Exemplar-Based Window-wise Onset/Offset Detection Model for Multi-Pitch Extraction

Final project for ECE477 – Computer Audition

Xiaohe Tian

I. Introduction

In this project, we build a multi-pitch extraction model that makes use of exemplar audio clips. The target audio input is supposed to be polyphonic with a single instrument/timbre. Apart from that input, we also introduce a short exemplar audio input containing the timbre information of the target audio input.

The motivation of this project comes from not only the versatile, creativity-inducing usages of the extracted multi-pitch data in music creation and production, but also from the difficult nature of multi-pitch extraction, where the overlapping harmonics confuse machine learning models beyond the intuition of human beings. Thus, we consider exemplar a good means for a breakthrough: an exemplar contains timbre information, which by spectral transformation can become certain patterns for the machine to recognize; it is easy to obtain from a full audio clip, along with pitch and volume information, for both human beings and machine learning models. Besides, state-of-the-art instrument separation techniques are advanced enough to make us believe single-instrument audio clips can be reliably obtained from multi-instrument music pieces. An exemplar can also help reconstruct music from extracted sheet data in the future.

One issue around implementing this idea is the dataset: none of the popular single-instrument and polyphonic music datasets contain exemplars, or similar representations of the timbre in the main audio. That is why we decide to build a music dataset by ourselves and make it well suitable for our tasks. We try to make that dataset as diverse in timbres as possible, as some instruments are transitional in sound upon playing while others are enduring.

For the model, we refer to the Deep Salience method in multi-pitch extraction [1]. We make the model based on windows rather than songs, with inputs taken from several frames of the audio spectrogram, so the processing of full audio of any length can be performed iteratively. Regarding this, we introduce frame-wise note event matrices, which contains note events, i.e., information of ongoing notes played at this frame. A note event matrix documents the pitch, velocity, and duration (frame passed since the note is played) of every active event at this frame. Those matrices, with their small size and frame-based nature, are suitable as recurrent inputs for our model.

This report is structured as this: Part II reviews the related paper works and datasets. Part III introduces the construction method for our dataset. Part IV details our proposed exemplar-based window-wise onset/offset detection model. Part V shares the results from this model and discusses potential methods for improving and/or extending the ideas in this project. Finally, Part VI draws the conclusions.

II. Related Works

Our work refers in a large part to the Deep Salience method for the multi-pitch extraction task by Rachel M. Bittner et al. [1]. Works related to this project also include joint onset and frame prediction with recurrent neural network by C. Hawthorne et al. [2], WaveNet autoencoder-based audio synthesis from music notes by J. Engel et al. [3], and differentiable digital signal processing by J. Engel et al. [4]. While we build our own dataset for this project, there are some related music datasets that are well established for multi-pitch extraction tasks, including MusicNet [5] and MAESTRO [6] datasets.

III. Dataset Construction

We build the dataset for this project through music creation process where single-instrument polyphonic songs are rendered from MIDI piano rolls in Reaper, while the instrument for playing on a piano roll is arbitrarily selected from VST plugin instruments. Namely, we use synth sounds from Vital VST instrument, which can emulate the sounds of multiple instruments as well as making customized sounds. Regarding the selected instrument, we also render a one-note MIDI to represent its timbre, which lasts slightly longer than 1 second.

We try to represent a collection of timbres as diverse as possible, and a key factor to take into consideration is the decay and sustain properties: sounds of some instruments sustain upon playing (string and horn) or at least linger for a while (piano and guitar), while sounds of other instruments vanish quickly, such as xylophone. For the latter case, offsets will mostly be meaningless. We have to consider whether the MIDI offsets will be meaningful for the real instrument in training and testing. Nevertheless, we restrain fancy effects on this VST instrument, such as flanger, phaser, distortion, reverb, etc. to make the sounds more representative for common, non-synth instruments.

3.1. Song-wise folder data

We create the data folders by songs: as mentioned, the songs are single-instrument and polyphonic. For each song, we create a folder which contains: (1) the full audio in .wav format; (2) an exemplar audio of ~1 second in .wav format, representing the timbre of the full audio, with its pitch and velocity noted in the file name; (3) a MIDI data sheet in .csv format, containing the start time, end time, pitch and velocity information of every note. The MIDI data sheet is extracted in Reaper using a .lua script.

The exemplar audio is named this way: “exemplar_XX_YY_ZZ.wav”. Here, XX is the pitch of this exemplar note, such as “C4”, “G#3” or “Bb5”. YY is the velocity of this exemplar note (an integer between 0 and 127). ZZ is the tempo of the project used to render the full audio, which is not used in this project but may have potential uses. Therefore, the pitch name should be transcribed into note numbers (0~128) later in parsing the file name.

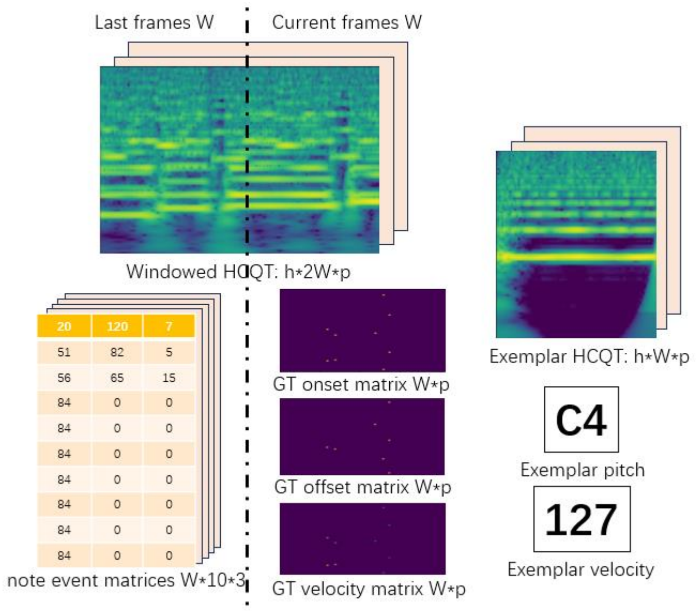
3.2. Window-wise PyTorch data

Then, we import the folder data and build the PyTorch Dataset class, whose items are as these: (1) windowed audio harmonic constant-Q transform (HCQT) of shape $(h, 2W, p)$, built with CQT function in librosa library; (2) exemplar audio HCQT of shape (h, W, p), built with CQT function in librosa library; (3) exemplar pitch (constant), obtained from exemplar audio file name; (4) exemplar velocity (constant), obtained from exemplar audio file name; (5) target (or ground truth) onset matrix of shape (W, p), which is a binary matrix indicating the frames and pitches where a note is set on; (6) target offset matrix of shape (W, p), which is a binary matrix indicating the frames and pitches where a note is set off; (7) target velocity matrix of shape (W, p), which contains numbers 0~127 indicating the velocity at the frames and pitches where a note event is ongoing; (8) note event matrices for the past frames of shape $(W, 10, 3)$, indicating matrices of shape (10, 3) for W frames; we will explain note event matrices later.

Here, h indicates the number of harmonic components of HCQT. For example, h is set to 5 in this project as we use (0.5, 1, 2, 3, 4) harmonic components as described in [1]. W is an integer indicating the window length, or number of frames from CQT transform for 1-second audio, regarding sample rate and hop size:

$$W = \text{ceil}\left(\frac{sample\ \ rate}{hop\ \ size}\right)$$

And in this project, as we use a sample rate of 44100 and hop size of 512, the resulted W is 87. p indicates the number of pitches for detection; in this project, we select p as 84, as the librosa CQT function forbids note frequencies higher than half of the sample rate. We select the lowest note as C1, and discard the 24 pitches lower than that and the highest 20 pitches of the original 128 pitches in MIDI data sheet. Hence, each windowed audio HCQT contains information for two seconds of audio, while each exemplar audio HCQT contains information for 1 second of exemplar audio, and we crop the exemplar audio to exactly 1 second in length.



20	120	7
51	82	5
56	65	15
84	0	0
84	0	0
84	0	0
84	0	0
84	0	0
84	0	0
84	0	0

Fig. 1: full contents of one set of data. The dashed line indicates the division between last and current frames. The note event matrices correspond to last frames while ground truth onset, offset, and velocity matrices correspond to current frames. The structure of note event matrix is also illustrated in this figure.

As there are $87 \times 2 = 174$ frames in windowed HCQT, the first 87 frames are regarded as “last frames” while the second 87 frames are regarded as “current frames”. In the model, our goal is to detect the onset, offset and velocity information of the current frames. Hence, the note event matrices correspond to “last frames”, and the target onset, offset and velocity matrices correspond to “current frames”. The full content of one set of data is demonstrated in Fig. 1.

3.3. Note event matrix

A note event matrix contains the complete MIDI data sheet information at a single frame. On the one hand, it keeps the complete information from MIDI sheet data while has a frame-based structure; on the other hand, it is significantly smaller in size than onset and velocity matrices. Therefore, note event matrices are suitable for our dataset and window-based model. The table at the bottom left of Fig. 1 illustrates the structure of a note event matrix.

Each note event matrix is (10, 3) in shape with every row representing an active note event. The three columns represent pitch, velocity and duration, respectively. The first value is the pitch of this active note between $0 \sim p - 1$. The second value is the velocity of this active note between 0~127. The third value is the number of frames passed since the note is set on, so that information about notes set on before the window is preserved. The number of rows indicates our designated maximal number of pitches played at the same time, which is 10 when the dataset is built as we assume it is the “maximal maximal” number of note events; we can later reduce the allowed maximal number of note events for the model (for example, 7 in this project). Other than rows representing active note events, rows representing no events are deemed “empty rows” and formatted as $[p, 0,0]$, where number p indicates no pitch since we use $0 \sim p - 1$ as pitch representations. Then in every matrix, rows are sorted by pitch numbers, and thus, active event rows stay on top and empty rows fall to the bottom.

IV. Multi-Pitch Extraction Model

We propose a window-based onset/offset detection (WOOD) model for the multi-pitch extraction task, which produces multi-head outputs of onset, offset and prediction matrices. The inputs for the model are: (1) windowed HCQT; (2) exemplar HCQT; (3) exemplar pitch; (4) exemplar velocity; (5) previous note events. They correspond to (1)(2)(3)(4)(8) of dataset items as described in Section 3.2. Here, we designate a new maximal number of events for the model no larger than that for the dataset (10), so accordingly, we take off the last $10 - n$ rows of the note event matrices. All the inputs are taken in by batches, so an extra dimension indicating the number of batches, B, is added to every input tensor.

The outputs for the model are: (1) predicted onset matrix of shape (B, W, p), where every element is a value between 0 and 1 predicting the existence of onsets at every frame and pitch, which will be applied a threshold later in testing; (2) predicted offset matrix of shape (B, W, p), where every element similarly predicts the existence of offsets at every frame and pitch; (3) predicted velocity matrix of shape (B, W, p), where every element is a value between 0 and 1 representing the normalized velocity at every frame and pitch. Note that velocity values should only take effect where onsets exist, as a velocity is only set where a note is set on.

4.1. Model structure

The model is structured as such: we build embeddings for four components, namely windowed HCQT, salience from windowed HCQT, exemplar contents (HCQT + pitch + velocity), and note event matrices. Each embedding has a shape of (B, W, d) so that they can be fused together (Here, d is the input dimensionality of the model, which we assign as 256). The fused embedding is then fed into a gated recurrent unit (GRU). Finally, we apply three heads to the GRU output, resulting in the three outputs of the model: onset, offset, and velocity matrices. The overall structure of the model is shown in Fig. 2.

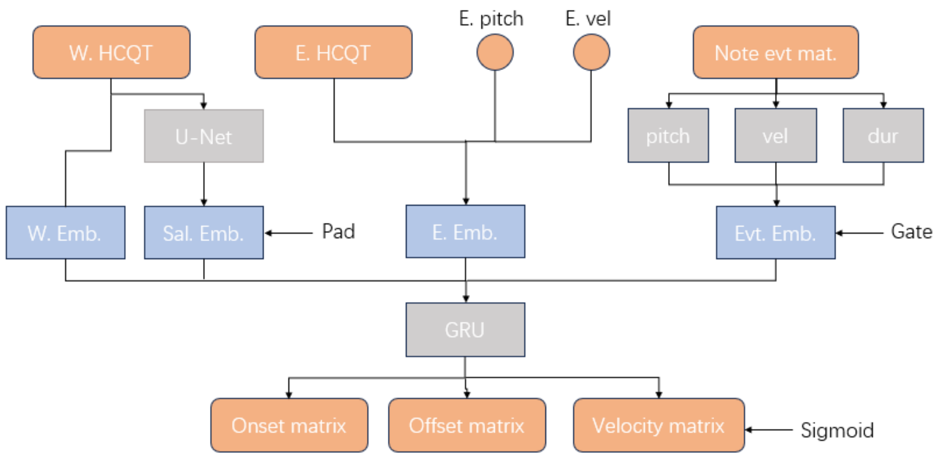


Fig. 2: overall structure of the WOOD model.

To look into the details of the model structure, the windowed HCQT embedding is made through an encoder that contains three convolutional layers followed by a ReLU activation function each, followed by meaning out the pitch dimension. The salience embedding is made by transforming windowed HCQT input into salience with a U-Net model, as with the Deep Salience method proposed by [1], and then applying a linear transformation. Note that since there are three size halving and doubling processes each inside the U-Net model, the window length at the input may not be the same as that at the output if it is not divisible by $2^3 = 8$. In this case, we apply a padding after the linear transformation to make sure its shape is fusion-ready. For the exemplar embedding, we process exemplar HCQT with an encoder similar to that for windowed HCQT, which is followed by squeezing out the pitch and window length dimensions. Then, the resulted tensor with shape (B, d) is added up with the embeddings of exemplar pitch through lookup table layer and exemplar velocity through linear transformation layer. Finally, the overall exemplar embedding is added a dimension and expanded by W to match the embedding shapes.

For the note event embedding, the three columns of note event matrices are separately made into embeddings, through lookup table layer (for pitch and velocity) and double linear layer transformation (for duration). The resulted embeddings are then combined together along d dimension to feed into a double linear layer transformation that reduces the dimensionality size from $3d$ to d, and the number-of-notes dimension n is meaned out. A gate is applied to the combined tensor to make its weight controllable, as different properties of input note event matrices are involved during training and testing. Finally, the tensor is expanded by W to get the note event embedding.

The fusion is done for all four embeddings above: salience, windowed HCQT, exemplar, and note event embeddings, along d dimension. The fused embedding then goes through a linear transformation to reduce the size of combined dimension from $4d$ to d before being fed into a gated recurrent unit (GRU) neural network. To the output of the GRU, we apply a head (linear transformation) to get each of onset, offset, and velocity matrices, which are the outputs of our model. All three matrices should be applied a sigmoid function to make values fall between (0, 1) as we expect, but because we will use BCEWithLogitsLoss function for onset and offset matrices, only velocity matrix is applied a sigmoid function.

4.2. Loss computation

For loss computation, we apply BCEWithLogitsLoss function in PyTorch's nn library to each of onset and offset matrices, while the velocity loss is calculated by

$$L_{vel} = \frac{\sum(\mathrm{abs}(\boldsymbol{V}_p - \boldsymbol{V}_t) * \boldsymbol{O}_t)}{\sum \boldsymbol{O}_t + \varepsilon}$$

where $\boldsymbol{V}_p$ is the predicted velocity matrix, $\boldsymbol{V_t}$ is the ground truth velocity matrix, and $\boldsymbol{O}_t$ is the ground truth onset matrix. The abs() here takes the absolute values from element-wise subtractions, and the binary $\boldsymbol{O}_t$ serves as a mask indicating velocities take effect only at frames and pitches where onsets exist. ε is a small number added to avoid zero denominator. The three losses are added together with a value λ to control the weight of the velocity loss:

$$L_{total} = L_{onset} + L_{offset} + \lambda \cdot L_{vel}$$

V. Results and Discussions

5.1. Training and testing

We train our model on Google Colab with T4 GPU. For the training part, we select a maximal number of note events for the model smaller than the maximal number assigned in the dataset (10). For example, we select 7 as the maximal note events here in the model, and so, we remove the last three rows of each note event matrix built from the dataset. We use Adam optimizer with a learning rate of 10^{-4}. We trained for 200 epochs, using λ=0.5 for the weight of the velocity loss. The resulted total loss curve and individual loss curves are shown in Fig. 3.

For the testing part, we build a slightly different dataset, where we base data on songs rather than windows. The number-of-frame dimensions of full (windowed) audio, onset matrix, offset matrix, velocity matrix, and note event matrices go from W to W_s, which is how many frames we get if we take CQT on the whole song.

We test one full song with the model in an iterative fashion: we iterate our testing loop and make model inputs window by window, with W frames taken in each time.

The frame time t iterates from 0 to W_s with a hop size of W. The windowed HCQT of window size $2W$ at corresponding position is taken from the full HCQT for the input. For the outputs, onset and offset matrices are applied with a sigmoid function and then a threshold each, and turned into binary matrices. The velocity matrix is multiplied by 127 rounded to integers to represent real velocities. Those three matrices can thus help build note event matrices of the W current frames. Those note event matrices are then taken as the input for the next iteration of this model.

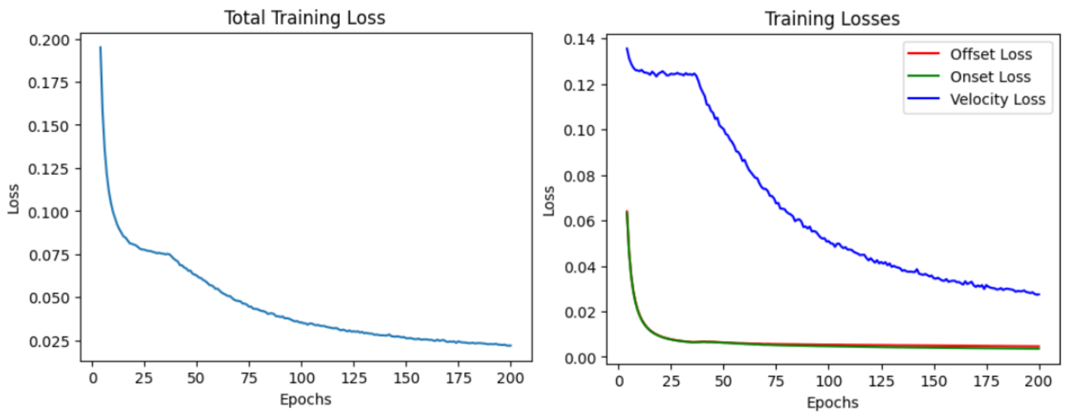


Fig. 3 (a) and (b): total training loss (with velocity loss weight λ=0.5) and three individual losses (onset, offset, velocity) versus epochs. Losses from first three epochs are removed to enhance the visibility of the plots.

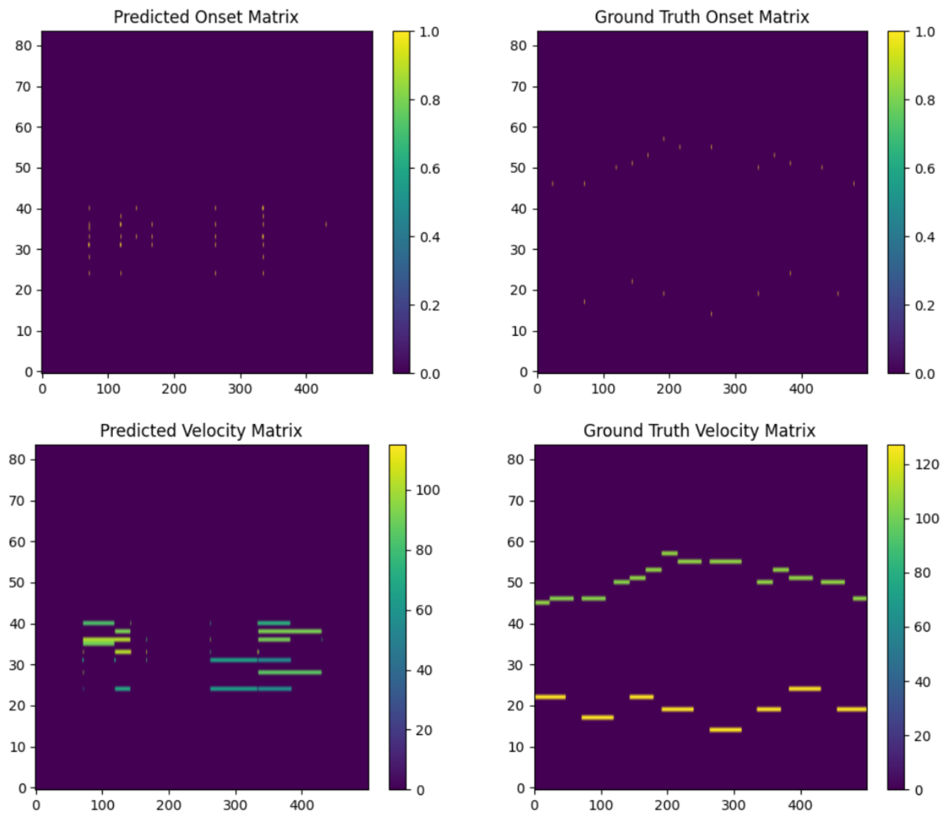


Fig. 4 (a) and (b): one example of predicted onset and velocity matrices in comparison with ground truth matrices. We only take the first 500 frames to see the iterative performance of the model.

After the model is run on the whole song, note event matrices for every frame are built. With them, we can construct the onset, offset and velocity matrices of the whole song, and then compare them with ground truth matrices. We tune the thresholds for onset and offset matrix binarizations to make the built matrices look reasonable.

One example of the onset and velocity matrices in comparison with the ground truths for one song are shown in Fig. 4. In this figure, we only look at the first 500 frames to get an illustration of how well the model performs iteratively. From the figures, we can see that the model is not suitable yet for reliable outputs. However, there are some clues showing we are making progress on this model: (1) the positions of onsets are mostly on point; (2) while the pitches are not accurately predicted, they are mostly within the reasonable interval. Meanwhile, through testing on multiple songs, the thresholds we set to get reasonable matrix plots are not consistent: for longer songs, the thresholds tend to be set lower.

5.2. Discussions

Given the novel nature of our multi-pitch extraction approach, we can think of many ways to improve the model:

- One challenge is how to decide long-term losses and train to model based on them. So far, during training, we have only evaluated losses for the current frames. However, since we test on full songs, we should be able to test on long-term losses too. Specifically, in the training process, we should make note event matrices for the next iteration from matrix be the outputs of this iteration. This process may cause some non-linearity and make loss back propagation impossible. Hence, this is an important yet challenging way to improve the performance of our model.
- As for exemplars, we only use one exemplar of one single note at a short length (1 second) so far. It is possible to improve our model by making use of multiple exemplar inputs, or more complex exemplars (e.g., one with multiple notes played together). It is also possible to use a longer exemplar. In addition, we make exemplar audios and full audios separately, but as we talked about, we think the feasibility of this model falls at extracting exemplars from the original audio. So, that is also a method we can try on. It may also be useful to predict the exemplar in some way to extend its length, as a real note played in songs can last longer than the length of exemplar.
- As for the dataset, we can and should work on creating more data and making the timbre/harmonic/rhythmic combinations more diverse. And several tunable parameters, including the gate for note event embedding, weights of the three loss metrics, and binarization thresholds, are still open to explorations and investigations.
- Once we obtain a model that outputs reliable sheet data, a natural next step will be to reconstruct the audio. The exemplar used in this project can be useful for this reconstructing process, while other timbres and instruments are good for this task too. An audio reconstruction scheme can be come up with to increase the perceptibility for testing the model as well as demonstrate the potential applications of this model.

VI. Conclusion

In this project, we build a window-wise onset/offset detection (WOOD) model meant for pitch extraction on single-instrument polyphonic music. The model makes use of exemplar audio, and regarding that, we build a song-wise dataset where an exemplar audio documents the timbre information of the respective song. We also introduce note event matrix that stores note event information in a frame-wise and memory-friendly way. Deep Salience method and gated recurrent unit (GRU) are applied in building this model. Results from testing the model on full songs shows that the model has some ability in locating the frames of onsets and the interval of pitches, although it has weaker performance in determining the pitch locations.

References

[1] Bittner R M, McFee B, Salamon J, et al. Deep Salience Representations for F0 Estimation in Polyphonic Music. *ISMIR. 2017: 63-70*. 2017.

[2] Hawthorne C, Elsen E, Song J, et al. Onsets and frames: Dual-objective piano transcription. *arXiv preprint arXiv:1710.11153*, 2017.

[3] Engel J et al., Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. http://arxiv.org/abs/1704.01279. 2017.

[4] J. Engel et al. "DDSP: Differentiable Digital Signal Processing". *International Conference on Learning Representations*. 2020.

[5] https://www.kaggle.com/datasets/imsparsh/musicnet-dataset

[6] https://magenta.withgoogle.com/datasets/maestro