Creating a musical trailer for an album

Ben Harrington
benmharrington@gmail.com

Sarah Staszkiel
staszkielsarah@gmail.com

Andrew Wiggins
xandrewwiggins@gmail.com

ABSTRACT

Using various methods such as key detection, tempo matching, and musical structure analysis, we will create a software that takes an album as input, and returns a single cohesive song that is created from segments of each song from the inputted set. Our project will choose sections by analyzing each song and determining verses and chorus sections in order to choose the portion that is to be added into the outputted song. Once these sections are determined, they will be stitched together into a single song by various possible methods, including fading in/out, downbeat repetition, or hard cuts. The order that the segments are placed in is yet to be determined, whether it be in the same album tracklist ordering, or an order that is deemed to be the most smooth for transitions between song segments. In order to create smooth transitions between song segments, the tempo and key of each segment must be detected in order to be able to adjust properly during transitions.

1. PROCESS

When searching for a new artist, many listeners choose to listen to the singles released preceding an album drop to determine whether or not listening to an album is worth their time. With the rising popularity of Spotify recommendation algorithms and curated playlists, it is easy to see the demand for a song that gives the listener a short preview of an album without requiring to listen to it front to back.

1.1 Selecting data sets

The first task for this project is selecting data sets. These data sets will be used for testing purposes throughout the development of the project and should therefore represent a wide variety of possible music inputs including multiple genres and tempos. Additionally, since the purpose of this project is to provide an audio summarization of an album, all of the data should be full length albums.

1.2 Album and song analysis

After selecting data sets, the next step to accomplish is analyzing each song within the set of songs, typically an album, given to the software. The first and foremost target of analysis is the tempo of each song. Using downbeat tracking [1-3] as well as using inspiration from neural networks [4], the tempo is the first step in extracting data from each song. After this the software will determine the key of each track [5-7], looking at collections of pitches throughout the song as well as in shorter segments to analyze the chord qualities of smaller sections [8].

1.3 Choosing the song segments

After analysis, the next step is to study the sections of each song in order to have a better understanding of which part to include in the output. This step consists mostly of determining chorus and verse sections by analyzing repeating segments of the audio signal [9-11], as well as analyzing the overall structure of the track [12] and isolating vocal portions to try to avoid cutting a track in the middle of a word or melodic line. [13]

1.4 Stitching together the segments

The final step of the process involves actually blending together the chosen segments into a separate cohesive musical structure. Some approaches to this include fading a segment into another in order to blend them properly, or doing automatic cuts using key blending and rhythm adjustments made to ensure the listener is not overly discomforted by a sudden switch [14, 15].

2. RESOURCES

In order to complete these steps, we will be using Python, as a number of Python libraries exist that are helpful for analyzing and manipulating audio. Some of these include PyChorus for analyzing chorus [10], as well as PyDub and PyAudioAnalysis for additional audio analysis and audio formatting [16-19].

3. GENERAL TIMELINE

Oct 18-24: Choose a short list of albums that spans various genres in order to do simple testing.

Oct 25-31: Brainstorm good segmentation choices/begin beat tracking code.

Nov 1-7: Finish beat tracking code that will track beats of each individual track within an album. This will help for segmentation.

Nov 8-14: Write code for basic stitching together of segments.

Nov 15-21: Complete code for proper segment determination.

Nov 22-28: Determine order of segment arrangement, and method(s) of segment stitching. Adjust code for more difficult genres. Final adjustments and testing.

Nov 29 - Dec 1: Presentations

Dec 2: Last Class

4. MID-TERM EVALUATION

4.1 Current progress

Our basic project presently accepts an album of songs in the form of a directory containing wav files. This directory is set as a variable within the code itself rather than being prompted by the user externally. After input, each file in the directory is subjected to a beat-tracking algorithm that estimates the beats from the songs. A segment is then cut from the track and added to an overall trailer number array. This array is then written to its own wav file and can be played using any conventional music software.

4.2 Positive aspects of the project so far

In its ongoing state, the software is able to accept any viable (wav files) album and create a simple trailer from said album. This gives the listener a preview of each track within the album and a general idea of what the album will sound like. The decision to use wav files was made as they are often used as a commercial format in which music is sold, and is one of the few lossless file formats that does not compress the music down. In its most basic form, the result is a barebones representation of what we are trying to accomplish.

4.3 Setbacks from original timeline

The actual segmentation of each track has been a task that is proving more difficult than originally anticipated. Ideally, segments from the tracks are created by counting beats based on the time signature of the song, and at an interesting section of the track, or at least not in the middle of a transition period (e.g. between a verse and a chorus). This has proven more complicated due to the natural difficulty of determining the time signature from each track, as well as the inherent errors within beat tracking software to provide accurate representations of beats within the songs. Moving forward, this would appear to be the most challenging task on the list, as each song is unique enough to make one overall algorithm for song segmentation reasonably difficult to implement.

In addition to this component of the project, another difficulty lies within the genre of the album. See Fig. 1 for the list of genres we are currently working with, and those which will prove more demanding than others to implement a proper segmentation choice. One of the challenges moving forward will be to ensure that our project works for all genres, rather than just those that contain generally strict structure and well-defined beats.

Reasonable	Challenging
Hip-Hop/Rap	Classical & Piano
Pop & Alternative	Jazz
R&B	
Electronic	

Figure 1. Genres and their respective difficulty in implementing segmentation.

Another difficulty that has arisen is the ability to pitch-shift and timestretch each song into one cohesive structure that is all in the same key and tempo. This appears to be more difficult than previously anticipated and would be a last-minute addition if everything else goes smoothly.

5. END-OF-TERM GOALS

5.1 Minimum goal

A worst case scenario for our project, assuming that it is too difficult to implement the remainder of our timeline, would be to use the software that we currently have. This would entail reading in an album and segmenting each song based on a general time/beat number. They would be stitched together using either a hard cut or a quick fade between segments, resulting in a clunky song that gives the listener an idea of what each individual track sounds like.

5.2 Reasonable goal

Considering the setbacks that we have encountered, our main goal now is to have each segment last a reasonable number of beats (e.g. 16/32 bars), and have the segments cut between each other directly on beat without any jarring transitions. Considering the difficulty of applying

this to more complex time signatures and unclear beats in certain genres, this goal appears to be reasonable without being trivial.

5.3 Stretch goal

After meeting the reasonable goal, there are a couple of ways that the software could be improved. These include choosing segments of each song that are interesting parts of the individual track (e.g. chorus), having them blend together on key (e.g during a rhythm section, from a relative dominant chord to its tonic, etc.), and time-stretching/pitch-shifting the tracks so that they all fit into one "master" key and tempo. Properly implementing any of these would be a good stretch goal if the reasonable goal comes along smoothly, but implementing all of them would be quite an impressive feat.

6. METHOD

The primary algorithm for creating an album trailer is broken down into four primary subsections: initialization, chorus detection, beat detection, and crossfading. Extra features were added to the algorithm after these were implemented in an attempt to further optimize the final product while maintaining the integrity of the audio.

6.1 Initialization

Specifying a directory is the first step in creating the album trailer. As of the time this paper was written, this is done by hardcoding the directory path in the beginning of the algorithm. If a command line interface was introduced, or a user interface, steps would be taken to improve the directory specification process.

Next, the specified directory location is read for wav files. Librosa is used to to load each of the files, and each one is converted into 32-bit floating point numbers for ease of use with Numpy.

6.2 Chorus Detection

In the second part of the process, each song is analyzed for a chorus section or, in the case that no chorus can be detected, an 'interesting' part of the song. A Python library called Pychorus is used to detect a chorus for each audio file [10, 20]. For the majority of genres, a chorus is easily detectable. Figure 1 shows the spectrogram of a rap song 'Gremlins' by Danny Brown which highlights the different sections present within a rap song. Pychorus uses a combination of the spectrogram and chroma to determine the logical sections of the audio.

Pychorus returns a timestamp that indicates the start of the chorus section. From this timestamp, we can assume an average length of the section and extract that portion of the audio file for use within the final trailer.

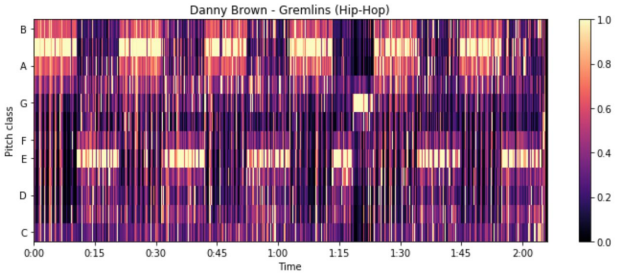

Figure 1. Spectrogram for 'Gremlins' by Danny Brown highlighting segments of a rap song.

In the case that a chorus cannot be detected, a fallback method is in place that utilizes the Vamp plugin Segmentino [21]. When this plugin is run on an audio file, it returns a Python generator that contains logical segments of the audio. Each of the segments is given a label to indicate which piece of the audio it is, whether that be a verse, chorus, bridge, etc. and similar segments are given the same label. These segments are then analyzed to determine which labeled segment occurs the most, and the first segment sharing that label is used in place of the chorus. Danny Brown's 'Gremlins' is broken down into segments according to segmentino using Audacity in Figure 2. Since label 'A' is the most common segment for this track, the first occurence of 'A' would be chosen as the 'interesting' part of the song.

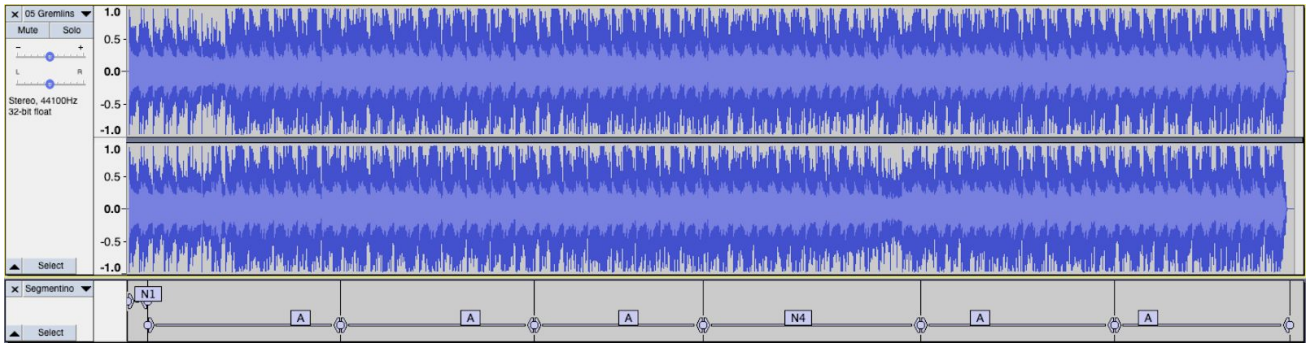

Figure 2. Segmentino plugin run in Audacity showing segments (bottom) for Danny Brown's 'Gremlins'

6.3 Beat/Tempo Detection

In order to prevent jarring transitions between each of the extracted segments, tempo and beat analysis has been performed using the librosa beat track function. Tempos were adjusted by doubling or halving accordingly in order to ensure they were within the range of 60<x<140 bpm. If accurate tempos are not found, the function returns a tempo of 0. When this occurs, the reordering of tracks and time stretching segments is deemed inaccurate by the code and it will not perform either of these functions (see 6.5: Extra Features). A data structure of timestamps of each detected beat is created for each track within the album. When the timestamp of the chorus is returned from the chorus selection algorithm, the program searches for a beat in the data structure that is within 3 seconds of the returned timestamp. If a beat cannot be found within 3 seconds, it is assumed that the beat tracking algorithm was not accurate enough to determine legitimate beats in the song and the segment chosen begins at the chorus start timestamp (not on a beat). If the beat is found, the timestamp is adjusted to be the timestamp corresponding to the closest beat rather than the original chosen timestamp. This ensures that each segment will start on beat. The final beat of each section is selected by counting a variable n number of beats (default 16) and returning an end segment timestamp based on the nth beat in the data structure. If the beat tracking was unsuccessful, the end timestamp returned is a time value corresponding to $1.5*n$. This represents an assumed average of 90bpm.

6.4 Crossfading

Crossfading is used to remove jarring transitions and is a relatively straightforward calculation where 'f' is the fade value specified by the user (converted from seconds to samples), and A and B are the audio arrays to be faded out and in respectively:

$$m = \frac{i}{f},\ i = 0, 1, 2, ..., f \quad (1)$$

$$A[(len(A) - f + i] = A[(len(A) - f + i] * (1 - m) \quad (2)$$

$$B[f - 1 - i] = B[f - 1 - i] * (1 - m) \quad (3)$$

A simple fade-in and fade-out is applied in a similar manner to the first and last tracks of the trailer respectively.

Figure 3 shows an album trailer with crossfading applied to each audio segment. Each segment is approximately the same duration and features a slow fade-in/fade-out technique to ensure a smooth transition. The amount of overlap between the two songs during the crossfade is a user controlled variable, as well as the duration of the crossfade itself.

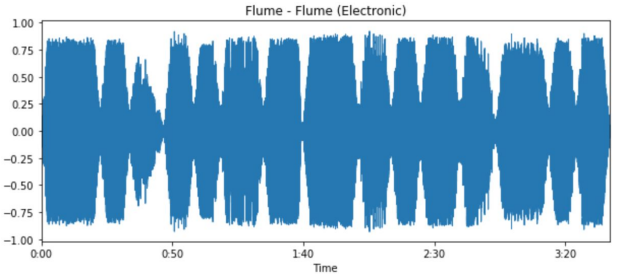

Figure 3. Example audio trailer for Flume's self-titled album highlighting the crossfades between segments.

6.5 Extra Features

With the primary algorithm written, a couple extra features were implemented in an attempt to improve upon the sound and feel of the final product. These included song reordering and time stretching. Song reordering is self-explanatory and simply rearranges the song in such a way that the tempo of each song increases as the trailer continues. The opposite is also an option, with the tempo of the tracks decreasing over time.

Time stretching is a slightly more ambitious goal. From before, the array of tempos returned from the beat tracking is analyzed and a mean value is taken. A simple function takes the mean tempo (m) and the tempo (t) for each track and returns the scaling value (s) for every song in the album.

$$s = m/t$$

This scaling value is used by the librosa timestretch function to stretch each segment in the trailer to match a master tempo (corresponding to the mean). The idea here was to make the entire trailer more consistent and feel like one piece of music rather than simply fading in between separate pieces of music. This feature is an interesting way to blend the trailer more smoothly into one cohesive track, but gives the listener less of a concrete sense of what each track on the album actually sounds like, as some may have been stretched by a large scale and sound inaccurate in regards to their original sound.

Both of these additional features were made optional with boolean variables at the start of the program, as the importance of maintaining the original order and tempo of the tracks depends on the preference of the user.

7. FUTURE WORK

One major aspect of the program that needs to be addressed in the future is the running time. At the moment, the audio files are loaded in once for the actual data processing and once for Pychorus to analyse. This is a major source of the program's long run-time. From our research, it appears that Pychorus requires the audio tracks to be loaded directly into its create_chroma function. In order to cut down on run-time either a work-around would need to be implemented to allow numpy arrays to be loaded into the create_chroma function, or a different Chorus detection library would need to be explored.

Another function that would be useful to add in the future is to be able to organize the audio segments in terms of musical key. Similarly to the optional tempo adjustments and tempo ordering currently in place in our program, a pitch and key detection algorithm would allow each segment to be ordered according to related musical keys. Pitch-shifting could also be implemented to allow each segment to be shifted to an 'average' key.

Extending the program to accept file formats other than wav files would also increase the utility of the program, but may introduce problems with mismatched sampling rates that would need to be addressed.

8. CONCLUSION

The final project encompasses much of our original goal. The program takes user input of a folder path containing wav files. With the combination of Pychorus and segmentino the program is very effective at choosing an interesting section (or chorus) of the audio. Each section will last a user-specified number of beats, or if beat tracking fails, a set length of time. Smooth crossfades are implemented between each track segment. We achieved a good portion of our stretch goal as well by implementing several optional functions in the program such as reordering the sections by either decreasing or increasing tempo, or time-stretching each segment to match an average tempo of the entire selection of tracks. A single wav file is the output of the program.

9. REFERENCES

[1] M. Davies, M. Plumbley, and D. Eck. "Towards a musical beat emphasis function." In 2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, pp. 61-64. IEEE, 2009. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5346462. [Accessed: 14-Oct-2019]

[2] M. Davies, and M. Plumbley. "A spectral difference approach to downbeat extraction in musical audio." In 2006 14th European Signal Processing Conference, pp. 1-4. IEEE, 2006. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7071189. [Accessed: 14-Oct-2019]

[3] B. McFee, H. C. Crayencour, J. Pablo Bello, M. Fuentes, S. Essid,. "Analysis of Common Design Choices in Deep Learning Systems for Downbeat Tracking." In ISMIR, pp. 106-112. 2018.

[4] H. Schreiber and M. Müller. "A Single-Step Approach to Musical Tempo Estimation Using a Convolutional Neural Network." ISMIR. 2018.

[5] A. Lykartsis, M. Stein and R. B. Gebhardt. "A Confidence Measure For Key Labelling." In ISMIR, pp. 3-9. 2018.

[6] R. Mahieu. "Detecting Musical Key with Supervised Learning." (2017).

[7] A. Faraldo, F. Hörschläger, M. Le Goff, P. Herrera, P. Knees, R. Vogl and S. Böck. "Two Data Sets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections." In ISMIR, pp. 364-370. 2015.

[8] F. Korzeniowski, G. Widmer. "Improved chord recognition by combining duration and harmonic language models." 2018.

[9] V. Jayaram, "Finding Choruses in Songs with Python", Medium, 2018. [Online]. Available: https://towardsdatascience.com/finding-choruses-in-songs-with-python-a925165f94a8. [Accessed: 12-Oct- 2019]

[10] M. Goto, "A chorus section detection method for musical audio signals and its application to a music listening station", IEEE Transactions on Audio, Speech, and Language Processing, vol. 14, no. 5, pp. 1783 - 1794, 2006 [Online]. Available: https://ieeexplore.ieee.org/document/1677997. [Accessed: 16- Oct- 2019]

[11] M. Bartsch and G. Wakefield, "To catch a chorus: using chroma-based representations for audio thumbnailing", Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575), 2001.

[12] J. Paulus, M. Müller, and A. Klapuri. "State of the Art Report: Audio-Based Music Structure Analysis." In ISMIR, pp. 625-636, 2010. [Online]. Available: http://paulus.kapsi.fi/pubs/paulus_ismir10_star.pdf. [Accessed: 14-Oct-2019]

[13] H. Fujihara, M. Goto, J. Ogata, and H. Okuno. "LyricSynchronizer: Automatic synchronization system between musical audio signals and lyrics." IEEE Journal of Selected Topics in Signal Processing, vol. 5, no. 6, pp. 1252-1261, 2011. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5876296. [Accessed: 14-Oct-2019]

[14] M. Davies, P. Hamel, K. Yoshii, and M. Goto. "AutoMashUpper: Automatic creation of multi-song music mashups." IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 22, no. 12, pp. 1726-1737, 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6876193. [Accessed: 14-Oct-2019]

[15] H. Ishizaki, K. Hoashi, and Y. Takishima. "Full-Automatic DJ Mixing System with Optimal Tempo Adjustment based on Measurement Function of User Discomfort." In ISMIR, pp. 135-140, 2009. [Online]. Available: https://ismir2009.ismir.net/proceedings/PS1-14.pdf. [Accessed: 14-Oct-2019]

[16] J. Robert, "jiaaro/pydub", GitHub, 2019. [Online]. Available: https://github.com/jiaaro/pydub. [Accessed: 16- Oct- 2019]

[17] "Mixing two audio files together with python", Stack Overflow, 2010. [Online]. Available: https://stackoverflow.com/questions/4039158/mixing-two-audio-files-together-with-python?noredirect=1&lq=1. [Accessed: 13- Oct- 2019]

[18] T. Giannakopoulos, "pyAudioAnalysis: An Open-Source Python Library for Audio Signal Analysis", PLoS ONE, vol. 10, no. 12, 2015 [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0144610. [Accessed: 14- Oct- 2019]

[19] R. Sagar, "Beginner's Guide To Building A Song Recommender In Python", *Analytics India Magazine*, 2019. [Online]. Available: https://analyticsindiamag.com/beginners-guide-to-building-a-song-recommender-in-python/. [Accessed: 16- Oct- 2019]

[20] "vivjay30/pychorus", *GitHub*, 2019. [Online]. Available: https://github.com/vivjay30/pychorus. [Accessed: 30- Nov- 2019]

[21] "Segmentino - Sound Software .ac.uk", *Code.soundsoftware.ac.uk*, 2019. [Online]. Available: https://code.soundsoftware.ac.uk/projects/segmenter-vamp-plugin. [Accessed: 04- Dec- 2019]