Earworm

Authors: Anja Kalaba, Wenting Chang, Nolan Hiehle

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of identifying a song based on a simple section of melody sung into a microphone by a user. This will be implemented with a combination of matching against preformatted MIDI files using melodic contour analysis where available, and using machine learning methods on a cross-similarity matrix of a song's chroma feature representation for songs without available MIDIs.**

***Index Terms*— Chroma feature, convolutional neural network (CNN), cross-similarity, dynamic time warping, machine learning, melodic contour, MUSART, query by humming, Roger Dannenberg, signals, sound, theme extraction**

I. Introduction

THIS project makes an application that will match a human's sung or hummed input to the existing song that they were trying to identify. Often times a person will have a tune stuck in their head and will not have any way to identify it unless they can remember some lyrics to search, so the application we are trying to build will try to solve this.

There are currently very few applications that will recognize a song through human singing or humming due to the complexity of the problem. The two most common apps to achieve something to this effect are Shazam and SoundHound. However, Shazam can only match to actual songs rather than human input, and SoundHound has poor performance and vague technical details. The approach we will be taking in our project borrows concepts from the query by humming project and chroma feature analysis with convolutional neural networks, which has been done for cover song identification. These two approaches use both approaches sequentially to match the input to an existing song. The system goals and metrics are to achieve 65% accuracy on matching singing or humming to a song and to match in under 90 seconds.

II. Design Requirements

The project will be packaged as a web app. This adds some constraints to what the user's experience should be like. From start to end, opening the app should lead to a straightforward interface, from which the user sings into the microphone. Qualitatively, the user shouldn't have to extensively worry about the sound quality or noisiness of their environment, but it should be reasonable. For example, if there are multiple other voices adding to the input captured by the microphone, the accuracy of the app is guaranteed to decrease. We will test our ability to filter out audio input noise by trying out multiple queries which are all consistent along the dimensions of singer, song, time frame, and instead vary the environment of recording.

Next, as part of the experience, past a seamless vocal input, the user shouldn't expect to wait more than 90 seconds for the app's response. So from our technical point of view, we hope to constrain the system to taking only 90 seconds for completion and decision. The components of the pipeline include taking in the input and filtering it, then performing chroma feature extraction, creating a cross-similarity matrix for all possible pairings of (sung_input_chroma, library_song_chroma), using the resulting matches to filter out the library for workable songs, retaking the raw input and bucketing the input to sampled frames that are empirically optimized, running the melodic contour and DTW algorithm against all pairings of (sung_input, filtered_library_song_midi), suggested by the neural network, sorting the outputs of this analysis and selecting a final top three song output list.
Finally, the user should see an ordered list of the top 3 songs that their sung input potentially matches against. The last critical constraint we plan for is obtaining 65% accuracy, with a success defined as the truly intended song, let's call this the *target*, appearing in the top 3 list. We plan to test this across multiple singers, songs, and song time frames, and begin with a small database of about 50 songs. Qualitatively, the user should see a simple to understand list, and on top of this, a data visualization image that indicates why the song matches that were given were made. The visual will indicate which parts of the user's query lined up with the matched songs, and to what degree the music lined up.

III. Architecture and/or Principle of Operation

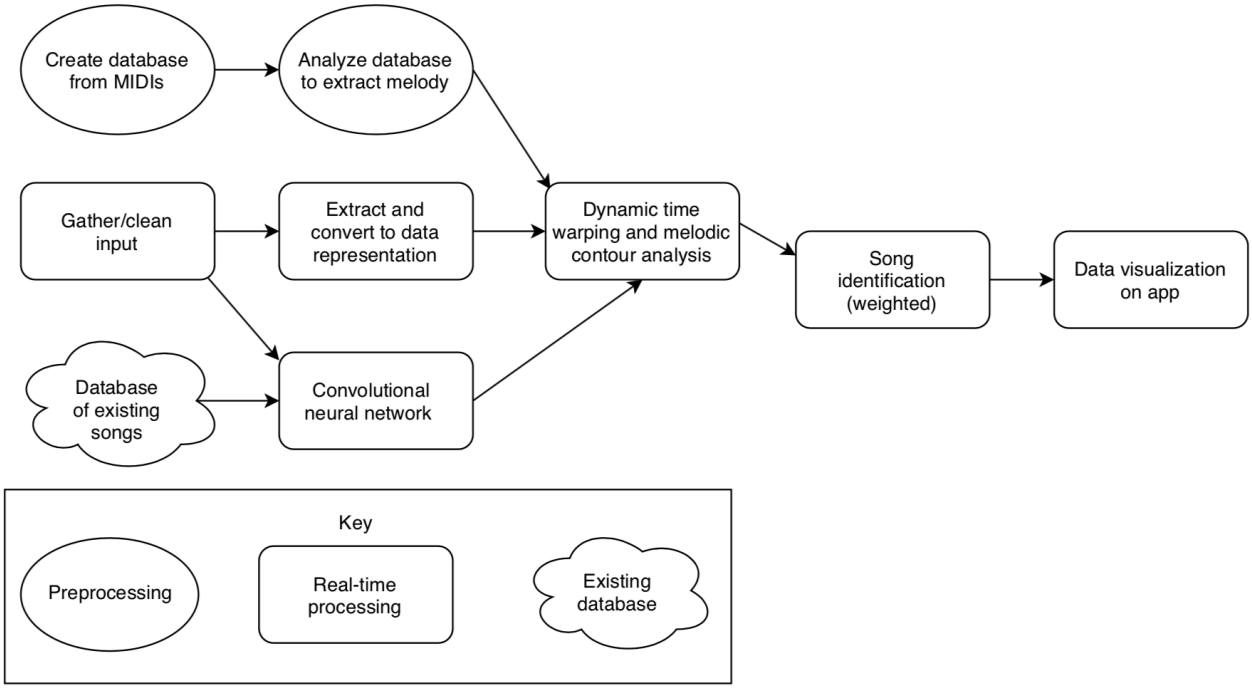
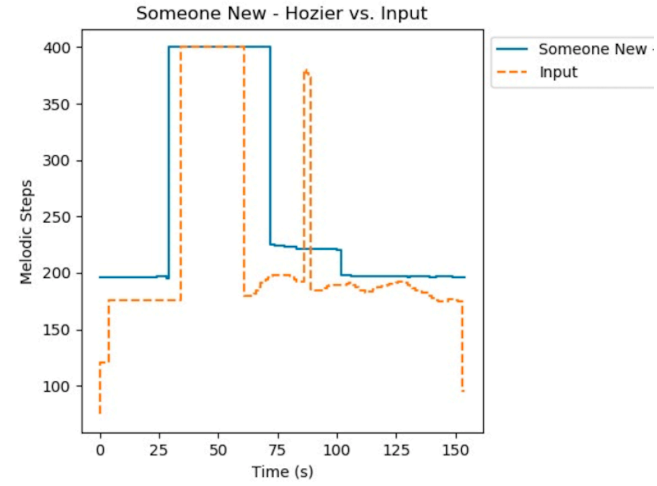


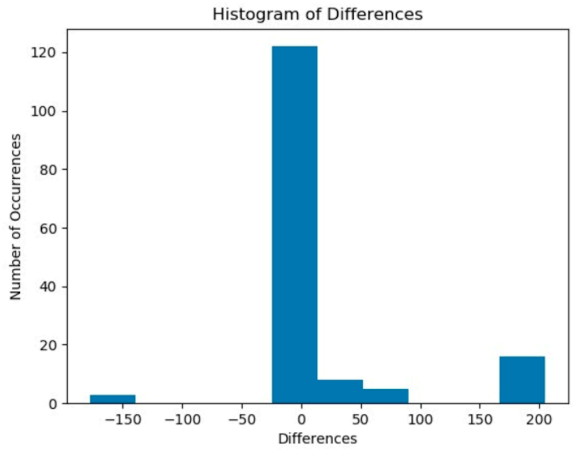
Fig. 1. System picture.

The flow of data begins with a user singing into a microphone. That input will be put through a filter to reduce background noise, and then a chroma feature plot will be generated. From this, a cross-similarity matrix is generated against the chroma feature of every song in the database. These are evaluated by the CNN to produce a set of most likely matches. The most likely matches, as determined by the CNN, are then sent to the melodic contour analysis portion of the system for further evaluation. Every song in the database has a pre-extracted melody line, which is used to compare against the melodic contour of the user input for the most likely matches. The final match is the song melody has the best match to the input sample after applying dynamic time warping.

Once this process is complete, the app shows a visualization of what was matched and how it was matched. The visualization consists of a plot of the melodic contours of the input and matched song, as seen in Fig. 2(a), and a histogram of differences, as seen in Fig. 2(b), for each of the matches.



(a)



(b)

Fig. 2. Example of data visualization (a) Melodic contour plot of matched song vs. input. (b) Histogram of differences between melodies

IV. Design Trade Studies

There are many parts to our design, including sequential sub-tracks to compute the decision more quickly. Design decisions like these were made on the basis of prioritizing efficiency and putting most emphasis on the time metric. In general, our trade offs were between accuracy and time of implementation or algorithmic time complexity, and helpfulness vs. time of implementation, and amount learned vs. time of implementation. Key categories that were logistically debated were the following:

- deciding on the algorithm to use for musical processing and comparison and potential modifications
- deciding how large to make the database
- deciding which parts of the project to borrow and which parts to implement ourselves
- deciding which test dimensions were the most important to analyze.

The design decision progression is outlined for the above compartments of the overall system in the following sections, along the debated lines of importance as mentioned before the list of key categories.

A. Algorithm and Database Size

We consulted the *Query By Humming* paper which did a proper testbench to compare multiple algorithms. As shown by their results in Table 1, the most successful algorithm was the Melodic Contour matching one, which we have selected to use in our project. The test suite for these results was fairly randomized and uniform, so it more closely resembled our expected range of input and usage. We found in another paper for it to have a mixture of 100% accuracy, also measured with the "contained-in-top-3" success definition, and 80% accuracy, for varying test suites of rock songs and jazz songs, respectively [4]. Since both of these values are above our target value of 75% for accuracy, this algorithm seemed sufficient.

Table 1. Table of MRR values for various algorithms from MUSART test bench

Search Algorithm	MRR
Note Interval	0.282
N-gram	0.110
Melodic Contour	0.329
Hidden Markov Model	0.310
CubyHum	0.093

A heavy point of consideration was the tradeoff in algorithmic time complexity and accuracy. While the Melodic Contour algorithm is accurate to the degree we wish to see, it is also significantly slow, as it is a linear search through the database per length of the input string. To hone in on a region of higher performance, we consult Fig. 3, wherein we notice that there is a plateau in algorithmic performance as a function of songs in the database. We also logically noted that having a smaller database would decrease the general search time for a linear algorithm, helping us achieve our goal of under 1.5 minutes potentially, so we were set on starting with a small database of roughly 50 songs and filter it to 25 by passing the query through the first algorithmic sub-track (the machine learning one). We also found in the paper an estimate of about 23ms on a 1.8GHz Pentium 4 Processor per song match for an algorithm that is a $O(mn)$ where m and n are string input lengths, as melodic contour will be since it tries every window frame for overlap. Considering the added runtime for the DTW on top of this, since it will not be constant but also a linear search of these lengths, we get $O(mn^2)$ so 529ms. Thus to obtain our under 90 seconds goal, our database would need to meet an optimized version of (1), with T_max set to 1.5 minutes measured in milliseconds. We found that the melodic contour algorithm actually took 1 second per song that it was compaing.

A third consideration was to include the CNN component in the algorithm pipeline first. Fortunately, training the network is the biggest bottleneck, which can be done ahead of time as a step for preprocessing, and we can make choices in our network architecture to ensure that, once trained, the network can emit classifications in a reasonably short amount of time. Our database of waveform-format songs can be preprocessed into chromagrams, so that the only calculation necessary will be:

1. Turning a sung query into a chromagram
2. Creating a cross-similarity matrix
3. Running the classifier (CNN).

Since the maximum length of a song is about 3-4 minutes, and the maximum length of a sung query will be 40 seconds, our cross-similarity matrix has some reasonable bounds on its size: we can expect that somewhere under 10,000 different distances need to be computed to create the matrix. However, as our database size increases, the model will need to be updated to account for new songs--there's no way to add a song to the classification output of the neural network without retraining the entire thing.

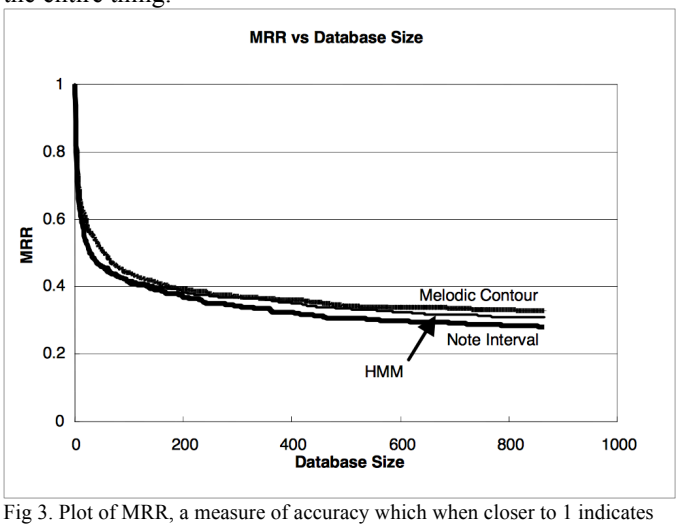


Fig 3. Plot of MRR, a measure of accuracy which when closer to 1 indicates greater accuracy, vs. database size.

$$T\_max >= 1000*database\_size \quad (1)$$

We chose to prioritize the time constraint over the accuracy one, and hence to change the original design in where the 2 subtracks ran in parallel to instead run in sequence, since our test data found that the time dimension was more off from

predicted than the accuracy dimension. So, in creating both pipelines, we found that the CNN one was very fast while not very accurate and that the melodic contour one was extremely slow (on the order of 5 minutes) but more accurate. Because of this, we went with the two-stage refinement that could speed up the more accurate algorithm but also not fall victim to its influence of being incorrect. That is, putting the 2 pipelines in sequential order has the fast but less accurate decision be made to filter out obviously incorrect songs, decrease the size of the database from 50 to 25, and this filtered database is then fed to a slower but more accurate part of the pipeline which can be sped up on the smaller database.

A final consideration was modifying the original DTW algorithm. While we initially wanted to set out to segment the song within a range of 0.5 to 2.0 (scale factor), that is bucketize the sample multiple times for a potentially better match along the time scale. If we had done this, however, we would have to do multiple queries against the same library, and then processing time would have linearly increased. So instead of going from the ideal minute and a half, we would at least go up to a few minutes. We decided this was not worth our desire to be fast, so we took this portion of the comparisons out.

B. Implemented vs. Borrowed Subsystems

Another consideration was deciding which parts would be borrowed and which parts would be implemented by us. We decided after looking at the extensive algorithm for MIDI processing in the Meek paper that the tradeoff for time of implementation was not worth doing that portion of the pipeline by ourselves [5]. Since we also were not able to interface with their code, we decided in the end to do melody extraction by hand. While initially we wanted the help of Professor Roger Dannenberg's graduate student, that did not end up being possible. Instead we came up with our own simple way that required limited resources, described in section V.5. Creating the whole library in this way took about 4 hours, which was reasonable.

The Chroma feature extraction portion was also not seen to be worth self-implementing, since successful MATLAB toolboxes existed already [2].

We don't have any equations to describe our decision making process for this category. It was instead a qualitative evaluation based on thinking about which portions of the pipeline were most vital to the integrity of the algorithms, which parts we were most interested in learning about, and which parts would take unnecessarily long amounts of time to complete. So in summary, with this in mind, we pruned MIDI file pre-processing and chroma feature extraction from our list of duties.

C. Test Dimensions

Here we discuss the test dimensions that were of most concern for evaluating project success. The dimensions we could enumerate were

- different singers
- different songs
- different time frame within a song
- duration of query
- noise-levels of environment of query

and we created tests to vary each in an isolated manner to see its performance. Since we have at least three test users, it was easy to compare a sung sample from each of us and ensure that it can generate accurate results a reasonable amount of the time.

Since we have two algorithmic paths, we performed tests on both algorithms individually as well as integration tests on the completed app, which were used in our final evaluation.

Testing different songs was also relatively straightforward once we have entered these songs into our database or trained our model on them. Once these tasks were complete, we confirmed that sung samples of melodies from each of the songs in the database can return a match. We can also test queries for the "main melody" of a song as well as other, less important musical sections, test with shorter queries that contain only a fragment of a melody, and test with queries in different environments--ideally, this would work even in a noisy, outdoor, public space.

Once our system was up and running, an automated regression testing suite was constructed using a handful of queries from each of us for several songs each. These canonical queries spanned the range of these potential limiting factors and ensured that any changes to our algorithm didn't affect our correctness guarantees (within reason). Fig. 4 depicts some outputs of matched contours during the testing process.

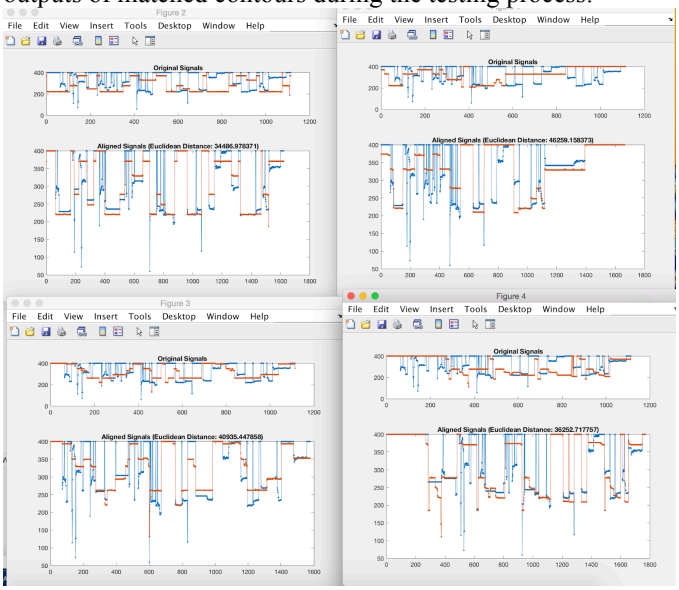


Fig 4. 4 comparisons of sung query (blue) to database entry (orange). For each of the four subplots, the top line is the original contours, the bottom plot is the aligned versions from DTW.

We found that the system was relatively robust to different singers--there were no statistically significant differences in accuracy across different individuals singing songs. However, the environment a singer is in was important. Our tests on demo day in a crowded gym were far less reliable than the preliminary tests we ran.

Our eventual product did not incorporate robust tests for either query duration (we tested queries of 30-40 seconds) or the section of a song being sung (although not every sample we tested was from the same section of the song we were matching to, matching different sections was never the focus of its own test suite). Once the system was integrated, we realized that we had plenty of work to do just to improve our accuracy numbers,

although we believe that these metrics would be important for a fully-featured product, they were a lower-priority for our team.

V. System Description

A. *Subsystem 1: Chroma Feature Analysis*

Chroma Feature analysis will be the first component of our song recognition pipeline. For this, we will convert both songs and sung samples into a popular audio processing data format known as a Chroma Feature (or chromagram). This is more or less a variant of a normal spectrogram: data is converted from the time domain into the frequency domain, and then is bucketed amongst the 12 notes of the Western chromatic scale.

Chroma Feature also combines all octaves into a single, 12-element vector, where each element corresponds to the total intensity of a note across all octaves. This is perfect for our purposes, since we want to allow for singers with different ranges.

There are several interesting post-processing techniques that can be applied to a chromagram [2], such as normalization along a single 12-dimensional chroma vector, normalization across different chroma vectors, changing note intensity to be represented with a logarithmic versus a linear scale (closer to how sound intensity is perceived by the human ear), and performing windowing functions across neighboring chroma samples to smooth notes out (this has the effect of removing much of the timbre/instrument information). These, plus the problem of how many samples to create in the first place, leave a lot of tuning even in the chroma feature step of this matching algorithm. For our system, we selected CENS (chroma energy normalized statistics), which takes averages over relatively large time windows. This is meant to introduce invariance to local tempo variations, such as those caused by different articulation or singing at slightly different speeds.

These Chromagrams will then be subjected to cross-similarity analysis. The cross-similarity matrix will essentially show the distance between EVERY point on the sung sample and EVERY point on the reference song. If the song is a match, it should be more likely to produce interesting patterns, like diagonal lines of similarity over time, and blocks where many adjacent samples are all similar to one another. A convolutional neural network (CNN) is trained to recognize these patterns in the cross similarity matrices and then to determine the extent of a match between the input song and all the songs in the library. In this step, half the library is eliminated as potential matches, and the remaining half is passed to the second subsystem as its starting library for melodic contour matching.

B. *Subsystem 2: Dynamic Time Warping and Melodic Contour Analysis*

The second component of our pipeline will be discretizing the input and applying a melodic contour / dynamic time warping (DTW) algorithm to the input. The first objective is to create a sequence of notes/pitches from the input. The input is to be sampled at 44100Hz, as this is a good standard. With an experimentally determined optimal window frame of 80ms, we use the Power Spectral Density function to find the intensities for each possible frequency. We analyze peaks within these PSD plots and find the maximum to determine the most likely pitch (frequency) for that window. Since this granularity is quite high, afterwards, the input is bucketized. We create a larger window that is on the order of one one thousandth the length of the query and one ten thousandth the length of the library entry, and for each frame that the sample audio is broken down into, a representative mode frequency is to be assigned to that frame, and thus a sense of bucketing the input is applied; we can call this creating a melodic contour on the input. The representative frequency for each sample frame is to simply be assigned the pitch of the note that occupies the most space in the frame. From this we obtain our sequence of n pitches for the query.

This same processing input, of creating a sequence we call the melodic contour, is to be applied to each song in the database as well. This procedure is to have been done ahead of time, so that with each song in the fixed database, a sequence of the melodic contour already exists as data to sift through. In detail, here is how we create and preprocess our library. It was done in two ways. The first was looking up existing MIDI files that could be opened with MuseScore, and the second was simply finding existing arrangements of the songs on a MuseScore database, from which we could download the MuseScore representation of the sheet music. One part of this that took a while was making sure the melody was correct and representative of what the song sounded like in common use, and this was simply done by ear. The biggest bottleneck for this process was isolating the melody then. Potential problems included the melody being polyphonic, and thus deleting extra parts, the melody coming in with extra instruments of the song, and thus muting the extra parts, the melody being split among multiple staffs and having to isolate regions of just the melody, and the melody being slightly inaccurate and correcting it by hand in the MuseScore program. After an accurate version of the melody was extracted in MuseScore, the instrumentation in the program was changed to ‘Grand Piano,’ the synthesizer setting were set to the dryest possible ones (turning up HF damping all the way, decreasing reverb to zero), and the tempo of playback was boosted by about 10% of the original song speed (even more if the original score was off tempo). And then from MuseScore, the song was exported as a WAV file, and the pitch extraction algorithm was run on this to obtain the data representation of the melodic contour.

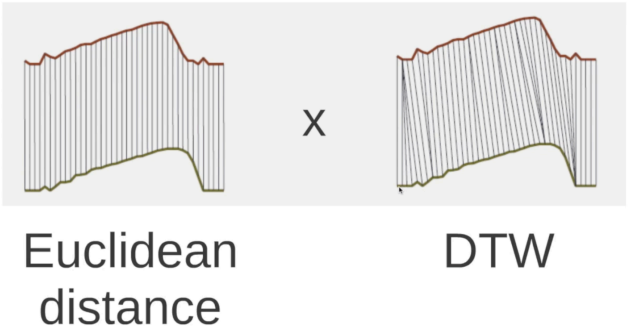


Fig 5. A visual of how the data across 2 time series are lined up differently between Euclidean distance and DTW comparison methods.

Next, we describe the concept behind dynamic time warping. The use of this comparison technique is to find a more intelligent measure of distance between 2 time series data sets. Instead of a simple naive Euclidean distance approach, DTW uses a matrix with each song along each axis to instead find which parts of one series more closely resemble other parts of the other series. This can be visualized in Fig. 5,
which clearly shows that DTW lines up parts of the data series which are more similar. Thus overall, using this approach is more helpful in finding melodic similarities considering that factors such as temporal alignment or scaling (which preserves shape but not length, but with DTW would be noticed), which shouldn't impact the accuracy, can now be naively dropped as influencing the search. Thus overall the match is more based on melodic integrity.

So, within our framework, the DTW algorithm, which is computed using a distance matrix and from this distance matrix finding the optimal/cheapest path to the final point, will be applied between the pitch vectors from the input and each song in the database. Moreover, it will be applied to each pair of input and database song for each n offsets of the data points, in a windowing effort. Then, with all these comparisons, an overall distance score will be computed, and the scores will be sorted in by smallest distance for all the possible combinations of matches between audio input and database song for all possible offsets. The top 3 smallest distances for distinct songs will represent the closest matches of the query to an element in the database.

C. *Post-processing: Data Visualization*

There will be a data visualization portion to see what the matching algorithm did regardless of whether or not it is able to generate a match. This will also serve as a form of visual debugging for the system so that we can see whether the system is generating garbage matches or whether the melodic contours do in fact seem similar. The two visualizations that are produced for each of the top three matches are (1) a plot of the input melodic contour and the match's melodic contour and (2) a histogram showing the distribution of differences between the contours of the input and match at each point in time. The contour plot is a very intuitive way of seeing where the singer was correct or incorrect as well as where they can potentially improve to sing the song correctly, assuming the song is matched. If the song is not a match, it is interesting to see what parts of the song resulted in the algorithm determining that it was a close match.

D. *Front-End Web Application*

The web application uses React for the front end and a Python Flask server for the back end. The system diagram and data flow can be found in Fig. 6. A user is able to start and stop recording themself singing, and when they press stop, the audio is sent to the Flask server via an HTTP POST request. The audio is saved into a .wav file which is then used for chroma feature analysis and DTW/melodic contour analysis. The results of the melodic contour analysis, which is the last step in the matching process, are written into a text file. These results are then read back out of the file and parsed to get the melodic contours, both of the input and matched songs, and song metadata. The contours and metadata are then passed on to generate the contour plots and histograms.

After the plots are generated, the results are sent back to the front end as JSON data. The front end then sends GET requests to the Flask server to retrieve the contour plots and histograms for each of the top three matches. Once this is complete, the web application then displays the results ranked 1-3, and the user can click to expand or collapse the plots for each matched song. Finally, they can submit a new query and begin this all over again. An illustration of the UI along with the data visualizations can be seen in Fig. 7 and Fig. 8, respectively.

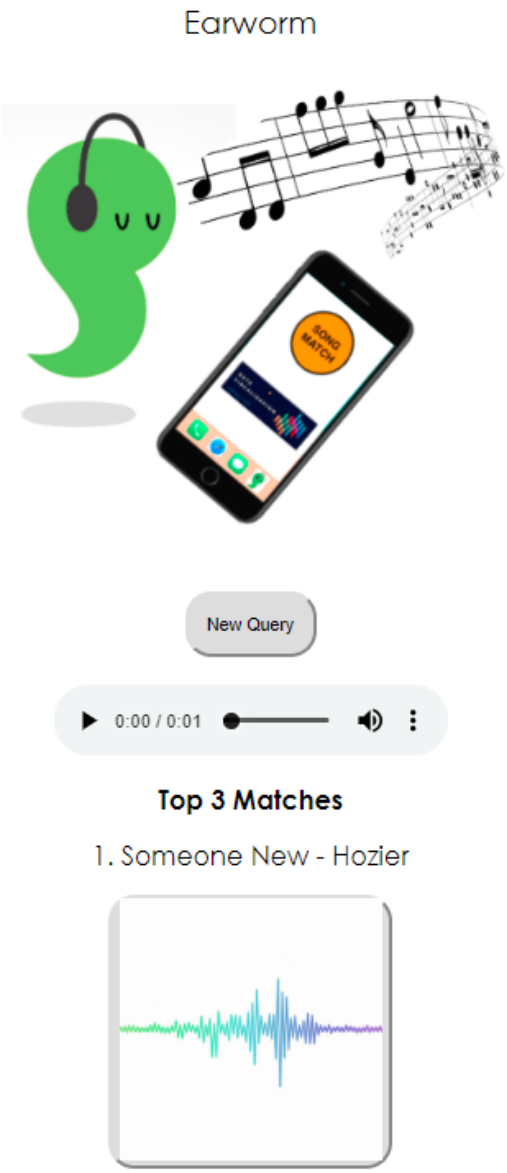


Fig. 7 User Interface of web application

VI. Project Management

A. *Schedule*

The schedule can be viewed in Fig. 3 at the end of the document. This project began with researching similar existing projects and contacting professors for their advice and

suggestions for our process. Using this information and advisance, we decided on the approach to take and began implementation.

The dynamic time warping and melodic contour analysis portion began with the pre-processing of MIDI files and filtering background noise from human input, and then proceeded to matching the input against the existing songs. The chroma feature analysis began with converting songs into the chroma format and examining the similarity matrices, and then training convolutional neural nets to recognize the similarities. Before training, however, all three of us had to sing the 50 samples from the database in order to have images to train on. Testing for both branches of the matching process was done separately for each portion to see their individual performance and then integrated together for the complete system, which was were we changed the pipeline to run the subtracks in sequence rather than parallel.

The data visualization and application portion begin with research and testing of existing data visualization methods to see what could be used and what concepts could be borrowed to create our own visualization technique. The design and creation of the app took place later in the process. After testing in use, interface refinement took place for the app too.

B. Team Member Responsibilities

Anja was responsible for the dynamic time warping and melodic contour analysis portion of the system. Additionally, she will be working on creating the database of songs that the system will match to.

Nolan worked on the chroma feature analysis and training the CNNs. This will involve generating the cross-similarity matrices for chroma features.

Wenting worked on the data visualization aspect of the project and the design and creation of the web application.

C. Budget

We did not use our budget.

D. Risk Management

The major risk of this project is its performance. Since there are no hardware dependencies, the points of failure are all in the algorithm not working or being very slow. We were aware of the risks from the beginning after looking at the performance of existing projects. For example, the original plan was to do polyphonic pitch tracking and break down sound files by ourselves, but that was found to be too difficult, so we switched to the back-up plan of just using existing MIDI files for analysis.

Our original design included using two parallel paths, the CNN and the melodic contour matching, which are detailed in Section V.A and V.B, in order to produce a match score weighted between the results of the two paths. Our thought had been that doing the matching twice would hopefully produce a better result, and we had yet to see how accurate each of those branches would be. However, we realized that the melodic contour analysis would be very slow, and it would be infeasible to perform it with all the songs in the library and still meet our time requirement. Making that branch considerably faster would require considerable effort and time that we believed was better spent in other aspects. We also found that the CNN was not as accurate in our problem space as it had been for cover song identification, but it was still very quick. With these two pieces of information, we decided to alter our design to make them sequential steps, trading off potentially better accuracy for speed.

Another risk that we encountered was having enough time to complete all of our tasks and to a high enough standard. The research portion of our project took more time than expected, but from our research, we found that some of the things we had allotted time for would not take as long as originally predicted. Therefore, we were able to alter our schedule to still complete the project on time and create a viable product.

VII. Related Work

Since the problem we are setting out to solve has no well-accepted solution, we are drawing most of our inspiration from related work in the computer music community. Our query by humming path featuring dynamic time warping and melodic contour analysis draws from a query by humming paper by Roger Dannenberg, a CMU professor [1], [4]. The area of extracting melody from music is a topic of much current research. Most melody recognition on waveforms is not advanced enough to handle a popular song, but there are several methods for extracting a melody from a MIDI file. One of these was used by Prof. Dannenberg in his query by humming paper [5]. While this program is no longer under development, we are hoping to find a program with similar features that is currently available. There are many such programs being developed by computer music research groups around the world. Most extract melodies by identifying repeated sections of music to return a list of potential important themes.

Chroma feature analysis is a thoroughly-researched topic, and many variations of it and post-processing techniques have been proposed for different applications by different research groups [2]. Also, chromagrams have been used to monster mash songs together in many different contexts: work has been done on matching a chromagram of a song fragment to the corresponding location in the song [3]. This team also considered matching songs to their covers, but they worked within a pretty specific library of different recordings of classical pieces--each “cover song” had the same orchestration and music, but different performers and conductors. Because of this, we should expect a match on our system (which will be guaranteed to have different music and performers, and will most likely also have some slight tempo variations) to look very different than a match on their system. Work has also been done using cross-similarity matrices and neural networks to identify cover songs [6]. This work also differs from ours because the matching is on two complete songs, with (presumably) complete instrumentation and also with all melodies and sections of the song intact, while our plan is to match a full studio-recorded song with a solo vocal artist who will only sing one important melody, or perhaps only a fragment of one.

The only commercial system we are aware of that performs a task similar to ours is called SoundHound. This is a mobile song-recognition app that offers features similar to Shazam (identifying a studio recording), as well as being able to recognize user-sung melodies as our project does. There is no data available (to our knowledge) on how SoundHound is engineered and what techniques they use to generate matches.

While SoundHound's library of melodies is much more extensive than ours, some experimentation with the app suggests that they use different techniques for matching sung samples vs matching full recordings. Very popular, well-known songs could mostly be matched via singing into SoundHound, but some slightly more obscure songs could only be recognized via playing the studio recording. Anecdotally, SoundHound also often confuses cover songs or different versions of a song with the song they are replicating, which suggests that even their algorithms for recording to recording identification incorporate some melody and theme analysis (or, at least, more than Shazam's do).

VIII. Future Work

There is lots of room for our improvement in our current system. The CNN portion of the algorithm was based on the cover song identification work in [6], and our CNN system mirrored theirs very closely. In addition to the obvious improvements of having a more robust and diverse set of training data (our training set consisted of only our 3 team members singing the songs in the database), there was probably some room for improvement in the system setup itself. The neural network was trained to classify match vs. no match, but the actual metric that mattered was that the correct song have the highest probability of a match. This led to clunky iteration patterns of training the network and then having to run a completely different set of tests to determine if it was working well. Future work should look into changing the loss function of the CNN so that we can train it to more closely match the metrics we care about--perhaps instead taking in a single cross-similarity matrix and spitting out a binary classification of match/no match, it could accept a set of matrices for every song in the database and emit a probability value for %match of each song.

Another area for improvement is our sequential algorithm: first CNN analysis followed by DTW analysis. This was a good decision for our project considering our constraints--specifically, neither of our two algorithms was quite robust enough to support the entire project on its own. Tying them together helped us to get a demo-able product, but is not really the best way to do things. Notably, this serialization of our algorithms means that instead of shoring up one anothers' weaknesses, the two approaches are now both affected by each others' shortcomings: for example, a song with poor training data in the neural network will rarely if ever be submitted to the DTW section, and a song with poor DTW performance will not be returned as a result even if the neural network can recognize it. A better solution should run the two algorithms in parallel and incorporate some sort of voting layer to combine the results and produce a final match.

As for the dynamic time warping, we see some potential for improvement in this area. For example, DTW does a great job at aligning the contours to the best of their capability. But the scoring upon this can be interpreted in many ways. For example, if the alignment snaps the two contours together, and the difference was they had purely identical shapes that were shifted apart vertically, this would ideally show a zero distance. As compared to a song that is definitely wrong, but the sum of its aligned differences might be smaller than the previously described one. So there are clear outliers where the distance is not a good evaluation tool. On alternative method we considered approaching was taking the two aligned samples and finding the cross correlation function. If the two samples were the same, this would be the same as finding the autocorrelation function. One key characteristic about the autocorrelation function is its symmetry. So finding an evaluation technique based on the symmetry value for the cross correlation of the dynamically warped and aligned samples might be one area of scoring to look into in the future.

A final potential for improvement is to decrease the search space. Currently, the algorithm compares the sample with very small offsets against the longer database entry, in an attempt to find all possible windows of match. A more intelligent approach might prune out possible matches found previously, or optimize the hop size so that the minimum amount of comparisons between the query and database entry can be made.

IX. Summary

Our system was able to meet our design specifications, although there is much that could be improved to do much better than the original specifications. The CNN step was fast but not accurate, while the time bottleneck was in the melodic contour matching. It was possible for the CNN to eliminate the correct answer from the library before the melodic contour matching step, which meant that it was impossible to be correct after that point. We acknowledged the tradeoff of this aspect of our design, and improvements to the CNN could help improve the app's performance.

Throughout this process, we have learned a number of valuable lessons. We received helpful advice from multiple sources: the people who wrote the cover song identification with CNNs paper [6] and CMU faculty and graduate students (Richard Stern, Raymond Xia, Tyler Vuong), which goes to show that people can be quite helpful in assisting others' research endeavors, so one should not be afraid to reach out to all the resources for help. We also learned to be flexible in our design and timeline. Our design changed multiple times throughout this process, and the final design was able to meet our desired metrics better than the original design ever could. We evaluated the tradeoffs in time and accuracy to reach our final design and became better engineers along the way. The original timeline did not allow us enough time for the amount of research that we ended up doing, though this also allowed us to better understand what we were doing. We also did not leave enough time for slack; some of the tasks we allotted time for in our schedule did not take as much time as anticipated, so we could have had more time at the end for testing, debugging, and further intellectual discussion.

References

[1] R. Dannenberg et. al, "A Comparative Evaluation of Search Techniques for Query-by-Humming Using the MUSART Testbed" http://www.cs.cmu.edu/~rbd/papers/musart-testbed-JASIST-2007.pdf

[2] M. Müller and S. Ewert, "Chroma Toolbox: MATLAB Implementations for Extracting Variants of Chroma-Based Audio Features" https://www.audiolabs-erlangen.de/content/05-fau/professor/00-mueller/03-publications/2011_MuellerEwert_ChromaToolbox_ISMIR.pdf

[3] M. Müller, F. Kurth, and M. Clausen, “Audio Matching Via Chroma-Based Statistical Features” http://resources.mpi-inf.mpg.de/MIR/chromatoolbox/2005_MuellerKurthClausen_AudioMatching_ISMIR.pdf
[4] D. Mazzoni and R. Dannberg, “Melody Matching Directly From Audio” https://www.cs.cmu.edu/~rbd/papers/melodymatching-ismir01.pdf
[5] C. Meek and W. Birmingham, “Thematic Extractor” http://ismir2001.ismir.net/pdf/meek.pdf
[6] S. Chang, J. Lee, S. Choe, and K. Lee, “Audio Cover Song Identification using Convolutional Neural Network” https://arxiv.org/pdf/1712.00166.pdf

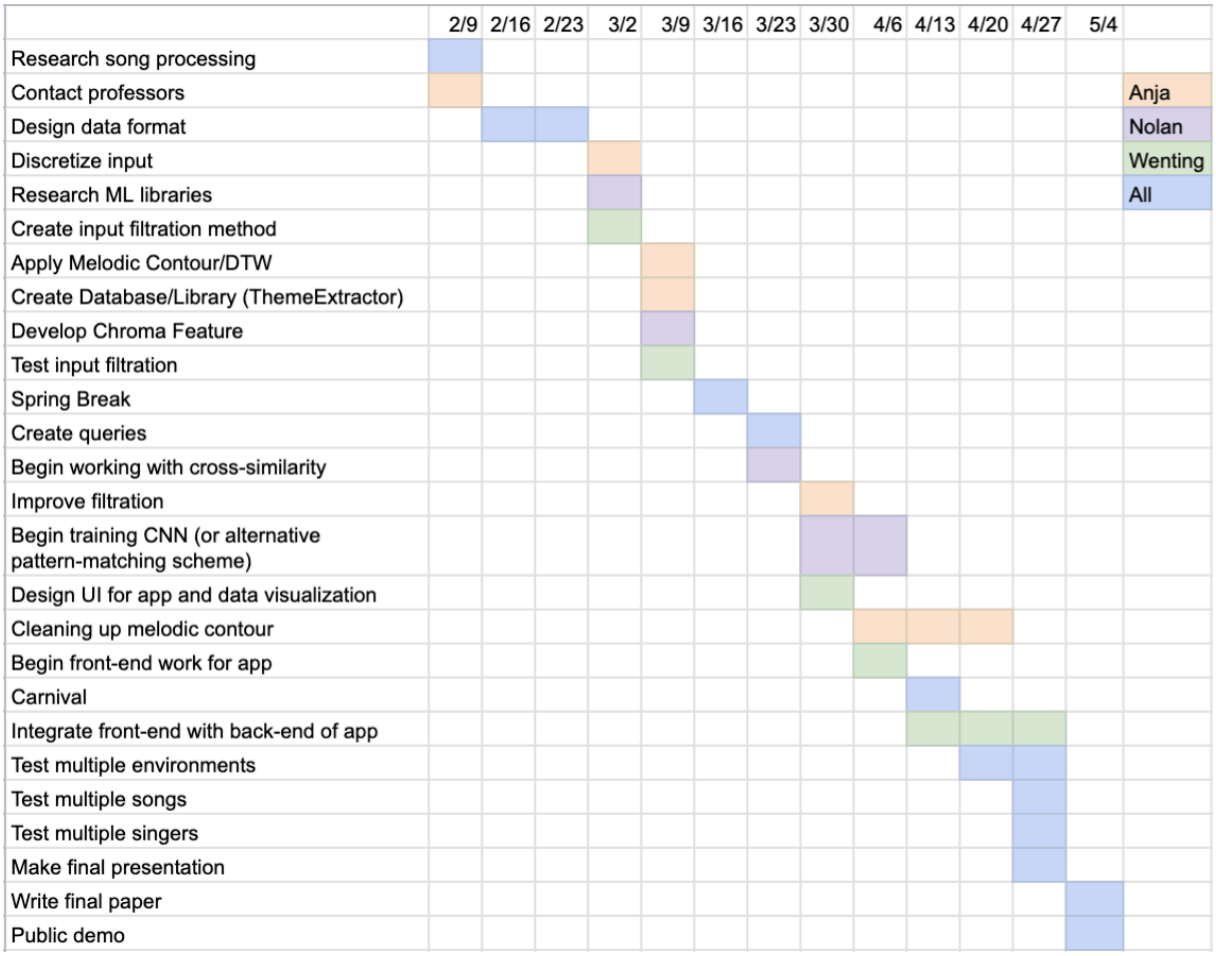


Fig. 3. Gantt chart of the schedule and division of tasks

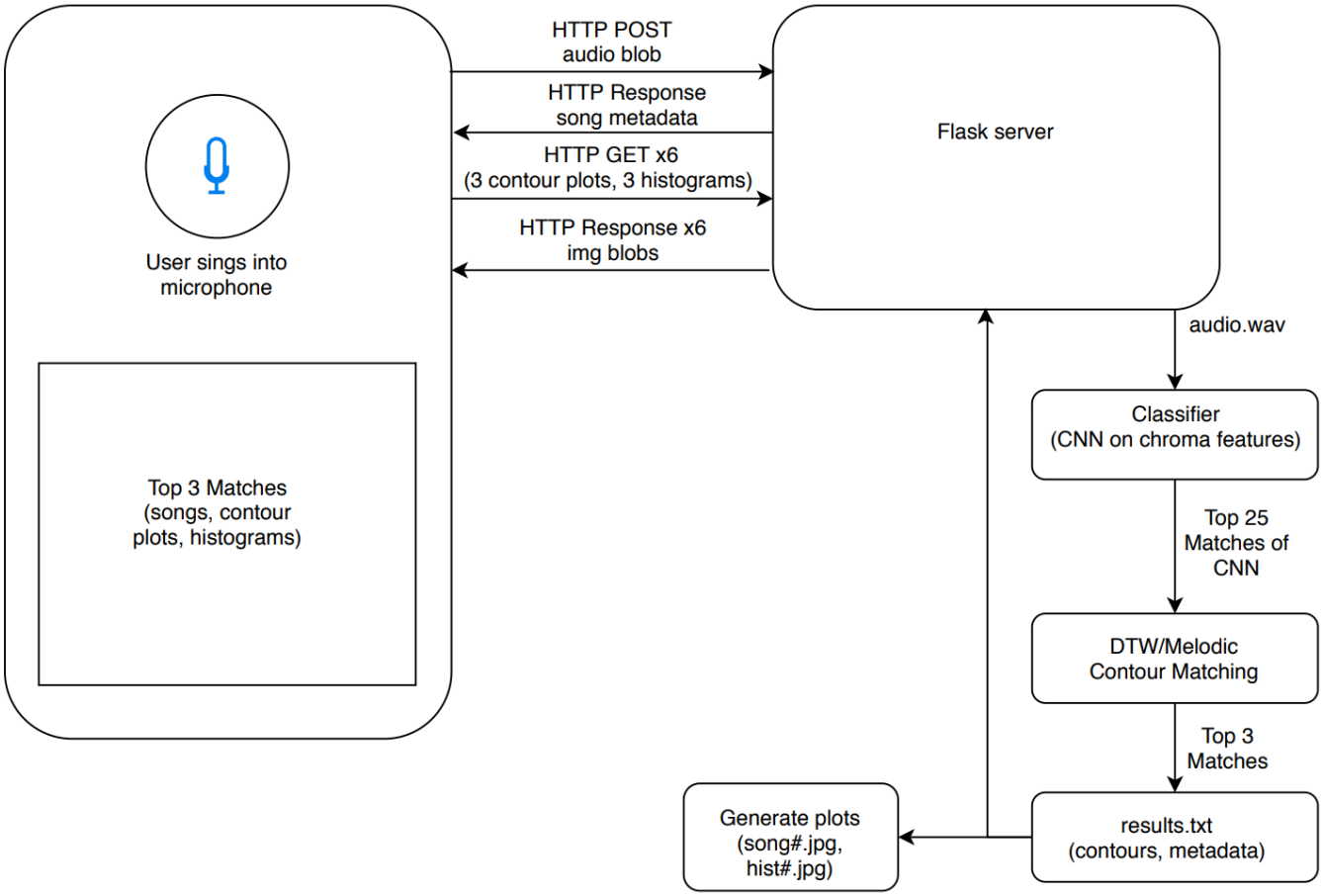


Fig. 6 System diagram and data flow of web application

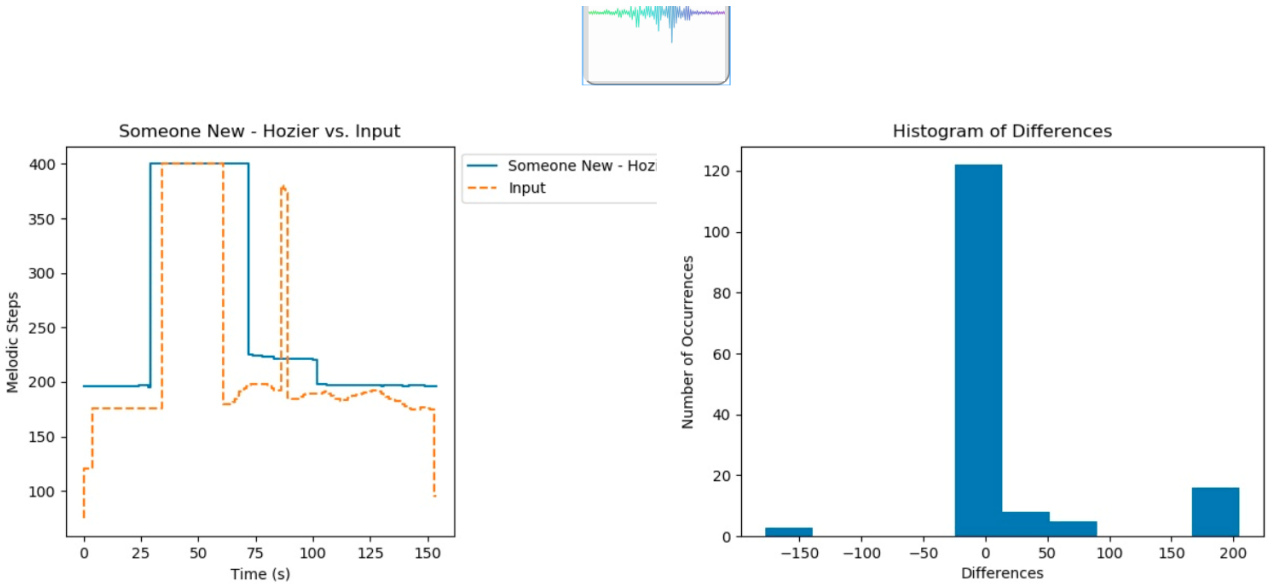


Fig. 8 Data Visualization: Melodic Contour of Match vs. Input (left), histogram of differences (right)