CS 4701: HUMmer

Shashaank Aiyer, Manvir Chahal, Neeharika Kotimreddy, Arnav Parashar

saa244, mkc222, nrk59, ap736

AI Keywords: Machine Learning, Sound Recognition, Convolutional Neural Networks

Application Setting: A web-based application that identifies songs from snippets of humming

Helpers: Likita Gangireddy: lg425 (hummer), Tejas Shah: tcs222 (hummer)

Video:

Part 1: Project Description

Core Idea

The goal of our project HUMmer is to develop an application capable of identifying songs based on short segments of human humming. Motivated by our passion for music and specifically the artist The Weeknd whom we all share a liking for, and targeting users who often find themselves unable to recall the title of a song they are humming, we designed our model to recognize songs from his discography. Initially, HUMmer aimed to cover three of his most popular albums–After Hours, Starboy, and Dawn FM–but we later narrowed our scope to 12 songs from Dawn FM (excluding Dawn FM, Every Angel is Terrifying, and Phantom Regret by Jim) and five additional tracks from his most popular songs–Blinding Lights, Starboy, Save Your Tears, I Feel It Coming, and Die For You.

We designed HUMmer such that interaction with the application is intuitive and seamless. Users access the HUMmer user interface where they can record a 10-second snippet of them humming their chosen song. This recorded humming input is sent to our backend server which converts the audio into a spectrogram and feeds that into our trained convolutional neural network, which processes the input and generates a prediction of the most likely song from the data set. The name of the predicted song is then displayed to the user, and users can continue to hum additional songs to be inputted as additional queries, with each new query and corresponding output being displayed on the screen. The UI also allows for users to upload 10 second hums of a specific song so that there is more training data for the model.

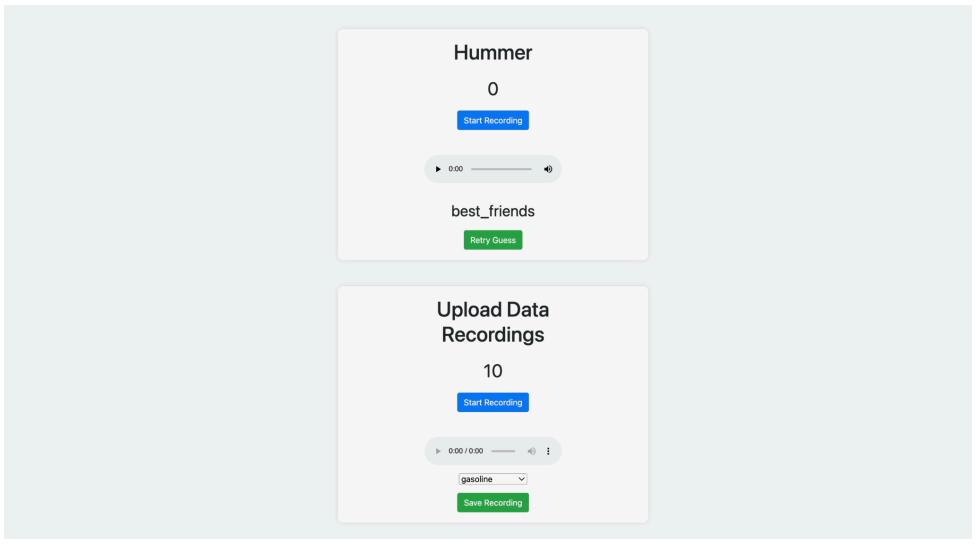

Fig. 1 HUMmer User Interface

Data Collection and Augmentation

We initially began by collecting audio recordings of people humming each of our chosen songs in their entirety. We then developed a program to partition each recording into ten second segments. The rationale for this segmentation process came from the fact that our model is aiming to predict songs off of short recordings of the users humming snippets of songs as opposed to songs in their entirety.

Unfortunately, we did not have enough quantity and variety with just our own recordings to be able to build a very robust model. Thus, we augmented our dataset by adding recordings from our friends, Tejas Shah and Likita Gangireddy, as well, enabling the model to generalize better across various users. We concluded by humming and adding specific short recordings of popular portions of songs in an attempt to simulate the way a user would interact with our project.

Data Formatting and AI Considerations

After creating our dataset of song recordings, we had to format our data for our model of choice. Initially, we took inspiration from Google's Hum to Search feature, where they create paired vector embeddings of hummed and studio recordings of songs. This technique involves transforming humming inputs into high-dimensional vector representations and then converting both the hummed input and corresponding studio recording into embeddings that capture the essence of the audio signal. These paired embeddings are then passed through a Deep Neural Network trained to recognize relationships between the hummed and studio recordings. This method is quite precise but leverages a large amount of preprocessed data and recordings in addition to a proprietary embedding algorithm, both of which present significant challenges for a smaller-scale project like HUMmer. We felt that we did not have enough songs or data for this to be successful.

Instead, we opted to take a conventional approach by creating image representations of our data. First, since our recordings of each song were initially .mp3 files, we built a system that automatically goes through each original recording, saves it as a wav file for processing purposes, then splits each wav file into multiple recordings of 10-second increments, and saves each in a folder for that specific song. The system then converts each of our 10-second recordings into a Mel Spectrogram, which is an image containing information about the progression of frequencies in the audio signal over time. Each spectrogram is saved in a folder for each song respectively, making it easier to inform the PyTorch dataset which class each image belongs to. Thus, our subsequent convolutional model differentiated between songs via their Mel Spectrogram representations. Our system allows for a more organized approach and the ability to simply record a song and have it added to our data automatically.

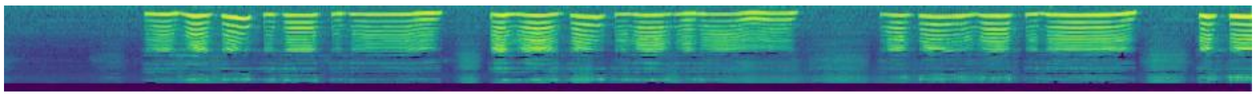

Fig. 2 Blinding Lights Spectrogram - Manvir

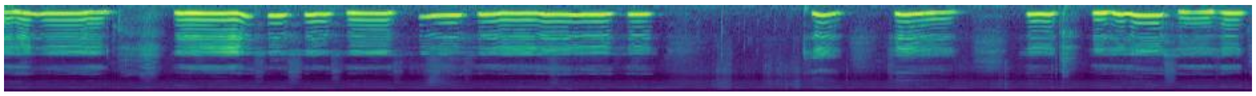

Fig. 3 Blinding Lights Spectrogram - Neeharika

Above are the spectrograms for the same 10-second segment of “Blinding Lights” hummed by Manvir and Neeharika, respectively. Notably, there are discernible differences in the manner in which each individual hummed the song. While the overall patterns exhibit similar rises and falls in corresponding regions of the song, both individuals introduced unique variations in their renditions, which elongated and shortened different parts of the spectrogram. These variations highlight the individual differences in vocalization, which present challenges for consistent audio pattern recognition.

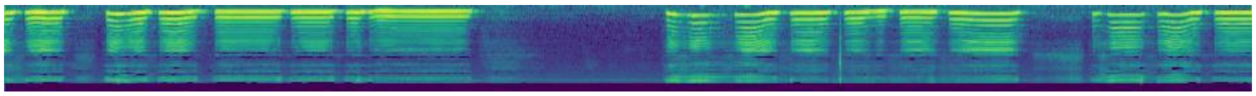

Fig. 4 Less Than Zero Spectrogram - Manvir

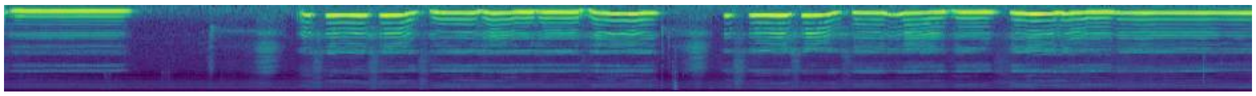

Fig. 5 Less Than Zero Spectrogram - Neeharika

Similarly, above are the spectrograms for the same 10-second segment of “Less Than Zero " hummed by Manvir and Neeharika, respectively. Here, the differences in their humming are quite pronounced. In Manvir’s spectrogram, we see fewer gaps in the graph, suggesting that she hummed both the song and the instrumental sections, whereas in Neeharika’s spectrogram, there are various gaps in the graph, suggesting that she focused on humming only the lyrical melody. This exemplifies situations where users may remember and hum the tune of lyrics while others may hum the background music. Diversity in humming patterns expands our dataset, and training the model on this data helps to enhance the model’s ability to accurately identify songs based on users’ varied humming inputs.

Convolutional Model

The version of our audio which gets processed by the model is the Mel Spectrogram of the audio. Specifically, the input to our model has the dimensions (batch_size, 3, 64, 938) which each correspond to RGB, the frequency bands of the spectrogram, and the length of time of recording. The model has 4 convolutional layers, each of which increase the image depth and reduce width and height through kernels and strides. The dimensionality of the data after going through all convolutions is (batch_size, 64, 4, 59). The model then uses an adaptive average pool to flatten the data to a shape of (batch_size, 64). Finally, the data is processed through the linear layer to get our output logits, making the output (batch_size, num_songs).

For training our model we ran a standard machine learning training algorithm. We created a dataframe listing the file paths to each of the images and the corresponding class. Each song was assigned an integer to represent its class. The data was then randomly partitioned into training and test data with a 70/30 split respectively. The batch size of the data was set at 16. We used a cross entropy loss function for the evaluation of the model's learning performance and an ADAM optimizer. The determination of the hyperparameters such as the learning rate of the optimizer and the number of epochs is described in the section below.

Grid Search for Hyperparameter Tuning

To determine the optimal hyperparameters for our model, we employed a grid search algorithm. This is a brute-force search algorithm that exhaustively considers all possible combinations of hyperparameter values to identify the best configuration. This technique constructs a matrix of hyperparameter values and evaluates the performance of the model for each combination, ultimately selecting the one that yields the highest performance metrics. The hyperparameters we focused on tuning were the learning rate and number of epochs, which are critical for the training dynamics and convergence of the neural network. The learning rate controls the step size during the gradient descent optimization process, which in turn affects how quickly the model updates its weights. The number of epochs determines the number of complete passes through the training dataset.

We defined a discrete range of values for each hyperparameter to explore. Firstly for learning rate, we tested the values 0.001, 0.01, 0.05, and 0.1. These values span a range from conservative to more aggressive learning rates, allowing us to observe how different step sizes impact the convergence and stability of the training process. For the number of epochs, we tested values 5, 10, 15, and 20. These values allow us to assess the model's performance with varying amounts of training, allowing us to find a balance between underfitting and overfitting.

Part 2: Evaluation

Evaluation Goals

Initially, we were unsure of how successful and accurate we could make our program, we wanted the model to at least process new inputs at a rate better than just the uniform probability of guessing a song. We anticipated that a large portion of errors would be due to the similarity of certain parts of different songs. In order to avoid overfitting our training data, it was crucial to appropriately split our dataset into training and testing subsets. At first, we decided on a 80/20 split, allocating 80% of the data for training and 20% for testing, to ensure robust evaluation of the model's performance on unseen data. During the training process, we monitored both the training and testing accuracy so that we could detect any signs of overfitting early and make adjustments as needed by tuning hyperparameters. We were seeing very similar accuracy results for training and testing data and not great results when we manually tested with newer audio. Thus in newer iterations of the model, we trained again with a 70/30 split and saw better results when manually testing.

Throughout the training process, we closely monitored the test accuracy to gauge the model's ability to make correct predictions on unseen data. After completing the training phase, the best version of our model achieved a test accuracy of about 87%, which indicates that the model was well-generalized and capable of making reliable predictions on new data.

In addition to developing our model, we also focused on implementing a simple user interface that integrates our trained model. Interactions with the model through the UI were not as successful as our test data, but we were able to improve over time. One reason we believe the full end to end usage of our UI and model to not be as great as our testing performance is because we think that the medium used to record for our UI, our laptops, results in slight changes in audio quality as compared to the means in which we all recorded the training data, our iPhones. This inspired our idea to add a section into our UI that allows a user to record ten second clips of songs and save them. This allows the user to add more points to our dataset, introducing a new set of voices and microphone qualities, thereby passively improving our model.

1. Evaluation Results

The current iteration of our model achieves a test accuracy of 87%. Below we describe the training and test results of various iterations of our model and the iterative changes we made to improvement results. In addition to evaluating our model on the test accuracy during training, we implement some community testing, asking users including ourselves to interact with our UI and test the performance of the model in real time. We describe the results in the community testing section, but they were not as strong as our test accuracy. Nonetheless, this opened our eyes to several possibilities of improving our model in the future.

	Model 1	Model 2	Model 3	Model 4	Model 5
Train Accuracy	66%	85%	99%	95%	94%
Test Accuracy	49%	84%	82%	83%	87%

Fig. 6 Train vs. Test Accuracy for Model Iterations

2. Improvements Based on Evaluation

We underwent numerous iterations of the model with various adjustments; however, in this report we chose to only mention the most significant modifications and label those as specific model versions (Model X). Initially, our model consisted of 3 convolutional layers, and through experimentation, we observed a significant improvement in accuracy when increasing the number of convolutional layers from three to four, which can be noted in the jump in accuracy between Model 1 and Model 2. Subsequently, we developed a grid search script to identify the most optimal hyperparameters and implemented these, leading to another improvement in accuracy from Model 2 to Model 3. This substantial increase in accuracy raised concerns about potential overfitting; thus, to address this concern, we adjusted our train/test split from 80/20 to 70/30, which can be noted in the transition from Model 3 to Model 4. Our final significant modification was the augmentation of the training and test datasets with additional data via the direct upload feature implemented in the user interface, which is reflected in the progression from Model 4 to Model 5, our final model.

3. Community Testing

To evaluate the effectiveness of HUMmer, we conducted a user study involving external participants. We asked 10 of our friends to hum 10-second segments of five different songs from our dataset of 17 songs. This provided us with a total of 50 distinct tests. Each participant's humming was processed by our application and the results of this evaluation demonstrated that HUMmer accurately identified 22 out of 50 of the hummed segments, resulting in an accuracy rate of 44%. This indicates that while HUMmer exhibits a decent level of accuracy, there is still much room for improvement in terms of bettering the model's ability to generalize across user inputs.

Final Takeaways

In terms of achievements, we successfully created a program that can determine which song a user is humming out of 17 total songs. We were able to implement a structured pipeline to collect humming samples and add diversity to our data set by allowing users to add training data. We were also able to segment our humming recordings into fixed-length snippets so that we input uniform data for the model. We successfully transformed our humming samples into spectrograms that captured spectral and temporal features of our humming samples, and we also successfully designed a CNN architecture that optimized for spectrogram analysis and extracted relevant features, and we conducted a grid search to optimize hyperparameters.

Because our primary focus was on developing a highly accurate backend model, a deprioritized goal of ours was an extensive user interface. We kept the UI simple and functional such that it provides basic interaction capabilities for recording and uploading humming inputs for users. Additionally, we intentionally limited the scope of our project to 17 songs by The Weeknd to prioritize accuracy over breadth. By focusing on a smaller dataset, we could ensure our model delivers higher precision in song recognition.

In the future, HUMmer could be expanded to include more songs by The Weeknd and discographies of many other artists. This would involve collecting larger and more diverse sets of humming samples across a wider array of songs and artists, including multiple variations of each song segment to account for different vocal pitches, timbres, and humming styles. Implementing crowd-sourcing techniques to gather humming samples from users across the world would also help increase the diversity of the dataset. Additionally, utilizing more sophisticated data augmentation techniques such as pitch shifting, time stretching or adding background noise, would simulate a more realistic scenario of how users would be using the application and make our model more robust. Furthermore, training the model on a more expanded and diverse data set will help improve the accuracy of the program as there is more training data for the model to learn from. Employing a more improved and targeted training tactic, such as curriculum learning or progressive training, both of which focus on training the model with simple tasks first before gradually increasing the difficulty of the pattern recognition. In its current iteration, the model is unable to correctly identify the song being hummed with 100% accuracy. With a more robust dataset and targeted training, these mistakes can be reduced and we can bring the model's accuracy closer to a perfect score.

Google's humming identifier program, Hum to Search, is similar to HUMmer except it essentially includes the discographies of almost all possible artists. In order to further optimize and differentiate HUMer from Google's application, our application could be developed to be personalized to songs that are within the user(s)' music taste and personalized to their humming styles. A user-specific model that adapts to individual user's humming styles and preferences with a feedback loop where users can correct the model's predictions can vastly improve the benefits of using our program. We could achieve this by expanding on the data collection section in HUMmer that we've already implemented where a user can record a sample of their humming and submit it to our dataset. Furthermore, we could also develop a recommendation system that learns the musical preferences of the user over time based on their humming history, and provides recommendations for songs that align with the user's tastes.

An immediate improvement we could make to the model is changing the classes that our model identifies between. We initially made the assumption that different chunks of songs are similar enough in background noise and beats so that we could make each song its own class. However, after evaluating our model manually, we believe that different chunks of songs are as different as chunks from different songs entirely. Thus, we think that creating a class for each 10-second chunk of each song and asking our model to classify hummed recordings into these smaller classes could improve accuracy. This, of course, would require a quick final step in the workflow of mapping any chunk of a song that the model outputs to the overall song. Although we feel that this approach will significantly improve results in practice, it will require significantly more training data per chunk of each song since we are multiplying the number of classes by ten fold.

Ultimately, we learned a great deal about artificial intelligence, system design, and teamwork from this project and are excited to continue improving our project in the future.

Resources

1. We utilized a blog written by Google about their application Hum to Search as research: https://blog.research.google/2020/11/the-machine-learning-behind-hum-to.html?m=1

2. PyTorch Documentation: https://pytorch.org/docs/stable/