Kyle Engelmann
kyleengelmann2018@u.northwestern.edu

Alyssa Liu
ljhan@u.northwestern.edu

foünd

EECS 352 - Northwestern University - Prof. Bryan Pardo

PROBLEM

Given a sample of a non-musical sound, such as a dog barking, our project software will synthesize music. It would use the overtones of a slowed down version of that sample to construct a musical piece that, when sped up, will resemble the input sound. While there are many programs that take in musical elements to produce music, we think it would be really interesting to create software that can take something non-musical to make something that at least resembles music. Moreover, many songs use random samples from everyday sounds (e.g. water splashes, animal noises) as embellishment. We want to take this a step further by turning such everyday sounds into music itself. It would also be really interesting to see what kind of varying results this one program gives using different inputs.

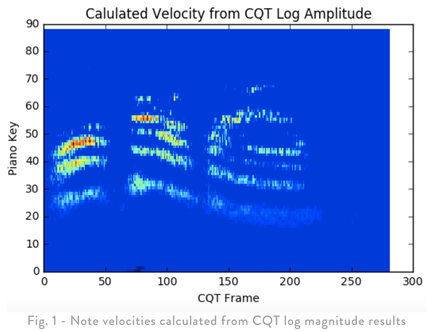

Fig. 1 - Note velocities calculated from CQT log magnitude results

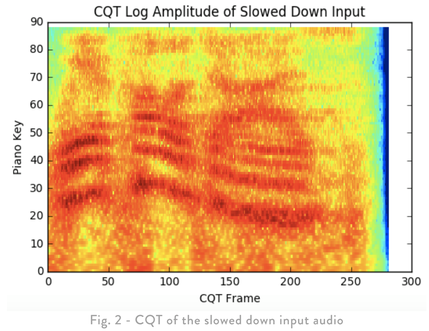

Fig. 2 - CQT of the slowed down input audio

METHOD

Our system will take an audio file as an input and slow it down significantly. It will then use the frequencies of that sample to map onto the 88 keys on a piano. These notes will be used to create a MIDI reconstruction with a user-specified instrument. The user can also pick the playback speed.

PROCESS

Our system will take a .wav audio file as an input. We focused on two types of audio - dog barks and a recording of the original “We Apologise.” Our software will then cut off everything past the first 5 seconds and slow the input file down to 10x the length using the PaulStretch algorithm and take the constant-q transform (CQT) of the result using the libROSA library. The CQT will then be remapped to the frequencies matching the 88 piano keys. It will then use a user specified instrument to build the spectrum of the input by using pyFluidSynth to create a MIDI note of matching fundamental frequency for each note. Using the user specified magnitude threshold, magnitude results from the CQT will be converted to note velocities. Since MIDI notes at higher pitches are quieter than those of lower pitches, the velocities are lowered linearly per key, descending. 88 pyFluidSynth channels are created with initial volume 0, with a maximum velocity note on event in each. For each CQT time frame, the channel’s velocity is set for each note to the correct volume using the calculated velocities, and recorded in an array. The MIDI notes will be created using a user specified S2F soundfont. Finally, PaulStretch is used to bring the reconstructed audio up to the playback speed chosen by the user (from 10x slower than the original to the speed of the original). To test results, the cross similarity matrix between the input audio and the MIDI audio will be computed. In addition, users can make qualitative comparisons between the input and output audio. The original “We Apologise” audio and choral reconstruction are used as a baseline to make these qualitative comparisons.

RESULTS

Our system works somewhat well for a few samples but not as well for others. It worked reasonably well on the dog howl, as well as on the original “We Apologise.” While our results were not as smooth or intelligible as the original choral reconstruction, experimentation with the parameters and selected soundfont improved our results. As seen in the figures below, the CQT of the output MIDI reconstruction has a very similar shape to that of the CQT of the input audio. However, there is no strong diagonal in the cross similarity matrix between the input and output. This signals there is no strong resemblance between the input and output. Furthermore, during user survey, the output was only recognizable to people who know what the input is and are listening for it.

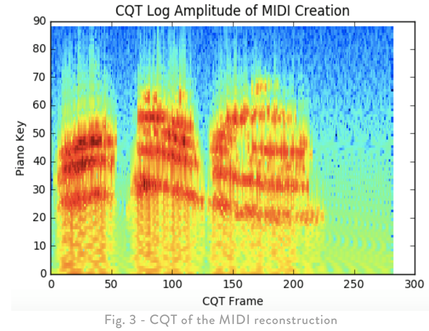

Fig. 3 - CQT of the MIDI reconstruction

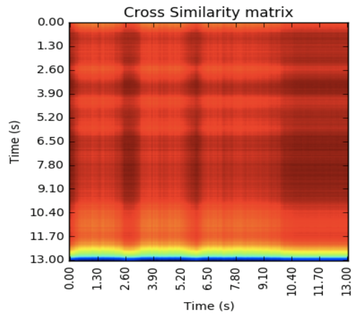

Fig. 4 - Cross Similarity matrix of the input and output