Documentation

Topic: Creating an optimized library for robust music analysis - including note recognition

Topic Description:
Throughout the semester we've covered many different techniques that can be used for music analysis. My goal in this project is to compile these, improve implementation efficiency, and make them more robust to the noise that is inherent to musical data. I aim to form a mini-library of sorts that I - or anyone else on the internet - will be able to use easily without needing to re-read the documentation over and over again.

Approaches Taken:

a) Library Design
   i) Object Oriented Programming
   > library is implemented as a class called musicAnalysis, with various static methods to manipulate audio and perform important audio processing calculations.
   > the class structure allows users to deal with these processes in an abstracted manner, giving them more time and flexibility to work on their project/goal instead of spending time on mathematical implementation details.

   ii) Documentation convention
   Every method has descriptive docstrings that state what it does and what each of the parameters are.

   iii) Plot label and title options
   Whenever a method shows a plot, there are text parameters to let us change the plot title, and boolean values to auto-change the scale of the units on the x, y, or z axis (z-axis being color).

b) Frequency template creation

   When working on Lab 10, I noticed that the method template(midi, N, sr) was implemented using a nested for loop where one loop iterates through the different possible harmonics of a note, and one iterates through each frequency bin. Because of this nested loop setup, the template creation part of the notebook took longer than most other parts. Since the operations done inside the loops were simple in mathematical nature, I decided to optimize template creation by making use of numpy arrays and relying on the speed-up that vectorized numpy code gives us (mostly because numpy operations run in C use static memory allocation for arrays).

c) Calculating log-likelihood with matrix multiplication

Another thing that grabbed my attention during Lab 10 was the log-likelihood calculation method which also used a nested for loop setup. I found a way to find the exact same likelihood matrix by transforming and rearranging the STFT and the list of templates such that the dot product of two matrices formed by these gives our likelihood matrix

d) Most likely note detection

The most likely note at any frame can be chosen as the one with highest magnitude in that FFT column. Since the frequency bin number is just the column index, I was able to use the numpy function argmax to efficiently find the most likely note in each frame. Some songs don't have much duration of silence, and the inclusion of the silence template sometimes leads to the algorithm predicting a silent note where it doesn't belong. Hence, I added a boolean option for whether or not the user wants to ignore silent notes.

Looking back, I see that our algorithm treats each frame as independent, and the highest bin in each frame is counted as the most probable bin. However, I think it would be an interesting project to build some kind of algorithm that incorporates the fact that consecutive frames are unlikely to have wildly varying note frequencies. Maybe some sort of Bayesian algorithm would be apt here?

e) Removing Spikes (noise)

One bug in note detection was that there were some frames where the prediction was very inaccurate, leading to 'spikes' in the predicted_note-time graph. To reduce this error, I ran a loop that looks through each element in the array that contains the "most likely note for each frame", and I compared each note's MIDI to its adjacent notes' MIDIs. If a particular note's MIDI differed from both of its neighbors' MIDIs by a significant threshold amount, the note would be changed to the average of its neighbors, to smooth the curve out. This is a start to what I was talking about in the previous subheading about incorporating neighboring notes' information into choosing this particular note.

f) Confusion between notes differing by octaves & Chromagrams

Another common error was the confusion between notes of the same pitch class but of different octaves, since playing a note A4 on most instruments will also produce a significant wave of A5, for example. Because of this, the "most likely note" across frames would often jump from one octave because of noise and coincidental interference from other frequencies being played at the time.

I came across the concept of chromagrams, which combine all notes of the same pitch class into one band, so they contain 12 bands for each semitone from C to B. After creating a chromagram from my spectrogram (using librosa) I could easily find the notes with highest frequency since there were only 12 options and less noise in this compressed space. However, I no longer knew which 'octave' of a particular note was most dominant (whether A4 or A5 was of stronger magnitude in the frame). A future improvement would be to find some way to first identify the pitch class dominant in a

frame using the chromagram but then somehow checking which octave of that pitch class would be the most accurate debit card prediction.

g) Spectral Centroid
The spectral centroid is the expected value of the frequency in a given frame (treating the template as a probability density function)

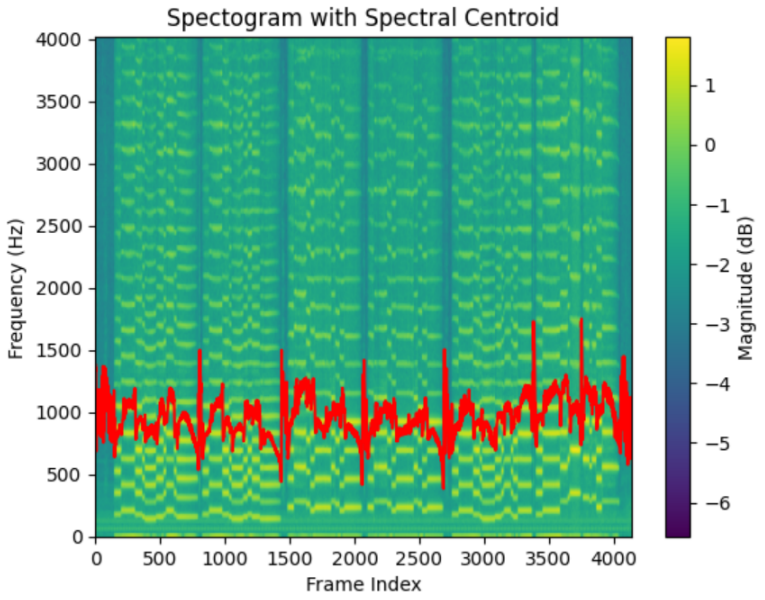
h) Multi-note recognition
A future direction to pursue would be to investigate better ways for multi-note recognition than just looking for the top N notes with highest magnitude. A possible idea that came to mind is thinking of each template as a sum of H gaussian probability density functions (or 'harmonic peaks'), and thinking of two simultaneous notes as the sum of 2 such groups of H gaussins each. I wonder if given the DFT of a frame, there's some way to find the optimal frequencies of 2 notes that we hypothesize to be playing in the frame at the moment, possibly using some sort of Maximum Likelihood Estimation.

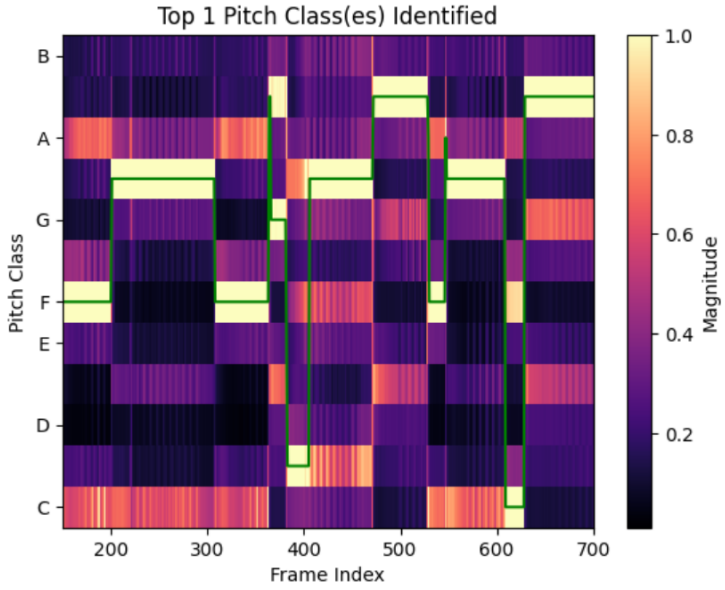
How to run this project

Package requirements: numpy, librosa, scipy, matplotlib, IPython
Setup:

1. Install necessary packages
2. Place "musicAnalysis.py", "CMSC 395 Final Project.ipynb", and the folder "Audios" all in one containing parent folder

Demonstrated results:





Tools/Libraries/Packages used:

Python
Numpy
Matplotlib
Librosa
Scipy
Jupyter
IPython