CS230

# PianoGAN - Generating Piano Music from Magnitude Matrices (Generative Modeling)

**Gavin Liu** jliu66@stanford.edu
**Laura Fee Schneider** lfee590@stanford.edu
**Sylvia Yuan** luyuan@stanford.edu

## Abstract

The goal of this project is to train a Generative Adversarial Network (GAN) to generate magnitude arrays, that can be inverted to 20 seconds of harmonic and consistent classical piano music. We experimented the extension of DCGAN [1], SpecGAN [2], and styleGAN [3] and discovered that adaptation of SpecGAN produces the best output audio clip.

## 1 Introduction & Motivation

Generating new music can ultimately be very useful for creators in particular to obtain unlicensed music specifically modified for their purpose.
The most established models in music generation, like MuseNet, apply language models to symbolic audio representations like MIDI (musical inference digital inference) files [4]. However, this approach is limited to electronic musical instruments because of its usage of MIDI files. We want to use a more general approach and employ GANs to generate cohesive piano music clips from magnitude matrices, generated from raw audio files.

## 2 Related Works

There exist autoregressive models like SampleRNN, WaveNet and MelNet that use raw audio files or spectrograms and are very successful on text-to-speech generation tasks [5] [6] [7]. MuseNet is an neural network that can generate music of different instruments and styles [4]. MuseNet uses the same method used in unsupervised language modeling to predict tokens in MIDI audio file, in order to generate realistic music audio. The goal of our project is similar to MuseNet. However, our proposed approach is different, as we plan use spectrograms and magnitude matrices from the short-time Fourier transform (STFT) instead of MIDI files, and do not use language processing methods.
Recent approaches utilizing GANs ( [8], [2], [9]) for generating music with consistent local and global features seem the most promising to us. In [2] SpecGAN is introduced which employs the DCGAN algorithm to produce 1 second long instrument sounds. [8] presents GANSynth, a model that produces 4 seconds long notes from various instruments, where the generator receives an additional input conditioning on pitch labels. The unconditional audio generation GAN from [9] can take a variable-length sequence of noise vectors as input and outputs a mel-spectrogram of variable length. Our approach is an unconditional GAN on a fixed length noise vector that tries to produce not only single notes but a harmonic melody. This is a very challenging task especially for 10 or 20 second long audio clips. To our knowledge, there is no publication about producing such long piano sequences from one graphical representation of raw audio.

## 3 Dataset and Preprocessing

For this project, we work with the MAESTRO dataset [10], which contains over 200 hours of piano music. The dataset includes raw audio wav-files and MIDI files of the piano music. We preprocessed the dataset for training in the following ways: we first divided each wav file into 20 (or 10) second clips, and used the STFT to convert the audio file into its time-frequency representation. In the next step, we apply the absolute value to the complex matrix obtained by the STFT to retrieve the magnitude matrix. We decided to use the magnitude matrix without considering the phase values, as our experiments showed that the quality of audio reconstruction was still good when using the librosa implementation of the Griffin-Lim algorithm. As suggested in [8], we apply an elementwise logarithm to the amplitudes. Further, we normalize to zero mean and unit standard deviation. To obtain values in [-1,1] to match the output of a tanh activation function, we follow the approach in [2] and clip the amplitude spectra to 3 standard deviations and rescale to [-1,1]. With these steps one can obtain differently large images depending on the sampling rate and the input parameters of the STFT algorithm defining the resolution in time and frequency dimension. To train our model efficiently, we experimented with various parameter settings to see how we can obtain high quality audio reconstruction from smaller images.
With the default parameters for librosa STFT method (sampling rate = 22050, n_fft=2048), we acquire arrays of size $(1025, 862)$. To downsample the arrays for computing efficiency when training, we first tested the quality of reconstructed audio clips from different downsampled STFT, and decided that the sizes $(512, 512)$ and $(256, 256)$ are sufficient for the Griffin-Lim algorithm to reconstruct reasonable audio clips of length 20 and 10 seconds respectively. Later, we decided to use a method to further reduce loss of quality, by using the parameters sampling rate = 10000, n_fft=1024, and hop length=256 for STFT, we acquire arrays of size (513, 782), and downsample the arrays to size (512, 512) for training DCGAN with 20 second audio. For the 10 second clips, which we used due to local computational restrictions, we choose the sampling rate to be 5500, n_fft=512, hop_length=128 and downsample to a size (256, 256) array. Figure 1 presents four examples from our training dataset for 10 seconds audio clips.

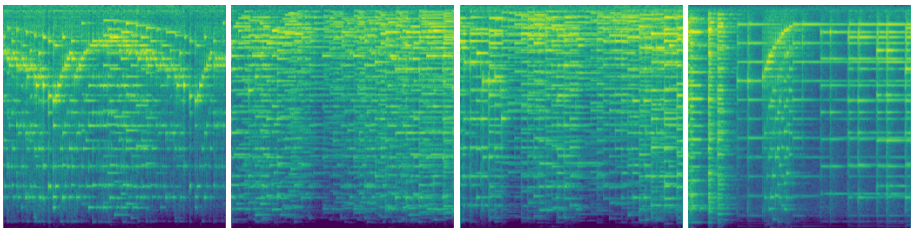

Figure 1: These spectrograms are examples from our training data of 10 second piano sounds.

## 4 Methods

Our goal is to find a GAN architecture that can learn to produce such magnitude matrices, which can be inverted into piano music sequences. As our baseline model we choose a deep convolutional generative adversarial network (DCGAN). We adapted the general structure of the keras-GAN implementation of DCGAN [1], modified the architecture, and started first trainings.

We used the binary cross-entropy loss in python tensorflow.keras library, the loss functions are [11]:

$$J^{(D)} = -\frac{1}{m_{real}} \sum_{i=1}^{m_{real}} \cdot \log(D(x^{(i)})) - \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} (1 - y_{gen}^{(i)}) \cdot \log(1 - D(G(z^{(i)}))).$$

$$J^{(G)} = \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(1 - D(G(z^{(i)}))).$$

The generator and discriminator architecture were adapted and modified from the DCGAN architecture in keras-GAN [1], as illustrated in Figure 2. The generator receives a 100 dimensional input vector and upsamples to produce a $512 \times 512$ dimensional matrix with values in $[-1, 1]$. The

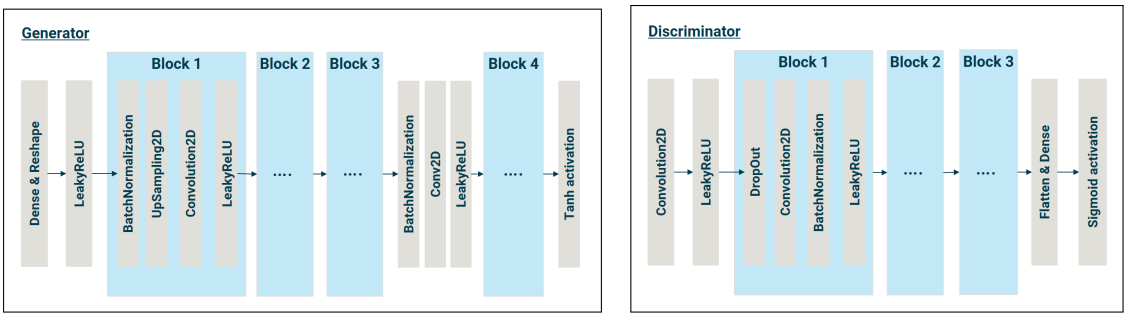


Figure 2: Illustration of the DCGAN baseline architecture

discriminator takes the matrix as an input and returns a single value calculated by a sigmoid activation function.

Another possibility is to use the architecture of SpecGAN as suggested in [2]. The structure of generator and discriminator is similar to the baseline model. The main difference is the use of Wasserstein loss instead of binary cross-entropy loss. The Wasserstein loss also called Earth Mover distance can be interpreted as the cost of moving mass in order to transform one distribution into another distribution. A Wasserstein GAN (WGAN) as defined in [12] uses the following value function obtained by the Kantorovich-Rubinstein duality,

$$\min_G \max_D \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r}[D(x)] \; - \mathop{\mathbb{E}}_{\hat{x}\sim\mathbb{P}_G}[D(\hat{x})], \tag{1}$$

where $\mathbb{P}_r$ represents the data distribution and $\mathbb{P}_G$ is the distribution defined by the generator. For the WGAN to work we remove all batch normalization layers from generator and discriminator and do not use any activation function in the discriminator's output layer. As an additional feature a gradient penalty term is added to ensure that the solution is closer to a 1-Lipschitz function, i.e. a differentiable function with gradients with norm at most 1 everywhere. This penalty term is justified by a theoretical statement about the optimal solution of the optimization problem in (1), see Proposition 1 in [12], and utilized in the SpecGAN approach as well.

The third option we consider is applying the StyleGAN architecture. For StyleGAN2 with adaptive discriminator augmentation (ADA), we used the library provided by NVIDIA Labs [13]. An adaptive discriminator augmentation mechanism can significantly stabilize training and prevent discriminator from overfitting or divergence during training, while ensuring the augmentations do not influence the distribution of generated images. Further, it uses StyleGAN architecture that introduces control over the style of generated images at different levels of detail. A rough overview of the architecture of this model can be found in Figure 7 in the appendix. We used two data processing and training approaches for this model in order to compare the results.

## 5 Experiments/Results/Discussion

To achieve better results, we conducted three parallel experiments. First experiment is to modify the architecture of DCGAN [1] and adapt the model to work for $(512, 512)$ magnitude matrices. The second experiment is to adapt the SpecGAN [2] model architecture to our purposes. The third experiment is to convert spectrograms to RGB images and train a styleGAN [3] model to generate images that can be converted back into spectrograms and audio clips.

### 5.1 Extension of DCGAN

We modified the model architecture provided in [1] for training with STFT magnitude matrices with size $(512, 512)$. We chose to use Adam optimizer with a learning rate of 0.0001, $\beta_1 = 0.5$, $\beta_2 = 0.9$. The intuition for the values of hyperparameters came from specGAN [2]. We decided to use a batch size of 32 to ensure reasonable updates to the generator parameters.The generator and discriminator architecture was adapted and modified from the DCGAN architecture in keras-GAN[8]. There are various ways to improve the power of this generative model and to adjust the architecture to be

more likely to produce the wanted images. One way is to use various tips and tricks to fine tune and enhance our baseline model. During the training, we encountered several issues with this baseline approach. We discovered there was a problem with exploding gradient, so we added gradient clipping to the Adam optimizer with clipnorm =1. However, the generated audio was still not close to piano music, and the discriminator loss approached 0 while its accuracy approached 1.
To weaken the discriminator, we attempted various methods. First is to flip part of real and fake labels to add noise to the discriminator input [14]. Another approach is to use label smoothing, replacing hard labels of 0 or 1 with soft labels from in the range of 0-0.1 and 0.9-1.0 [15]. We also attempted to concatenate generated noise input to the real or fake examples, and generated random labels to the real or fake labels, and feed the concatenated inputs to the discriminator.
We constantly examined outputs from different adjustments to our models. Figure 3 shows outputs from four different failed models, converted to spectrograms for visualization purposes. Though

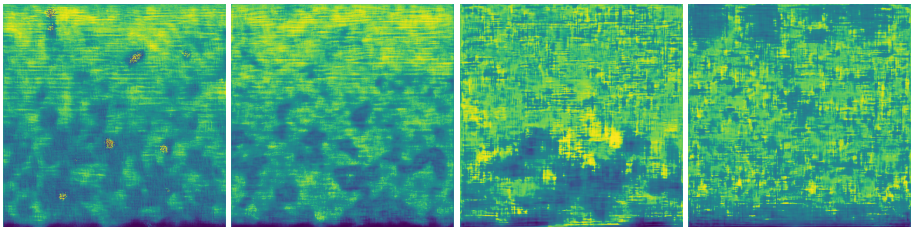

Figure 3: Generated outputs from failed DCGAN models

we were unable to fix the model collapse issue, we believe that likely with more time to fine-tune the hyperparameters and model architecture, the model will be able to learn to generate reasonable matrices.

### 5.2 SpecGAN Version

For our experiments with the SpecGAN architecture, we used the code provided by [16] and modified it for our purpose. We adjusted the code to work with the new tensorflow API and modified it to require less memory space. Further, SpecGAN used an additional label defining the category of each spectrogram. We removed this part from the architecture and changed the generator output and discriminator input size to $256 \times 256$. The hyper-parameters are the same as in the original SpecGAN implementation, e.g. Adam optimizer with a learning rate of $0.0001$, $\beta_1 = 0.5$, $\beta_2 = 0.9$.
The training of the SpecGAN version turned out to be computationally more expensive than the baseline model and we decided to train it on 10 second audio clips represented by $256 \times 256$ matrices. In [2], it is described that training converged after 1750 epochs. Although our dataset is larger and less epochs might be sufficient, the 340 epochs that we were able to train could be too few. It seems the GAN is still learning and improving after these epochs. The resulting audio clips sound more and more like produced by a piano, but do not contain a harmonic melody yet.

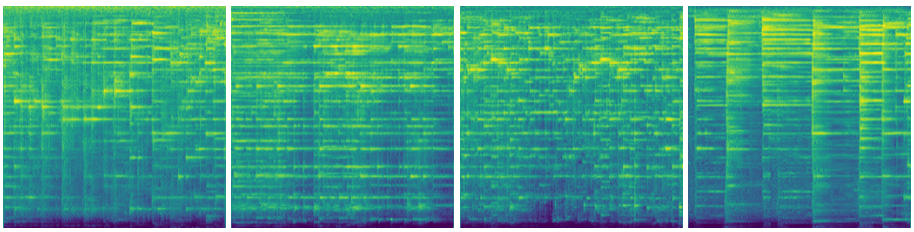

Figure 4: These spectrograms are produced by the generator of our SpecGAN approach after 340 epochs of training.

### 5.3 StyleGAN2 Version

We used two data processing and training approaches in order to compare the results.

### 5.3.1 Approach 1 - RGB Images

For this approach, we normalized STFT matrices generated from 10 second audio clips, resized them into $256 \times 256$, then converted them into RGB images for training. The model achieved the lowest fid50k score of 53.49 (lower the better) after training for 800k images, but modal collapse occurred after. The generator was not able to generate a variety of images regardless of input.

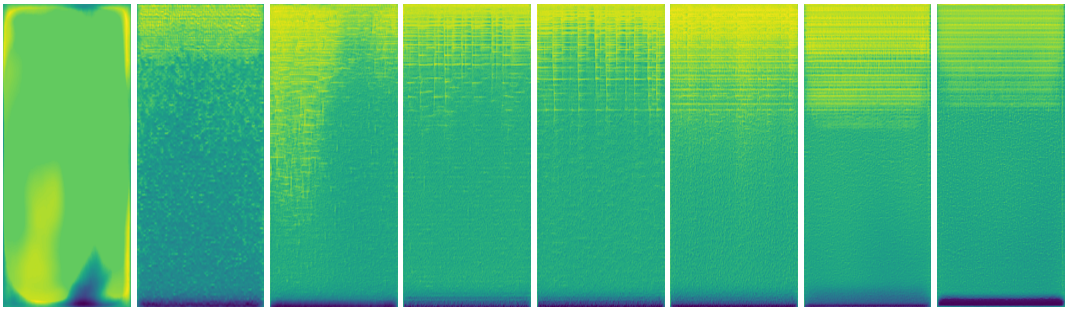

Figure 5: Generated RGB images over 1.4M images trained with the original dimensions (200K image difference from left to right)

### 5.3.2 Approach 2 - Gray Scale Images

For this approach, instead of normalizing the STFT matrices, we scaled up the brightness of important pixels, then resized them into $512 \times 512$ gray scale images for training. Due to time constraint we were only able to train the model for 400k images. We scaled the brightness of some pixels back down, and used noisereduce library [17] to reduce white noise in the audio clips.

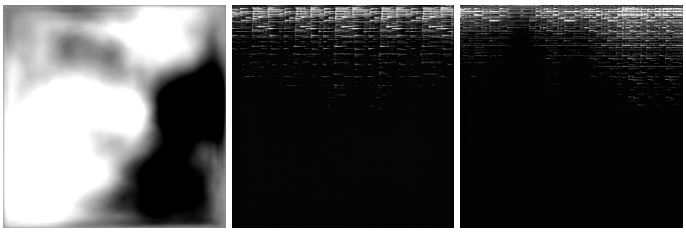

Figure 6: Generated Gray Scale Images (0k, 200k, 400k images trained)

## 6 Conclusion/Future Work

Judging from the spectrogram visualization and quality of the audio clips, which can be found in our github repository, we conclude that the best results came form the adaptation of the SpecGAN model. More training epochs may still be required to achieve better results. However, it seems that the model already learned how to reproduce the local features of piano sounds. These sounds are just not combined to a harmonic melody yet. It is likely that a more evolved architecture is necessary to learn these global compositions of the piano sounds.
In contrast to the methods in [8] and [2], we decided not to transform our magnitude matrices to the Mel-scale, because the inversion of our images led to less loss in audio quality. It is possible that training with mel-spectrogram could be more effective. However, our results suggest that this should not be a major factor for audio quality.
Another aspect that could help produce better audio, is categorize the piano music clips. As stated in [18], adding labels usually improves the results of a GAN. One could use different categories for different types of piano melodies to help the generator learn global harmonic features.

## 7 Source Code

https://github.com/GavinLiu-AI/PianoGAN.git

## 8 Contributions

Gavin Liu:

1. Researched on digital audio signal processing and Fourier Transform
2. Processed data and generated customized data set from Maestro v3 data set
3. Trained networks based on StyleGAN2-ADA-pytorch library using RGB and gray scale images
4. Worked on image to audio reconstruction

Laura Fee Schneider:

1. Researched on audio representations and audio reconstruction.
2. Contributed to code in preprocessing.py, util.py and pianoGAN_SpecGAN_version.py
3. Trained the SpecGAN version of our model locally on 10 second audio clips.

Sylvia Yuan:

1. Tested STFT with different parameters and different downsampling parameters/methods to check audio reconstruction quality
2. Contributed to code in preprocessing.py, util.py and pianoGA_dcGAN_version.py (modified from keras-GAN [1])
3. Trained the modified DCGAN model with AWS EC2 instance

## A Appendix

Figure 7 presents a graphical illustration of general idea an architecture of StyleGAN2-ADA, as it can be found in [3].

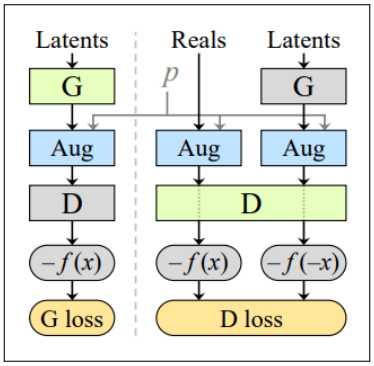


Figure 7: Illustration of the StyleGAN2-ADA architecture from [3]

## References


[1] E. Linder-Norén, "Keras-gan," 2017. [Online]. Available: https://github.com/eriklindernoren/Keras-GAN

[2] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," *arXiv preprint arXiv:1802.04208*, 2018.

[3] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401–4410.

[4] A. Topirceanu, G. Barina, and M. Udrescu, "Musenet: Collaboration in the music artists industry," in *2014 European Network Intelligence Conference*. IEEE, 2014, pp. 89–94.

[5] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio, "Samplernn: An unconditional end-to-end neural audio generation model," *arXiv preprint arXiv:1612.07837*, 2016.

[6] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[7] S. Vasquez and M. Lewis, "Melnet: A generative model for audio in the frequency domain," *arXiv preprint arXiv:1906.01083*, 2019.

[8] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "Gansynth: Adversarial neural audio synthesis," *arXiv preprint arXiv:1902.08710*, 2019.

[9] J.-Y. Liu, Y.-H. Chen, Y.-C. Yeh, and Y.-H. Yang, "Unconditional audio generation with generative adversarial networks and cycle regularization," *arXiv preprint arXiv:2005.08526*, 2020.

[10] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the maestro dataset," *arXiv preprint arXiv:1810.12247*, 2018.

[11] K. Katanforoosh, "Cs230: Lecture 4 attacking networks with adversarial examples[pdf document]," 2017. [Online]. Available: https://cs230.stanford.edu/fall2020/lecture4.pdf

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[13] NVlabs, "Nvlabs/stylegan2-ada." [Online]. Available: https://github.com/NVlabs/stylegan2-ada

[14] S. Chintala, E. Denton, M. Arjovsky, and M. Mathieu, "How to train a gan? tips and tricks to make gans work," 2016.

[15] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016.

[16] N. Tokui, "Specgan," 2018. [Online]. Available: https://github.com/naotokui/SpecGAN.git

[17] "noisereduce." [Online]. Available: https://pypi.org/project/noisereduce/

[18] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," 12 2016.