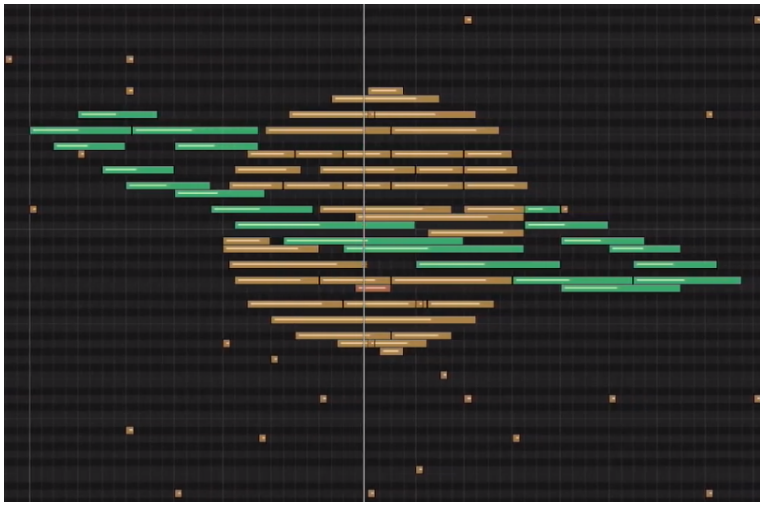
wav2midi: An Unsupervised Music Format Translation Model



Peter Schaldenbrand, Husni Almoubayyed
{pschalde, halmouba}@andrew.cmu.edu

Abstract

We created a deep, convolutional model that can translate waveform audio into music in the MIDI protocol which is comparable to a piano roll. The translation model, `wav2midi`, is trained using a dataset of waveform audio files and a dataset of MIDI files that are not explicitly paired together based on the song. `wav2midi` is able to capture the pitches of the incoming song but struggles to predict rhythm. In note prediction without considering rhythm, `wav2midi` is comparable to baseline models that were trained in a supervised fashion.

1 Project Definition

We attempt to translate a waveform audio file with arbitrary instruments into a vector representation of music that still retains key details of the song. We have vast data sets of music in either the .wav or .mp3 format, which we will refer to as waveform audio in this paper. Our goal was to translate this into a representation that more resembles how humans process music using only unpaired MIDI and .wav files using only unpaired waveform and MIDI files as training data.

Our target representation of music is a list of vectors which is comparable to the MIDI protocol. Each vector contains information about an instrument playing a note or notes: which note or chord, how loud to play it, and which instrument to use. This representation of music is far more intelligible to humans visually.

We can use a synthesizer to translate MIDI music back into waveform audio (`midi2wav`), but the reverse is currently not possible. We learn this `wav2midi` function using only unpaired datasets. Similar to CycleGAN [9], we feed waveform into our model, translate them into MIDI, then translate them back in to waveform. Our model learns using the loss between the original waveform audio and the predicted waveform. Similarly we input MIDI data and get MIDI data out.

Even though our model tries to perfectly recreate the input song with the MIDI representation, it still may be considered creative. The vector representation does not have all the instruments available that the original song had so the model needs to be creative to try to recreate the input song with limited resources. We can personify the model as an instrumentalist who is trying to perfectly recreate a given song using only the instruments they have on hand. The model could be creative in its choices such as instrument, note, or rhythm while trying to recreate a song.

2 Technical Challenges

The obvious method to try to translate audio from waveform to vector representation is to find a large data set of waveform to vector song pairs and treat it as a supervised learning problem. For example, you could have the Beatles discography of .mp3 files and some MIDI files that correspond to each of these. The closest thing to a dataset like this is the MAESTRO dataset [3] which perfectly pairs piano waveform audio and the MIDI representation of it. But using this dataset will not allow our model to generalize to arbitrary instruments and will not satisfy our goal of translating the music without using paired files. Our goal in this project was to utilize the abundance of waveform and MIDI files to accomplish this translation task.

We use the FMA data set [6] which is composed of 106,574 tracks from 16,341 artists and 14,854 albums. A large data set is necessary for this translation task due to the complexity of waveform music and the need for the model to generalize beyond the training set. .wav files, like the ones in the fma data set, are often sampled at $\sim$40,000 data points per second leading to the full size of

Note/Pitch	ΔTime	Instrument	Is Drum	Velocity	Duration	Is Chord
One Hot (128)	Float	One Hot (52)	Binary	Float 0-128	Float	Binary

Table 1: Feature representation of a MIDI file. Can be converted to MIDI and back without loss.

our data set being 917 gigabytes. This provided a huge technical challenge to be able to train our model on all of these songs. We tackled this challenge using Amazon Web Services resources and a GPU cluster at Carnegie Mellon University.

Defining the vector representation of music, which can be seen in Table 1, was a difficult challenge. The representation needed to be complex enough to capture the nuances of music, but needed to be simple enough so that the model is able to learn. The vector representation can be easily turned back into waveform audio using a synthesizer, however, we needed to be able to convert it using a differentiable network in order to back-propagate the error term through our whole model. Creating a network that perfectly mimics the synthesizer was a great challenge. We can create samples of the vector representation to have an infinite amount of data to train this `midi2wav` network.

3 Related Work

There are currently several online tools that aim to convert mp3 to midi files, using deterministic methods, however trials have shown underwhelming results[1]. Most use deterministic methods to match pitch and assume single instruments. Their results are often only accurate when using piano and limiting usage of chords and outside noise.

As machines become more powerful, using raw waveform data in machine learning models has become more possible whereas it was too high dimensional to use on weaker, older machines. WaveGAN and SpecGAN [1] are two examples of two models that have the same task but output different formats of audio data. Both models generate audio but WaveGAN outputs raw waveform data while SpecGAN outputs a spectrogram which can be converted in a lossy way to waveform data using the Griffin-Lim algorithm. The creators of the models conclude that the SpecGAN was likely learning more about audio than the WaveGAN, but the SpecGAN results sounded worse due to the artifacts introduced from the Griffin-Lim algorithm. For this reason, we decided to use spectrograms in our project. We are not outputting raw waveform audio so we want to use an audio format that will help our model learn more and faster.

Our first thought at approaching our translation goal was to directly convert waveform audio into MIDI using paired samples. This was the method used to train the Wave2`midi2wave` [] model. What makes Wave2`midi2wave` possible is the MAESTRO [3] dataset which consists of piano recordings with their corresponding MIDI representation captured by using a special piano with electronic sensors to record the key strokes to create the MIDI file. Wave2`midi2wave` works extremely well on piano recordings but will not generalize to other instruments. Additionally, it relies on the paired waveform and MIDI dataset which was extremely difficult and costly to create, leading to suspicions that it will not be reproduced with other instruments.

CycleGAN [9] is a translation model that has been effective in translating images from one type to another such as apples into oranges. CycleGAN does not require picture pairings between the classes as it only requires a large dataset of pictures from each class. CycleGAN utilizes two

[1] `https://www.bearaudiotool.com/mp3-to-midi`

translation networks to translate an image of class A into an image of class B, then back to A. The model can calculate the loss between the original image of class A and the translated image and back propagate the loss through the model. This is the approach that our model will use to train. CycleGAN only works on image classes that are very similar whereas MIDI and waveform audio are extremely different. We utilize a synthesizer to help train the `midi2wav` portion of our model which helps the network overcome these huge differences in music format.

`pix2pix` [4] is another image translation model that learns by training on paired images of the two classes. The `pix2pix` model is a deep convolutional architecture that down samples and convolves seven times then upsamples and deconvolves for six times to output an image of the same size but of a different class. `pix2pix` utlizes a U-Net [7] architecture which has "skip connections" that allow information from previous layers to skip some layers and connect to later layers. In the case of `pix2pix` the k^{th} layer is connected directly to the $(k+1)^{th}$ layer and the $(n-k)^{\text{th}}$ layer where n is the number of layers. In the `midi2wav` and `wav2midi` models, we do not want any bottlenecks of information. The music data needs to travel all the way through the network since it is valuable to the output as opposed to a model that might classify a song using a boolean value. We employ the `pix2pix` architecture for both `midi2wav` and `wav2midi` models referencing `https://github.com/eriklindernoren/Keras-GAN/blob/master/pix2pix/pix2pix.py` for code implementation.

4 Methods

A representation of our model can be seen in Figure 1. We chose spectrograms to represent the waveform audio because spectrograms are easier for neural networks to train on [1]. The spectrograms are of size 200×160 meaning that the frequency vector is size 200 and the time vector is of size 160 which corresponds to about 4.5 seconds of audio. We represent the MIDI files as a list of vectors of size 7 as seen in Table 1. There are 40 notes in each MIDI sample. Each note has an associated pitch, time since previous note, instrument, possibility to be drums, a loudness, and a Boolean for whether it is part of a chord or not.

Our translation models, `wav2midi` and `midi2wav`, were both modelled after the pix2pix [4] architecture, and each has around 10,000,000 parameters. The `midi2wav` model outputs a spectrogram of size 200×160 and uses no activation function since the spectrograms are made up of values in a normal distribution. The `wav2midi` model has 7 outputs corresponding to the MIDI feature representation in Table 1.

We start by training the `midi2wav` model using MIDI files and their corresponding spectrograms produced by the FluidSynth[2] synthesizer. We used 100,000 samples created from real MIDI files and generate 400,000 random MIDI samples by selecting values at random to train the `midi2wav` model. A large dataset is necessary to train this model in order to cover the near infinite different possibilities of MIDI input files. We had a held out set of MIDI samples that we computed the loss for but did not train on. The `midi2wav` model was considered trained when this held out set's loss stopped decreasing.

Once the `midi2wav` model was trained, we began to train the `wav2wav` and `midi2midi` models which are made up of the `midi2wav` and `wav2midi` models chained together. When training `wav2wav` and `midi2midi`, the `midi2wav` parameters are fixed/non-trainable. These parameters are fixed for two reasons: (1) the `midi2wav` model is only supposed to mimic the synthesizer, so its training was complete in the previous step (2). If we train the parameters of both the `midi2wav` and

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

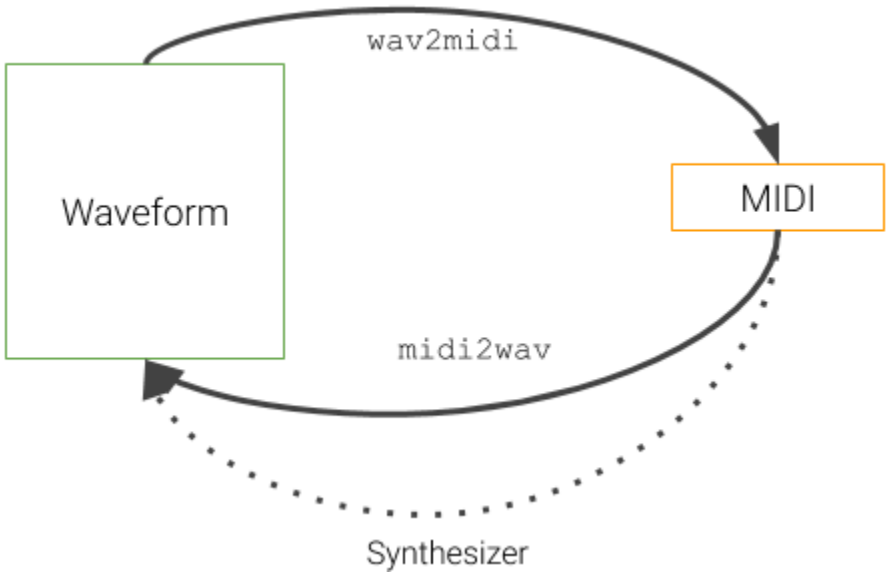


Figure 1: Diagram of our translation model. Our model has similar architecture to CycleGAN. Note that while we do train each part of the cycle, our goal is to have an accurate `wav2midi` model.

`wav2midi` models, then the `wav2wav` and `midi2midi` models will just become auto-encoders and the intermediate values will not be meaningful.

5 Changes Since Last Presentation

Our previous spectrograms generated from MIDI files only covered the top left corner of the spectrograms. This was not going to generalize well to the wav spectrograms since the wav spectrograms fill out the whole spectrogram. We remade our MIDI dataset so that the spectrograms are more full. We did this by generating longer MIDI files and cut off some of the duration in the spectrograms.

We experimented with using ReLU6 instead of Sigmoid as the activation function for predicting the Δ time MIDI feature. These results were not successful, though, as the model would only output values of 0 for the Δ time feature. The Δ time values are incredibly complex and are often 0 or a small value such as 0.09 seconds, and we believe that our model that used ReLU6 got caught in a local minimum that only outputted 0 seconds since this is close to the average Δ time value.

In our past reports, we were normalizing the spectrograms to 0-1. We tried transforming the spectrograms to a normal distribution and the model trained 2-10 times faster and the results looked and sounded much better. Instead of using Sigmoid outputs to generate the spectrograms, we now use a linear activation function. The normalized spectrograms provide a much larger range and a unit standard deviation, and so this allows our model to learn much better. Additionally, this normalization helps bring the distribution of waveform spectrograms and MIDI spectrograms closer together.

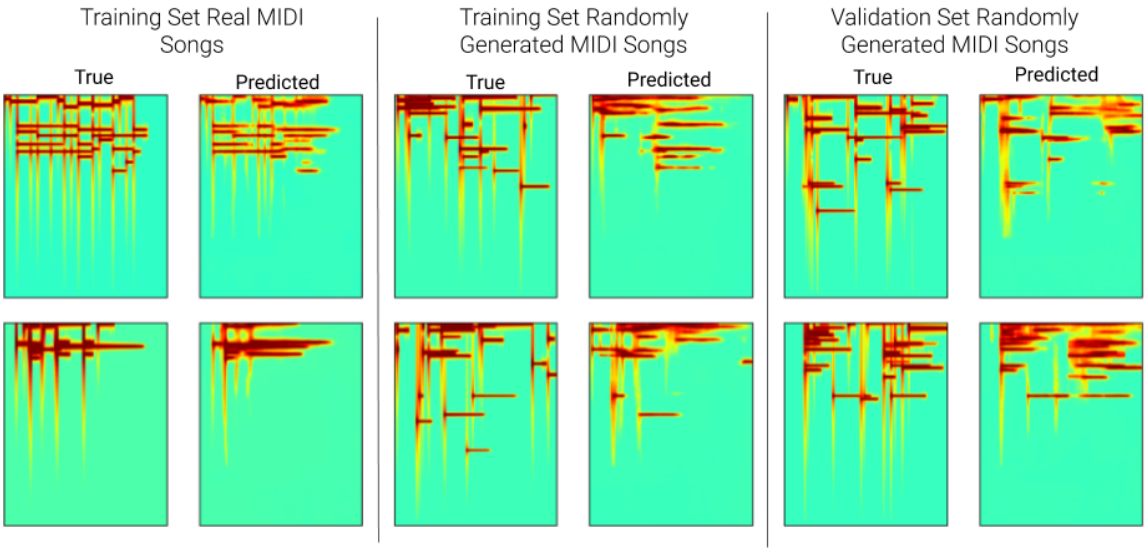


Figure 2: Results from training the `midi2wav` model. Results from this model are generally successful, and are fixed for the remainder of the paper such that the training is done on the remaining models.

6 Results

The results of training the `midi2wav` model are shown in Figure 2. The model took 48 hours on a Titan GPU to train. The model would not over-fit, instead the validation loss stayed constant after a while. The training MIDI to spectrograms are able to be reproduced with high fidelity and the model generalizes to a held out validation set quite well.

Once the validation loss for the `midi2wav` model stopped increasing, we connected the `midi2wav` and `wav2midi` models, made the `midi2wav` model non-trainable, and began training the full models (`midi2midi` and `wav2wav`). The `midi2midi` model was able to recover a midi file quite well. As seen in Figure 3, the notes are accurate. One thing that can be lost in the `midi2midi` model is small details like short, quiet notes and percussion. We know that when losing the phase information by taking only the magnitude of the spectrogram, we lose certain details. In our experiments with converting the spectrograms back to waveform audio, we lose details such as percussion and short notes, and these are very similar to the details that are lost in the `midi2midi` model.

The `wav2wav` results were not as promising as the `midi2midi` results. The spectrograms were not able to be recovered well as seen in Figure 4. This is likely a testament to how difficult is is for the loss to travel through the pretrained `midi2wav` model to optimize the `wav2midi` parameters. The loss in the `midi2midi` model immediately can affect the trainable parameters in the `wav2midi` model, however, the loss needs to travel back through the `midi2wav` model when training the `wav2wav` model (see Figure 1). To test if the model was able to recover the spectrogram at all, we trained the model using only a small subset of wav spectrograms. The model was able to over-fit and the `wav2wav` model outputted spectrograms that looked very similar to the inputs.

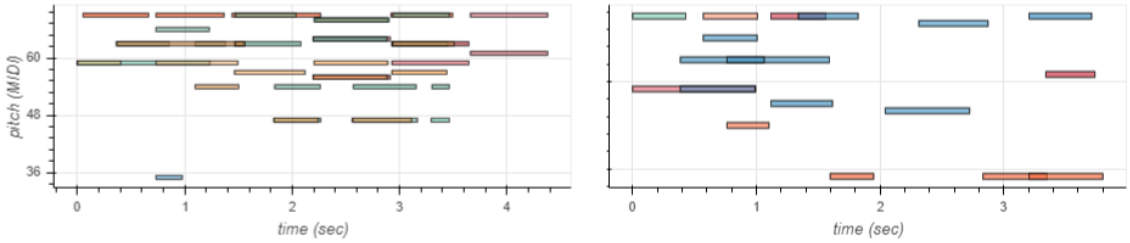


Figure 3: `midi2midi` results. The true MIDI sample is on the left and the right is the `midi2midi` output.

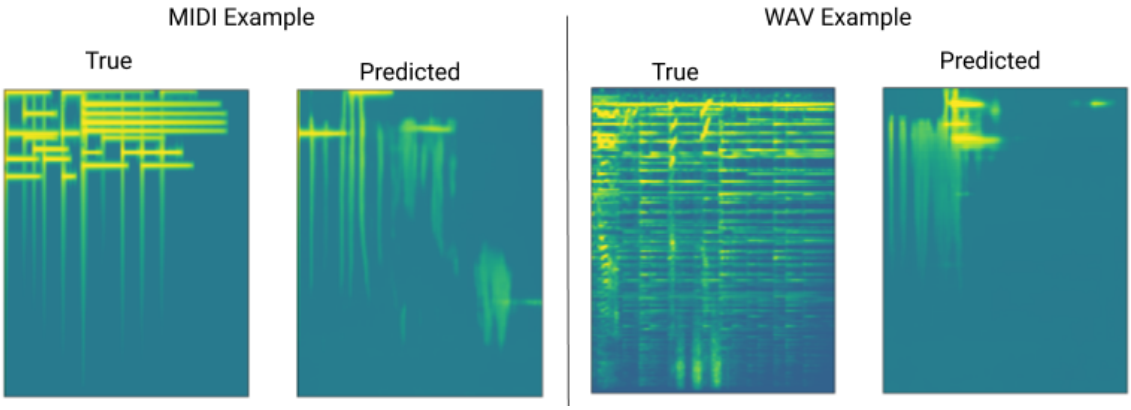


Figure 4: Results after training the `wav2wav` model. The model tries to convert the input spectrogram (True) into MIDI then back in to a spectrogram (Predicted). The left shows an example using a MIDI spectrogram produced using FluidSynth, and the right shows a spectrogram from the FMA dataset.

6.1 Qualitative Results

We tested our `wav2midi` model using training spectrograms from the FMA dataset. In a way, this model cannot over-fit since the FMA dataset does not have true MIDI pairs, and so if our model produces MIDI files that sound like it, it would not be a failure if the model only works on these training samples and not held out validation samples. The MIDI files often share some similarities with the input waveform file. One thing that is immediately apparent is how diverse and strange the FMA dataset is. The songs come from many different genres. Many of the songs are not songs that you would ever care to have a MIDI transcription of since they are filled with noise and strange sounds rather than discrete notes, melodies, or other common aspects of music.

When using a file that has clear pitches and rhythm to test the `wav2midi` model, we do see some promising results. The model is able to learn pitches from the input file, but often fails to recreate the rhythm. The rhythm can be so bad and the notes can be so good sometimes that it can sound like the MIDI file is just a different part of the same song. Because the spectrograms are normalized, the model fails to accurately reproduce the dynamics and volume of a song. If the input song is quite, the output will still be loud because of this normalization.

Playing the results of the `wav2midi` model to our class, many students were in agreement that the model was able to learn some of the pitches, however, everyone was in agreement that the model was failing to translate the song with high fidelity.

6.2 Quantitative Results

Using the MAESTRO dataset, we tested our model for note prediction. Hawthorne et al.[2] outlines a method of evaluating note prediction. They look for a window in time around a note in a song and see if the model was able to predict the correct note in this time window. They used a time window of 50 milliseconds, and their results are outlined in Figure 2. All the models except for ours in this table are supervised training neural networks that learned using wav-MIDI paired training examples. Our model learned without using the paired data. Our model was significantly worse at predicting notes in this small window of time.

	Precision	Recall
Sigtia et al., 2016 [8]	44.97	49.55
Kelz et al., 2016 [5]	44.27	61.29
Melodyne[3]	62.08	48.53
Hawthorne et. al 2017 [2]	84.24	80.67
`wav2midi` (ours)	7.38	11.22

Table 2: Note prediction results on the MAESTRO data set and a 50ms window. Results other than ours were reported in [2]

To test if our model's poor results in Table 2 were caused by our models inability to predict rhythm well, we increased the window size incrementally. As seen in Table 3, increasing the window size drastically improves the precision and recall of our model. The infinite ("inf") window size means that we're looking to see how many of the notes in the true MIDI file were found in the translated MIDI file. With the large window size our results are comparable to some of the results in Table 2. This leads us to conclude that our model is mediocre at predicting notes and very bad at predicting when to play the notes.

Window (sec.)	Precision	Recall
0.05	7.38	11.22
0.1	12.83	18.18
0.5	33.04	39.58
1	44.82	48.75
inf	56.02	62.38

Table 3: Changing the window size and evaluating our `wav2midi`, note prediction model on the MAESTRO dataset.

7 Impact (Technical or Social Contribution)

Our technical contributions lie in developing a new method for translating raw waveform audio into midi files. More specifically, we were able to reconstruct midi-generated wavefiles with good accuracy. We were the first in using a pix2pix- and cycle-GAN-inspired model in music translation; whereas currently, it is mainly used for image translation. We also created a differentiable synthesizer that works well and is vastly different than the ones that are currently available in music software.

Our social contributions lie in that our end-to-end model, once fully operational, can be used by aspirational artists to learn/transcribe their favorite songs, allowing them to play them; as well as professional artists who would like to transcribe their full-band music on-the-go. The model also has its own character when relating music; adding interesting variations to the input audio that could inspire an artist.

8 Limitations and Future Directions

The difference between spectrograms that come from wavefiles (FMA dataset) and MIDI files are extreme, as can be seen in Figure 4. These spectrograms certainly come from different distributions and likely are contributing to our model not performing well. We have two ideas of how to get over this difference in spectrograms. One way is to add whether the spectrogram is a MIDI spectrogram or a wav spectrogram to the `wav2midi` model. This could just be a Boolean value that helps the model overcome the difference in distribution.

The other way to combat the difference between wav and MIDI spectrograms is to create a translation model between them. We could use CycleGAN to translate the two classes of spectrogram and add it to our pipeline. Now, our cycle would be MIDI converts to MIDI spectrogram converts to Wav spectrogram then converts back to MIDI.

None of our translation models in this iteration of this project over-fit. At best, the validation error would simply stop increasing and flat line. This is an indicator that our models need to be more complex. Increasing the number of parameters in our models is a technical challenge since our models are already extremely large.

Many of the songs in the FMA dataset are songs that nobody would ever wish to convert to MIDI. For instance, many songs are extremely distorted or are just ambient sound rather than traditional instruments and voice. If we were to filter these noisy, useless songs out of the dataset, our model would have an easier time learning.

The `wav2midi` model struggles to capture the rhythm of the song. This could be because of the

way that we represent time in our MIDI features. For every note in the MIDI song, our feature representation has a Δ time value which is how long to wait before playing the note from the previous note. This could be extremely challenging for our model to predict. Instead, we could try discretizing time, for example saying that for one second of audio, there will be at max 100 notes evenly distributed in time in the translated MIDI file. Many of the 100 notes will be "turned off", and this builds in time to the model implicitly rather than explicitly.

9 Team Bio

Husni and Peter have worked together on multiple music related projects in machine learning courses at Carnegie Mellon. Husni is a PhD student in the McWilliams Center for Cosmology, and Peter is a technical staff member of the Human-Computer Interaction Institute. Both are interested in bringing state of the art machine learning techniques to an under-researched domain of music. Husni and Peter will each equally contribute to programming, writing, and presenting this work.

References

[1] Chris Donahue, Julian J. McAuley, and Miller S. Puckette. Synthesizing audio with generative adversarial networks. *CoRR*, abs/1802.04208, 2018.

[2] Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, and Douglas Eck. Onsets and frames: Dual-objective piano transcription. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, 2018*, 2018.

[3] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[5] Rainer Kelz, Matthias Dorfer, Filip Korzeniowski, Sebastian Böck, Andreas Arzt, and Gerhard Widmer. On the potential of simple framewise approaches to piano transcription. *CoRR*, abs/1612.05153, 2016.

[6] P. Vandergheynst X. Bresson M. Defferrard, K. Benzi. Fma: A dataset for music analysis, 2016.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[8] Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 24(5):927–939, May 2016.

[9] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv e-prints*, page arXiv:1703.10593, Mar 2017.