi) Fingerstyle Guitar [3YmilMn9dBk]_chorus.wav

MELODY_Coldplay - Hymn for the Weekend - Kelly Valleau Fingerstyle Guitar [6Z8bFqlc4GQ]_chorus.wav

- Butterworth filter approach : output file name BF_low and BF_high



BF_high_Skyfall (Adele) Fingerstyle Guitar [UPZD37KlxSO]_chorus.wav

BF_low_Always (Bon Jovi) Fingerstyle Guitar [3YmilMn9dBk]_chorus.wav

BF_low_Coldplay - Hymn for the Weekend - Kelly Valleau Fingerstyle Guitar [6Z8bFqlc4GQ]_chorus.wav

BF_low_John Lennon - Imagine - Kelly Valleau Fingerstyle Guitar [MbW9WsbgNpg]_chorus.wav

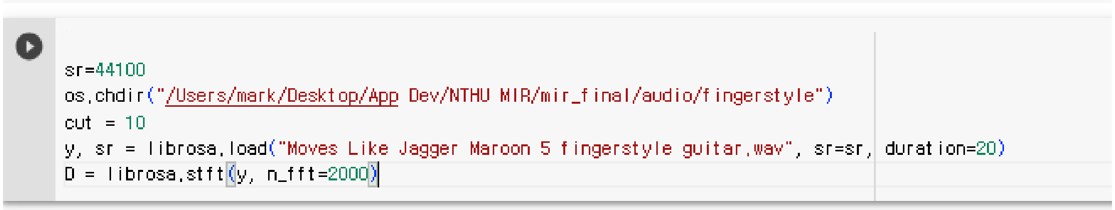
- Method:
  - Frequency separation - STFT-based approach

    Overall, this approach using the STFT and cutoff frequency cut allows you to selectively extract and separate low-frequency and high-frequency components from the original signal

  - Filter precision - Butterworth filter approach

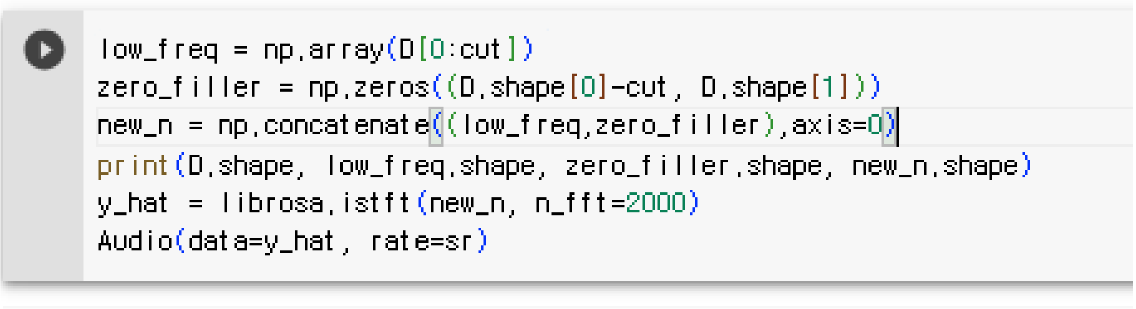
    The Butterworth filter is designed to have a maximally flat magnitude response in the passband, which means it introduces minimal distortion to the amplitude of the signal within the desired frequency range. It achieves this flatness by minimizing variations or ripples in the passband. Also, you can proceed with the low-frequency band cutoff setting and the high-frequency band cutoff setting separately.

- Algorithm Overview:
  - STFT-based approach



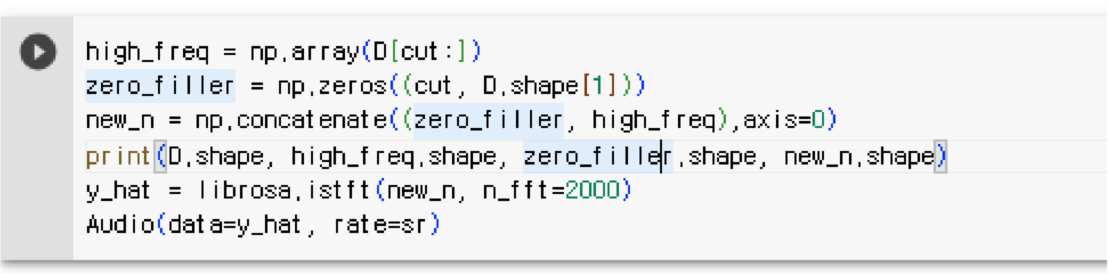
```
sr=44100
os.chdir("/Users/mark/Desktop/App Dev/NTHU MIR/mir_final/audio/fingerstyle")
cut = 10
y, sr = librosa.load("Moves Like Jagger Maroon 5 fingerstyle guitar.wav", sr=sr, duration=20)
D = librosa.stft(y, n_fft=2000)
```

Set the sample rate to 44100. Set the cutoff to 10 (approximately 10 to 12). Load the audio file using librosa.load. And use librosa.stft to compute the STFT of the audio.



```
low_freq = np.array(D[0:cut])
zero_filler = np.zeros((D.shape[0]-cut, D.shape[1]))
new_n = np.concatenate((low_freq,zero_filler),axis=0)
print(D.shape, low_freq.shape, zero_filler.shape, new_n.shape)
y_hat = librosa.istft(new_n, n_fft=2000)
Audio(data=y_hat, rate=sr)
```

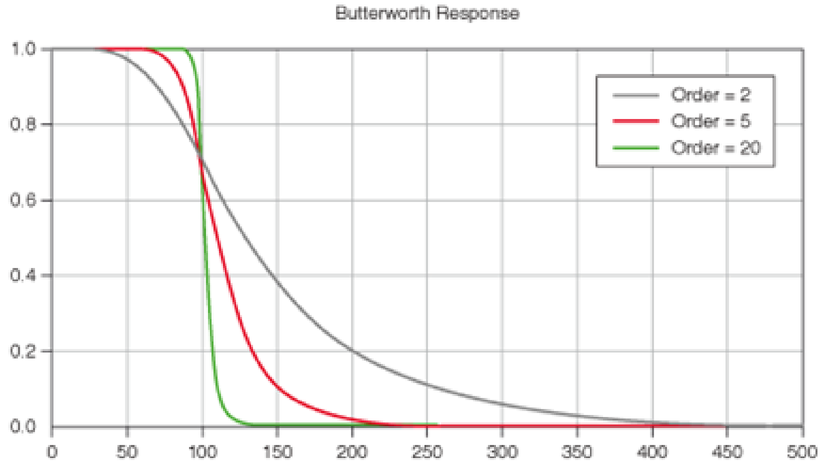
The low-frequency component (low_freq) of the STFT is extracted by slicing (based on cutoff) the 'D' array. Then create an array filled with zeroes. Then connect the two components to get the low frequency (bass).



```
high_freq = np.array(D[cut:])
zero_filler = np.zeros((cut, D.shape[1]))
new_n = np.concatenate((zero_filler, high_freq),axis=0)
print(D.shape, high_freq.shape, zero_filler.shape, new_n.shape)
y_hat = librosa.istft(new_n, n_fft=2000)
Audio(data=y_hat, rate=sr)
```

The high frequency part is similar to the low frequency part described above. (Contrary, the high_freq variable sets the remainder from the array cutoff.

- ○ Butterworth filter approach



$$|H_a(j\Omega)|^2 = 1/\left\{1+\left(\frac{\Omega}{\Omega_c}\right)^{2N}\right\} \qquad (10)$$

First, try differentiating the expression of the Butterworth filter

$$|H_a(j\Omega)| = \left[1+\left(\frac{\Omega}{\Omega_c}\right)^{2N}\right]^{1/2} \qquad (11)$$

Second, the original expression of the Butterworth filter was summarized.

$$\frac{d}{d\Omega}|H_a(j\Omega)| = \left[-\frac{1}{2} + \left(\frac{\Omega}{\Omega_c}\right)^{-3/2}\right]\left[2N\left(\frac{\Omega}{\Omega_c}\right)^{2N-1}\right]\left(\frac{1}{\Omega_c}\right) \tag{12}$$

$$\therefore \left|\frac{d}{d\Omega}|H_a(j\Omega)|\right|_{\Omega=0} = 0 \tag{13}$$

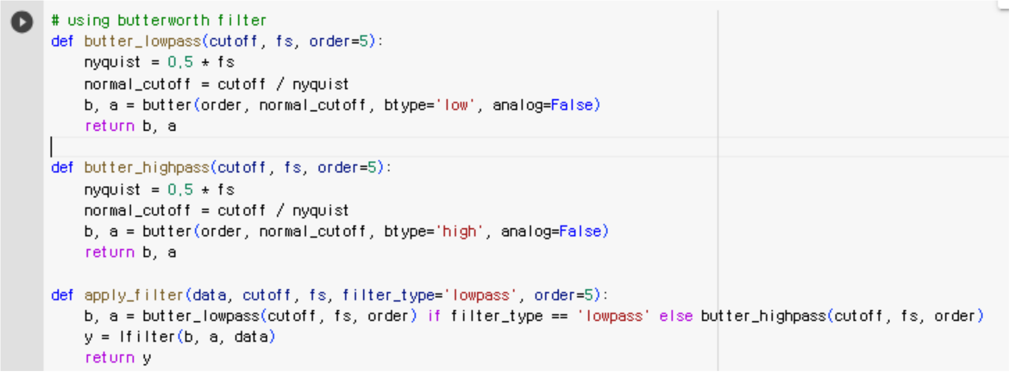
(It can be seen that it is flat, and it can be seen that it is maximally flat because it is all effectively 0 even when it is continuously differentiated.)

$$|H_a(j\Omega)| = 1/\sqrt{1+\left(\frac{\Omega}{\Omega_c}\right)^{2N}} \tag{14}$$

when Ω = 0(13), and then try differentiating it when Ω = ∞ (14)to find out how the slope changes at both ends.

$$\lim_{\Omega\to\infty} |H_a(j\Omega)| = -20N\log_{10}\left(\frac{\Omega}{\Omega_c}\right) \tag{15}$$

That is, as an asymptote proportional to the value of -20N, it can be considered to take the form of gradually decreasing. Also, for N, it can be seen that the larger the size of N, the faster the size of the denominator increases, so that the transition drops very steeply and approaches the shape of the ideal filter.



```
# using butterworth filter
def butter_lowpass(cutoff, fs, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

def butter_highpass(cutoff, fs, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return b, a

def apply_filter(data, cutoff, fs, filter_type='lowpass', order=5):
    b, a = butter_lowpass(cutoff, fs, order) if filter_type == 'lowpass' else butter_highpass(cutoff, fs, order)
    y = lfilter(b, a, data)
    return y
```

Butterworth lowpass filter and high pass filter function The cut off variable is the desired cutoff frequency for low frequencies. The filter order was set to 5. The apply_filter function applies a low-pass or high-pass filter to the input data.

- Shortcomings and Future Work:

  At first, we approached the frequency separation (STFT-based cutoff) method. A cutoff parameter was set to differentiate between low and high frequencies. However, it is necessary to set different values for the cutoff parameter according to the sound quality or component of the sound source file. However, there is a condition to this approach, which is that "high and low frequencies are separated." However, it is not clearly classified in all sound source files, and in the case of low-pitched songs, it was impossible to distinguish between melody and bass.

Another way was to find a filter method. Unlike the previous approach, the Butterworth filter made it possible to classify melody and bass more precisely, as each parameter for the cutoff for low frequencies and the cutoff for high frequencies could be specified separately, even if the low and high frequencies were in the same sound range.

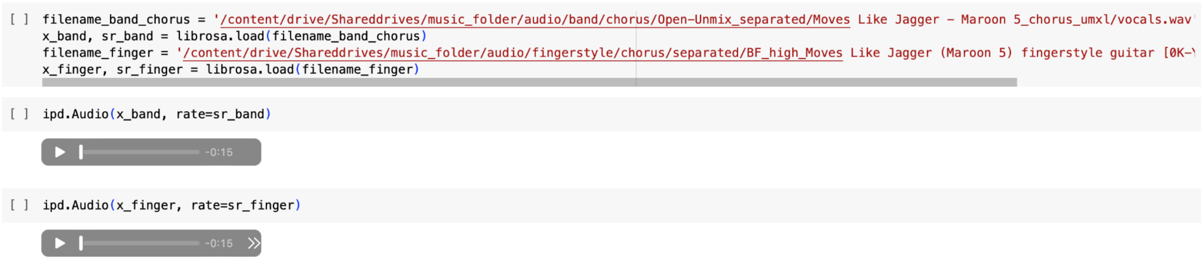
- Conclusion:

  Both the STFT-based approach and the Butterworth filter approach provide methods for separating the signal into different frequency components, enabling the distinction between melody and bass. The STFT-based approach offers a more flexible and customizable way to analyze the time-frequency characteristics of the signal, allowing for a detailed understanding of its frequency content over time. On the other hand, the Butterworth filter approach provides a more direct and straightforward separation of low-frequency and high-frequency components using a well-defined filter design.

  The choice between these approaches depends on the specific requirements of the application and the desired level of control over the frequency separation. The STFT-based approach may be more suitable for applications where detailed time-frequency analysis is needed, such as music transcription or audio effects processing. The Butterworth filter approach may be more appropriate when a simpler and efficient classification of low-frequency and high-frequency components is sufficient for the task at hand.

IV. Music Transcription

- Input: mp3 or wav types of audio file

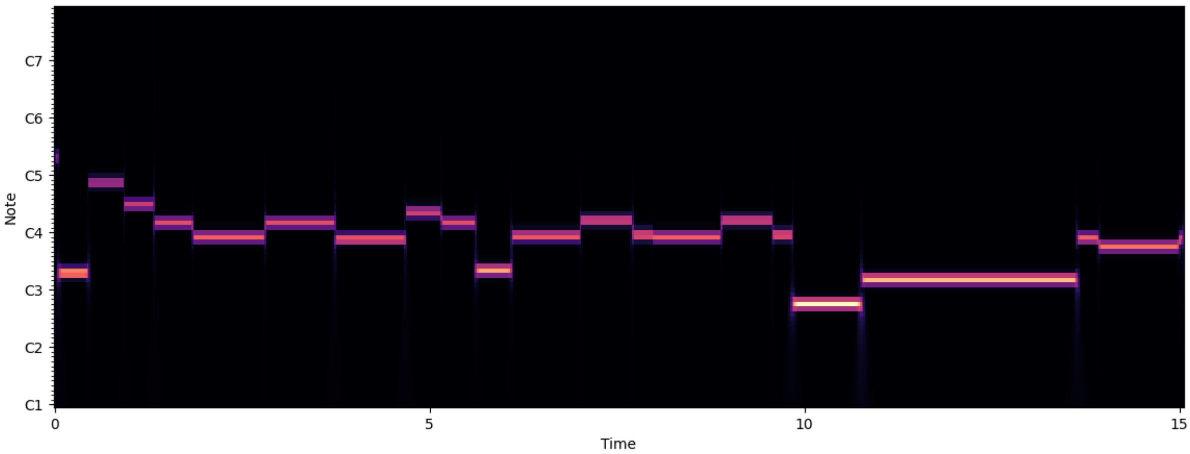


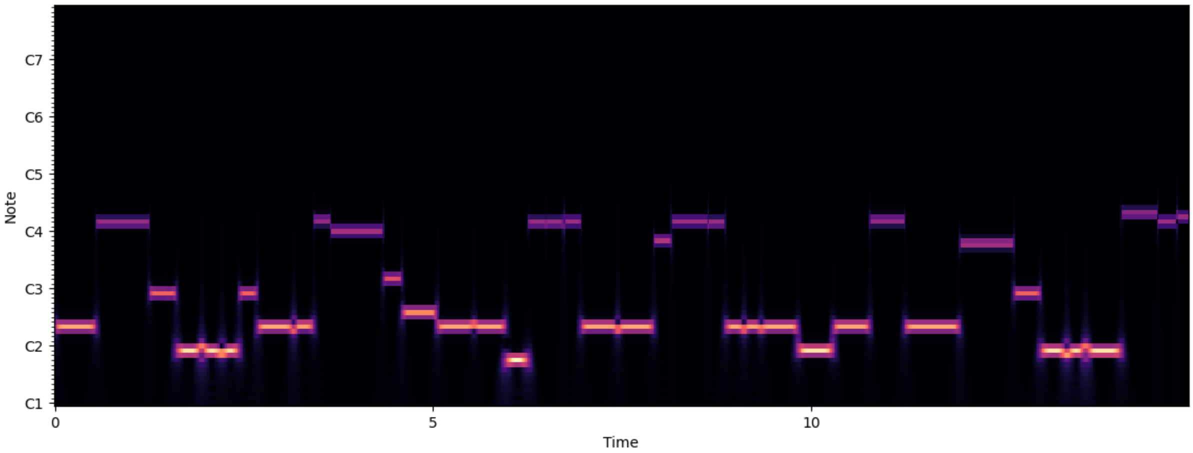
```
[ ] filename_band_chorus = '/content/drive/Shareddrives/music_folder/audio/band/chorus/Open-Unmix_separated/Moves Like Jagger - Maroon 5_chorus_umxl/vocals.wav'
    x_band, sr_band = librosa.load(filename_band_chorus)
    filename_finger = '/content/drive/Shareddrives/music_folder/audio/fingerstyle/chorus/separated/BF_high_Moves Like Jagger (Maroon 5) fingerstyle guitar [0K-
    x_finger, sr_finger = librosa.load(filename_finger)

[ ] ipd.Audio(x_band, rate=sr_band)

[ ] ipd.Audio(x_finger, rate=sr_finger)
```

- Output: pitch map





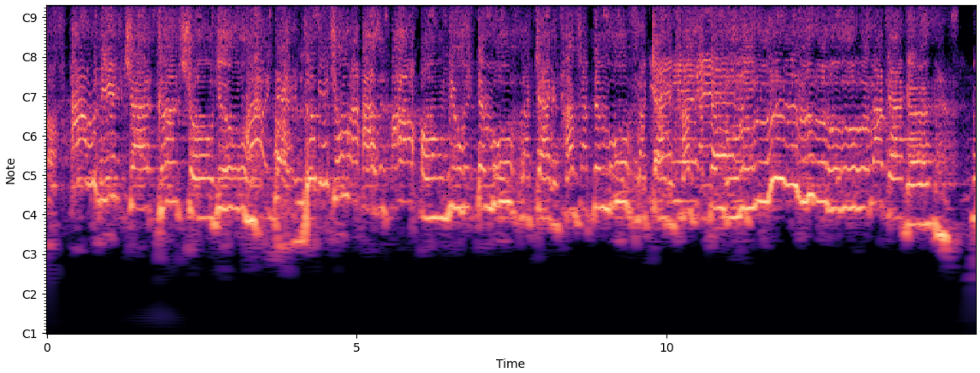
- Method:

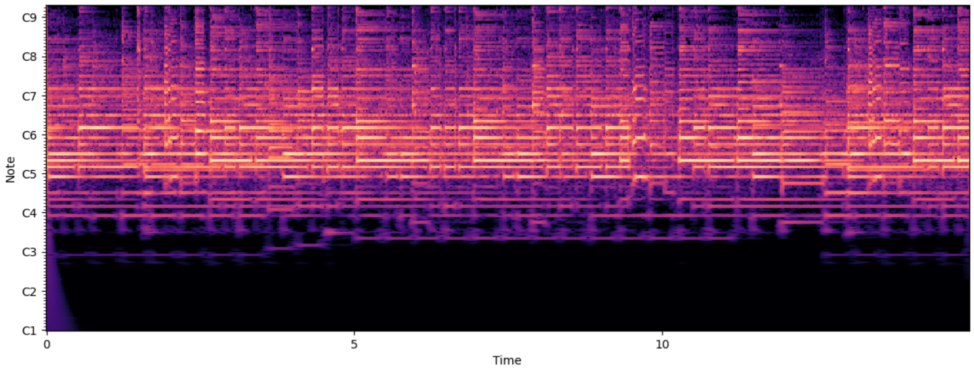
  I utilized various techniques in music analysis using librosa, including printing the spectrum, pitch analysis using librosa.cqt, and beat analysis using librosa.onset_detect.

- Algorithm Overview:

  1. Spectrum Analysis:

  By using librosa's functions, I extracted the spectrum of the audio recording. The spectrum represents the distribution of frequencies present in the audio signal. Printing the spectrum allowed me to visualize the energy distribution across different frequencies. This information can be useful for understanding the overall spectral characteristics of the music and identifying prominent frequency components.





2. Pitch Analysis:

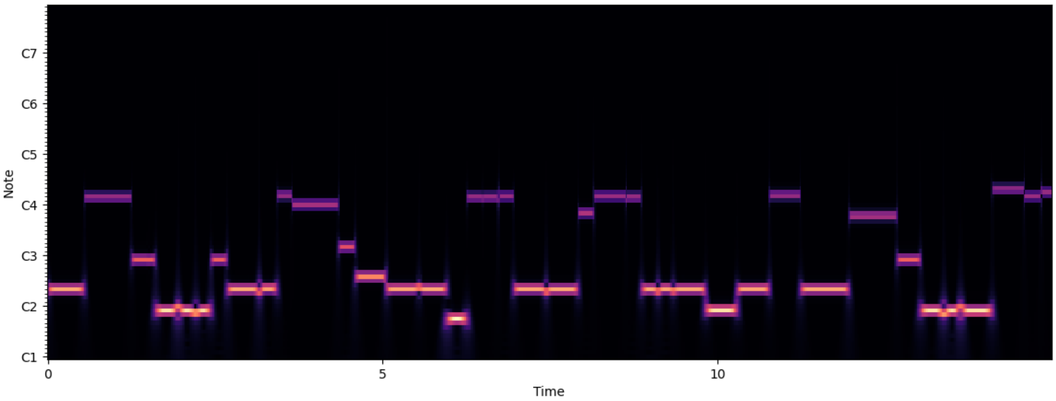
To analyze the pitch of the audio recording, I employed librosa's function librosa.cqt (Constant-Q Transform). CQT is a time-frequency representation that provides a logarithmically spaced frequency axis, which closely resembles the human auditory system's perception of pitch. By applying CQT to the audio recording, I obtained a spectrogram-like representation with pitch information. This allowed me to identify the pitch content of the music at different time frames, which is essential for music transcription.

3. Beat Analysis:

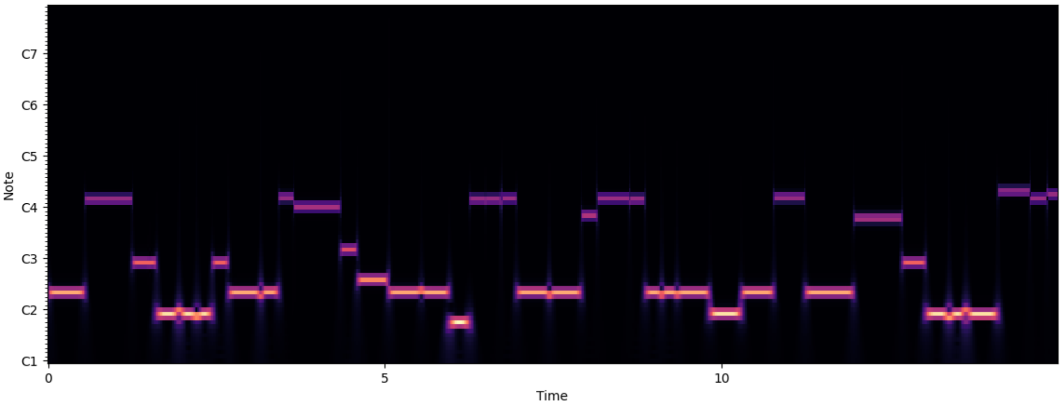
For beat analysis, I utilized librosa's function librosa.onset_detect. This function detects the onset events, which correspond to the beginning of musical beats or rhythmic changes in the audio recording. By analyzing the detected onsets, I could identify the underlying rhythmic structure of the music. This information is crucial for determining the tempo and aligning the transcribed notes with the correct timing.

By combining these techniques and utilizing librosa's powerful audio analysis functions, I was able to extract valuable information from the audio recording, such as the spectrum, pitch content, and beat structure. These insights can greatly assist in the accurate transcription of music and provide a solid foundation for further analysis and interpretation.

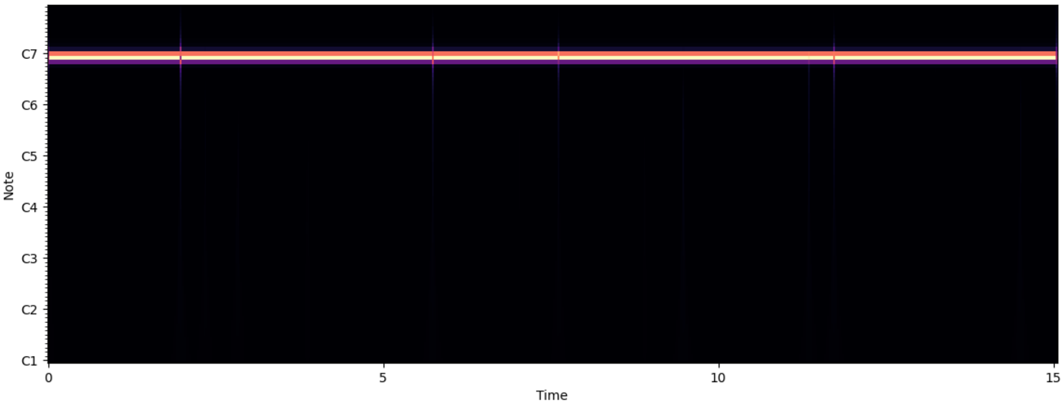
example: moves like jagger results



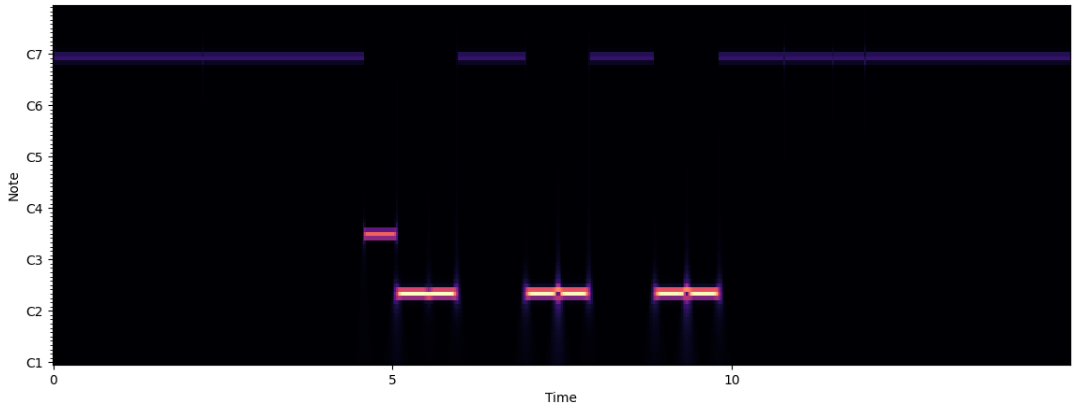
(origianl-demucs-vocal)



(fingerstyle-melody)



(open-unmix-bass)



(fingerstyle-bass)

- Discovery:

First let’s see the melody part:

From the above results, it can be seen that the vocal pitch map generated by using the demucs method on the original song fragment has a very clear melody. However, the melody pitch map obtained from the fingerstyle audio will have some fixed rhythm monotones mixed into the melody. Therefore, the generated map will also include a portion of the rhythm at the bottom, making the comparison between the two images quite evident.

Then we see the results of bass part:

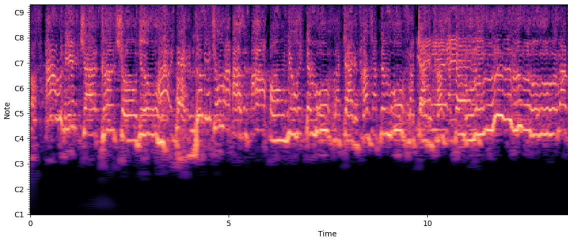
From the results obtained, it can be observed that there are some differences in the generated bass pitch maps between the original song fragment generated using the open-unmix method and the separated bass from the fingerstyle audio. While the fingerstyle audio still resembles the original audio to some extent, there are slight discrepancies in the judgments, which are reflected in the generated pitch maps.

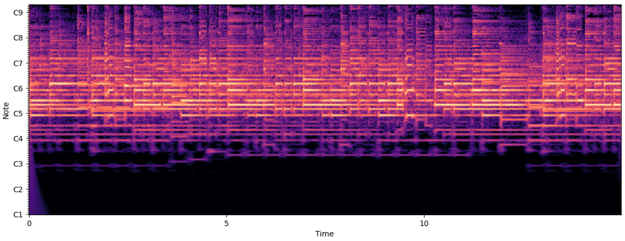
In comparison to the melody, the bass section's pitch map does not exhibit a clear melodic pattern. Although we can perceive pitch variations by listening to it, we suspect that the low frequency range makes it difficult to discern individual pitches, resulting in most of the identified pitches being displayed at the same pitch level.

More details in process:

Previously, it was mentioned that before generating the pitch maps, we use librosa.specshow to visualize the spectrum of the audio file and observe the spectral differences. Interestingly, in the case of the fingerstyle audio, we found that its spectrum is segmented cleanly.

These two diagrams represent the vocal and melody parts of "Moves Like Jagger." On the left, we can see that the spectrum is similar to some of the spectra generated in previous assignments. However, on the right side, the lower half of the diagram is completely devoid of patterns. A similar situation can also be observed in the bass spectrum.





- Shortcomings and Future Work:

Shortcomings:

While the current methods and techniques used for music transcription and analysis have shown promising results, there are still some shortcomings that can be addressed:

1. Robustness to Variations:

The existing algorithms may struggle with variations in music styles, instrumentation, and performance techniques. When faced with complex musical passages, polyphonic textures or non-standard tuning systems, these algorithms may not perform up to the mark. It is crucial to enhance their robustness in handling such variations; this area requires attention from researchers as a potential scope for future work.

2. Accuracy in Note Detection:

Note detection in music transcription can still be challenging, especially when dealing with overlapping or rapidly changing notes. Improving the accuracy of note detection algorithms and reducing false positives or false negatives is crucial for achieving more precise transcriptions.

Future Work:

To further enhance music transcription and analysis, several areas of future work can be explored:

1. Deep Learning Techniques:

Continuing research in deep learning approaches can lead to advancements in music transcription algorithms. Exploring different network architectures, training strategies, and incorporating additional contextual information can help improve the accuracy and robustness of the models.

2. Integration of Contextual Information:

Incorporating musical context, such as harmonic analysis, chord recognition, and phrase segmentation, can provide a richer representation of the music and aid in more accurate transcription. Developing algorithms that can effectively leverage such contextual information is an area worth exploring.

3. Multimodal Approaches:

Integrating multiple modalities, such as audio and symbolic representations (e.g., sheet music), can lead to more comprehensive music transcription systems. Combining audio analysis with symbolic information can help improve the accuracy of transcription and bridge the gap between audio and notation.

4. Real-time Transcription:

Developing real-time music transcription systems that can process and transcribe music in real-time opens up possibilities for live performance applications, music education, and interactive music experiences.For future research, it could be fascinating to create algorithms that can handle streaming audio and offer timely transcriptions.

By addressing the present flaws and exploring novel avenues of enquiry, researchers can continue advancing the field of music transcription and analysis. This eventual advancement will lead to more accurate and comprehensive systems for interpreting and comprehending music.

- Conclusion:

We utilized the librosa library and its functions to generate spectrograms and analyze pitch in music. Librosa proved to be a powerful tool for these tasks, offering a range of functions that facilitated our analysis.

By using librosa.specshow, we were able to visualize the spectrograms of audio files, allowing us to observe the frequency content and spectral differences between different music pieces. This visual representation provided valuable insights into the structural characteristics of the music.

Furthermore, we employed librosa functions such as librosa.onset_detect and librosa.cqt to analyze pitch and identify the melodic aspects of the music. These functions allowed us to detect the onset times of musical events and extract the pitch information accurately.

The results demonstrated that librosa functions were effective in capturing the pitch variations in the music and providing valuable information for further analysis. The pitch maps helped analyze various audio sources and revealed differences in melodic patterns, highlighting the distinctiveness of each piece.

However, it's important to recognize that pitch analysis accuracy can be influenced by factors like background noise, audio recording quality, and complexity of music. Adjusting parameters of librosa functions to optimize pitch detection for specific characteristics can improve accuracy.

In conclusion, librosa functions offer a reliable and versatile approach to generate spectrograms and analyze pitch in music. They provide valuable tools for studying the spectral content and melodic aspects of audio recordings. By leveraging the capabilities of librosa and exploring its functionalities further, researchers and music enthusiasts can gain deeper insights into music composition, performance, and analysis.

Conclusion

This work involves several signal processing techniques, including music segmentation (chorus extraction), source separation, signal filtering and music transcription. Through the use of different algorithms and techniques, we explore the pros and cons of these algorithms, allowing for a deeper understanding and analysis of fingerstyle music compositions.

We found that there’s no single algorithm that dominates in music information retrieval like BERT model in NLP. For example, when conducting source separation, we have tried out Demucs, which is known to be the most successful source separation tool available. However, we found that even Demucs has its strengths and limitations. Demucs excels in processing specific types of music, while encountering difficulties in other contexts. Therefore, choosing the appropriate method and tool requires considering the characteristics, objectives, and requirements of the music being analyzed.

We have also found that regardless of the method used, there are common challenges and limitations, such as the accuracy of note detection and adaptability to variations and contexts in music transcription. Future research can delve into various avenues such as deep learning techniques so as to come close to human-level judgment.

In summary, the study of fingerstyle music helps us understand how music segmentation, source separation, signal filtering and music transcription are implemented in code. Furthermore, music transcription involves onset detection and pitch detection, which we explored in homework separately, and now come together. We hope that this study helps future researchers understand how to approach fingerstyle music composition. Finally, they are able to enhance the precision of our work.

By knowing the difference between fingerstyle music composition and their original music, we hope that in the future human beings are able to develop a model that can automatically compose fingerstyle music. As a result, even guitar players who are not professional composers can easily generate fingerstyle music from any music he/she likes.