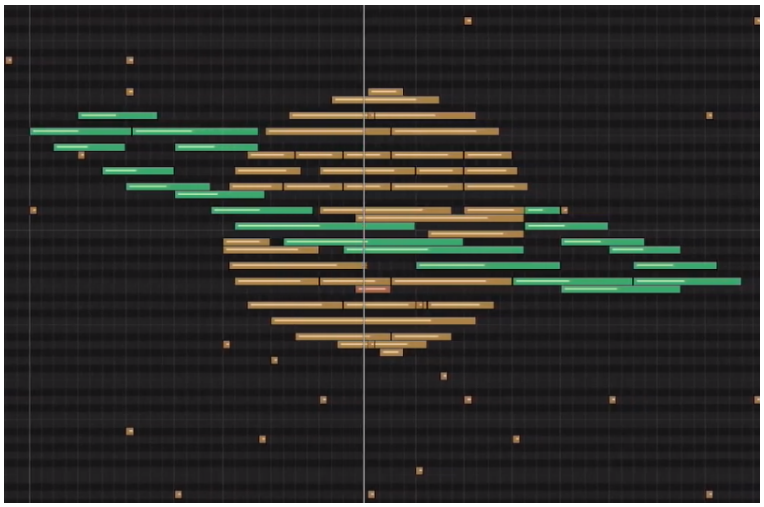
Creative AI
CMU 2019 Fall
Final Project

wav2midi: An Unsupervised Music Format Translation Model



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

Abstract

Write abstract of the paper here.

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 is to translate this into a representation that more resembles how humans process music using only unpaired MIDI and .wav files.

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 [8], we can feed waveform into our model, translate them into MIDI, then translate them back in to waveform. Our model will be able to learn using the loss between the original waveform audio and the predicted waveform. Similarly we can input MIDI data and get MIDI data out.

Even though our model will be trying to perfectly recreate the input song with the output vector representation, it still may be considered creative. The vector representation will not have all the instruments available that the original song had so it will need 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 now 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 is to utilize the abundance of waveform and MIDI files to accomplish this translation task.

We will be using 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 our data set being 917 gigabytes. This will provide a huge technical challenge to be able to train our model on all of these songs. We will attempt to tackle this using Amazon Web Services resources.

Defining the vector representation of music is a difficult challenge. The representation needs to be complex enough to capture the nuances of music, but needs to be simple enough so that the model is able to learn. This vector representation can be easily turned back into waveform audio using a synthesizer, however, we will need 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 will be a great challenge. Fortunately, 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. An example of these can be found here: `https://www.bearaudiotool.com/mp3-to-midi`. 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 Wave2Midi2Wave [4] model. What makes Wave2Midi2Wave 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. Wave2Midi2Wave 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 [8] 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 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` [5] 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 Complete Description of Technical Approach

The following is a list of technical challenges our team has faced, and how we have solved them:

- We had to make the choice of using either spectrograms or wavefile data. We opted to using spectrograms due to the fact that their dimensionality is much less and they are easy to handle given the progress that has been made in machine learning on images. We were able to transform spectrograms back into audio using Librosa's implementation of the Griffin-Lim algorithm.
- Spectrograms coming from synthesized midi files are vastly different than ones coming from raw audio files. The latter ones are much noisier, extend to higher frequencies, while spectrograms coming from synthesized midi files are much cleaner and appear more discrete. This made it very difficult for a model trained on spectrograms synthesized from midi files to generalize to raw wave files. We have attempted to combat this by (a) restricting our data to less noisy samples and (b) generate random midi files that appear more similar to wave files than actual midi songs.
- The training was taking a very long amount of time, we used standard scaler normalization of the spectrograms which cut the training time by a factor of 4.
- Evaluating music-type models on general datasets is usually not a smart choice, because of the huge variance in genres of music as well as the lack of quality assurance on large numbers of paired midi and wavefiles. We used a piano-only dataset for this reason called MAESTRO, which has been proven to be consistent in its quality.

...

5 Experimental Setup and Evaluation Methods

A representation of our model can be seen in Figure 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. 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. This is outlined in the table below.

Our translation models wav2midi and midi2wav were both modelled after the pix2pix [5] architecture. Each has 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

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

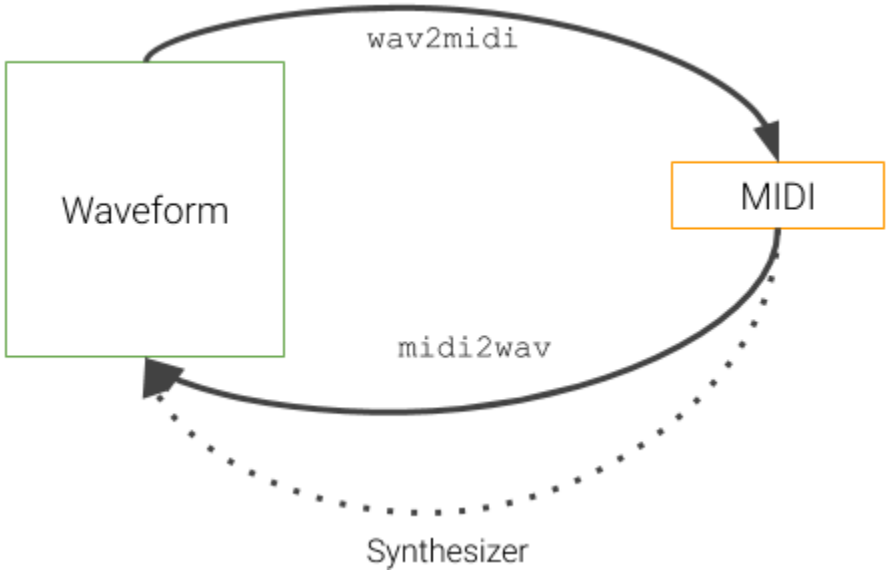


Figure 1: Diagram of our translation model

distribution. The wav2midi model outputs 7 values corresponding to the MIDI feature representation in the table above.

We start by training the midi2wav model using MIDI files and their corresponding spectrograms produced by the FluidSynth synthesizer. We use 100,000 samples created from real MIDI files and generate 400,000 random MIDI samples by selecting values at random.

When the midi2wav model is trained, we begin 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.

6 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 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.

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.

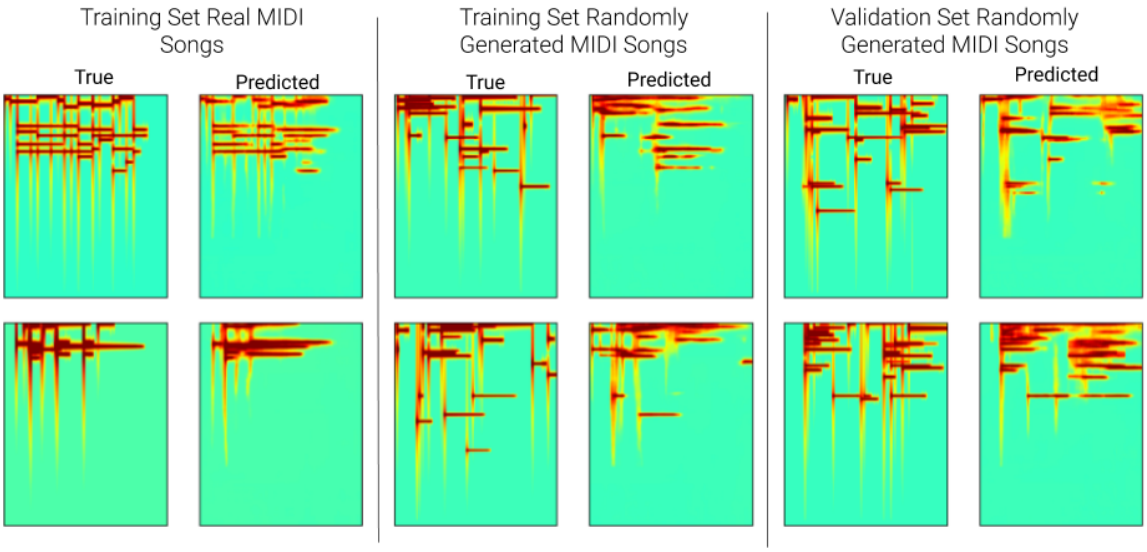


Figure 2: Results from training the midi2wav model

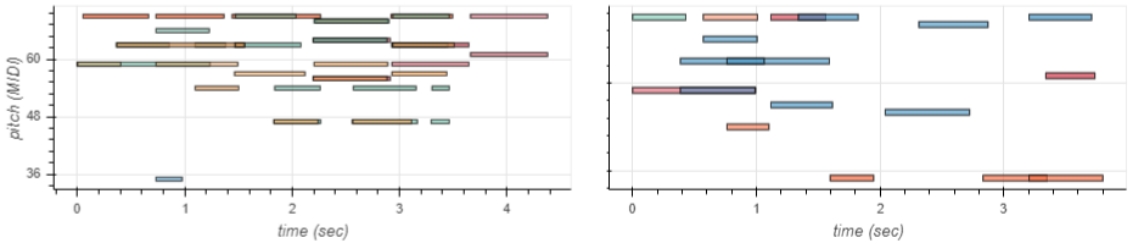


Figure 3: midi2midi results. The true MIDI sample is on the left and the right is the midi2midi output.

7 Final 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 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.

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.

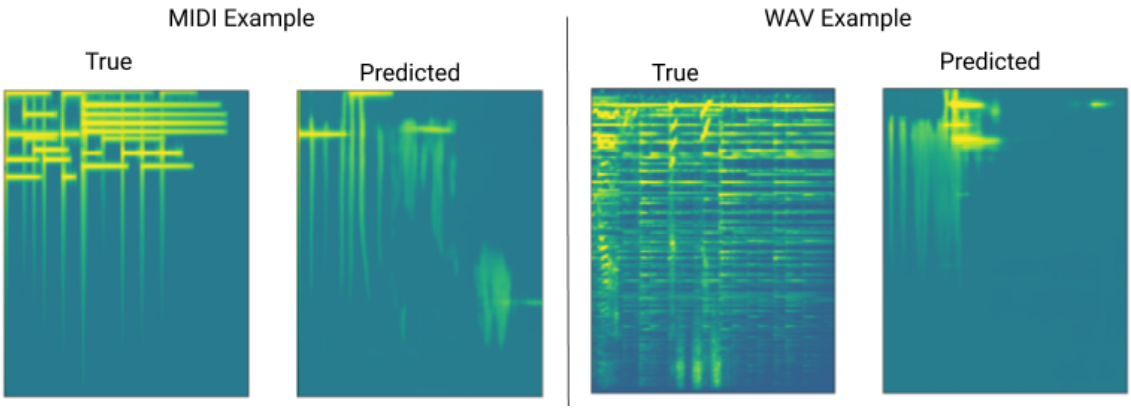


Figure 4: wav2wav results

	Precision	Recall
Sigtia et al., 2016	44.97	49.55
Kelz et al., 2016	44.27	61.29
Melodyne*	62.08	48.53
Onsets and Frames (Hawthorne et. al 2017)	84.24	80.67
Ours	5.35	9.08

Figure 5: Note prediction results on the MAESTRO data set

8 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 5. 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.

To test if our model's poor results in Figure 5 were caused by our models inability to predict rhythm well, we increased the window size incrementally. As seen in Figure 6, increasing the window size drastically improves the precision and recall of our model. This leads us to conclude that our model is mediocre at predicting notes and very bad at predicting when to play the notes.

Window	Precision	Recall
0.05s	5.35	9.08
0.1s	9.33	14.55
0.5s	26.70	32.06
1s	42.02	37.13
inf	50.75	58.77

Figure 6: Changing the window size and evaluating our note prediction model

9 Impact (Technical or Social Contribution)

Our technical contributions lie in developing a new method for transforming raw waveform audio into midi files. More specifically, we were able to reconstruct midi-generated wavefiles to 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.

10 Limitations and Future Directions

Our model finds it difficult to generalize on noisier wavefile data. While we restrict our work to clearer audio files (e.g., piano recordings), noisy full-band pop/rock music transcription is out of scope and can be considered as future work.

11 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] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse H. Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. *CoRR*, abs/1810.12247, 2018.

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