**Instrumentalize**

CS489 Computational Audio

## Introduction

There are many existing algorithms for transferring the style of one audio file into the content of another file. For example, making music played by one instrument sound like its being played by another instrument while maintaining the melody. Most of them use some form of neural style transfer applied to the spectrograms of audio, also known as audio style transfer.

Originally, neural style transfer was used for images to make photos look like paintings. Existing implementations of audio style transfer are similar to neural style transfer, except the content file would contain the melody and the style file would contain the sounds of the instrument. Since the models only work for two-dimensional data while audio is one dimensional, the algorithm runs neural style transfer on the spectrogram of the audio.

There are many open-source implementations of audio style transfer. I will use an implementation (https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer/) available on Github as a baseline. I've tested it using a content file (data/baseline/lune.wav) containing a snippet of *Au Claire de la Lune* in piano and a style file (data/baseline/flutes.wav) containing flute sounds. The result (data/baseline/ulyanov.wav) is unsatisfactory, since it's really noisy and it doesn't always sound like flute. I could still hear piano for many of the notes. If I use a busier flute sample (data/baseline/style.wav) as the style, the result (data/baseline/ulyanov2.wav) no longer sounds like the *Au Claire de la Lune* melody. The algorithm seems to be combining the two melodies together.

My own implementation is systematic instead of using a neural network. It consists of three steps: note detection, assigning notes to instruments such that they don't overlap, and generating output using instrument samples. This implementation seemed more successful than the baseline. It was able to keep the melody of the content file while sounding like it's being played by the specified instrument. My algorithm works for converting music to flute, guitar or musicbox, and for transferring the style of one piece of music to the content of another.

## How to use

To run my program on the commandline:

```
instrumentalize.py -i <content file path> -s <instrument>
```

The <instrument> argument can take values:

tone – outputs notes in content file replaced by pure tone
beep – notes in content file replaced by 100ms beeps, good for debugging
flute – convert to flute music

guitar – convert to guitar music
musicbox – convert to musicbox music
<style file path> – style transfer

You can specify multiple instruments with extra -s arguments, like this for 2 instruments:

```
instrumentalize.py -i <content file path> -s <instrument> -s <instrument>
```

Optional arguments:

-o <output file> (default 'out.wav')
-s <one of [transfer, tone, beep, flute, guitar, musicbox] or filepath> for each instrument
-t <style file> for each transfer instrument
-f <frame size> (default 512)
-a <minimum amplitude cutoff between 0 and 1> (default 0.01)
-j <jingle threshold between 0 and 1> (default 0.1)
-c <octave shift>
-b <note blend>
-t <envelope tau>

## Data

**Input:**

The music used for testing this project all came from Youtube. These files are located in the folder data/in/

lune.wav – Au Claire de la Lune, monophonic piano (only one note played at a time)
https://www.youtube.com/watch?v=euyTUTxdRxU

summer.wav – One Summer's Day, polyphonic piano (multiple notes played at a time)
https://www.youtube.com/watch?v=gPhRvw4aIRg

futari.wav – Futari no Kimochi, flute music https://www.youtube.com/watch?v=Ju4S5yFeBiU

style.wav – Suisenka, flute music with bells https://www.youtube.com/watch?v=MQ17gScDvsU

always.wav – Always Mine by Two Steps from Hell, string instrument
https://www.youtube.com/watch?v=e0TR6xk3DWo

moonlight.wav – Into the Moonlight by Two Steps from Hell, orchestral strings
https://www.youtube.com/watch?v=u1qpGpfbtaE

highwayman.wav – The Highwayman by Loreena Mckinett, vocals
https://www.youtube.com/watch?v=Ixi4jz0Gn4E

**Instrument samples:**

The instrument samples are .wav files for notes from C1 to B7 with types y, ysoft, ylong, yhard, yvib. I filled out the missing notes by pitch-shifting the nearest available note by running librosa.effects.pitch_shift(y, fs, n_steps) in a python script, where n_steps is the number of half-steps to shift by (one octave being 12 half-steps).

They are located in wav/instruments/<instrument name>/

ysoft/ - this contains samples for short quiet notes

yhard/ - this contains samples for short loud notes

ylong/ - this contains samples for long quiet notes

yvib/ - this contains samples for long loud notes

Flute:

y, yvib: http://theremin.music.uiowa.edu/MIS-Pitches-2012/MISFlute2012.html

ylong, yhard: https://archive.org/details/LisaLim

Guitar:

y: http://freepats.zenvoid.org/Guitar/acoustic-guitar.html

yvib, ylong, yhard: https://archive.org/details/OpenPathMusic44V3

Musicbox:

y: https://archive.org/details/Berklee44v1

**Output:**

I've run my program with each of the input files as the content file and the instrument (flute, guitar or musicbox) as the style. The output is located in data/out/<input file name>/<instrument>.wav

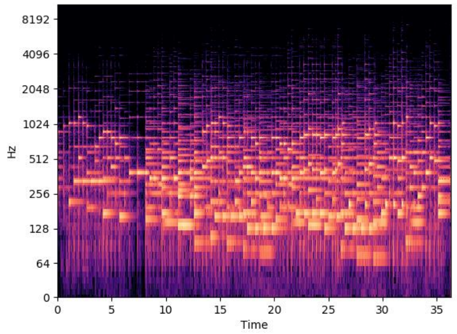
I've also run my program with some of the input files as style files. The output is located in data/out/<input file name>/<style file name>.wav

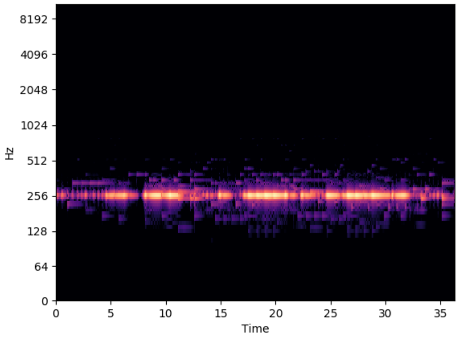
For music played with multiple instruments, the data/out/<input file name>/tone/ folder contains output generated with pure tones for each instrument, with instrument 0 (0.wav) being the main instrument and instrument 1 (1.wav) being the secondary instrument. This shows that for polyphonic music, some notes will be missing if only played by one instrument.

## Algorithm

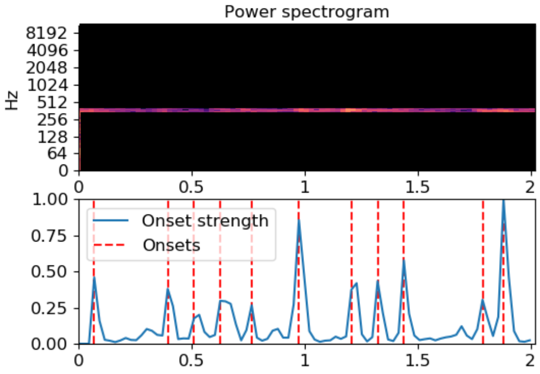
### Note detection:

In transcribe.py, my algorithm first uses a peak filter to split the content file by frequencies for each note. For example, it calls the peak filter on frequency 261Hz for the note C4 to isolate only the notes played in this frequency. It does this for the frequencies from C1 to B7.

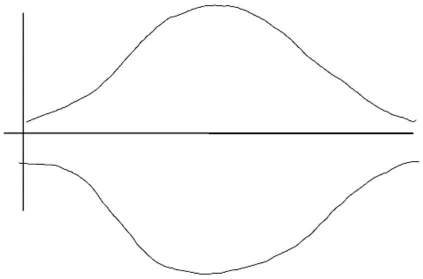




After isolating each frequency, it calls librosa.onset.onset_detect for each frequency. This would get the beginning of each note being played so that my algorithm can separate out each individual note.



The onset_detect function from librosa is too sensitive so it often splits the notes in the middle due to peaks in the onset strength caused by noise. My algorithm would then merge some of the notes into one note based on its shape. The amplitude of a complete note should rise in the beginning, have a hump and taper off in the end.



My algorithm detects if the beginning and end of a note is decreasing or increasing by splitting the note into frames of 512 samples and getting the maximum amplitude of each frame, then determining if the frame amplitude is greater than or less than the next frame.

If the start of the note is increasing and the end is decreasing, this is a complete note. If the end increases, merge it with the next note. If the start decreases, merge it with the previous note. This way my algorithm is able to accurately detect the start and end of notes.

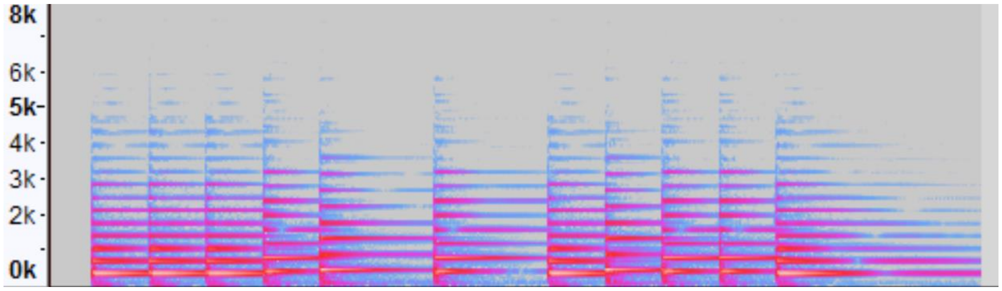
The reason I use a peak filter to isolate the note instead of using pitch detection for each onset is that for polyphonic music, multiple notes would be played at the same time. Pitch detection would take the fft of the onset and determine the pitch based on the fundamental frequency, the first peak in the fft frequencies. This would only get one pitch when multiple are played at the same time, and this pitch may not be the most important one in the melody, so using this method would cause the outputted melody to sometimes have the wrong notes. Isolating each note frequency would avoid this problem.

I've tested my program with summer.wav as the content file using pitch detection without filtering, pitch detection after bandstop filtering to isolate each octave, and isolating each note frequency (pitch detection would be unnecessary for this). These were generated using the transcribe-tests.py script which runs all three note detection algorithms and generates output using pure tones for each detected note.

From pitch-detect.wav, we can see that pitch detection without filtering results in so many incorrect notes that the melody is no longer recognizable.

The output for filtering by octave (octave.wav) sounds decent enough to recognize the melody, however compared to the output for isolating each note frequency, it's still missing some notes.

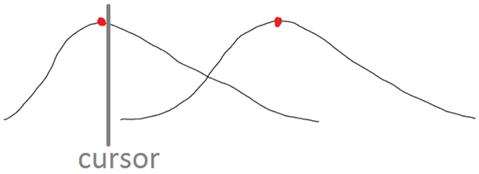
Isolating each note frequency (notes.wav) works well in preventing notes from being missing or incorrect in the output. The problem with this algorithm is that it would detect one note as many notes being played at the same time, one at each frequency of the harmonics. In the spectrogram below of the monophonic piano music from lune.wav, only one note is played at a time and each note has many frequencies (the harmonics of a piano). I was able to get past this problem in the instrument assignment step by only taking the loudest notes at each time.



**Instrument assignment:**

This step is done so that each instrument in the output plays only one note at a time. In real life, one instrument wouldn't play multiple notes at the same time. Polyphonic music is played using multiple instruments (or multiple fingers in the case of a piano or guitar), so my algorithm tries to simulate this.

In transcribe.py, it detects if notes are played at the same time based on whether the peaks (maximum amplitude) of the notes overlap.



It starts with a list of unassigned notes consisting of all the notes obtained from the note detection step. It splits the length of the entire content file into frames of 512 samples and moves the cursor (represented by a variable containing an index) frame by frame until if finds a note where the start of the note is before the cursor and the end of the note is after the cursor. It then calculates the score and cost of the note. If the score is greater than the cost, it assigns the note to the instrument, removes the note from the list of unassigned notes and moves the cursor to the index of the note's peak. It would then repeat this process until the last frame.

The score of a note is calculated based on the loudness of the note (its maximum amplitude) and how similar it is in length and frequency to the rest of the notes of the instrument. It compares the length of the note to a moving average of the lengths of the notes already assigned to the instrument. It compares the frequency to the frequency of the loudest note in the list of unassigned notes at the beginning of the algorithm.

The cost of the current note is the list of all notes that couldn't be assigned to this instrument if the current note is assigned. That is, all the notes whose peaks are after the cursor and before the peak of the current note. It would get the score of all of these notes and assign the note with the highest score to the instrument.

If the user specified for the output to be played with multiple instruments, it would run the algorithm using the unassigned notes for the number of instruments specified by the user. Each run of the algorithm would remove notes from the list of unassigned notes.

**Generating output:**

If the user specifies the name of the instrument (flute, guitar or musicbox) it generates output using the instrument samples in wav/instruments/

If the user specifies the path to a .wav style file as the instrument, it runs note detection on the style file to get a list of notes from that file, then it picks the best sample of each note based on higher amplitude and longer length, or if no samples are available for that frequency, it generates a sample by pitch-shifting the closest available sample. It gets samples for each type of note (long, soft, hard, vibrato) based on the length and amplitude of the sample. These samples could then be used to generate output in the same way as the instrument samples.

For each instrument, it loops over all the notes and selects the sample to play the note. For each note, it chooses the sample based on the note's frequency, maximum amplitude and length. The frequency of the sample must match the note, the sample gets multiplied by the maximum amplitude of the note, and the sample gets trimmed to the length of the note. For it to sound smoother, the algorithm multiplies the sample from the note's peak to the end by an envelope so that it fades out in the end. The envelope for fading out is the decay envelope:
$A(t) = A\,e^{-t/\tau}$

It also selects the type of the sample based on the max amplitude and length of the note: the samples from folder ysoft/ if it's short and quiet, yhard/ if it's short and loud, ylong/ if it's long and quiet, yvib/ if it's long and loud. If the type isn't available for the instrument, it uses the regular samples in folder y/

## Tests and Analysis

### What worked well and what worked poorly

I ran the files in data/in/ to generate the output in data/out/ using the command

```
        instrumentalize.py -i <content file path> -s <instrument> -s <instrument>
```

The outputted file is data/out/<content file name>/<instrument or style file name>

```
instrumentalize.py -i data/in/summer.wav -s <instrument> -s <instrument>
```

Worked really well for all instruments. The melody is clearly recognizable and the output doesn't sound chaotic with any unnecessary notes.

```
instrumentalize.py -i data/in/lune.wav -s <instrument>
```

This is a basic test and worked well for all instruments. It sounds good, but it did get some extra notes in the 5th octave when they should all be in the 4th octave

```
instrumentalize.py -i data/in/always.wav -s <instrument> -s <instrument>
```

Sounded good until 00:45 when the percussion begins

```
instrumentalize.py -i data/in/moonlight.wav -s <instrument> -s <instrument> -s <instrument> -s
<instrument>
```

The flute output sounded okay. The melody was recognizable, but there were many extra notes making it sound messy. The other instruments (guitar and musicbox) are unrecognizable. This is probably because this is a chaotic piece of music with many instruments playing at the same time.

```
instrumentalize.py -i data/in/futari.wav -s <instrument> -s <instrument>
```

The melody was usually clear. The flute sounded a bit jittery, probably because the content file has a lot of vibrato throwing the note detection off. The guitar and musicbox sounded really good. It was better before 00:50 than after, probably because more instruments were playing in the content file in the second part, so note detection was less accurate.

```
instrumentalize.py -i data/in/style.wav -s <instrument>
```

It sounded decent for flute. The content file was pretty noisy, but melody was captured recognizably with only one instrument. The melody was still recognizable for guitar and musicbox, but this kind of music sounds better with the long lingering notes of the flute.

```
instrumentalize.py -i data/in/highwayman.wav -s <instrument> -s <instrument>
```

I was surprise it still managed to capture the melody well despite the content file being vocals. The flute is messy but the guitar sounded almost like how a person would play it

```
instrumentalize.py -i data/in/summer.wav -s data/in/style.wav -s data/in/style.wav
```

With bells+flute music as the style file, it did style transfer somewhat well. I could hear the jingle and it sounds a little like flute. It could still use improvement since the style of the output still sounds clearly different compared to the actual music of the style file

```
instrumentalize.py -i data/in/style.wav -s data/in/summer.wav
```

With piano music as the style file, the notes sounded short like a piano but didn't really have the texture of it. This may be because the peak-filtering in the note detection of the style file filtered out most of the harmonics.

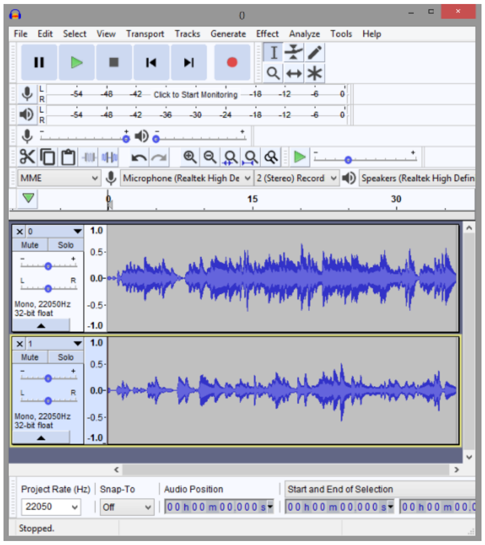
In summary, my algorithm was able to capture a recognizable melody for all of the input I tested. With a style file, the melody is recognizable, but it doesn't transfer the texture of the style well.

**How many instruments are required to play a melody?**

The user could specify the number of instruments playing at the same time in the output that my program generates. If there are not enough instruments, some notes might be missing causing to melody to sound wrong. If there are too many instruments, the extra instruments would barely play anything and be unnecessary, or too many instruments would be playing at the same time, making it hard to hear each instrument's texture.

I've run tests with different content files as input to determine how many instruments need to be specified for my algorithm to output a melody that sounds like the content file.

The test involve running the algorithm with various numbers of instruments and generating output as pure tones for each note assigned to the instrument. It outputted each instrument separately into the folder data/out/<content file name>/tone/ so they could be imported into Audacity to test the instruments separately or together.


0
File Edit Select View Transport Tracks Generate Effect Analyze Tools Help
-54 -48 -42 Click to Start Monitoring -18 -12 -6 0
-54 -48 -42 -36 -30 -24 -18 -12 -6 0
0 15 30
0
Mute Solo
1.0 0.5 0.0 -0.5 -1.0
1
Mute Solo
1.0 0.5 0.0 -0.5 -1.0
Stopped.

MME
Microphone (Realtek High De
2 (Stereo) Record
Speakers (Realtek High Defin
Mono, 22050Hz 32-bit float
Project Rate (Hz) 22050
Snap-To Off
Audio Position 00 h 00 m 00.000 s
Start and End of Selection 00 h 00 m 00.000 s 00 h 00 m 00.0

As previously mentioned, 0.wav contains the main (loudest) instrument, 1.wav contains the secondary instrument, and so on.

The quality of the output is subjective. I determined the number of instruments needed based on the minimum number to recognize the melody in the output without missing important notes.

| Content file | Number of instruments |
|---|---|
| lune.wav | 1 |
| summer.wav | 2 |
| style.wav | 1 |
| futari.wav | 2 |
| moonlight.wav | 4 |
| always.wav | 2 |
| highwayman.wav | 2 |

The test showed that monophonic piano music (lune.wav) can be played with one instrument for my algorithm. Monophonic flute music may require two instruments if the flute has a lot of vibrato, since the main instrument 0.wav of the content file futari.wav sounds really wavering, but sounds better when played together with 1.wav. Monophonic strings music (always.wav) may need two instruments, since 0.wav sometimes sounds too low or skips notes by itself. Most polyphonic music can usually be played with two instruments (with one instrument 0.wav by itself, it would have missing notes). Chaotic orchestral music (moonlight.wav) would need four instruments to capture the melody.

**Using vocals as content or style files**

My algorithm doesn’t work well with singing. As a content file (highwayman.wav), it’s still able to output a recognizable melody as shown by flute.wav, guitar.wav and musicbox.wav in the highwayman/ output folder, but it sounds muffled and there are many extra notes that shouldn’t be there. The muffledness may be because human voice doesn’t match the frequency of a note exactly, so the peak filter is getting the edges of the frequency that the singer is trying to hit rather than the loudest center. The reason there are extra notes may be because the harmonics of voice is messier than instruments, and because the start and end of notes is less clear.

As a style file (see data/out/summer/highwayman.wav), it doesn’t sound like vocals at all. This is because when getting the notes from the style file, my algorithm gets the peak-filtered notes rather than all of the harmonics.