5-2017

Music Feature Matching Using Computer Vision Algorithms

Mason Hollis
University of Arkansas, Fayetteville

Music Feature Matching Using Computer Vision Algorithms

Mason Hollis
College of Engineering, University of Arkansas
mdhollis@uark.edu

Abstract

This paper seeks to establish the validity and potential benefits of using existing computer vision techniques on audio samples rather than traditional images in order to consistently and accurately identify a song of origin from a short audio clip of potentially noisy sound. To do this, the audio sample is first converted to a spectrogram image, which is used to generate SURF features. These features are compared against a database of features, which have been previously generated in a similar fashion, in order to find the best match. This algorithm has been implemented in a system that can run as a server, processing "query" clips as they come in and returning to the end user a specific song which matches the input query audio.

1. Introduction

As the prevalence of personal computers was growing in the early 2000s, more and more people began to be interested in the idea of identifying music digitally. Both in academia [1, 2] and with popular commercial services such as Shazam [3], the best way to accomplish this goal was a heavily researched topic. In 2008, Shazam launched its widely-used iPhone app bringing the problem into the general public's view, and inspiring even more research in the area. In contrast, research in computer vision has been a popular research topic for much longer, beginning in the 1960s when artificial intelligence research was just beginning [4]. Partially because of this, and partially

because of the wider scope of potential applications, I believe the computer vision field is ahead of the audio recognition field in many aspects. Rather than continue to develop independently, recently some researchers have begun using algorithms traditionally developed for image recognition in order to recognize music [5]. This is still a relatively unexplored approach to audio recognition but one with a lot of potential.

The goal of this paper is to establish a reliable method of using computer vision to identify the original song from a short query clip containing the a portion of the song. A classic use-case of this goal is the case of an end-user hearing a song that they don't necessarily know the name or artist of, but would like to find out. Using the algorithm discussed here, my program could take a short clip of audio as input and return to the user the name, artist, and other information related to the song. A more industry-facing use case is the case of companies such as record labels wanting an algorithm that can help identify their songs in the context of videos and audio sharing sites online in order to cut back on illegal use or piracy of their song.

This paper discusses my process for developing an algorithm designed to accomplish these goals using computer vision. Section 2 discusses my approach to preparing an audio sample to be used by a computer vision algorithm by converting it to an image representation. Sections 3 and 4 discuss the process of generating features for these images and finding matching features in the database. The success of this approach is evaluated in section 5. Possible applications are discussed in section 6. And finally, I talk about plans I have for the improvement of this approach in section 7.

2. Converting Audio to Images

The first challenge faced in this project was finding the best way to actually represent audio in a way that computer vision techniques could use. Audio is normally represented as a one dimensional signal representing volume as a function of time. Computer vision techniques are designed to work with two dimensional images, where pixel values correspond to color or brightness at each location. There are essentially only two popular ways of doing this - a standard waveform (Figure 1) and a spectrogram (Figure 2). A waveform representation is essentially an image representation of the actual sound wave. A simple, computer-generated, single tone will be shown as a sine wave as can be expected with a simple audio signal. However, when complexities such as resonant frequencies, overtones, harmonies, and multiple instruments are added in, the aggregate waveform is often only useful for seeing the general amplitude of the

Figure 1: Waveform for sample of "Me and Your Mama" by Childish Gambino

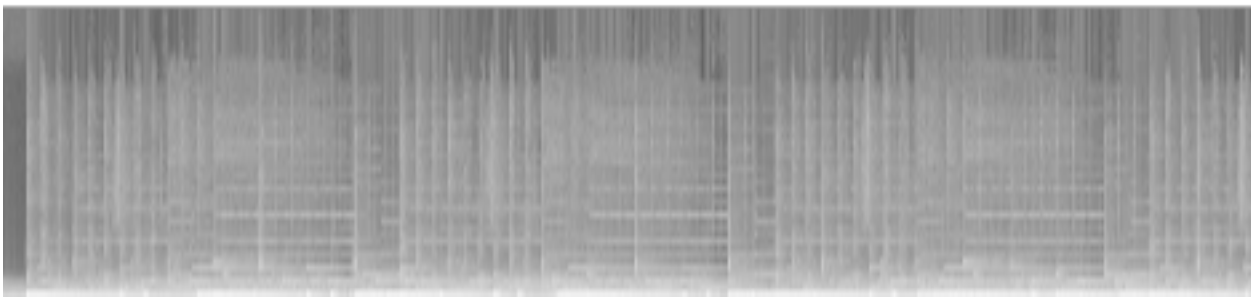

Figure 2: Spectogram for sample of "Me and Your Mama" by Childish Gambino

song. For example, this type of file is great for identifying the loudest point of a song. Without an unreasonable level of resolution for every song, this type of graph doesn't provide nearly as much information as a spectrogram, which plots the amplitude of particular frequencies across a given time. These graphs generally have time on the x-axis, frequency on the y-axis, and the amount of signal at that frequency as a color scale at that point (Figure 2). This allows us (or a well-designed computer algorithm), to see not only the overall amplitude of a song like a waveform image gives us, but also the range of frequencies that are most prominent at any time. Because of this, I chose to use spectrograms as my medium of converting audio. There are many tools already in existence that accomplish this task fairly well, given the right parameters so I won't go into too much detail here about that process. Most of them use a "Short-time Fourier Transform", which takes microseconds of an audio file at a time and finds the best sine wave functions at each "window" of time in order to find the frequencies present at that moment. In my experiments, I used pyplot from the Matplotlib library with a sampling frequency of 16 samples per time unit. I used an NFFT of 512, meaning that 512 points are used to calculate each Fourier Transform. I also used an overlap of 300, meaning that there is a 300 point overlap between each Fourier Transform. Lastly, I changed from the default Hanning window to a Hamming window, which helped provide more distinction between "dark" areas and "light" areas. This process was used to generate figures for each song that are similar to the one seen in figure 2.

3. Feature Detection in Spectrograms

Once the spectrograms were created, the next step was to generate features that the computer could use to compare two images. To do this, I initially tested two of the most popular methods used in general computer vision problems - SIFT and SURF. A recent paper [5] uses a custom feature detection algorithm based on Adaboost and Viola-Jones features, which does seem to perform well but re-implementing that approach was determined to be outside the scope of this project. Both SIFT and SURF are in the category of “blob detection algorithms” that use a “Laplacian of Gaussian” measurement to find the center of blobs in images. The Laplacian is found by adding the second derivative in the x direction to the second derivative in the y direction for each point on a gray scale image. It’s been found that the center of a “blob” in an image generally corresponds to relative maxima in the Laplacian. This method only finds extremely specific and uniform blobs in an original image. However, if a Gaussian blur is first applied to an image before finding the maxima of the Laplacian, it is possible to find larger and more generic blobs. By finding the Laplacian maxima at differing amounts of Gaussian blur, it is possible to calculate a set of key points that represent these blobs and store them in a way that can be compared to other images. SIFT and SURF then calculate edge strengths and orientations in a neighborhood around each key point and store this information in a feature vector that characterizes the local image structure near the key point. This information can be used to identify objects in other images with similar geometric properties. For the purposes of this paper, I applied the SURF algorithm directly to a spectrogram just like any other image, with one major exception. Instead of calculating SURF on the spectrogram for the whole song, I split the image

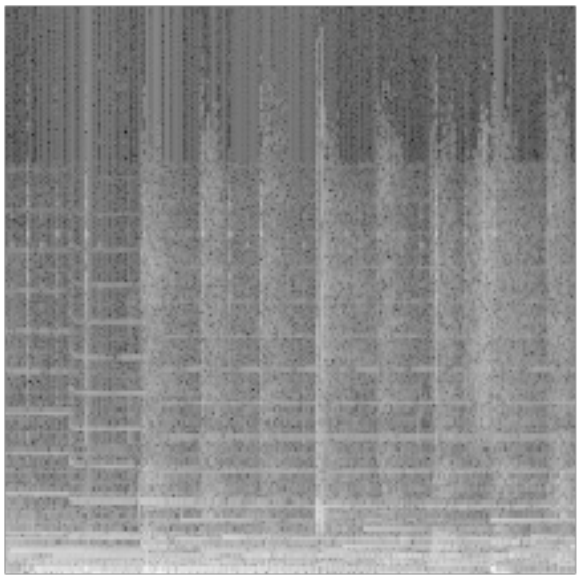

Figure 3: Spectogram representation of "Me and Your Mama" by Childish Gambino clip starting at 60 seconds and ending at 65 seconds.

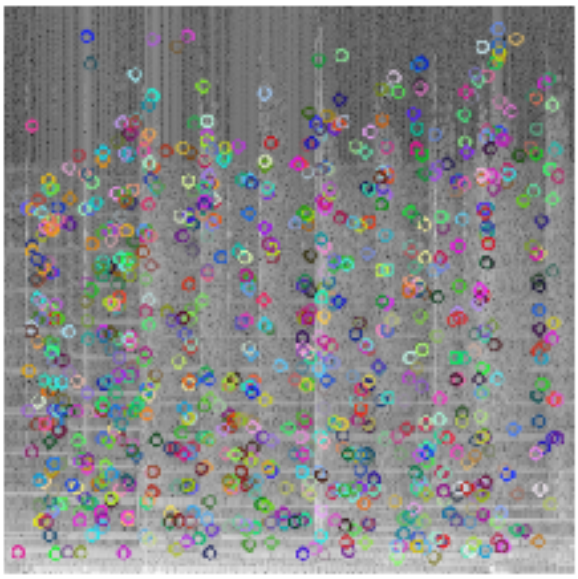

Figure 4: Spectogram shown in figure 3 overlaid with features detected by the SURF algorithm

vertically into a large number of images, each with a width that corresponds to 3 seconds on the x-axis, before finding key points on each sub-image individually. This is to ensure that the features found by the algorithm are evenly distributed throughout the song so that later on, segments from any part of the song can be matched up to these features. I do this on the fly by splitting each spectrogram as its created into these segments and calculating the SURF features while the image is in memory without ever actually saving an image file. I then save the key points into a text file with the following structure, where descriptor and keyPoint are objects generated by the OpenCV2's SURF algorithm:

```
entireSong: [
   subImage (3 second windows): [(
      descriptor,
      keyPoint: [
         point: [x, y]
         size
         angle
         response
         octave
         class_id
      ]
   )]
]
```

4. Feature Matching

The next logical step in the process is actually using these features to identify snippets of songs that are passed in. Much of this process looks pretty similar to generating the features in the original song. The incoming audio file is transformed into a spectrogram and SURF is used to calculate a list of features for the image. I then open my text file used to store features and loop through each songs' 3 second sub-images in order to find the best match. "Best match" is calculated by first running OpenCV's brute force matcher to compare the query features to each sub-image's features. It is an admittedly sub-optimal algorithm in terms of time complexity that compares each feature in the query image to each feature in the "saved" song's image and returns a list of the key point pairs that best match each other. Once we have a list of the best matching SURF features, we need to go back to the corresponding images to verify that the feature points are in the right relative spot chronologically in both audio samples. To do this, I take the top 10 matches from each image, as calculated by OpenCV's Brute Force matcher, and calculate the median time offset between "matching" features (Figure 5). To calculate the error, I then iterate through all the matching points and see how closely

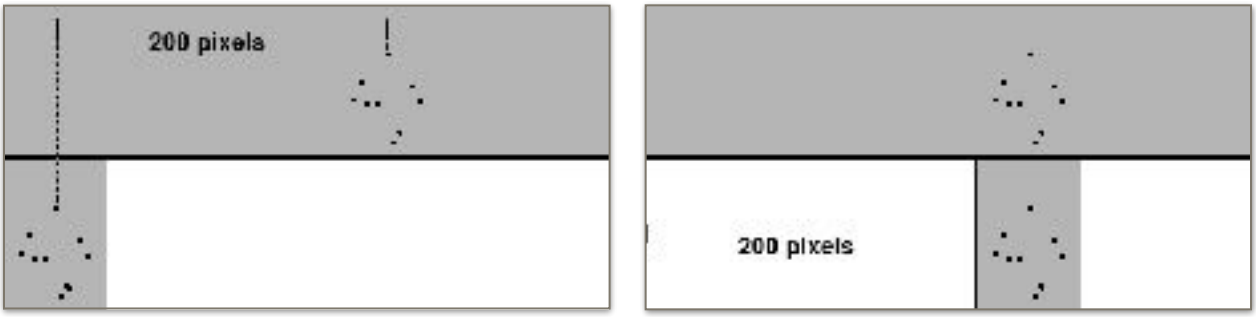

Figure 5: Simplified version of the alignment process used as the first step in determining the right relative temporal placement of key points. The left image represents the points returned from OpenCV's brute force matcher. The offset is then calculated from the median points of each set and the query "image" is "moved" so that the median points are in the same spot (right).

the horizontal positions of each point corresponds to the position of its respective "match", after the previously calculated offset is considered. After looking at each song's sub images, I return the song that had the frame with the lowest total error (as long as it is under a certain threshold; I used 50 pixels for this experiment). Figure 6 shows a graph of this error value for each 3 second frame in a matching song. Figure 7 shows this graph for a song that is not a match. The most important take-away from this is graph is the spike in Figure 6 that is clearly a lower error than everything else, getting as low 19.78 as compared to Figure 7's lowest point of 93.96. This point is indeed the segment of the song that the query wav file was taken from. The points surrounding Figure 6's low point are, as could be expected, multiples of four measures away from the actual clip and sound nearly identical to a human listener so I don't view these spikes as an error. In fact, I was surprised at the consistency that this method identifies the correct point in the song rather than a similar point in a song that involves a lot of repetition. As noise is introduced into the query audio clip, this gets slightly less distinct but it is still possible to accurately match the query to the original song up to a certain

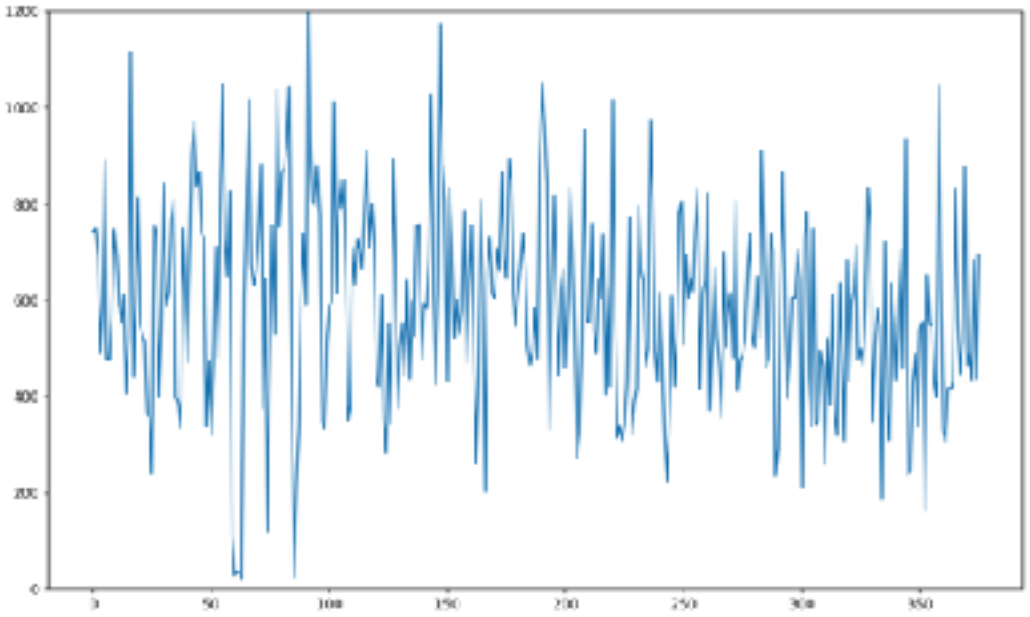

Figure 6: A graph of the calculated error value for a clip starting at second 60 of “Me and Your Mama” by Childish Gambino compared to the same song’s saved features

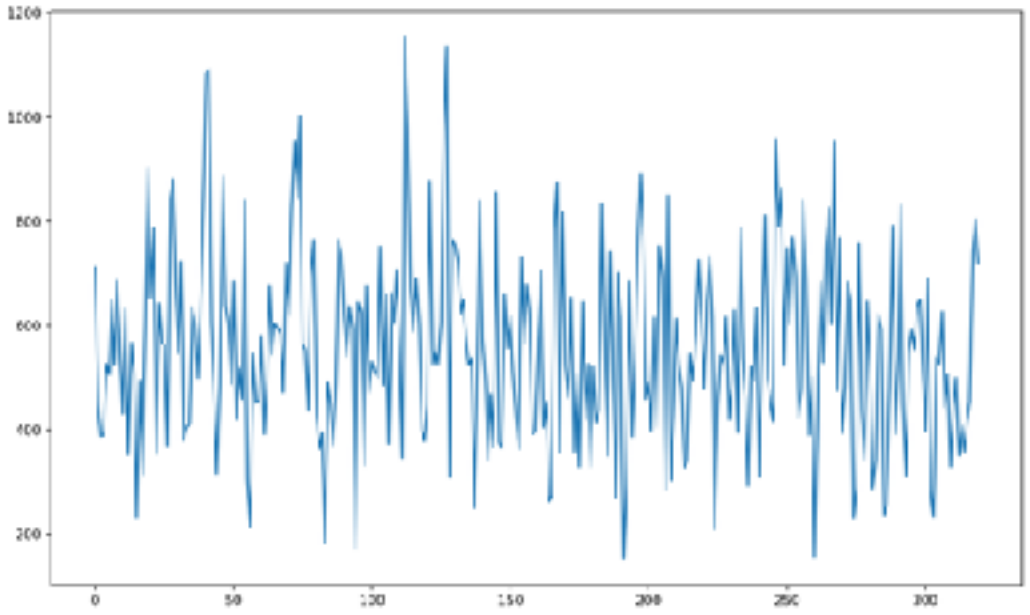

Figure 7: A graph of the calculated error value for a clip starting at second 60 of “Me and Your Mama” by Childish Gambino compared to the saved features of “Bridges” by Broods

threshold of noise. Figure 8 shows the error graph for a successful match on an audio clip that contains fairly noticeable background noise. It is still able to clearly identify the best match at 60 seconds, although the calculated error value at that point is slightly higher at 33.23. This number could be further improved by a various number of pre-processing techniques designed to remove background noise that are outside the scope of this project [8].

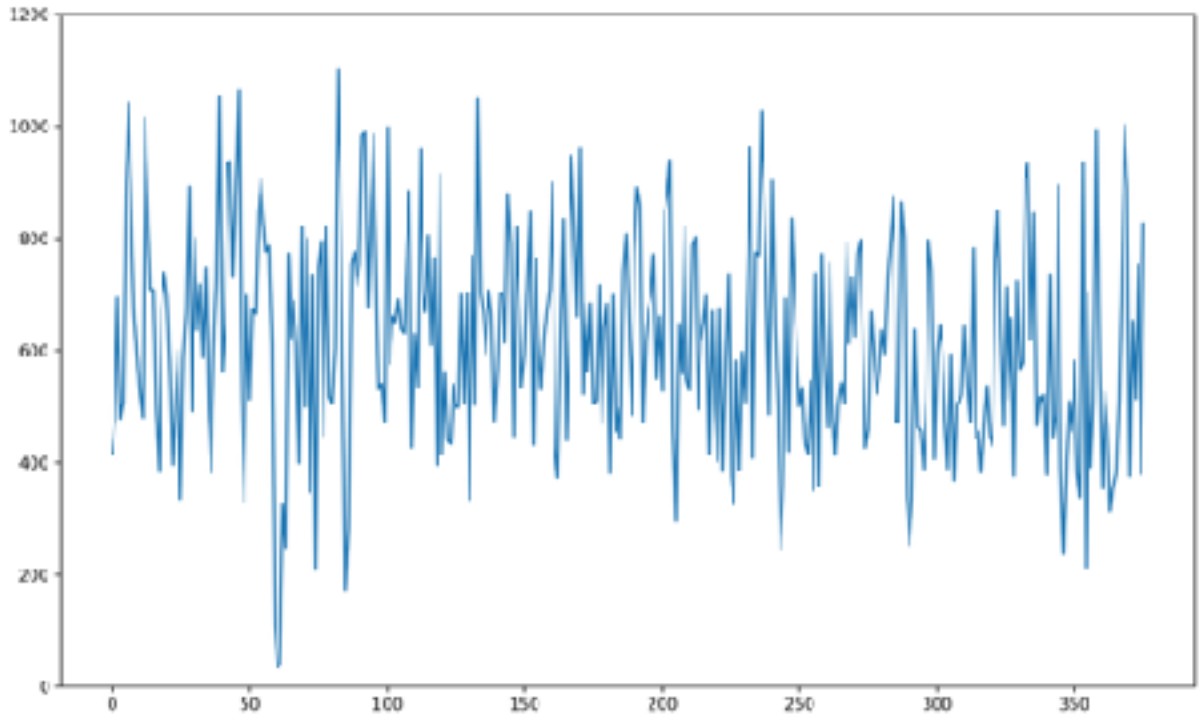

Figure 8: A graph of the calculated error value for a clip starting at second 60 of "Me and Your Mama" by Childish Gambino (with digitally added background noise) compared to the same song's saved features

5. Measuring Success

To measure the success of this methodology, I looked at two main use cases. First, any query clip that contains at least three seconds of possibly distorted audio from any of the songs in my database should be recognized as the original song, and only the original song. Second, any song not meeting the above use case should not erroneously match to a song - that is, I should not have any false positives. For the scope of this paper, I added 10 songs to my database, picked a random point from each song and gradually added noise until it was not able to be recognized. To add this noise I took an audio clip of a large crowd [6], overlaid it onto the original song file and used Audacity [7] to adjust the gain of the background noise file in order to get differing levels of noise (in 5dB increments). In every case, the correct song was recognized without background noise and with limited background noise. While the exact amount of background noise that the algorithm could handle varied with each song, and is difficult to measure, they seemed to all stop correctly identifying songs when, to my ear, the background noise volume overtook the song noise volume. For the other case, I simply input query clips that were not a part of songs in my database and made sure that none of them returned a match. I tried 10 different clips varying from 3 seconds to 10 seconds and found that none returned a match.

6. Conclusion

The results of this project establish the validity of using computer vision techniques as a viable option in the identification of music and other audio samples. While the technique described in this paper does not offer any immediate improvements over current state-

of-the-art technology for audio recognition, it does offer comparable performance using very rudimentary computer vision algorithms. Possible improvements and next steps that are discussed in the following section seem very promising in terms of offering real improvements in terms of speed and accuracy of audio recognition. Additionally, the use of computer vision in this way opens the door to looking at some of the unique benefits of that particular technology, such as its ability to identify various transformations and images in various contexts. Utilizing this kind of algorithm could move the field to be better able to recognize “transformations” of songs such as live performances, covers, and digitally altered versions in a way that audio recognition has not yet been able to accomplish. Overall, I view this paper as a success in terms of accomplishing most of what I set out to do and setting the foundation for a wide number of potential uses and improvements in the future.

7. Shortcomings and Future Improvements

7.1 Time and Space Complexity

In the future, I would like to see this code expanded in a number of ways. First, and perhaps most importantly, there are a number of ways the efficiency, both in terms of time complexity and space complexity, of the process could be improved. Currently, all the stored key points and descriptors of known songs are stored in a multi-dimensional nested list that is looped through in an extremely naive proof-of-concept sort of way. Because of this, there is a O(n^2) operation to find the best key point matches on each 3 second frame of each song in the database. This could be significantly improved using trees and/or hash tables and a more efficient matching algorithm. Additionally,

although multithreading is currently being used to analyze more than one song at a time, moving the project to a cluster of computers could significantly improve the performance boost that this provides.

7.2 Feature Detection Algorithm

Second, because the point of this project was to prove the possible benefits of using computer vision in general as applied to music recognition, I did not spend too much time proving one techniques superiority over another's. My final code uses the SURF feature detection algorithm because it was a slightly simpler implementation than other methods. However, more advanced algorithms would almost certainly outperform the current implementation so there is a lot of room for improvement in that area.

7.3 Recognition of Different Alterations

Lastly, one of my original stretch goals in this project was to use some of the unique advantages of computer vision to allow for recognition of potentially altered songs. Given my time constraints, I had to focus on the case of added noise to the original sample. However, I believe these types of algorithms could also be applied to cases where the song has been modified in other ways such as a changed tempo or a change in pitch.

8. References

[1] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of algorithms for audio fingerprinting. *In Workshop on Multimedia Signal Processing*, 2002.

[2] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. *In Proceedings of International Conference on Music Information Retrieval*, 2002.

[3] Shazam Entertainment. http://www.shazam.com/

[4] S. Papert. The Summer Vision Project. *MIT AI Memos,* 1966.

[5] Y. Ke, D. Hoiem, R. Sukthankar. Computer Vision for Music Identification. 2005

[6] Crowd Talking 1. https://www.soundjay.com/ambient-sounds.html

[7] Audacity. http://www.audacityteam.org/

[8] B. Boashash, editor, *Time-Frequency Signal Analysis and Processing – A Comprehensive Reference*, Elsevier Science, Oxford, 2003