LARGE-SCALE CONTENT-BASED MATCHING OF MIDI AND AUDIO FILES

First author
Affiliation1
author1@ismir.edu

Second author
Retain these fake authors in submission to preserve the formatting

Third author
Affiliation3
author3@ismir.edu

ABSTRACT

MIDI files, when paired with corresponding audio recordings, can be used as ground truth for many music information retrieval tasks. We present a system which can efficiently match and align MIDI files to entries in a large corpus of audio content without using any metadata. The core of our approach is a neural network-based cross-modality hashing scheme which transforms feature matrices into sequences of vectors in a common Hamming space. Once represented in this way, we can efficiently perform large-scale dynamic time warping searches to match MIDI data to audio recordings. We evaluate our approach on the task of matching a huge corpus of MIDI files to the Million Song Dataset.

1. TRAINING DATA FOR MIR

Central to the task of content-based Music Information Retrieval is the curation of ground-truth data for tasks of interest (e.g. timestamped chord labels for automatic chord estimation, beat positions for beat tracking, prominent melody time series for melody extraction, etc.). The quantity and quality of this ground-truth is often instrumental in the success of MIR systems which utilize it as training data. Unfortunately, creating appropriate labels for a recording of a given song by hand typically requires person-hours on the order of the length of the song. This often arguably makes the available training data a bottleneck to success for a given content-based MIR task.

It has previously been observed that MIDI files, when time-aligned to corresponding audio recordings, can be used to infer ground-truth information about a given song [8, 24]. This is due to the fact that a MIDI files can be viewed simplistically as a timed sequence of note annotations or a piano roll. It is much more straightforward to estimate, e.g., beat locations, chord labels, and the predominant melody from these representations than one which was derived from an audio signal. Unsurprisingly, a handful of tools have been developed for inferring this information from MIDI files [6, 7, 15, 17].

In [9], it is argued that some of the biggest successes in machine learning are thanks to the fact that "...a large training set of the input-output behavior that we seek to automate is available to us in the wild." The main motivation behind this project is that this crucial availability of data holds true for MIDI files - through a large-scale web scrape, we obtained 140,910 unique MIDI files, which is orders of magnitude larger than the datasets typically used for MIR research. We believe this proliferation of data is largely caused to two factors: First, that karaoke files are typically distributed as MIDI data and that karaoke is wildly popular, and second, that transcribing popular music as MIDI files is a common pastime for hobbyist musicians.

1.1 Wrangling MIDI files

The mere existence of a large collection of MIDI data is not enough, however. As noted above, in order to use MIDI files as ground truth, they need to be both matched (paired with a corresponding audio recording of the same song) and aligned (adjusted so that the timing of the events transcribed in the file match the audio recording). The latter problem has seen a great deal of research effort [8, 24], and will not be a main focus of this work.

Given large corpora of audio and MIDI files, the task of matching entries from each may seem to be a trivial problem involving fuzzy text matching of the files' metadata. However, MIDI files have no formal mechanism for storing metadata (apart from text meta events, which are rarely used), and as a result the best-case scenario is that the artist and song title are included in the filename or subdirectory. While we found some examples of this in our collection of scraped MIDI files, the vast majority of the files had effectively no metadata information. Figure 1 shows a random sampling of subdirectory and filenames from our collection.

Fortunately, the goal of matching MIDI and audio files is to find pairs which have *content* in common (i.e., the MIDI file is a transcription of the audio file), an information source which is available regardless of metadata quality. However, comparing content has the potential to require much more computation than a fuzzy text comparison, and NM comparisons must be made to match a MIDI dataset of size N to an audio file dataset of size M. Motivated by this, we propose a system which can *efficiently* match MIDI files to audio based solely on their content. Our system learns to hash MIDI and audio content

```
J/Jerseygi.mid
V/VARIA18O.MID
Carpenters/WeveOnly.mid
2009 MIDI/handy_man1-D105.mid
G/Garotos Modernos - Bailanta De Fronteira.mid
Various Artists/REWINDNAS.MID
GoldenEarring/Twilight_Zone.mid
Sure.Polyphone.Midi/Poly 2268.mid
d/danza3.mid
100%sure.polyphone.midi/Fresh.mid
rogers_kenny/medley.mid
2009 MIDI/looking_out_my_backdoor3-Bb192.mid
```

Figure 1. Random sampling of 12 MIDI filenames and their parent directories from our corpus of 455,333 MIDI files scraped from the Internet.

to a common Hamming space where sequences of vectors can be compared efficiently using dynamic time warping (DTW).

The idea of using DTW distance to match MIDI files to audio recordings is not new. For example, in [11], MIDI-audio matching is done by finding the minimal DTW distance between all pairs of chromagrams of (synthesized) MIDI and audio files. Our approach differs in a few key ways: First, instead of using chromagrams (a hand-designed representation), we learn a common representation for MIDI and audio data. Second, our datasets are many orders of magnitude larger (hundreds of thousands vs. hundreds of files), which necessitates a much more efficient approach. Specifically, by mapping to a Hamming space we greatly speed up distance matrix calculation and we receive quadratic speed gains by implicitly downsampling the audio and MIDI feature sequences as part of our learned feature mapping.

In the following section, we detail the dataset of MIDI files we scraped from the Internet and describe how we prepared a subset for training our hasher. Our cross-modality hashing model is then described in Section 3. Finally, in section 4 we evaluate our system's performance on the task of matching files from our MIDI dataset to entries in the Million Song Dataset [3].

2. PREPARING DATA

Our project began with a large-scale scrape of MIDI files from the Internet. We obtained 455,333 files, of which 140,910 were found to have unique MD5 checksums. Most of these files were transcriptions of pieces of music of varying duration and quality. As mentioned previously, the majority of these files had very little useful metadata information. The goal of the present work is to develop an efficient way to match this corpus against the Million Song Dataset (MSD), or more specifically, to the short preview audio recordings provided by 7digital [19].

In order to evaluate our system, we need a collection of MIDI-audio pairs which we know are correctly matched. The accuracy of our approach can then be judged based on how accurately it is able to recover these pairings using the content of the audio and MIDI files alone. Fortunately, we identified a subset of files where the subdirectory of the file indicated the artist and the filename indicated the song title. By cleaning up this subset and using the resulting metadata to match its entries against the Million Song Dataset, we can obtain the requisite ground-truth set of MIDI-audio pairings. We will refer to this collection of files as the "clean MIDI subset".

A further data requirement for our system is a collection of feature vectors derived from audio and MIDI data which should be mapped to similar hashes. This dataset will be used to train our model for hashing MIDI and audio features to a common Hamming space (described in Section 3). In our application, we will be matching sequences of hashes using dynamic time warping, so we need to obtain pairs of sequences of feature vectors where the nth vector in one sequence should be mapped to the nth vector in the other. This necessitates a collection of MIDI files and audio recordings which we are confident are well-aligned in time. From this collection, we can extract features for each modality and be sure that there is a direct correspondence between the resulting hash sequences and the original feature vector sequences.

2.1 Metadata matching

We first extracted the song title and artist from each entry in the clean MIDI subset based on each MIDI file's filename and subdirectory. The resulting metadata was still somewhat messy; for example, "The Beatles" appeared as an artist along with "Beatles, The", "Beatles", and "The Beatles John Paul Ringo George". To normalize these issues, we applied some manual text processing and resolved the artists and song titles against the Freebase [5] and Echo Nest [1] databases. This resulted in 17,243 MIDI files for 10,060 unique songs.

As noted above, we will leverage the clean MIDI subset in two ways: First, to obtain ground-truth pairings of MSD/MIDI matches, and second, to create training data for our hashing scheme. Note that for the latter purpose, we do not need to restrict ourselves to audio data from the MSD, and we can reasonably expect that by using more training data we may be able to learn a better representation. As a result, we combined the MSD with three benchmark audio collections: CAL500 [25], CAL10k [23], and uspop2002 [2]. To match these datasets to the clean MIDI subset, we used the Python search engine library Whoosh to perform a fuzzy matching of their metadata. This resulted in 26,311 audio/MIDI file pairs corresponding to 5,243 unique songs.

2.2 Synthesized MIDI-to-audio alignment

Fuzzy metadata matching is not enough to ensure that we have correctly paired MIDI and audio files due to the following potential issues: The metadata could be incorrect, the fuzzy text match could have failed, the MIDI could be a very poor quality transcription (e.g. missing instruments or sections), and/or the MIDI and audio data could correspond to different versions of the same song. These pit-

[1] http://developer.echonest.com/docs/v4

falls necessitate a method to ensure the correctness of each MIDI/audio file pairing. In additional, we need to align MIDI files to their corresponding audio recordings to use them as training data for our hashing model. A common method for performing this kind of alignment is to compute the dynamic time warping (DTW) path between sequences of feature vectors [8, 11, 24]. Conveniently, DTW reports a score which represents the quality of the alignment, and unsurprisingly this score tends to be very poor when non-matching sequences are attempted to be aligned. An overview of DTW and its application to music can be found in [16].

For our purposes, the reliability of the DTW confidence score is crucial because we will use it to decide when an audio/MIDI file pairing is valid and when an alignment can be used as training data for our hashing model. We experimented with a variety of approaches, and converged on the following system for aligning a single MIDI/audio file pair: First, we synthesize the MIDI data using `fluidsynth`.[2] We then estimate the MIDI beat locations using the MIDI file's tempo change information and the method described in [17]. To circumvent the common issue where the beat is tracked one-half beat out of phase, we double the BPM until it was at least 240. Using `librosa` [14], we compute beat locations for the audio signal with the constraint that the BPM should remain close to the global MIDI tempo. We then compute log-amplitude beat-synchronous constant-Q transforms (CQTs) of audio and synthesized MIDI data with semitone frequency spacing and a frequency range from C3 (65.4 Hz) to C7 (1046.5 Hz), also using `librosa`. The resulting feature matrices are then of dimensionality $N \times D$ and $M \times D$ where N and M are the resulting number of beats in the MIDI and audio recordings respectively and D is 48 (the number of semitones between C3 and C7). Example CQTs computed from a 7digital preview clip and from a synthesized MIDI file can be seen in Figure 2(a) and 2(b) respectively.

We then use dynamic time warping to find the lowest-cost path through a full pairwise cosine distance matrix $S \in \mathbb{R}^{N \times M}$ of the MIDI and audio CQTs. This path can be represented as two sequences $p, q \in \mathbb{R}^L$ of indices from each sequence such that $p[i] = n, q[i] = m$ implies that the nth MIDI beat should be aligned to the mth audio beat. Traditional DTW requires that this path includes the start and end of each sequence, or equivalently that $p[1] = q[1] = 1; p[L] = N; q[L] = M$. This approach is not valid when one sequence may be a subsequence of the other, which is often true in our problem setting due to the fact that, for example, the MSD 7digital audio recordings are cropped preview song clips. We therefore modify this constraint so that either $gN \leq p[L] \leq N$ or $gM \leq q[L] \leq M$ where $g \approx 1$ is a parameter which provides a small amount of additional tolerance. We also include the typical modification of applying an additive penalty ϕ for "non-diagonal moves", i.e. path entries where either $p[i] = p[i+1]$ or $q[i] = q[i+1]$. For synthesized MIDI-to-audio alignment, we used $g = .95$ and set ϕ to the 90th percentile of S. The cosine distance matrix and the lowest-cost DTW path for the CQTs shown in Figure 2(a) and 2(b) can be seen in Figure 2(e).

[2] `http://www.fluidsynth.org`

2.3 DTW cost as confidence score

The "cost" of a DTW path p, q through S is calculated by the sum of the distances between the aligned entries of each sequence, or

$$c = \sum_{i=1}^{L} \begin{cases} S[p[i], q[i]] + \phi & p[i] = p[i-1] \\ S[p[i], q[i]] + \phi & q[i] = q[i-1] \\ S[p[i], q[i]] & \text{otherwise} \end{cases}$$

As-is, the DTW cost is not an appropriate confidence score for alignment/match quality in our application primarily for two reasons. First of all, the path length can vary greatly across MIDI/audio file pairs depending on N and M. We therefore compute the *mean* distance between aligned constant-Q spectra across the path instead of the sum of distances by dividing c by L. Secondly, the scale of S (and therefore the DTW cost) depends on how similar the synthesized audio is to the "real" audio recording, which is not an important factor in the alignment quality. This can be mitigated by normalizing the DTW cost by the mean value of the submatrix of S through which the DTW past travels:

$$\mathcal{B} = \sum_{i=\min(p)}^{\max(p)} \sum_{j=\min(q)}^{\max(q)} S[i, j]$$

These normalizations were also discussed in [11]. Combining the above, we get our modified DTW cost

$$\hat{c} = \frac{c}{L\mathcal{B}}$$

In practice, we found these two normalization steps to be of critical importance for obtaining a reliable confidence score.

In order to use our normalized DTW cost to determine when a match and alignment was successful, we need a threshold above which the DTW cost indicates a failure. To estimate this threshold, we manually auditioned 125 alignments and noted whether each was successful, where "success" requires that the audio and MIDI were well-synchronized for their entire duration. From this, we obtained an AUROC score of 0.986, indicating a highly reliable confidence metric. As a threshold, we chose 0.78, above which no alignments where labeled as successful. We discarded all MIDI/audio matches whose alignment resulted in DTW costs above this threshold, resulting in 10,035 successful alignments.

As mentioned above, we will be using these matched and aligned MIDI and audio files for two purposes: First, as training data for our hashing model; and second, to evaluate our efficient content-based matching system. For fair evaluation, it is critical that we do not test the matching system using any songs which were used to train the hashing model or to adjust the parameters of the matching system. We therefore split out 50% of the successful

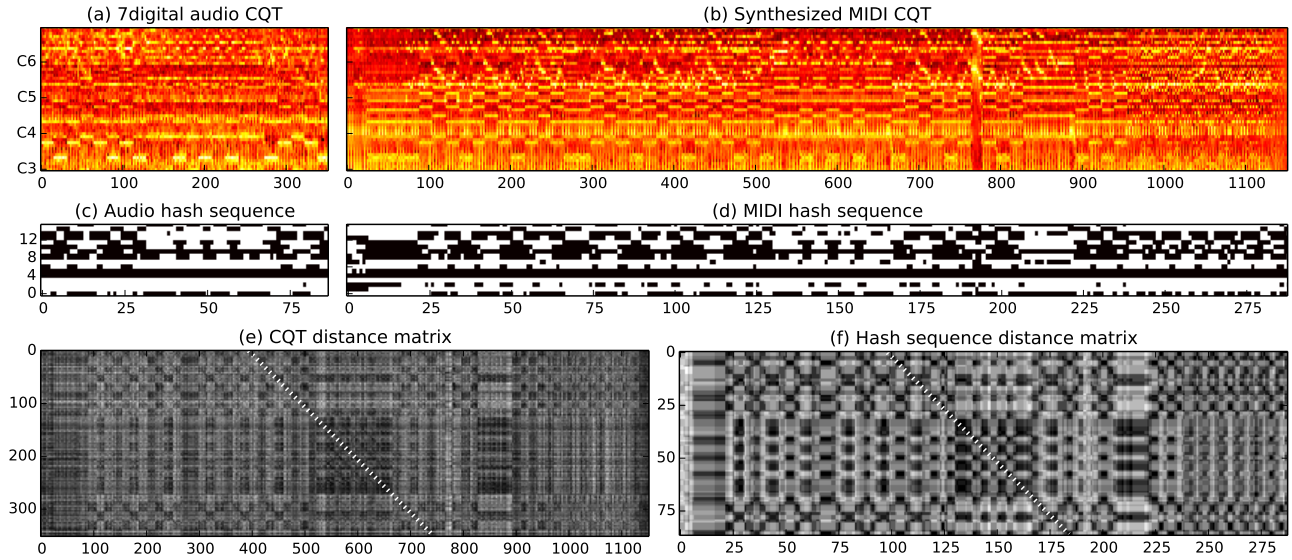

Figure 2. Audio and hash-based features and alignment for Billy Idol - "Dancing With Myself" (MSD track ID `TRCZQLG128F427296A`). (a) Normalized constant-Q transform of 7digital preview clip, with beat indices and semitones on the horizontal and vertical axes respectively. (b) Normalized CQT for synthesized MIDI file. (c) Hash bitvector sequence for 7digital preview clip, with pooled beat indices and Hamming space dimension on the horizontal and vertical axes respectively. (d) Hash sequence for synthesized MIDI. (e) Distance matrix and DTW path (displayed as a white dotted line) for CQTs. Darker cells indicate smaller distances. (f) Distance matrix and DTW path for hash sequences.

alignments to use as training data, 25% as a "development set" to tune the content-based matching system, and the remaining 25% to use for final evaluation of our system. Due to duplication within and across datasets, care was taken to split based on *songs*, rather than by entry.

3. CROSS-MODALITY HASHING OF MIDI AND AUDIO DATA

It would be straightforward to use our alignment scheme to match MIDI files with no metadata to entries in a dataset of song recordings. The size of our entire MIDI dataset (140,910 files) suggests that we should use as large of an audio corpus as possible in hopes that an audio recording of the song corresponding to each MIDI file appears in the corpus. Towards that end, we will be matching against the Million Song Dataset, the largest standard corpus of popular music recordings. This necessitates about one million distance matrix and DTW cost calculations for each MIDI file. Given the size of our MIDI dataset, if a single MIDI-file-to-audio-dataset matching takes more than a few seconds, matching our entire MIDI dataset would be infeasible. This means that both computing the distance matrix and calculating the DTW cost for each MIDI/audio pair must take at most tens of microseconds.

Unfortunately, the method described above does not meet this need. The median number of beats in the 7digital preview clips from the MSD and the MIDI files are 186 and 1218 respectively. Computing a cosine distance matrix of this size (with a feature dimensionality of 48) using the highly optimized C++ code from `scipy` [12] takes on average 9.82 milliseconds on a Intel Core i7-4930k processor. When implemented using the LLVM just-in-time compiler Python module `numba`,[3] the DTW cost calculation described above takes on average 892 microseconds on the same processor. It follows that matching a *single* MIDI file to the MSD using this approach would take just under three hours. Clearly, a more efficient approach is necessary.

Calculating the distance matrix and the DTW cost are both $\mathcal{O}(NM)$ in complexity. The fact that computing the distance matrix takes about 10 times longer is then likely due to the fact that computing each entry in the distance matrix involves D each of multiply and addition operations. It follows that calculating the distance between feature vectors is a bottleneck in our system. In addition, any reduction in the number of feature vectors (beats) in each sequence will give quadratic speed gains for both DTW and distance matrix calculations.

Motivated by these issues, we propose a system which learns a common representation for the audio and MIDI CQTs in a Hamming space. By replacing constant-Q spectra with bitvectors, we replace the expensive distance computation with an exclusive-or operation followed by simple table lookup. This is thanks to the fact that computing the exclusive-or of two bitvectors a and b will yield a bitvector with 1s in the dimensions where a and b differ and 0s elsewhere, and that the number of 1s in all bitvectors of length D can be precomputed and stored in a table of size 2^D. In the course of computing our Hamming space representation, we also implicitly downsample the sequences over time, which provides speedups for both distance matrix and DTW calculation. Our approach has the additional potential benefit of learning a representation for comparing

[3] `http://numba.pydata.org/`

audio and MIDI constant-Q spectra, rather than simply using the cosine distance.

3.1 Hashing with convolutional networks

Our hashing model is based on the Siamese network architecture proposed in [13]. This approach considers the situation where we have feature vectors from two modalities, and two sets $\mathcal{P}$ and $\mathcal{N}$ which contain pairs of feature vectors from each modality such that if $(x, y) \in \mathcal{P}$ then x and y are considered "similar" and if $(x, y) \in \mathcal{N}$ they are "dissimilar". A nonlinear mapping is then learned from each modality to a common Hamming space such that similar and dissimilar feature vectors are respectively mapped to bitvectors with small and large Hamming distances. One possible objective function for this technique is

$$\mathcal{L} = \frac{1}{|\mathcal{P}|} \sum_{(x,y)\in\mathcal{P}} \|f(x) - g(y)\|_2^2 - \frac{\alpha}{|\mathcal{N}|} \sum_{(x,y)\in\mathcal{N}} \max(0, m - \|f(x) - g(y)\|_2)^2$$

where f and g are the nonlinear mappings for each modality, α is a regularization parameter which controls how important it is that the dissimilar pairs be mapped far apart, and m is a threshold above which dissimilar pairs are considered sufficiently distant.

Finding the nonlinear mappings f and g involves optimizing their parameters with respect to $\mathcal{L}$. The exact form of these nonlinearities is a system design choice; in [13] they are chosen to be multi-layer nonlinear networks. In the present work, we will use convolutional networks due to their ability to exploit invariances in the input feature representation. CQTs contain invariances in both the time and frequency axes, so convolutional networks are particularly well-suited for our task.

In order to apply this technique, we also need to obtain the collections of pairs of similar and dissimilar feature vectors $\mathcal{P}$ and $\mathcal{N}$. In our problem, our two feature modalities are CQTs from synthesized MIDI files and CQTs from audio files. We therefore build $\mathcal{P}$ by choosing pairs of constant-Q spectra from simultaneous beats in successfully aligned MIDI/audio pairs. For example, if $A, B \in \mathbb{R}^{N\times D}$ are CQTs of an aligned MIDI and audio file respectively, then $\mathcal{P}$ will include $(A[1], B[1]), \ldots, (A[N], B[N])$. The choice of $\mathcal{N}$ is less obvious. We found that simply randomly choosing CQT spectra from non-aligned beats in our collection of aligned MIDI and audio files achieved satisfactory results.

3.2 System specifics

Training our hashing model involves presenting it with training examples and backpropagating the gradient of $\mathcal{L}$ through the model parameters. Our dataset is the collection of successfully aligned MIDI and audio CQTs described in Section 2. We split this collection into 90% train and 10% validation, where the validation set was not used to optimize the model parameters. We z-scored the entire training set across feature dimensions and re-used the mean and standard deviation from the training set to z-score the validation set.

For efficiency, we used minibatches of training examples; each minibatch consisted of 50 subsequences, each of of length 100. These minibatches were extracted by choosing a random offset for each training sequence pair and cropping out a subsequence of length 100. For $\mathcal{N}$, we simply presented the network with mismatched pairs of subsequences; that is, if $(x, y) \in \mathcal{N}$ then x and y were subsequences from different songs. Once the network had trained on minibatches from the entire training set (one epoch), we repeated the random sampling process. For optimization, we used RMSProp, a recently proposed stochastic optimization technique [22]. Every 100 minibatches, we computed the loss $\mathcal{L}$ on the validation set. If the validation loss was less than 99% of the previous lowest, we trained for 1000 more iterations (minibatches).

While the validation loss is a reasonable indicator of network performance, its scale will vary depending on the α and m regularization hyperparameters. To obtain a more consistent metric, we also computed the distribution of distances between the hash vectors produced by the network for the pairs in $\mathcal{P}$ and those in $\mathcal{N}$. If these distributions are well-separated, we can assume that the network is performing well. We therefore used the Bhattacharyya distance between these distributions as an alternate performance metric [4].

The hashing networks for each modality were chosen to have the same architecture: A series of alternating convolutional and pooling layers followed by a series of fully-connected layers. All convolutional and fully-connected layers used rectifier nonlinearities except the final output layer, which (as in [13]) used a hyperbolic tangent nonlinearity. This choice allows us to obtain binary hash vectors by testing whether each output unit is greater or less than zero. We chose an output dimensionality of 16, motivated by the fact that bitvectors of this dimensionality can be efficiently stored as 16 bit unsigned integers. The first convolutional layer had 16 filters with a filter size of five beats by twelve semitones, which gives our network some temporal context and octave invariance. As advocated by [20], all subsequent convolutional layers had 2^{n+3} 3x3 filters, where n is the depth of the layer. All pooling layers performed max-pooling, with a pooling size of 2x2. Finally, as advocated by [10], we initialized all weights with normally distributed random variables with zero mean and a standard deviation of $\sqrt{2/n_{in}}$, where n_{in} is the number of inputs to each layer. Our model was implemented using `theano` [1] and `lasagne`.[4]

To ensure good performance, we optimized all model hyperparameters using Whetlab,[5] a web API which implements black-box Bayesian optimization [21]. We used Whetlab to optimize the number of convolutional/pooling layers, the number and size of the fully-connected layers, the RMSProp learning rate and decay parameters, and the α and m regularization parameters of $\mathcal{L}$. As a hyperparam-

[4] `https://github.com/Lasagne/Lasagne`

[5] `http://www.whetlab.com`

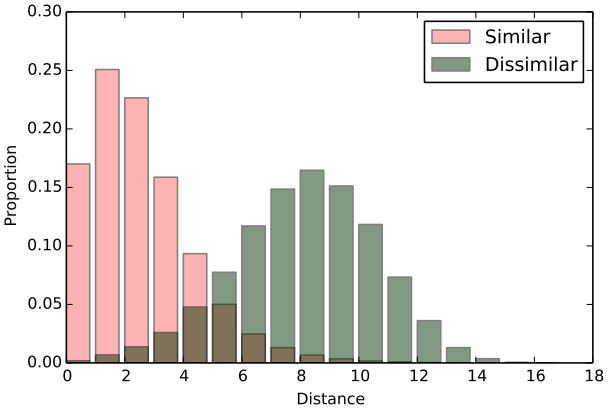

Figure 3. Output hash distance distributions for our best-performing network.

eter optimization objective, we used the Bhattacharyya distance approach described above. The best performing network found by Whetlab had 2 convolutional layers, 2 "hidden" fully-connected layers of with 2048 units in addition to a fully-connected output layer, a learning rate of .001 with a decay parameter of .65, and $\alpha = .5$ and $m = 4$. This hyperparameter configuration yielded the output hash distance distributions for $\mathcal{P}$ and $\mathcal{N}$ shown in Figure 3, which resulted in a Bhattacharyya distance of 0.488.

4. MATCHING MIDI FILES TO THE MSD

After training our hashing system as described above, the process of matching MIDI collections to the MSD proceeds as follows: First, we precompute hash sequences for every 7digital preview clip and every MIDI file in the clean MIDI subset. Note that in this setting we are not computing feature sequences for known MIDI/audio pairs, so we cannot force the audio's beat tracking tempo to be the same as the MIDI's; instead, we estimate their tempos independently. Then, we compute the DTW distance in the manner described in Section 2.2 between every audio and MIDI hash sequence. We can evaluate our system's performance based on how well it recovers the MIDI-audio pairs whose beat-synchronous CQTs produced a DTW score below our chosen threshold.

We tuned the parameters of the DTW cost calculation based on the results using the "development" set of successfully aligned MIDI/MSD pairs. For g, we found it beneficial to use a smaller value of $g = 0.9$. Using a fixed value for the non-diagonal move penalty avoids the percentile calculation on the simiarity matrix, so we chose $\phi = 4$. Finally, we found that it did not help to normalize by the average distance value $\mathcal{B}$, so we skipped this step.

4.1 Results

Bitvector sequences for the CQTs shown in Figure 2(a) and 2(b) can be seen in 2(c) and 2(d) respectively. Note that because our networks contain two pooling layers, each of which downsamples by 2 in the time dimension, the number of bitvectors is $\frac{1}{4}$ of the number of constant-Q spectra for each sequence. The Hamming distance matrix and lowest-cost DTW path for the hash sequences can be seen in Figure 2(f). For this example, it exhibits the same structure as the CQT-based cosine distance matrix shown in 2(e), and the same DTW path was recovered successfully.

Rank	1	10	100	1000	10000
Percent $\leq$	15.2	41.6	62.8	82.7	95.9

Table 1. Percentage of MIDI-MSD pairs whose hash sequences had a rank better than each threshold.

To evaluate our system, note that we know ahead of time which MSD entry each MIDI should match to based on the metadata-based matching and CQT-based DTW score we precomputed. A useful metric for is therefore to rank MSD entries according to their hash sequence DTW distance to a given MIDI file. We can then determine the rank of the correct match for each MIDI file. In this setting, a common way to evaluate performance is to compute the mean reciprocal rank; on our test set, we achieved a value of 0.241, indicating that the correct matches tended to be ranked highly. We find that a more informative metric is to compute the percentage of MIDI files in our test set whose correct match ranked below a certain threshold; this metric is shown for various thresholds in Table 1.

Studying Table 1 reveals that we can't rely on the hash sequence for the correct MSD entry being matched to MIDI hash sequence exactly, i.e. the DTW distance for correct matches is only the smallest across the entire MSD about 15.2% of time. Furthermore, for a significant portion of our MIDI files, the correct match did not rank in the top 1000. We found that this was usually caused by the MIDI file being beat tracked at a different tempo than the audio file, which inflated the DTW score. For some of the MIDI files whose ranks were small but not 1, the MIDI file was erroneously matched to a different version (cover, remix, etc.) of the correct entry. Finally, some degree of inaccuracy can be attributed to the fact that our hashing model is not perfect and the MSD is large. So, in a relatively small proportion of cases, the MIDI hash sequence ended up being very similar to many MSD hash sequences, and so the correct entry was not ranked first.

Because our system does not uniformly assign the lowest DTW score to the correct MSD entry for our MIDI files, it is more realistic to look at our system as a pruning technique; that is, it can be used to discard MSD entries which we can be reasonably confident are not the correct match for a given MIDI file. For example, if we use our system to compute the hash-based DTW score between a MIDI file and every entry in the MSD, we can discard all but 1% of the MSD and only risk discarding the correct match about 4.1% of the time. Pruning methods are only valuable when they are substantially faster than performing the original computation; fortunately, our approach is orders of magnitude faster: For the median hash sequence lengths, calculating a Hamming distance matrix between hash sequences is about 400 times faster than computing the CQT cosine

distance matrix (24.8 microseconds vs. 9.82 milliseconds) and computing the DTW score is about 9 times faster (892 microseconds vs. 106 microseconds). These speedups can be attributed to the fact that computing a table lookup is much more efficient than computing the cosine distance between two vectors and that, thanks to downsampling, our hash-based distance matrices have $\frac{1}{16}$ of the entries of the CQT-based ones. In summary, a straightforward way to describe the success of our system is to observe that we can, with high confidence, discard all but 1% of the entries of the MSD by performing a calculation that takes about as much time as matching against 1% of the MSD.

5. FUTURE WORK

Many pruning techniques have been proposed for the general case of large-scale DTW search; an overview can be found in [18]. Applying these approaches to our current system would allow for further speed improvements. Unfortunately, most of these techniques rely on the assumption that the query sequence is of the same length as all of the sequences in the database, so they would need to be modified before applying them to our problem. In terms of accuracy, as noted above most of our hash-match failures can be attributed to erroneous beat tracking. With a better beat tracking system or with added robustness to this kind of error, we could improve the pruning ability of our approach. Even without these improvements, our system can, in a reasonable amount of time, tackle the problem we set out to solve: Resolving our huge MIDI collection against the MSD.

6. REFERENCES

[1] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] Adam Berenzweig, Beth Logan, Daniel P. W. Ellis, and Brian Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

[3] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24-28, 2011, Miami, Florida*, pages 591–596. University of Miami, 2011.

[4] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.

[5] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[6] Michael Scott Cuthbert and Christopher Ariza. `music21`: A toolkit for computer-aided musicology and symbolic music data. In *Proceedings of the 11th International Conference on Music Information Retrieval*, pages 637–642, 2010.

[7] Tuomas Eerola and Petri Toiviainen. MIR in Matlab: The MIDI toolbox. In *Proceedings of the 5th International Conference on Music Information Retrieval*, pages 22–27, 2004.

[8] Sebastian Ewert, Meinard Müller, Verena Konz, Daniel Müllensiefen, and Geraint A. Wiggins. Towards cross-version harmonic analysis of music. *IEEE Transactions on Multimedia*, 14(3):770–782, 2012.

[9] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[11] Ning Hu, Roger B. Dannenberg, and George Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 185–188. IEEE, 2003.

[12] Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for Python. 2014.

[13] Jonathan Masci, Michael M Bronstein, Alexander M Bronstein, and Jürgen Schmidhuber. Multimodal similarity-preserving hashing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(4):824–830, 2014.

[14] Brian McFee, Matt McVicar, Colin Raffel, Dawen Liang, and Douglas Repetto. librosa: v0.3.1, November 2014.

[15] Cory McKay and Ichiro Fujinaga. jSymbolic: A feature extractor for MIDI files. In *Proceedings of the International Computer Music Conference*, pages 302–305, 2006.

[16] Meinard Müller. *Information retrieval for music and motion*, volume 2. Springer, 2007.

[17] Colin Raffel and Daniel P. W. Ellis. Intuitive analysis, creation and manipulation of MIDI data with `pretty_midi`. In *Proceedings of the 15th International Conference on Music Information Retrieval Late Breaking and Demo Papers*, 2014.

[18] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

[19] Alexander Schindler, Rudolf Mayer, and Andreas Rauber. Facilitating comprehensive benchmarking experiments on the million song dataset. In *Proceedings of the 13th International Conference on Music Information Retrieval*, pages 469–474, 2012.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[22] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

[23] Derek Tingle, Youngmoo E. Kim, and Douglas Turnbull. Exploring automatic music annotation with "acoustically-objective" tags. In *Proceedings of the international conference on Multimedia information retrieval*, pages 55–62. ACM, 2010.

[24] Robert J. Turetsky and Daniel P. W. Ellis. Ground-truth transcriptions of real music from force-aligned MIDI syntheses. *Proceedings of the 4th International Conference on Music Information Retrieval*, pages 135–141, 2003.

[25] Douglas Turnbull, Luke Barrington, David Torres, and Gert Lanckriet. Towards musical query-by-semantic-description using the CAL500 data set. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 439–446. ACM, 2007.