AUTOMATIC MODAL SHIFTING USING CHROMAGRAMS AND A CONVOLUTIONAL NEURAL NETWORK

Miller Hickman
University of Rochester
mhickman@ur.rochester.edu

Alex Mancuso
University of Rochester
amancus9@ur.rochester.edu

ABSTRACT

While methods of manipulating sound files, such as pitch shifting, are readily available in several DAWs, changing the modality of a song is not as accessible of a feature. We propose a method that will allow the user to automatically change the modality as well as the key of a song by simply inputting a sound file and a target mode. Our implementation was based on an ideal target due to time constraints and is open for further advancements.

1. INTRODUCTION

Several accessible software applications allow the manipulation of a song's features, such as the length or pitch of the file, but few, if any, allow the user to change the song's mode. The *mode* of a song refers to the intervals between notes in a scale, as well as the *tonality* (major, minor, diminished, etc.) of the chords at that interval. For all of our musical references, we will be basing them off of tunings and scales commonplace in Western music, i.e. music that uses twelve-tone equal temperament.

On inference, this project will take in separate audio files for each instrument. Ideally, the user would not need to input separate tracks for all of the instruments, but we do not have source separation methods clean enough to yield usable results. Assuming the input is a group of stems, the first step involves two different possible routes based on whether the source is monophonic or polyphonic, i.e. playing one note or multiple notes at a time. For monophonic sources, the pitches just need to be detected and shifted based on the desired mode. It gets more complicated for polyphonic instruments since only certain notes in the chord may need to be shifted. For polyphonic modal shifting, we used a convolution-based neural network to implicitly shift the tonality. After changing modes, the audio is pitch shifted to the desired key.

For the sake of time, we have limited the scope of chorded instruments to just synthesizer sounds. As it stands, the only models being trained for tonality shifting are from major chords to minor chords as well as from minor chords to diminished. Once those models achieve desired results, additional models will be trained to shift between other chord tonalities.

1.1 Related Works

Modality shifting, and more specifically tonality shifting, are not tasks that have been addressed at all within the research community. Thus, inspiration and direction for how to actually approach these tasks required looking at other computer audition tasks.

Audio source separation turned out to be a great direction, as tonality shifting and source separation are very similar from a high level. Training source separation models relies on learning a difference in the frequency content of an audio mixture and an isolated voice or instrument. Shifting from one tonality to another relies on learning the difference in the frequency content between the pitches in those tonalities.

Many different models have been used to train source separation models, but Convolutional Neural Networks (CNN) seemed like the most promising method. As far as CNNs, Wave-U-Net [5] was the most promising model, which we aimed to emulate. Wave-U-Net was itself based on the U-Net model for image segmentation [4]. With some adjustments, our model for tonality shifting does take heavy inspiration from the U-Net architecture.

2. DATA

Data collection was a surprisingly difficult task for this project. There are no sufficient datasets of chords with multiple tonalities available, which left data generation up to us. Currently, the dataset used for training consists of chords generated from PySynth [2], which is a programmable synth module in Python with six different timbres. The different timbres are designed to sound like a flute, piano, electric piano, plucked string instrument, and bowed string instrument. Pysynth is publicly available code under the GNU General Purpose License.

Chords were generated with root notes ranging from midi notes 24 to 80 in major, minor, and diminished tonalities. Each chord was also generated with a corresponding version that includes the seventh interval of it. Ideally, the data set will be expanded to include more instruments and timbres as well as different effects applied to the generated chords, which will further generalize the tonality shifting model.

Other datasets were also considered, like GuitarSet [3] and others that included chords played on piano or guitar. However, it seemed that each set had some aspect that made them not ideal. For example, some guitar chord sets would have various tonalities played in the same position of the guitar, but with different fingering positions that made comparing corresponding samples impossible.

3. METHODS

This section describes the methods necessary to perform three main tasks that exist within our implementation: identifying the input pitches and determining if they are monophonic or polyphonic, shifting the tonality of polyphonic chords using a CNN, and shifting the new version of the song to a different key.

3.1 Pitch and Chord Identification

To properly shift the pitches, it is necessary to know the note name and how it relates to the original mode and the desired mode. It is also needed for pitch shifting to a new key if that is desired. For detecting pitches, we used data from chromagram calculations, which were derived using a constant Q transform. The algorithm, taken from an open-source code based on the concepts of Brown and Puckette's paper [1], returns twelve values per frame that represent the strength of each note detected in the twelve-tone scale regardless of octave. With a monophonic source, for example, the maximum value amongst the twelve outputs is the theoretical evaluated pitch. This is beneficial since it limits the scope of the notes to the quality of the pitch (note name) rather than both the quality and its height (octave). The chromagram also allows for pitch detection of monophonic and polyphonic inputs, a very important feature for dissecting chords.

We will refer to the location in which energy is represented for a note in a chromagram as that note's energy *bin*. If the energy in a bin passes a certain threshold, it is considered a note that is being played at that time. This threshold is implemented to distinguish between notes being played and silence, or 'blank' frames. It also provides a way to tell the difference between monophonic and polyphonic inputs. If multiple notes pass the threshold, that is considered a chord rather than a single note. For our testing, we found the best results using a threshold of 0.032 for the energy in a bin and using other thresholds for certain sections that were factors of the energy threshold.

It is possible that multiple pitches will pass the threshold even when a monophonic source is playing. Several condition statements were included in the note detection algorithm to check the behavior of the surrounding frames and nullify the measurements in the multi-pitch frame when necessary. It is very possible that a piano player will play single notes followed by chords and that not every new musical event will involve the same amount of notes, so several conditions need to be put in place to account for different scenarios.

An example of a chromagram measurement is seen in Figure 1. The audio used in this case was a file of a distorted guitar recorded in Reaper. The original audio consists of a single melody of the guitar playing the notes D-A-D. The file was duplicated twice and pitch-shifted within Reaper to add in the third and fifth above each note. This is not a realistic input for a guitar, but this was used just to show an example and to test if the methods worked in any capacity.

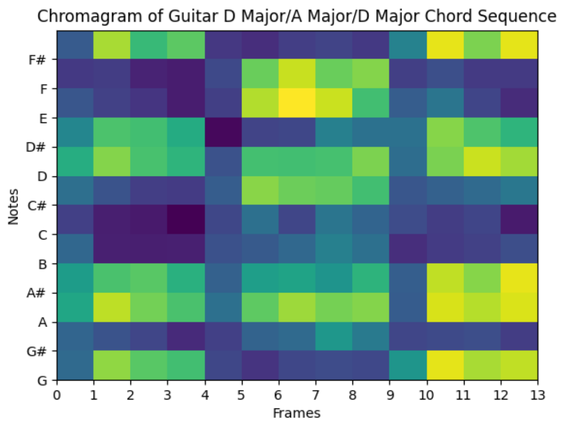

Figure 1. Output from a distorted guitar playing a 3-chord sequence. The chords included just the root, the third, and the fifth of the chords.

As seen in Figure 1, the chromagram implementation is subject to smearing. This means that the note a half step above the ground truth often also has a lot of energy, which can lead to pitch detection errors even with thresholding. Thus, peak detection is implemented to find the maximum value in a set that lies above the threshold. Due to misreadings and smearing, it is possible that the bin above the ground truth is picked as the note being played, so the lower note is chosen when two energies are above the threshold and right next to each other. To distinguish when a note or chord starts and stops, empty frames are detected as checkpoints and the number of populated frames between empty frames is considered the length of that note or chord. Distinct changes in note streams are also used as checkpoints to determine chord changes in the case that there is no space between chords.

In the case of chords, chord names and each of their chord qualities were stored. The *quality* of a chord refers to its tonality (major, minor, or diminished), not to be confused with the quality of a single pitch. The notes detected as part of the chord were extracted and run through a function from the pychord library that allows chord names to be extracted given their pitches. The quality of each chord was found using a function from the music21 library.

3.2 Tonality Shifting

The overall implementation of the tonality shifting model is based on the U-Net [4] model to perform image segmentation. The U-Net architecture has also been implemented with audio to perform source separation in works like Wave-U-Net [5]. However, instead of computing the loss between the spectrogram of an audio mixture and isolated vocal or instrument, the tonality shifting model compares the spectrograms of chords between the input and target tonality.

To effectively create a mask that converts one tonality to another, like major to minor, the intended conversion lies in isolating the frequencies in the input chord that shift in the target. For a shift from major to minor, this would involve generating a mask that reduces the energy at the major third and major seventh notes, while introducing the frequencies lying in the minor third and minor seventh notes, which are each one semitone down. To effectively train this mask, the only difference between the input and target chords must be only those notes. Thus, all pairs of input and target chords share the same timbre, duration, and octave.

Our method implements the layers as shown in Figure 2. U-net was originally designed for image segmentation, so convolution layers are performed in 2-D. To adapt this for audio, the 2-D layers were reduced to 1-D convolutions.

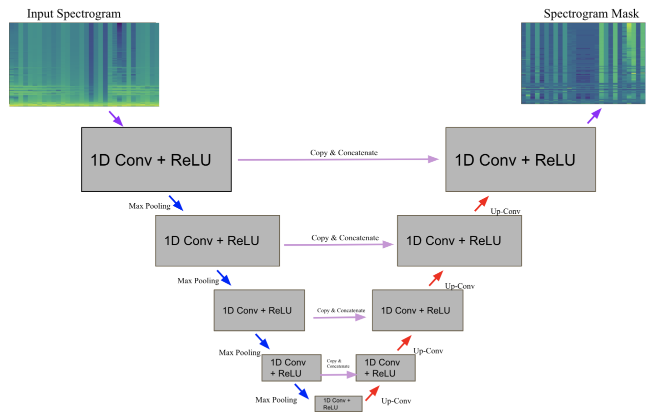

Figure 2. Diagram describing the layers of the model.

As can be seen in Figure 2, there are three encoding layers, which reduce the data to a 64 sample feature vector. Each convolution block actually includes two separate convolutions and ReLUs. Between each encoding layer is a max pooling function. Once the data is reduced, there is a final encoding before upwards convolution. Decoding the feature vector requires 3 layers of convolution with a layer of upwards convolution between each one. At each decoding layer, the corresponding encoding is also concatenated with the upwards convolved feature vector. This ultimately results in a spectrogram mask that can then be convolved with the input to hopefully shift its tonality. While training, the loss between the estimated chord and the target tonality is evaluated using mean squared error loss.

The model is only trained with chords of the same root note, C, across multiple octave ranges to contextualize and specify the spectrogram mask. If all notes were used in training, it would likely be impossible to achieve a reliable and quality output. Thus, on inference, the input chord will be pitch shifted such that the root note is C, evaluated and processed with the model to shift its tonality, then pitch shifted once again to the desired note.

A separate model is trained for each pairwise combination of tonalities: major to minor, major to diminished, minor to major, etc.

3.2 Pitch Shifting

We used a pitch-shift function from the librosa library to shift the audio in increments of semitones. It provided quick pitch-shifting with few artifacts compared to other sources. The need for pitch shifting comes from the idea of training the models with chords that are all based around a root note of C to achieve better performance. We also wanted to implement the option of shifting the output afterwards to a brand new key. The most we would ever need to pitch-shift a song or a chord, assuming octave does not matter, is six semitones, so we chose librosa's pitch-shift function because it can perform quick pitch-shifting of that magnitude with little added noise.

It is important to note that we purposely did not implement a pitch shifting algorithm to change the tonality of chords in the input file. Using a CNN was the preferred method for this specific case because the task of separating and resynthesizing the notes of a chord is very challenging and would likely lead to unsatisfactory results compared to training a model. However, if separating and resynthesizing the notes of a chord was a viable option, it certainly would be easier since pitch shifting is not a taxing task.

4. RESULTS

4.1 Pitch Identification

Pitch identification was fairly accurate after a couple of modifications to the code based on [1]. The results in Figure 3 are from an excerpt that is around 30 seconds long, providing confidence that this can work for full-length songs.

For polyphonic sources, the pitch detection was still accurate. Figure 3 shows a demonstration with an extended version of the melody from Figure 1 and the third and fifth above the melody added in. All pitches were detected correctly for each frame at an arbitrary and adjusted frame size of 4096 samples. All other parameters within the chromagram calculation functions were set to default for the sake of simplicity. From here we can extract the note names and use a library called pychord to extract a chord name and a library called music21 to retrieve the tonality from the given notes so that we know which model to feed the data to (major to minor, minor to major, etc.).

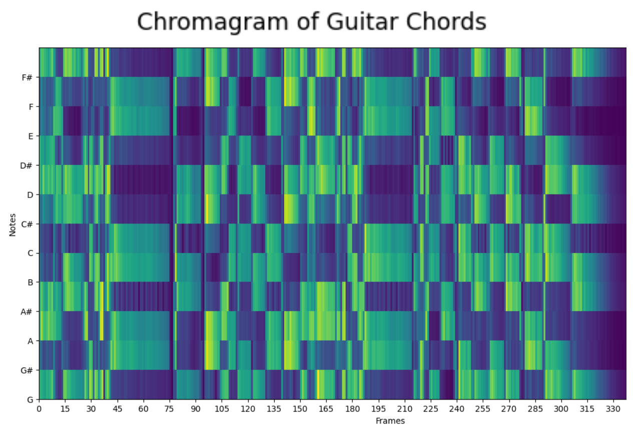

Figure 3. Chromagram of an extended version of the guitar melody from Figure 3, with a third and fifth above the original melody added.

4.2 Tonality Shifting

Preliminary results for the tonality shifting model are promising, but have not reached our desired goal yet.

With the current training parameters, the model has been able to achieve a training loss of 0.1512 and validation loss of 0.1573 after significant training. Though, these values have reached a plateau and are not improving, despite additional training for a large amount of time. Similar values were reached for the model trained to convert from minor to diminished. Unfortunately, the model trained to shift from diminished to major was never able to reach an acceptable quality.

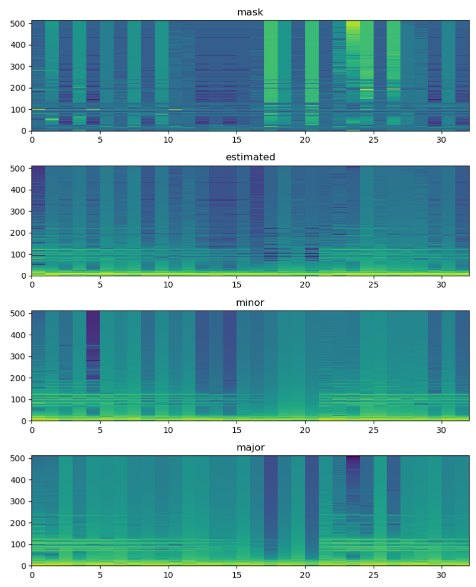

Figure 4. Example output during training. Top to bottom: Mask, Estimated, Minor, Major. Estimated is Mask convolved with Major, where Minor is the target. The x-axis is frames and y-axis is frequency bin.

Figure 4 displays the output from the model during training. The estimated minor chord, with input of a major chord, does get pretty close to achieving the desired spectrogram. However, the actual output on inference is still not ideal. For the major to minor model, the output sounds noticeably shifted to the desired tonality. The model trained to transform minor tonalities to diminished is not as subjectively clear as the major to minor model, but it does appear to be quite effective. It sounds as though the fifth scale degree, the one shifting between the minor and diminished tonalities, mostly disappears instead of being shifted. It does have a slight twinge of the lowered fifth scale degree. The diminished to major tonality shifting model is extremely poor. It just produces a large amount of distortion and artifacts. Clearly the data was not sufficient enough to properly train a major to diminished model.

These results are not fully satisfactory yet. The problem lies in data size and diversity. With the resources available, it was not possible to obtain or generate a dataset that has a wide breadth of instruments, timbres, and instances of chords playing. However, the results give promise that with a more robust dataset, a really high quality result can be achieved.

5. CONCLUSIONS

We've proposed a method for shifting the mode of a song through the use of a convolutional neural network. Our work has potential to be successful, but there are certain limitations we must overcome. The chord identification is effective, but our chord extraction only works for up to three different notes in a chord as of now, which is not ideal for real-world applications. The chord extraction is also very threshold-dependent, meaning not all audio files work with the same threshold; some process to normalize input will likely be required for the method to work properly. Even with normalizing, the chord extraction in its current state works extremely well for both chords and melodies with the guitar data we have but does not produce promising results for Musescore-generated piano melodies and chords. Even turning off the built-in reverb and trying different sounds did not improve the results, meaning modifications need to be made to account for different types of input files.

In particular, the tonality shifting model also needs further development and fine tuning. Some of the parameters we intend to continue training with include more frequency bins in the magnitude spectrum, meaning a larger block and hop size, as well as weighted learning rates that decay over the course of the training. These adaptations, along with more time spent training the model, should increase the models' ability to learn the more minute differences between the estimated spectrogram and the target spectrogram, which should decrease loss further and improve the overall output.

In the future, we'd like to achieve the ultimate goal of the user being able to simply input an audio file, as well as a target key and mode, and have their input be returned in the new key and mode without an excessive amount of artifacts. Automatic key detection is an additional feature that would have to be implemented along with smoothing audio during reconstruction to avoid audible 'pops' in the output. The model will also need to be trained with more instruments and timbres of those instruments to account for various types of input. Better detection of the start and end of notes would account for embellishments such as bends in a melody or a chord, and this could be achieved by linking an onset detection algorithm to the current methods. A lot of these propositions were not explored in depth, or at all, due to time constraints, but our methods prove that there is considerable belief that automatic modal shifting can become an accessible feature for audio editing in the future.

6. REFERENCES

[1] Brown, Judith & Puckette, Miller. (1992). "An efficient algorithm for the calculation of a constant Q transform". Journal of the Acoustical Society of America. 92. 2698. 10.1121/1.404385.

[2] PySynth, "PySynth - a Music Synthesizer for Python," https://mdoege.github.io/PySynth/ (accessed Dec. 5, 2022)

[3] Q. Xi, R. Bittner, J. Pauwels, X. Ye, and J. P. Bello, *"Guitarset: A Dataset for Guitar Transcription"*, in 19th International Society for Music Information Retrieval Conference, Paris, France, Sept. 2018.

[4] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science(), vol 9351. Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28

[5] Stoller, Daniel & Ewert, Sebastian & Dixon, Simon. (2018). Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation.