Patrick Ferry

Automatic Music Transcription Tool

Technical Manual

Student number: 15496488

Date Finished: 18/05/2019

Abstract

My project is a web application, built using Flask, in which a user can take upload a file or enter a youtube of a song they would like to transcribe to midi. There are three tools, piano, drums, and full song. Users will have the option of sectioning the song. This will tell the system to split the output into sections such as intro, chorus, verse. For the drum tool it uses a neural network to classify each drum hit then transcribe it.

Motivation **3**
Research **3**
Design **4**
High Level Design 4
Language Choice and Libraries 4
Module Design 5
Component Design 5
Database Design 6
Dataflow Design 6
Flask App Design 7
Jupyter Notebook 7
Feature extraction of data 7
Implementation **8**
Frequency binning 8
Salience function 8
Midi Output 9
Harmonic/Percussive Separation 9
Sectioning function 10
Drum Classifier 12
Problems solved **12**
Hardware & Scalability 12
Extra Noise 13
Onset Detection 13
Windowing 13
Results, Testing, and validation **14**
Unit Tests 14
Functional Tests 14
User Tests 14
Drum classifier accuracy 16
Transcription accuracy 16
Future work & Conclusion **16**

Motivation

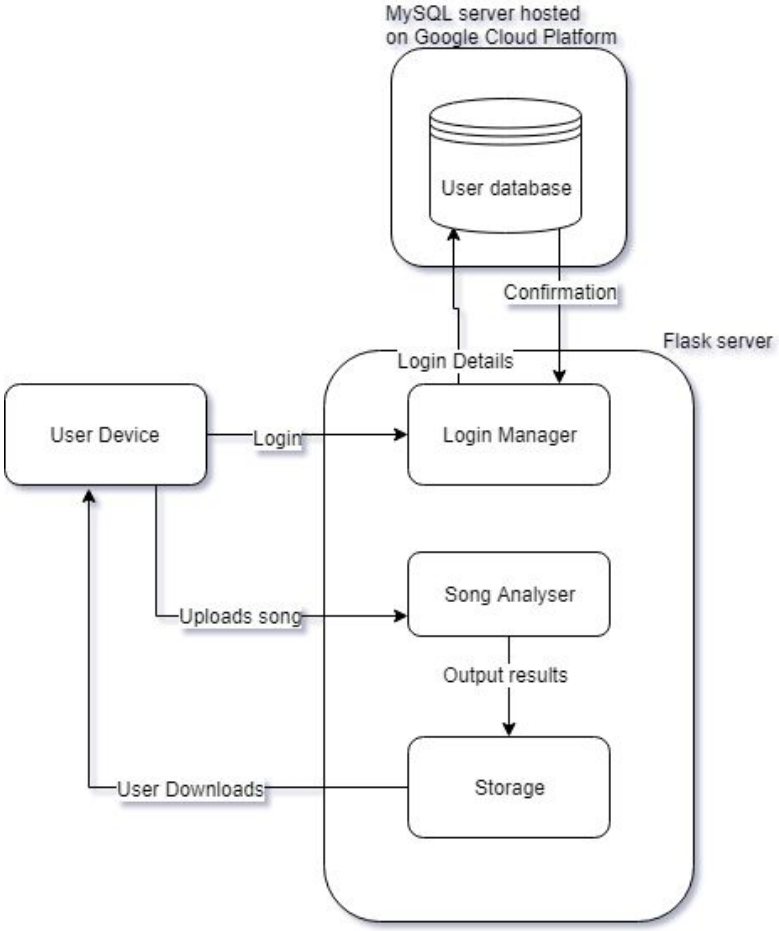
Digital signal processing is a subfield of mathematics that I had no knowledge about until I found out about the Fourier transform from a video I had watched. I began to read into it and realised that it seemed like a very interesting field and that I would like to know about it. I had been playing with digital audio workstations, such as Ableton/FL studio, for producing music and I seen that they used MIDI files as inputs for the instruments. So I decided I would try and use the Fourier transform to try and transcribe songs. The main goal was to be able to transcribe piano pieces and learn more about DSP. Using a web application as my front end was decided because I was more comfortable with that where as I have never created a desktop application with a proper GUI.

Research

I researched papers which involved onset/novelty detection, salience, and melody tracking. To first get a better understanding of some of the terms and such I read a good few of the basic chapters from 'The Scientist and Engineer's Guide to Digital Signal Processing' by Steven W. Smith. This helped me understand the papers I was previously reading much better. I then found a book called 'Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications' by Meinard Müller which contained references to almost all of the papers I had previous read but had them all summed up so it was a fantastic help. The chapters I mainly focused on were “Tempo and Beat Tracking” , “Music Structure Analysis”, and “Musically Informed Audio Decomposition“. As I began to understand how a lot of these techniques worked I realised I might need something with more detail so I went through the references in the book and I found a few papers that were a great help but the main paper that I focused on was “Melody Extraction from Polyphonic Music Signals using Pitch Contour Characteristics” by Justin Salamon and Emilia Gomez. This was the technique I used for the melody tracking part of my project. I looked into a library called Librosa which contains many tools for projects like mine but after talking to my supervisor we thought it would be best to not lean on this library and write code for these functions myself to show that I truly understood what I was doing.

Design

High Level Design



This system uses Flask web framework. There is a logins table that is being hosted on Google Cloud Platform. All output files are stored on the server. Input files are deleted after being analysed as they are large and I am unsure if it's legal to gather up a load of free music from others.

Language Choice and Libraries

I decided to choose python for this project as it is the language with which I am most comfortable and as I had a lot to learn about this domain I decided it would be best to not have to learn a new language or too many new tools at the same time. I chose Flask as my front end as it is not the most difficult framework to pick up but still gives you a lot of power. For my song analysis tools I decided not to use a library called Librosa after talking with my supervisor so I could prove that I properly understood the functions I was calling. Looking back at this choice I like to think that implementing the functions myself was definitely because some of the functions in the library were quite confusing and had many parameter which I didn't understand until I implemented the function myself. I chose Keras, a library

built on top of TensorFlow, to build my neural network as it was the similar to tutorials and such that I had looked and I thought it would be easiest to use. This project relied heavily on NumPy and SciPy. A full list of the libraries used can be found in the requirements.txt file.

Module Design

```
├── song/
        ├── beat.py
        ├── decomp.py
        ├── drum.py
        ├── midi_tools.py
        ├── read.py
        ├── spectral.py
        ├── structure.py
        └── utils.py
```

This is the set of modules which contain all the components for functionality for the process of transcription of songs. These modules get called from the flask app when they are needed. Each one of these modules contains functions relevant to their title. I organized it this way for the sake of keeping things a bit neater and also to stop me having to import every function as once as I did not always need all of them.

- beat.py
    - Novelty functions
    - Although not all functions are used. Some of them were implemented to test their accuracy.
- decomp.py
    - This module contains functions needed to decompose a song into it's harmonic and percussive elements
- drum.py
    - Functions needed for feature extraction and drum classification
- midi_tools.py
    - All functions needed output internally modified songs to midi
- read.py
    - Reading, converting, downloading files
- spectral.py
    - Functions for modifying spectrograms
- structure.py
    - Functions for analysing song structure
- utils.py
    - Small functions that may have been needed throughout other modules but were too awkward to classify

Component Design

A lot of the functions that I created for processing the music were based off of different formulas and algorithms which I found within Müller's book. Functions such as the salience function, onset detection, structure analysis, harmonic/percussive separation and many

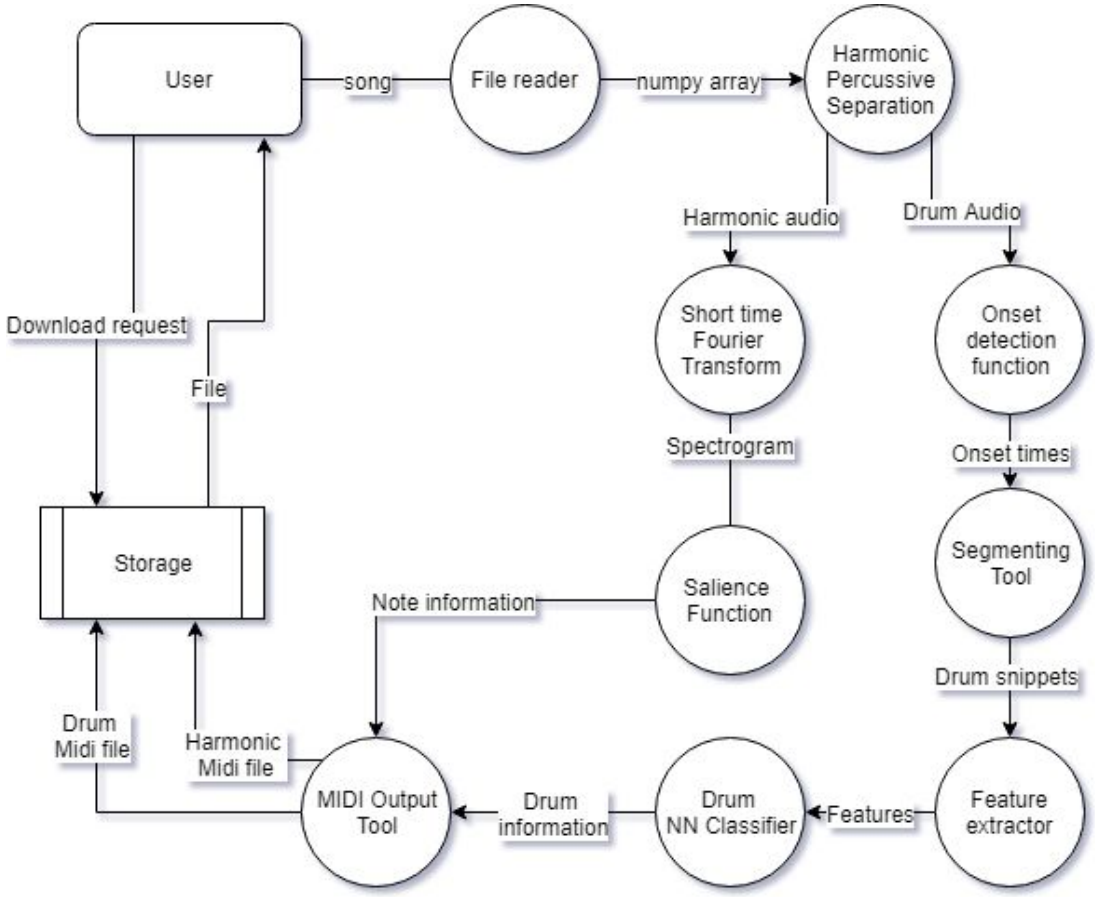
more smaller functions. These functions were written as maths formulae in the book so it was quite difficult to convert some of them into code.

Database Design

I used a database for user logins. This was a MySQL database with just one table. I decided to use this inside of just a local SQLite file if I wanted to expand and create posts. The table contains user names and hashed passwords. It is integrated using flask-sqlalchemy, which is a handy library that makes it easy to build and connect to SQL databases using flask.

Dataflow Design

Below is a data flow diagram to show the process of converting a whole song to midi. A user uploads the song and it is split into different versions of itself, one that contains the drum audio and one that contains the harmonic elements, i.e. piano/guitar. The harmonic version goes through a salience function which find the strong notes and keeps track of them to then pass it onto a function that converts it to midi. An onset detection algorithm is used on the drum audio and the audio is split into small sections each containing one onset. The features are extracted from these and then passed to the neural network to be classified. Once classified they are paired with their onset times to be output to a midi file.



Flask App Design

I created my Flask app in by creating a main app file and then building around with files such as route.py, which contained all the behaviours of each page. I used file templates with flasks built in jinja as well as Bootstrap as a CSS framework. Each tool page contains a two forms one of which takes a file input and the other which takes YouTube link. The form that takes a YouTube link has regex validators that won't allow to input a link that doesn't belong to YouTube.

Jupyter Notebook

I used jupyter notebook quite often during this project for testing out certain designs and functions as it was quicker than running a whole file over and over again. I used jupyter notebook:

- To create a dataset from the drum samples I had collected
- To create and train up my neural network
- And for testing different parameters for certain functions where I could find no clear answer online and would have to test by trial and error.

Feature extraction of data

For this process I used Pandas and this is when I decided to use Librosa as I need to extract many different features from all of the drum samples I had gathered and time was a constraint.

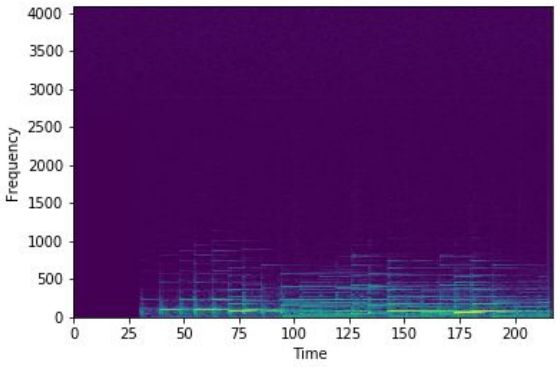
Implementation

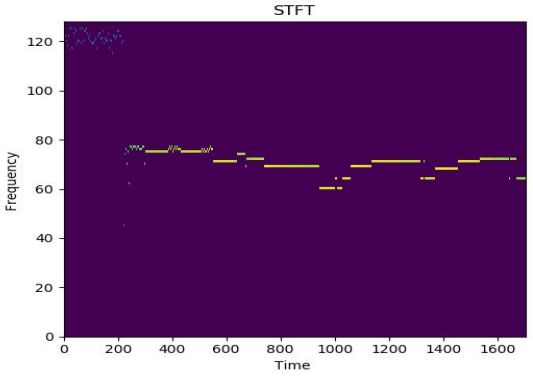
My system reads in songs using either file format or a youtube link. It will take any file, but preferable wav and if it is not a wav file then it will use FFMPEG to try and convert it. When using youtube links I used a library called PyTube to download the video/audio and convert it to wav. The wav files are read using `scipy.io.wavfile.read()`.
My song analyser works by taking a song and getting the short-time fourier transform of it. I decided on a hopsize of 512, a window size of 2048, and an fourier transform length of 8192 as these are similar numbers to what was suggested in many of the research papers on melody tracking. I used a hann window and used `scipy.signal.stft()` function.

Frequency binning

Once I had short-time fourier transformed my audio I would have a spectrogram of one frame (column) for every 512 samples in the audio and 4096 frequency bins (rows). The highest row relates to the to Nyquist frequency 22050. This is as high as humans can possibly hear. I decided to try and bin the frequencies into less amount of rows, 128 one for each note possible on a MIDI instrument. Below is the basic version of the formula I used to choose which new bin to put my values.

$$Bin(w) = \lfloor 12 \cdot log_2(\frac{w}{440}) + 69.5 \rfloor$$





Example of normal frequency domain → Example of binned frequency domain
Notice the row numbers

Salience function

The salience function described in papers I have researched relies on harmonic summation. When a note is played on a instrument the note we hear is what is called the fundamental frequency or F0 but this is not the only frequency that is present. There are other frequencies created called harmonics or overtones. This frequencies exist at double, triple, quadruple, etc. the fundamental frequency. Harmonic summation is when we take the peaks

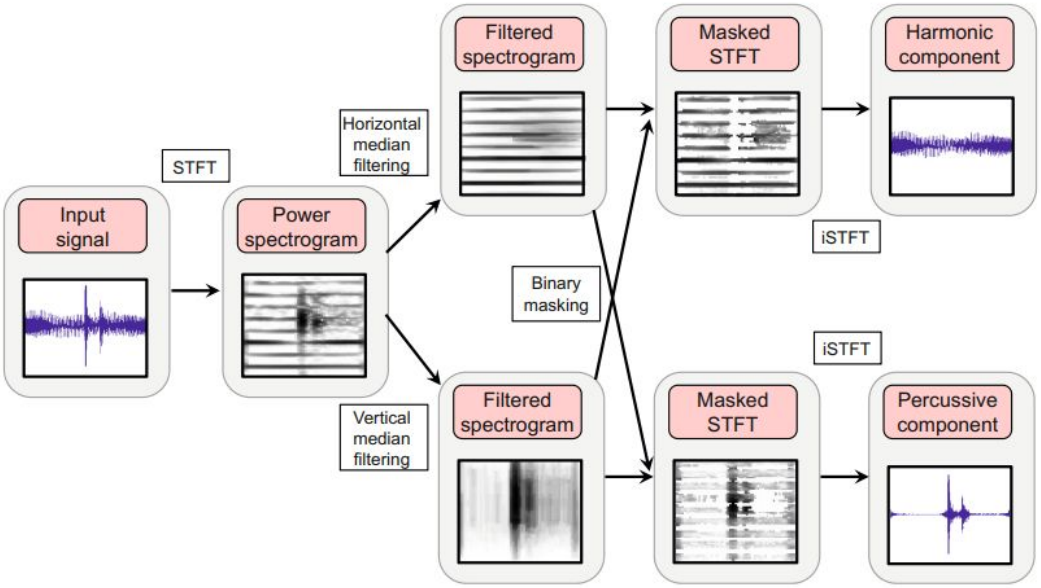
from our frame and we add the values at each harmony bin to our peak. Each harmonic has a different weight by which it is multiplied. Once this function has completed we will have a 128 frequency bin spectrogram and there will be higher values in places we think there may be a fundamental frequency. We then parse over the spectrogram to remove any weaker peaks.

```
def harmonic_summation(frame, peaks, weights):
   for peak in peaks:
       harms = np.array(([peak]*4))
       harms += np.array([12, 19, 24, 28])
       fbins = 128
       for harm, weight in zip(harms, weights):
           if harm < fbins:
               frame[peak] += frame[harm]*weight
```

Midi Output

The spectrogram only containing the supposed fundamental frequencies is passed to a function which looks through each row and enters a start and an end time into a dictionary with the note (row number) as the key. These are then passed to a function built using the midiutil library. This is a library that allows you read, and write midi files. This function will output the file to a folder under the users name and will be available for download after it it's completion. As each key in our dictionary represents a midi note it is very easy to read through it and create a note for each start and end time.

Harmonic/Percussive Separation

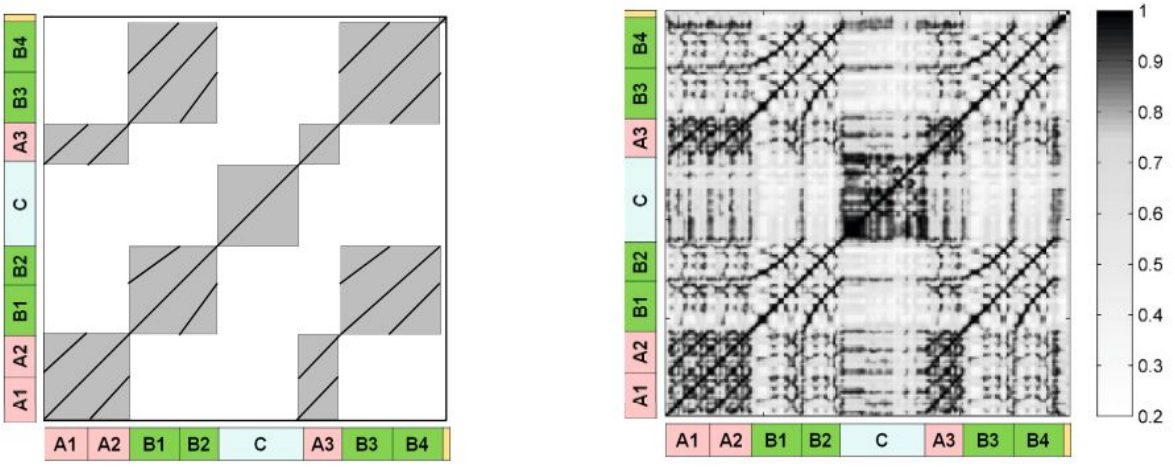


In music there are two main types of instruments, harmonics which play actual notes and melodies (guitar, piano, violin, etc) and percussive which usually play the rhythm (drums).

Once you have created a spectrogram of a piece of audio harmonic elements may be observed as horizontal structures and percussive element show as mainly vertical structures. My technique for splitting these into two different track was by using a two different types of median filters to create a mask. I found out about this technique through Derry FitzGerald's research paper "Harmonic/percussive separation using median filtering". A median filter will select the the median value in it's each of it's kernels and set all values to that value. By using a kernel that is of the shape 1x50 it will keep places in the spectrogram where there are harmonic or horizontal structures and for a kernel 50x1 it will keep only vertical structures. These new versions of the spectrogram can used to create binary masks or combined to create soft mask. These masks are then applied to the original spectrogram to create two pieces of data.

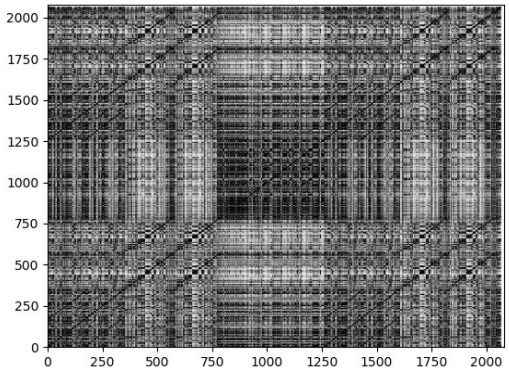
Sectioning function

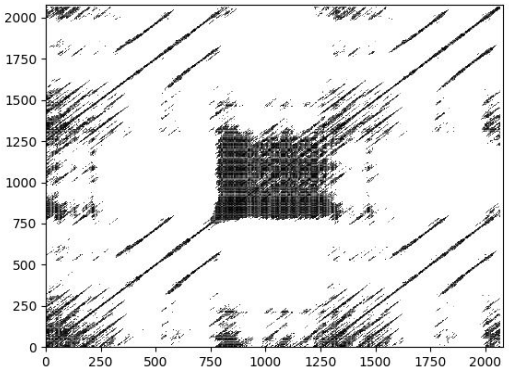
This function aims to separate audio into different sections such as chorus, or verse, etc. This all works off of the idea that there will be repeated melodies in a song where such as when the chorus repeats. I would take a spectrogram of a song and convert it into a chromagram. A chromagram is similar to my binned version of a spectrogram I mentioned earlier but instead of having 128 bins it now has 12, one for each semitone (C, C#, D, …, A, A#, B). I then created a self similarity matrix comparing every frame in this chromagram to itself and every other. This creates diagonal lines in places where the melody repeats and can show you the form of the song if there is one.



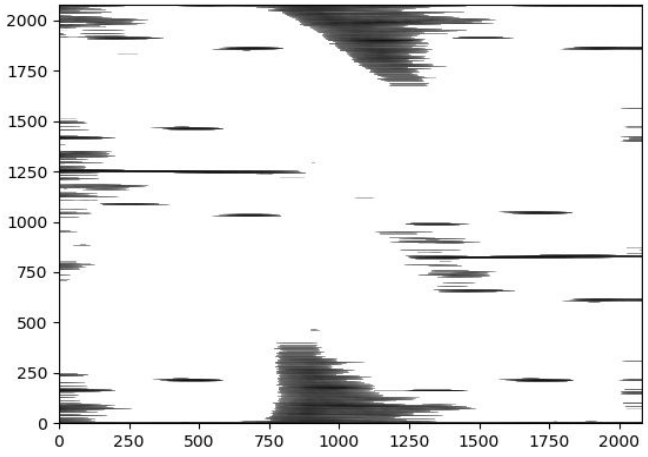
Above is an example of ideal similarity matrix (left) and a real one (right).

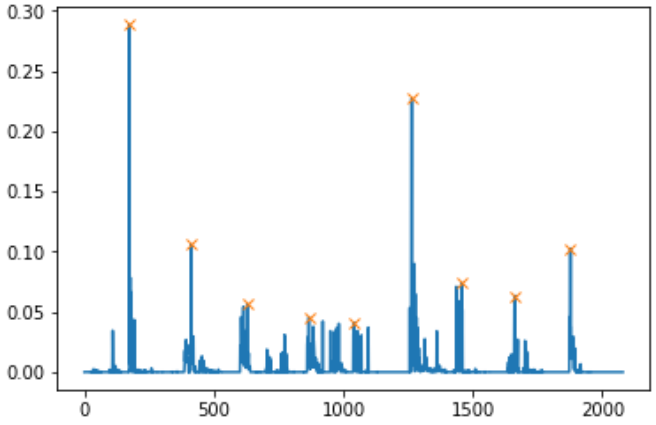
I compared each frame using cosine similarity and placed it in an n x n matrix where is the number of frames in the spectrogram. You can see the result below on the left. You can make some of the diagonal line but some are still very hard to see so I created an 2D array that was all zeros except for the main diagonal from bottom left to top right. I used this array as a footprint for a median filter and applied it to a copy of the original matrix. I removed any points left in this that were below .85 similarity and I was left with the graph below on the right.





Now that I had the diagonals I needed to find where different lines began and ended but after much brainstorming I could not think of a simple way to do it but I eventually stumbled upon a solution where you convert your matrix into a lag representation allowing you find the beginnings and endings of lines.





On the left here you can see what you get after we convert the above filtered similarity matrix to a lag matrix. We then parse through through this lag matrix column by column comparing each one to the last using `numpy.linalg.norm()` . This gives us the energy change and where there is a big increase it means it is the beginning of a new line and we mark that as a beginning of a segment. Below you can see what the energy graph looks like and the peaks that were selected.

Drum Classifier

My drum classifier is a basic neural network built with Keras. I began collecting drum samples here and there from when I began the project and then spent some time solely doing this. I collected 23,000 audio drum samples of 8 different types of drums, clap, closed high hat, open high hat, kick drum, crash, ride, snare, and tom. I searched for the best features to extract from audio to try and classify it and these are what I found

- Chroma STFT
    - This is the average value in its chromagram
- RMSE(Root Mean Square Error) spectral
- Spectral centroid
    - indicates where the "center of mass" of the spectrum. Often related to the brightness of the sound
- Spectral bandwidth
- Rolloff
- Zero crossing rate
    - Number of times a signals passes through a value of zero
- a column for the average value in every mel frequency cepstral coefficient (MFCC)
    - MFCCs helps to describe the timbre of a sound and is often used in voice recognition.

I extracted these features from each of the samples I collected. Dataset can be found here.
I created a keras sequential neural network and trained it on this dataset. I used a softmax last layer in the hopes for multi-classification for clips that contained multiple layers but because of the data I had gather and the way I was extracting features this was not very possible. I needed to gather more data for this model especially for the drums that are not kick drums, high hats, snares (as these drums are very often used it was much easier to find samples of these and I gather an uneven proportional). Once I had a model made I saved it along with the feature scaler.
I used an onset detection function to find the hits of drums in a drum clip and then I segmented each of these into smaller clips and let my neural network try to classify them. I then took each of these descriptions along with it's related onset time and converted it to midi. Unfortunately due to this design and the lack of multi-classification in the model I could only transcribe one drum per onset. If I had more time I think I would have brought in a some low-pass, mid-pass, and high-pass filters to try and improve the accuracy on actual drum clips.

Problems solved

Hardware & Scalability

When I first began transcribing songs I quickly found out that some larger songs would take too much memory 8gb+ and would cause a memory overflow error. This is when I decided to look into sectioning the songs by verse, chorus, repeated melodies etc. This allowed me to break the song into chunks and not have too much memory being used at once.

Extra Noise

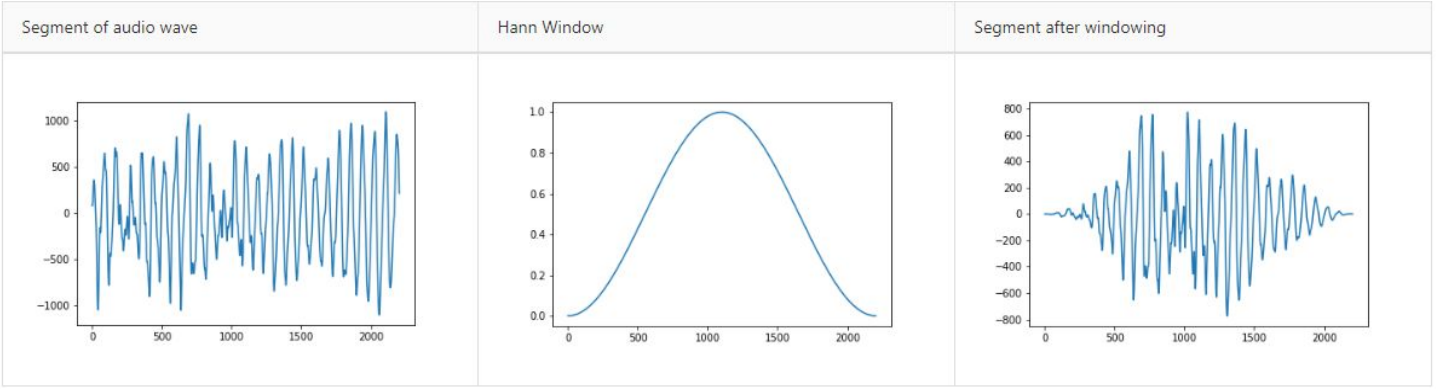
There was many very small extra notes that were coming through in my output and were caused by small pieces of noise that had high values. These small noise were very short so I created a median filter of a size bigger than the biggest piece of noise and created a mask using this to remove most of the noise.

Onset Detection

My first idea for note tracking was to look for the onsets of notes and then trying to segment each note and fourier transform that segment to find the note. I then learned that this is a bad way of trying to achieve my goal so I began to research new ideas.

Windowing

When I began the project I played around in jupyter notebook with the fourier transform and I could never get the result I expected. I was getting extra peaks everywhere and i just didn't understand it. This is when I learned about windowing. Windows the small section I was trying to analyse fixed the problem and I learned that those extra frequencies were caused because of the sharp cut off at the beginning and end of the clip which is eliminate by a window.



Results, Testing, and validation

Unit Tests

I used pytest as my testing framework. I wrote unit tests for many of my functions. I had issues in writing tests for certain functions due to the oracle problem. For example I didn't know exactly the spectrogram of a function would look like unless I used my function for creating a spectrogram so this defeated the purpose of some tests so for some of my unit tests I just ensured that the function ran successfully.

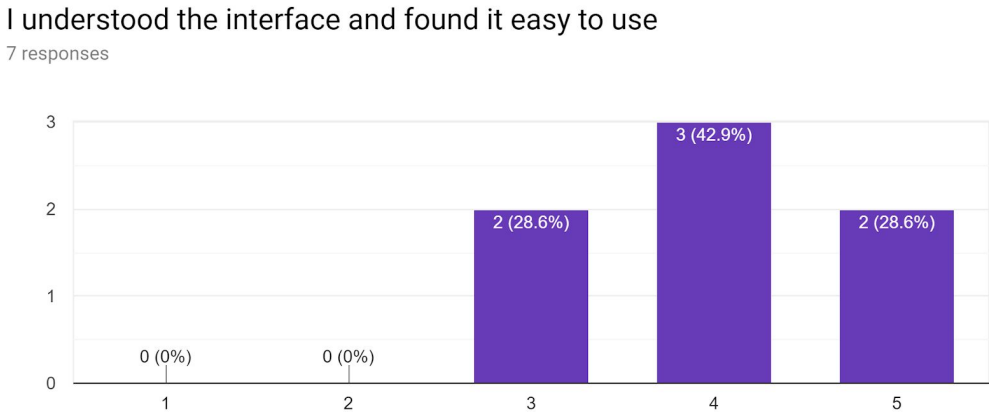
```
========================= 35 passed, 271 warnings in 57.32 seconds ==========================
```

Functional Tests

I wrote tests that check the functionality of my Flask app. I checked things such as user logins, creating an account, etc.

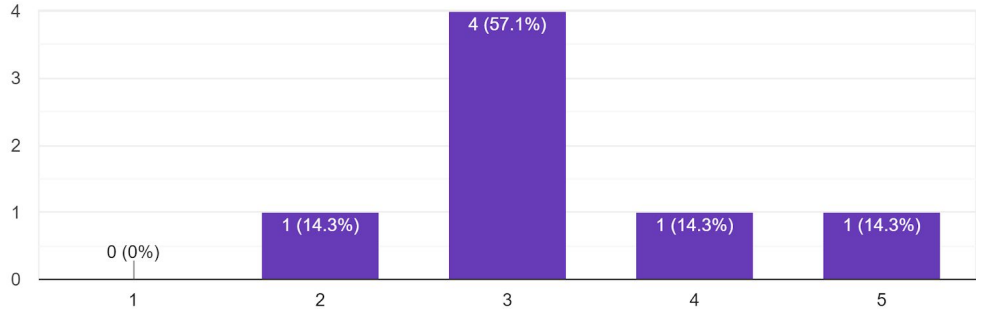
```python
def test_login(client):
    """Test user login"""
    response = login(client, "test", "test", red=False)
    assert response.status_code == 302

def test_valid_user_registration(client):
    """Test account registration"""
    # TODO: Fix this response data
    response = register(client, 'test', 'test@test.com', 'testing', 'testing')
    assert response.status_code == 200
    assert b'Congratulations, you are now a registered user!' in response.data
```

User Tests



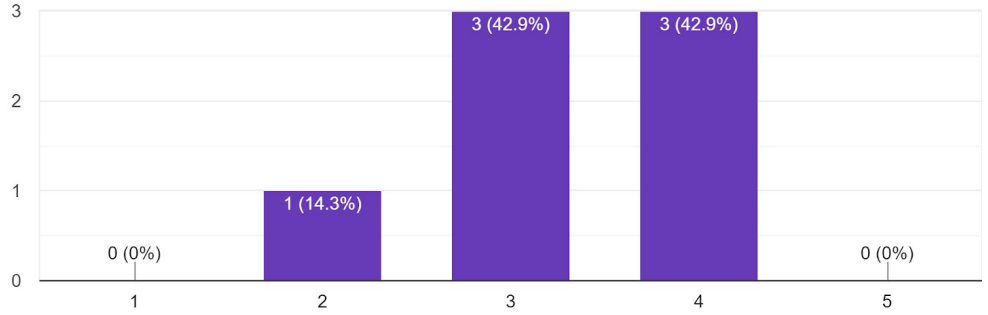
The design of the interface looked good

7 responses



How would you rate the accuracy of the output(s)?

7 responses



Can you identify any issues with the output and if so what are they?

7 responses

I could definitely hear extra notes that werent supposed to be

there were notes missing

The correct rhythm was basically there but the drums were not all the same

Too many missed notes

It was recognisable but a lot of notes were missing

you would not recognise the song in any part besides the chorus

It sounded much better than I expected

Drum classifier accuracy

I used Keras' evaluate function to check how accurate my model is. 86% was higher than I was expecting to get but I did realise that a lot of the data it is being trained on are very high quality, clean clips of audio.

```
In [15]: test_loss, test_acc = model.evaluate(X_test,y_test)
         print('test_acc: ',test_acc)

         4360/4360 [==============================] - 0s 38us/step
         test_acc:  0.8619266055045871
```

Transcription accuracy

To evaluate the the transcription accuracy of my system I search online until I found a clean, manually made midi file for a certain song. My system does not account for tempo changes and finding songs without any tempo changes proved difficult. Another issue was that I needed to know the exact version of the song so I could also put that through my system. I eventually had to settle for putting a file through my system that was not of the cleanest audio but I could not find any other options. Midi files are not comparable the way they are so I searched and found a library that converted MIDI files to csv. I transformed both midi files to follow the same format and had to manually change some of the tempo values. I was not able to evaluate my system on more than one song because of these issues.
These were the results that I got. The song that I used to evaluate was Moonlight Sonata.

```
Recall: 0.39246658566221143
Precision: 0.1936450839328537
F1 Score: 0.25933360096346847
```

Recall: How many of the correct notes did my system return over how many there are
Precision: How many correct notes did my system return over how many it returned

Future work & Conclusion

With some more tuning I think I could increase the accuracy of my system and if possible it would be interesting to transcribe a whole set an artists songs and gather info on each track and attempting to train a model to produce a bar of music maybe using LSTMs, or Markov models that is similar to the artist. I would like to create a desktop version of this project as at the moment it is very unscalable as it uses so much memory. I think it could be a good idea to revisit the piano transcribing part of the application but involve some machine learning. Maybe to revisit this idea but use a lower level language or creating then calling compiled files from python might be a faster approach. I would have liked to spent more time searching for data for evaluation.