The Song Search

Team members

1. Isha Hemant Arora (arora.isha@northeastern.edu)
2. Praveen Kumar Sridhar (sridhar.p@northeastern.edu)

Introduction

Taking inspiration from Shazam, we hope to create a project that would be replicate the inner workings of Shazam and perform a retrieval task on musical data. The core of this IR system comes from finding efficient representations for songs and performing a retrieval task with the said representations. While trying to figure out our plan, we came across the amazing resource Tensorflow Magenta and were intrigued by music transcription models so we decided to explore and experiment with a couple of these models.

Methods

To start off, we started with trying to understand and reproduce the models for Piano and Drum Transcription (Onsets and Frames) and then the models of Multi-instrument transcription (MT3). This is where we actually hit a snag, we weren't very familiar with audio data and MIDI transcriptions and the original idea of trying to use string implementations over lyrical music and using it for matching audio clips had to be reformed.

We restructured our idea to work solely with instrumental data, all after trying to understand how the models parsed lyrical data (it was parsed as an instrument, generally the piano), with a pitch similar to the data.

Working with instrumental data, we decided that we could try and retrieve the vector representation of each song in our dataset and try using these representations to match with query vectors.

Dataset

We decided to use the GTZAN dataset as available on Kaggle. The dataset consists of 10 genres of music, each genre having 100 instrumental audio clips, each being 30 seconds long. All files were originally in the .wav format, a format around which the model was created.

After transcription, we stored the data as such:

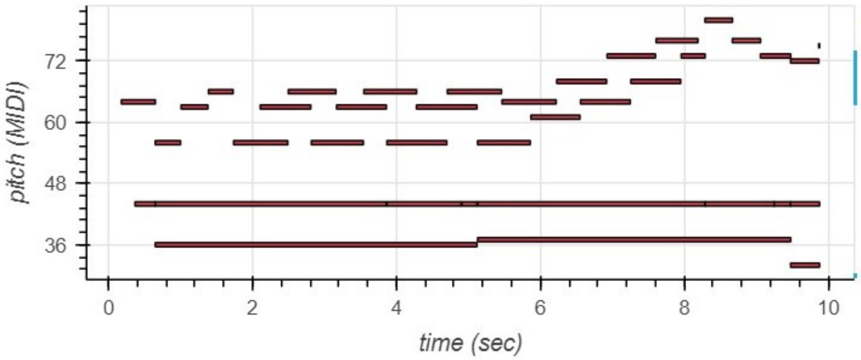
```
{
  "audio_file_path": "sample1.wav",
  "date_added": "11/23/2022",
  "meta_data": {
    "artist": "Bach",
    "duration": "00:00:30",
    "genre": "classical",
    "transcribed_json_path": "transcribed_sample1.json"
  }
}
```

For each song, the format of transcribed JSON would be similar to (as created after transcription):

```
notes {
  "pitch" : "40",
  "velocity" : "127",
  "start_time" : "3.33",
  "end_time" : "3.45",
  "instrument" : "2",
  "program" : "42"
}
notes {
  "pitch" : "29",
  "velocity" : "127",
  "start_time" : "3.45",
  "end_time" : "3.56",
  "instrument" : "2",
  "program" : "42"
}
```

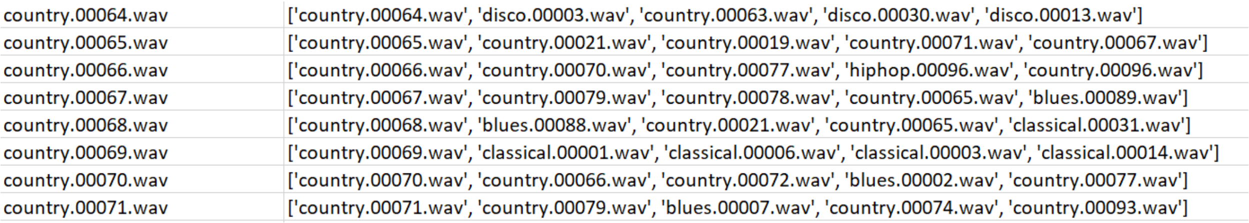
Queries

A 10-second audio file was introduced as the query. The file, also expected to be a .wav file was parsed and transcribed into a similar JSON format. The following is a visual representation of the audio input for the query. This image is NOT the input but instead a representation of the .wav file.



Candidate Result Sets

Since the features for each of the audio files were added as a separate .csv file with the dataset, treated them as the annotated features and used them to find our candidate set. This was done with the help of cosine similarity. The following is a snapshot of the results.



country.00064.wav	['country.00064.wav', 'disco.00003.wav', 'country.00063.wav', 'disco.00030.wav', 'disco.00013.wav']
country.00065.wav	['country.00065.wav', 'country.00021.wav', 'country.00019.wav', 'country.00071.wav', 'country.00067.wav']
country.00066.wav	['country.00066.wav', 'country.00070.wav', 'country.00077.wav', 'hiphop.00096.wav', 'country.00096.wav']
country.00067.wav	['country.00067.wav', 'country.00079.wav', 'country.00078.wav', 'country.00065.wav', 'blues.00089.wav']
country.00068.wav	['country.00068.wav', 'blues.00088.wav', 'country.00021.wav', 'country.00065.wav', 'classical.00031.wav']
country.00069.wav	['country.00069.wav', 'classical.00001.wav', 'classical.00006.wav', 'classical.00003.wav', 'classical.00014.wav']
country.00070.wav	['country.00070.wav', 'country.00066.wav', 'country.00072.wav', 'blues.00002.wav', 'country.00077.wav']
country.00071.wav	['country.00071.wav', 'country.00079.wav', 'blues.00007.wav', 'country.00074.wav', 'country.00093.wav']

Evaluation

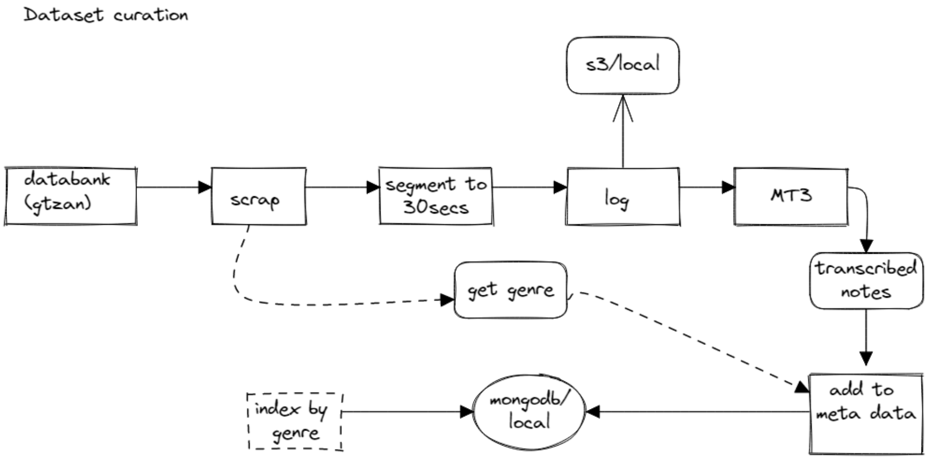
We used the metrics for **Accuracy in Top 5** and **MAP** to test the Information Retrieval system that we had created.

Accuracy: It is an extremely intuitive performance measure. Simply, it is a ratio of the number of queries with the best/ideal result in one of the top 5 fetched results to the total queries. We chose this because we wanted to measure how accurately our model fetches results from the dataset for a given query with a definite ideal answer, this is sort of like a known retrieval problem because we don't really value ambiguity in the search. We have the correct/best result for each query.

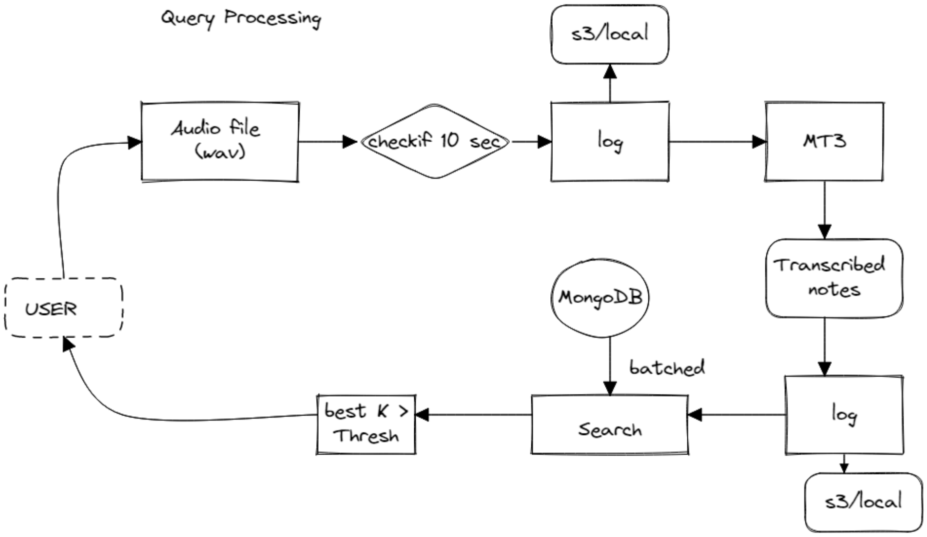
MAP (Mean Average Precision): The general definition for the Average Precision (AP) is finding the area under the precision-recall curve. MAP is the average of the Average Precision (AP).

Architecture Diagram

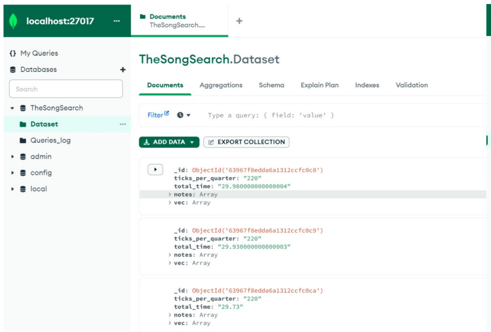
For curating and transcribing the Dataset



For transcribing and matching the Query



We saved the dataset and query log for our project on our local MongoDB:



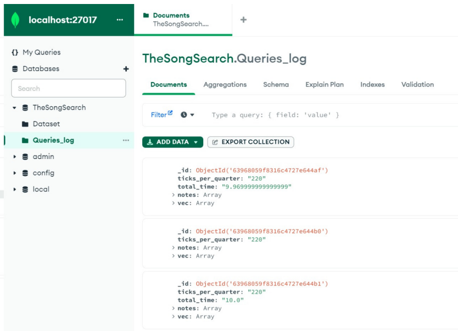
TheSongSearch.Dataset

```
Type a query: { field: 'value' }
```

```
_id: ObjectId('63967f8edda6a1312ccfc0c8')
ticks_per_quarter: "220"
total_time: "29.980000000000004"
> notes: Array
> vec: Array

_id: ObjectId('63967f8edda6a1312ccfc0c9')
ticks_per_quarter: "220"
total_time: "29.930000000000003"
> notes: Array
> vec: Array

_id: ObjectId('63967f8edda6a1312ccfc0ca')
ticks_per_quarter: "220"
total_time: "29.73"
> notes: Array
> vec: Array
```



TheSongSearch.Queries_log

```
Type a query: { field: 'value' }
```

```
_id: ObjectId('63968059f8316c4727e644af')
ticks_per_quarter: "220"
total_time: "9.969999999999999"
> notes: Array
> vec: Array

_id: ObjectId('63968059f8316c4727e644b0')
ticks_per_quarter: "220"
> notes: Array
> vec: Array

_id: ObjectId('63968059f8316c4727e644b1')
ticks_per_quarter: "220"
total_time: "10.0"
> notes: Array
> vec: Array
```

Models Used

The papers that we are focusing on were written with the aim to achieve AMT (Automatic Music Transcription). AMT is valuable in not only helping with understanding but also enabling new forms of creation. The MT3 is the Multi-task Multitrack Music Transcription (Music Transcription with Transformers). As an overview we were able to see that while it is possible to separate different instruments and transcribe them separately, the architecture for different instruments would be different, thus making the process to implement different models per instrument long and tedious. More on this is explained in the next sections. The MT3 can process audio with multiple instruments and transcribe multiple different instruments to the MIDI standard.

We also tried understanding the model developed by Shazam and trying to relate it to the method we were trying to implement (the one where we process and transcribe audio data).

1. **Onsets and Frames**

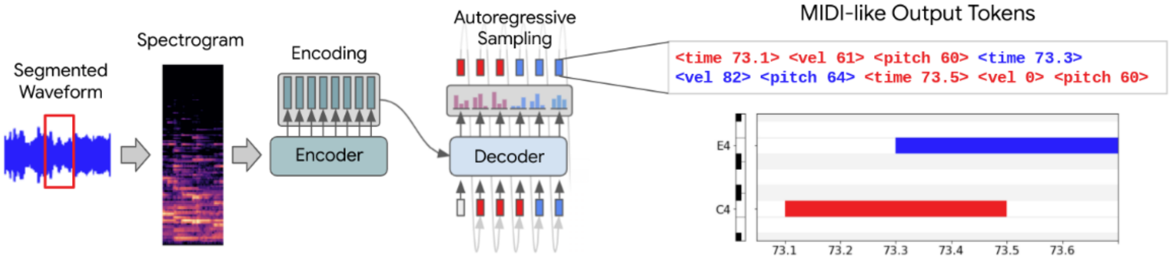
   The dual objectives are learned on each stack:

   Frame Prediction: Trained to detect every frame where a note is active.
   Onset Prediction: Trained to detect only onset frames (the first few frames of every note)

   Each instrument needs a new architecture, but it is tedious to build a custom arch for each instrument. Thus, the MT3 model was created.

2. **MT3 (Multi-Task Multi-track Music Transcription)**

   It uses off-the-shelf transformers, as they work well if not better than custom neural networks as we had seen for Piano/Drums. They are modeled to take spectrograms as input and output a sequence of MIDI-like note events. It was modeled as a sequence-to-sequence task, using the Transformer architecture from T5. The major benefit of this model was if needed to retrain this architecture for newer instruments, we would only need to change the vocabulary of the output.



What is MIDI?

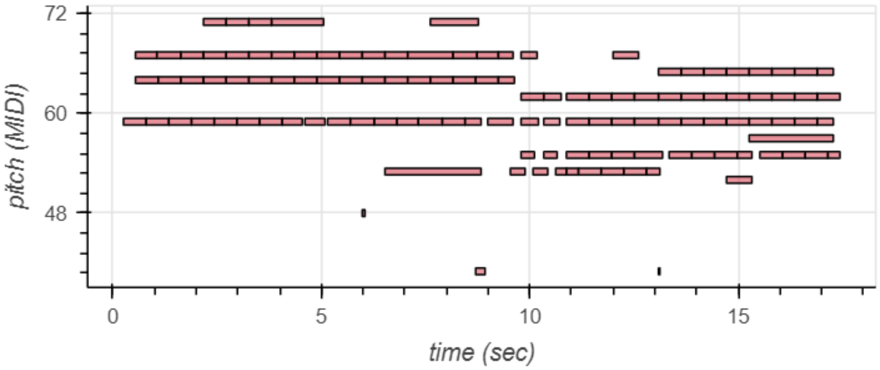
MIDI is a communication standard that allows digital music gear to speak the same language. MIDI is short for Musical Instrument Digital Interface. It's a protocol that allows computers, musical instruments, and other hardware to communicate.

Experiments

To begin with, we first tried to understand the models for Onsets and Frames and MT3 and the MIDI transcription.

Piano Transcription

On transcribing a piano audio file (as added below), a pitch-against-time visualization was created.



The piano transcription model was created around two stacks.

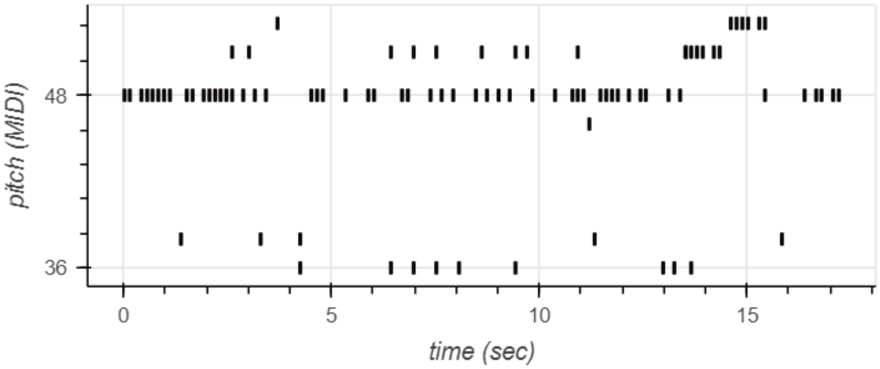
Using the stacks for inference:

- The raw output of the onset detector is fed into the frame detector as an additional input
- The final output of the model is restricted to starting new notes only when the onset detector is confident that a note onset is in that frame.

Finally, the loss function used is the sum of two cross-entropy losses: one from the onset side and one from the frame side.

Drum Transcription

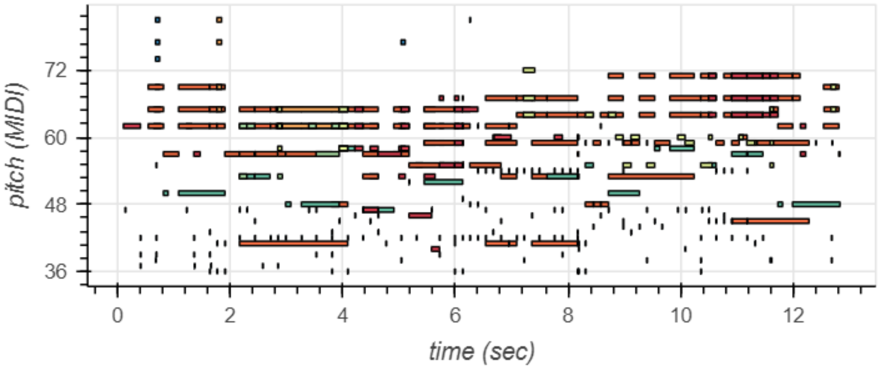
On transcribing a drum audio file (as added below), a pitch against time visualization was created.



We experimented with piano and drums transcription using the customized networks as suggested in the papers and observed the output (pitch vs time and transcription) on manually created audio files using the GarageBand software.

With this, we were able to see how these models are transcribing the audio it receives as input. The way these models transcribe audio is by converting them to MIDI format.

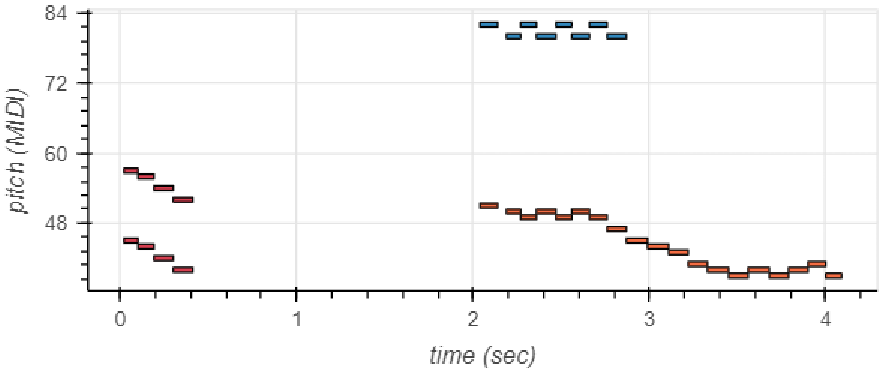
Multi-task Multitrack Music Transcription (MT3)



On creating a snippet of an audio file with multiple instruments (again on GarageBand), we were able to create a similar pitch against time visualization, this time for all the instruments in the audio file.

Wanting to experiment further, we wanted to see how the model would process audio with vocals (MIDI instrument numbers do not consider vocals).

To gain further clarity, we tried passing a pure audio clip of us speaking (no background instruments included). We noticed that the transcription and the MIDI audio were classed as that of a piano.



Genre Classification

The original idea was to pull vectors for each audio transcribed in the dataset and then match it to the query vectors and thus we tried implementing it. Since we were struggling with the same (the model was a little too complex for us), we then decided on trying to create a genre classifier, which we believed would give us vectors too, ones that we could try to match with the query vectors that would be created using this genre classifier.

Yet, when we created the genre classifier, we realized that the data that we were working with (1000 audio clips, 30 seconds each) was extremely small for us to be able to train a proper model (we tried multiple models; all the way from simple ANNs to LSTMs). Thus, the genre classifier itself was not well-trained and the vectors, irrespective of the genre, were very similar. Thus, query vectors when created, almost all had a cosine similarity of over 0.85 and best matches at the first position were not always true.

Vector Creation

Since our idea of using a genre classifier was not exactly successful, we tried yet another approach. Since we had converted our audio files into Mel Spectrograms, we got the matrix representations of that and then flatten them - that gave us vectors that we were able to use for matching with query vectors (which were also created using the Mel Spectrogram representations).

Piano Matching (Single instrument) v/s GTZAN clips (Multi-instrument)

As a part of the string matching with JSON, we considered two cases:

1. Single Instrument (Piano files)

   The first thing we saw was that the JSON created did not include the instrument (even though MIDI does have multiple types of instruments for a single style, so we were expecting to see its existence in the JSON). We used the single instrument data as a proof of concept for building a matching algorithm. This matching algorithm was created keeping the idea of Rabin-Karp in mind. We call this algorithm from now onwards, the **notes-matching algorithm**.

2. Multiple Instruments (GTZAN dataset)

   This dataset, when we first transcribed it into the JSON format, did have the instrument data. Testing the matching algorithm on this dataset, we noticed that there were a lot of issues and that the matching algorithm was not performing as well as we had hoped. To improve on it, we made a few more changes to the matching by tweaking a couple of hyperparameters we built into the notes-matching algorithm.

Results

When testing the MT3 model, we observed that:

- MT3 has an average frame F1 score of 0.85 across different datasets
- MT3 has an average onset F1 score of 0.8 across different datasets

When implementing the string matching algorithm on the JSON files created using:

1. Single Instrument **(notes-matching algorithm)**

   Even when a single song was cut into 30-second clips, we were able to see an accuracy of 79%. This was a little expected as the models were able to transcribe audio on a single instrument way better.

2. Multiple Instruments

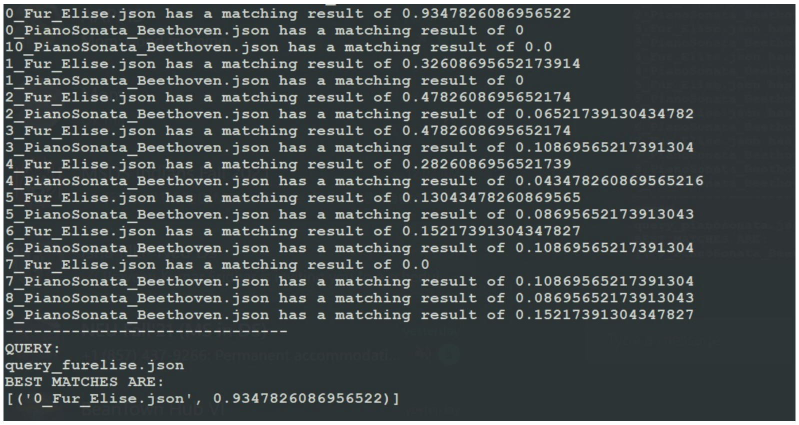
   a. **Notes-matching algorithm**: The IR system with the MT3 model has an overall accuracy of 74% in the top 5 candidate set and a MAP of 0.68

   b. **Mel-Vector similarity**: Using the vector representations from the Mel Spectrograms had an overall accuracy of 51% in the top 5 candidate set

Understanding Query Matching Results

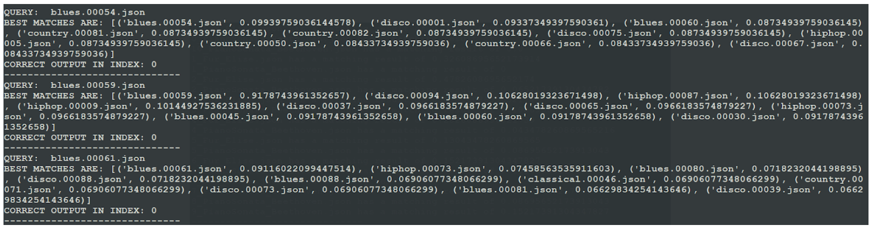
1. Single Instrument Results (notes-matching algorithm)

   Each query had the top 1 best-matched results based on the notes-matching algorithm without multiple-instrument fine-tuning. If the correct result was found, we get the file name with the highest matching score.



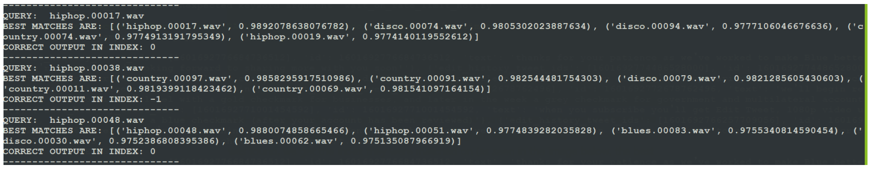
```
0_Fur_Elise.json has a matching result of 0.9347826086956522
0_PianoSonata_Beethoven.json has a matching result of 0
10_PianoSonata_Beethoven.json has a matching result of 0.0
1_Fur_Elise.json has a matching result of 0.32608695652173914
1_PianoSonata_Beethoven.json has a matching result of 0
2_Fur_Elise.json has a matching result of 0.4782608695652174
2_PianoSonata_Beethoven.json has a matching result of 0.06521739130434782
3_Fur_Elise.json has a matching result of 0.4782608695652174
3_PianoSonata_Beethoven.json has a matching result of 0.10869565217391304
4_Fur_Elise.json has a matching result of 0.2826086956521739
4_PianoSonata_Beethoven.json has a matching result of 0.043478260869565216
5_Fur_Elise.json has a matching result of 0.13043478260869565
5_PianoSonata_Beethoven.json has a matching result of 0.08695652173913043
6_Fur_Elise.json has a matching result of 0.15217391304347827
6_PianoSonata_Beethoven.json has a matching result of 0.10869565217391304
7_Fur_Elise.json has a matching result of 0.0
7_PianoSonata_Beethoven.json has a matching result of 0.10869565217391304
8_PianoSonata_Beethoven.json has a matching result of 0.08695652173913043
9_PianoSonata_Beethoven.json has a matching result of 0.15217391304347827
------------------------------
QUERY:
query_furelise.json
BEST MATCHES ARE:
[('0_Fur_Elise.json', 0.9347826086956522)]
```

2. Multiple Instrument Results
   a. Notes-matching algorithm: Each query had the top 5 best-matched results based on the notes-matching algorithm. If the correct result was found, we get the index they were matched on (as seen in the image below, the best match was found on index 0).



```
QUERY:  blues.00054.json
BEST MATCHES ARE: [('blues.00054.json', 0.09939759036144578), ('disco.00001.json', 0.09337349397590361), ('blues.00060.json', 0.08734939759036145)
, ('country.00081.json', 0.08734939759036145), ('country.00082.json', 0.08734939759036145), ('disco.00075.json', 0.08734939759036145), ('hiphop.00
005.json', 0.08734939759036145), ('country.00050.json', 0.08433734939759036), ('country.00066.json', 0.08433734939759036), ('disco.00067.json', 0.
08433734939759036)]
CORRECT OUTPUT IN INDEX: 0
------------------------------
QUERY:  blues.00059.json
BEST MATCHES ARE: [('blues.00059.json', 0.9178743961352657), ('disco.00094.json', 0.10628019323671498), ('hiphop.00087.json', 0.10628019323671498)
, ('hiphop.00009.json', 0.10144927536231885), ('disco.00037.json', 0.0966183574879227), ('disco.00065.json', 0.0966183574879227), ('hiphop.00073.j
son', 0.0966183574879227), ('blues.00045.json', 0.09178743961352658), ('blues.00060.json', 0.09178743961352658), ('disco.00030.json', 0.0917874396
1352658)]
CORRECT OUTPUT IN INDEX: 0
------------------------------
QUERY:  blues.00061.json
BEST MATCHES ARE: [('blues.00061.json', 0.09116022099447514), ('hiphop.00073.json', 0.07458563535911603), ('blues.00080.json', 0.0718232044198895)
, ('disco.00088.json', 0.0718232044198895), ('blues.00088.json', 0.06906077348066299), ('classical.00046.json', 0.06906077348066299), ('country.00
071.json', 0.06906077348066299), ('disco.00073.json', 0.06906077348066299), ('blues.00081.json', 0.06629834254143646), ('disco.00039.json', 0.0662
9834254143646)]
CORRECT OUTPUT IN INDEX: 0
------------------------------
```

   b. Mel-Vector similarity: Each query had the top 5 best-matched results based on the cosine similarity scores calculated using the Mel Spectrogram vectors created. If the correct result was found, we get the index they were matched on (as seen in the image below, the best match was found on index 0 and -1 if the best-matched result was not in the top 5).



```
------------------------------
QUERY:  hiphop.00017.wav
BEST MATCHES ARE: [('hiphop.00017.wav', 0.9892078638076782), ('disco.00074.wav', 0.9805302023887634), ('disco.00094.wav', 0.9777106046676636), ('c
ountry.00074.wav', 0.9774913191795349), ('hiphop.00019.wav', 0.9774140119552612)]
CORRECT OUTPUT IN INDEX: 0
------------------------------
QUERY:  hiphop.00038.wav
BEST MATCHES ARE: [('country.00097.wav', 0.9858295917510986), ('country.00091.wav', 0.982544481754303), ('disco.00079.wav', 0.9821285605430603), (
'country.00011.wav', 0.9819399118423462), ('country.00069.wav', 0.981541097164154)]
CORRECT OUTPUT IN INDEX: -1
------------------------------
QUERY:  hiphop.00048.wav
BEST MATCHES ARE: [('hiphop.00048.wav', 0.9880074858665466), ('hiphop.00051.wav', 0.9774839282035828), ('blues.00083.wav', 0.9755340814590454), ('
disco.00030.wav', 0.9752386808395386), ('blues.00062.wav', 0.975135087966919)]
CORRECT OUTPUT IN INDEX: 0
------------------------------
```

The queries used can be seen in the image (.json for the notes-matching algorithm and .wav for Mel-Vector similarity).

The best match is the same song from which the query was created this is the *ground truth.*

For example,

- If blues.00054.json is the search query then the best match is found in index 0 ‘blues.00054.json’ is the song from which this query was created.
- If classical.00004.wav is the search query then the best match is found in index 0 ‘classical.00004.wav’ is the song from which this query was created.

Contributions and Future Work

We realize that the project did not include songs that contained lyrics as we were unable to parse and transcribe them in a way that was particularly useful. As a future extension, we hope to be able to extend this project of matching songs to include songs with lyrics. Also we want to swap out the Robin-Karp based notes-matching algorithm for a Smith-Waterman based notes-matching algorithm.

GitHub Link

https://github.com/PraveenKumarSridhar/TheSongSearch

References

1. **Main Paper/Blog**
   - Links:
     - Hawthorne, Curtis, et al. "Sequence-to-sequence piano transcription with Transformers." *arXiv preprint arXiv:2107.09142* (2021).
     - Gardner, Josh, et al. "Mt3: Multi-task multitrack music transcription." *arXiv preprint arXiv:2111.03017* (2021).
     - https://magenta.tensorflow.org/transcription-with-transformers (the original blog)
   - Source Code:
     - https://github.com/magenta/mt3
     - https://github.com/google-research/text-to-text-transfer-transformer/ (T5)
2. **Auxiliary Papers/Blogs**

   a. https://towardsdatascience.com/3-reasons-why-music-is-ideal-for-learning-and-teaching-data-science-59d892913608 (Max Hilsdorf)

   b. Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

   c. Gong, Yuan, Yu-An Chung, and James Glass. "Ast: Audio spectrogram transformer." *arXiv preprint arXiv:2104.01778* (2021).

   d. M. Awiszus, “Automatic music transcription using sequence to sequence learning,” Master’s thesis, Karlsruhe Institute of Technology, 2019.

   e. https://magenta.tensorflow.org/onsets-frames

   f. https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition

   g. https://www.makeuseof.com/how-does-shazam-work/