CS230

Audio Transfer Learning with Voices

Fabian Boemer *
fboemer@stanford.edu

Eric Gong *
ericgong@stanford.edu

Youkow Homma *
yhomma@stanford.edu

Abstract

The Neural Style Transfer method by Gatys et al. enabled the transfer of style from one image to another by iteratively changing the raw pixel values of a noisy image. While deep learning from raw values has been a mainstay in the image domain, this has not been the case in audio. One key step toward working with raw audio values was the introduction of the WaveNet model which generates state-of-the art, natural sounding audio by autoregressively predicting raw audio values. The NSynth model uses the WaveNet model to encode audio in a latent space that allows for interpolation between different sounds. In this project, we combine the image-based Neural Style Transfer methodology with the NSynth autoencoder architecture to investigate a framework for transferring style from one raw audio file to another.

1 Introduction

In this project, we explore a framework for transferring audio style from one audio file to another audio file. More specifically, our input is two audio files of three seconds, a content audio file x_C and a style audio file x_S. The output is a new, three-second long audio file x_G that has the content of x_C and the style of x_S. We are primarily interested in the case where x_C is audio of human speech and x_S is an instrument. In this way, we aim to construct understandable human speech that sounds less like a human and more like a specific instrument. Additionally, we explore the setting where x_C and x_S consist of one or more pitches of the same instrument. We foresee that this type of audio synthesis could be interesting for applications in electronic music.

2 Related work

Style transfer applied in the audio domain has been attempted under a few guises with many of the most notable methods summarized by Grinstein et al. in [GDOP17]. One successful methodology is to transform one-dimensional audio into two-dimensional spectrograms, and then feed the result into the image-based Neural Style Transfer method by Gatys et al. [GEB15]. This technique was explored by Ulyanov et al. [UL] where audio was transformed into spectrograms through a short-time Fourier transform and then passed through a texture synthesizer with random weights. The authors describe this technique as "style transfer by analogy" because it does not aim to audibly orthogonalize style and content as in the image case.

Mital [Mit17] provided a more comprehensive study of style transfer in the audio domain by exploring different pre-processing steps for the technique by Ulyanov et al. Additionally, Mital applied neural style transfer on both the WaveNet decoder and NSynth encoder, similar to the approach we study in this work, but was unable to achieve meaningful synthesis. We will expand further on key differences between our work and Mital's in Section 4.

*Institute of Computational and Mathematical Engineering, Stanford University

Foote et al. [FYR] also discussed incorporating Neural Style Transfer with the WaveNet architecture, but mentions difficulties with the residual connections and the discretization of audio sample values as reasons for not pursuing the idea further. We also expand on this discussion in Section 4.

3 Dataset and Features

For this project, our style dataset is the NSynth dataset which was used by the Magenta project [Ten] to train the NSynth model weights. The NSynth dataset contains single-note pitches generated by neural networks in the style of various instruments. Each pitch is sampled at 64 kHz, and is between three and four seconds long. We focus on two acoustic vocal pitches and three synthetic flute pitches in the NSynth test set, as a representative basis for learning. Figure 2a and Figure 2d show spectrograms of two audio files in the test set. Our content dataset is a mix of NSynth data, a 1 kHz sine wave sound [Nata, Natb], and a recording of a team member's voice. An 8-bit μ-law algorithm is used to encode all audio before use in our algorithms.

4 Methods

4.1 NSynth Architecture

WaveNet [VDODZ$^+$16] is a generative, deep neural network for raw audio which uses causal, dilated convolutions to expand its receptive field to yield state-of-the-art, natural sounding speech. NSynth [ERR$^+$17] is an autoencoder with stacked non-causal, dilated convolutions as the encoder and the WaveNet model as the decoder which produces encodings in a new latent space for single tones. Interpolation in the embedded space enables blending of audio samples resulting in, for instance, a blend between a bass and flute sound, or bass and organ sound.

For this project, we use the encoder portion of the NSynth model, depicted in Figure 1. This choice was made for a few reasons: First and foremost, the WaveNet decoder is autoregressive, meaning predictions are produced serially with predictions at one time step depending on all previous time steps. This makes working with the decoder very slow. On the other hand, the NSynth encoder is non-causal, making it much faster. Second, the encoder architecture is more similar to the VGG-16 model used in Gatys et al. [GEB15] because the input to the encoder is the raw audio much like how the input to the VGG-16 model is the raw pixels. The input to the decoder is the encoding of the raw audio, which does not lend itself as nicely to the Neural Style Transfer framework. We re-purpose the Tensorflow Magenta code to build the encoder and use the pre-trained weights from Magenta for our project. Further discussions on the code adaptations can be found in Section 7.

4.2 Neural Style Transfer

Our framework is an adaptation of Gatys et al.'s Neural Style Transfer method [GEB15] to the NSynth encoder described in Section 4.1. The initial values for the generated sound, x_G, is a linear combination of the content audio, x_C, and random noise uniformly distributed from $[-10^{-6}, 10^{-6}]$ at each time step.

For the content layer, we considered both the encoding of the NSynth encoder and the final ReLU activation and found the final ReLU to retain the content more clearly. In particular, in our vocal examples, we found the white noise was significantly reduced when silence is present, hence better maintaining the signal-noise ratio.

Unlike Mital [Mit17], we took the style layer values to be the values at the ReLU activation units, rather than the values at the skip connections of the residual blocks, as demonstrated in Figure 1. We believe that restricting the value of the layers to non-negative values more closely mimics the Neural Style Transfer method, and allows for further extensions such as model normalization [GEB15] and instance normalization [UVL17]. As in image-based Neural Style Transfer, earlier layers of the NSynth encoder capture lower-level information, while deeper layers capture higher-level information.

Let $S(\cdot)$ denote the style encoding, consisting of 60 layers which correspond to the two ReLU activations for each of the thirty skip connections. Then, we have that the content cost is

$$\mathcal{L}_C(x_C, x_G) = \frac{1}{\text{number of entries}} \sum_{\text{all entries}} (C(x_C) - C(x_G))^2$$

. We also consider two style costs. The first is based on the Gram matrix due to Gatys et al. [GEB15]:

$$\mathcal{L}_{S_G}(x_S, x_G) = \sum_S w_s \left[\frac{1}{\text{number of entries}} \sum_{\text{all entries}} (\mathcal{G}(S(x_S)) - \mathcal{G}(S(x_C)))^2 \right]$$

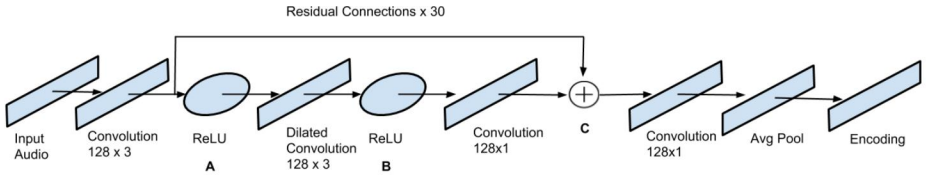


Figure 1: NSynth encoder architecture. We use the activations denoted by A and B as our content and style layers, while previous work by Mital [Mit17] used the output of the residual layer, denoted by C, as the content and style layers.

, where $\mathcal{G}$ is the Gram matrix of the activations. The second style cost is based on the L2 loss:

$$\mathcal{L}_{S_{l_2}}(x_S, x_G) = \sum_S w_s \left[\frac{1}{\text{number of entries}} \sum_{\text{all entries}} (S(x_S)) - S(x_C))^2 \right].$$

The total loss function is then defined as

$$\mathcal{L}(x_C, x_S, x_G) = \mathcal{L}_S(x_S, x_G) + \alpha \mathcal{L}_C(x_C, x_G)$$

where α is a hyper-parameter indicating the relative weight of style loss to content loss. We use gradient descent on x_G using Adam Optimization to minimize $\mathcal{L}$, generating a sequence $x_G^{(0)}, x_G^{(1)}, \ldots$ of audio at each iteration. The x_S and x_C are kept fixed to ensure a faster runtime, and that results do not depend on weights in the NSynth network. For simplicity, we fixed the optimization to run for 100 iterations, at which a learning rate of 0.05 yielded monotonic decrease in the loss.

5 Experiments and Results

We first investigate the feasibility of style transfer described in Section 4 in two settings: pitch and chords. As NSynth was trained only on individual, monotone pitches, we expect the most promising results in these simple settings. We then experiment with speech audio to obtain some preliminary results in the vocal domain.

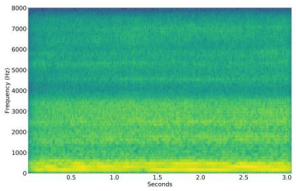
5.1 Pitch-Pitch

We take a pair of vocal pitches, one high and one low, from the NSynth test set. We set one of the pitches to x_S, and the other to x_C, and minimize the style cost $\mathcal{L} = \mathcal{L}_{S_{l2}}$ using $w_s = 1/5$ for each of the first five encoding layers, and starting at $x_G^{(0)} = x_C$. Empirically, this setting tends to produce a continuous blend between x_S and x_C from $x_G^{(0)}$ to $x_G^{(99)}$. We consider two settings: transitioning from the low pitch to the high pitch, and transitioning from the high pitch to the low pitch. Figure 2 shows the spectrograms from the produced audio, at an intermediate iteration 50, and at the 99th iteration. By this final iteration, x_G and x_S match quite closely in pitch, though with a considerably noisier timbre. The spectrograms reflect this noise as less distinct horizontal stripes. This result also suggests that for single pitches, the encoder used in this way can act as a noisy, but faster decoder.

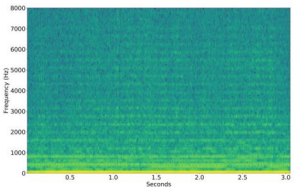
At iteration 50, both methods produce a reasonable blend between the low and high audio signals, sounding like an overlay of the two pitches. The spectrograms show the blended pitches (2b, 2e) contain several overtones, characteristic of the high pitch (2d) and visualized as many horizontal stripes. The blended pitches also show somewhat less intensity above 3.5 kHz, as in the low pitch (2a). By iteration 50, the loss has been reduced by 99.7% of the total reduction by iteration 100 (see 7 for losses).

5.2 Chord-Pitch

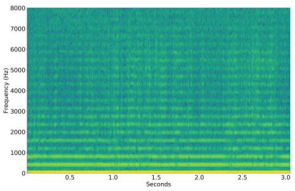
We also considered the setting of learning a chord. Using the same loss $\mathcal{L} = \mathcal{L}_{S_{l_2}}$ and weights w_s as described in Section 5.1, we transition from a single pitch to a chord and then back. In particular, a simple sine wave sound [Nata, Natb] is set as the initial content sound. The sine wave sound is composed of a low pitch and a high pitch, where the lower pitch is played, a brief silence follows, and a higher pitch is played. This is reflected in Figure 3a, where a noticeable silence is seen at the two second mark. This content is trained against a C Major chord as



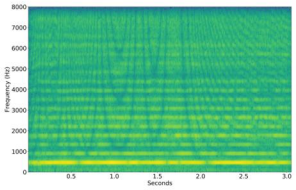
(a) Low Pitch https://youtu.be/PmYJgGLwvNQ



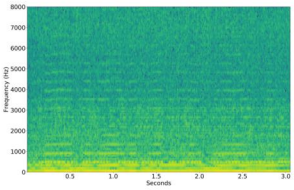
(b) Low to High Iter. 50 https://youtu.be/GmZ6Au5gJpU



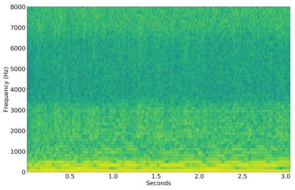
(c) Low to High Iter. 99 https://youtu.be/yivuh1V5y6M



(d) High Pitch https://youtu.be/UlhZtyqm2Pk



(e) High to Low Iter. 50 https://youtu.be/bXM7JZq-GXk

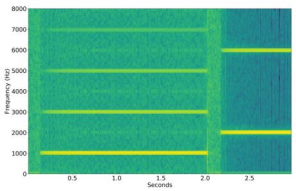


(f) High to Low Iter. 99 https://youtu.be/SBYNES2e4qc

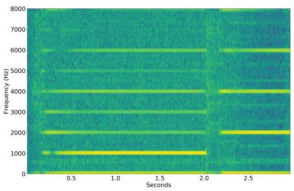
Figure 2: Spectrograms from ascending and descending pitch. Audio is at the posted links.

its style. The C Major chord sound is created by stacking together C, E, and G pitches from the NSynth dataset into a single audio file. Figure 3 shows that spectrograms from the produced audio, at iteration 30, and at the final iteration 100. Contrary to the results in Pitch-Pitch, x_C learns x_S within 30 iterations, but begins to lose the signal after further iterations. That is, the generated audio sound demonstrates the multi-pitch sound of a chord but then degenerates to white noise in further iterations. This is seen in Figure 3b), where spectrogram shows only one pitch.

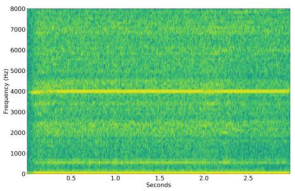
Conversely, the same success was not replicated going from a chord to a single pitch. As seen in Figure 3d - 3f, the chord sound becomes less and less prominent iteration by iteration. By the end of the 100 iterations, the generated audio is white noise.



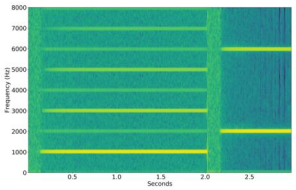
(a) Learning Chord, Iter. 0 https://youtu.be/LL3rntJ9N1Q



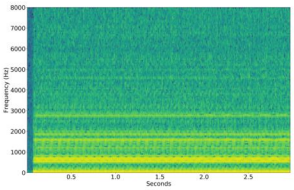
(b) Learning Chord, Iter. 30 https://youtu.be/bpN6lKLRUy0



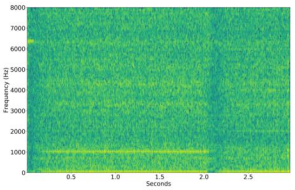
(c) Learning Chord, Iter. 99 https://youtu.be/T_JC7uQKqZI



(d) Learning Pitch, Iter. 0 https://youtu.be/F0QpTkEmCUE

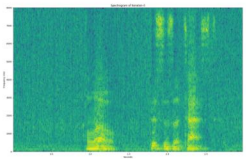


(e) Learning Pitch, Iter. 30 https://youtu.be/jXxJVMcZUT4

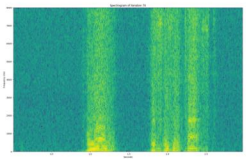


(f) Learning Pitch, Iter. 99 https://youtu.be/c9ycTKgTX1U

Figure 3: Spectrograms from learning a chord and learning a pitch. Audio is at the posted links.



(a) Voice Content, Flute Style, Iter. 0 https://youtu.be/hmoX15iVxeo



(b) Voice Content, Flute Style, Iter. 74 https://youtu.be/6aKVFQq1ZkM

Figure 4: The following spectrograms show the generated audio after the first iteration and 75th iteration of training with vocal audio as content and a flute tone as the style. We find that the vocal audio using the Gram matrix loss with style layers at the first ReLU activation units preserves the content audio temporally (we can still understand what is being said) but the original frequencies are blurred to other frequencies.

5.3 Vocal Audio

Finally, we consider a preliminary example of vocal audio, which is a team member's voice as the content, and a flute tone from the NSynth test set as the style. We utilize the Gram matrix loss with the content layer as the final ReLU and the first ReLU activation in each residual block as the style layers, equally weighted across the 30 layers. We can see from the spectrograms and by listening to the audio in Figure 4 that the content is temporally preserved and is still understandable, but the frequencies are distorted and blurred. This phenomenon was also observed when using the Gram matrix, rather than the L2 loss, in the chord-pitch experiments from Section 5.2 (the Gram matrix loss results are at the following links: Iteration 0, Iteration 99). This suggests that the Gram matrix loss is able to somewhat orthogonalize the content and the stylization.

6 Conclusion/Future Work

In this project, we studied how to adapt the Nsynth encoder into the framework of style transfer by Gatys et al. [GEB15]. We investigated two loss functions for stylization, an L2 loss and loss based on the Gram matrix. The former results in a noisy decoder for single pitch styles while the latter preserves temporal content but produces distorted frequencies. We also investigate using activation values within the residual blocks and find that this enhances the signal-noise ratio of the generated audio for vocal audio.

One potential avenue for further exploration would be to better understand what is being learned at each layer of the NSynth encoder as it relates to the notion of style in audio by understanding the implications of matching the Maximum Mean Discrepancy via the Gram matrix [LCC+17].

Other potential improvements would be to include stabilizing terms to the loss function. Risser et al. proposed using histogram losses to minimize parameter tuning and blurring of images in the Neural Style Transfer method [WRB17]. A similar augmentation might improve clarity for the audio domain as well. An additional stabilization that was shown to be helpful in Neural Style Transfer for audio spectrograms was the inclusion of losses based on weighted energy contour and frequency energy contour [VS18]. Adding these terms may also be helpful in our methodology.

7 Code

For this project, we leveraged the python libraries NumPy [WCV11] and scipy [JOP14] for computations with arrays. We used matplotlib [Hun07] to construct the cost graphs. Additionally, we leveraged the pre-trained NSynth autoencoder model provided by the Magenta project [Ten] as well as Tensorflow [ABC+16]. Additionally, we used ffmpeg [BN+12] for audio manipulation.

The full code base is accessible to CS 230 staff at https://github.com/youhom/CS230-project-submission.

Appendix

Experiment		Iteration		
		0	50	99
Pitch-Pitch	Low to High	8286706	4409072	4399098
	High to Low	10242564	4323082	4302873
Chord-Pitch	Pitch to Chord	34289436	2726922.75	2539626
	Chord to Pitch	11554859	7192789	7180529

Table 1: Loss for Pitch Experiments. We can see the losses decrease monotonically in each experiment.

Contributions

Fabian setup the ICME GPU and AWS computing instances, implemented the style cost and conducted the pitch-pitch experiments. Eric collected the dataset, wrote-up the milestone and conducted the pitch-chord experiments. Youkow implemented the initial model and experimented with the architecture.

Acknowledgements

We would like to thank the Winter CS 230 course staff, particularly Zahra Koochak, for helpful discussions throughout this project, as well as providing Amazon Web Services computing credits. We would also like to thank the Institute for Computational and Mathematical Engineering for providing computing resources.

References

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[BN+12] Fabrice Bellard, M Niedermayer, et al. Ffmpeg. *Availabel from: http://ffmpeg. org*, 3, 2012.

[ERR+17] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders, 2017.

[FYR] Davis Foote, Daylen Yang, and Mostafa Rohaninejad. Do androids dream of electric beats? `https://audiostyletransfer.wordpress.com/2016/12/14/do-androids-dream-of-electric-beats/`.

[GDOP17] Eric Grinstein, Ngoc Q. K. Duong, Alexey Ozerov, and Patrick Pérez. Audio style transfer. *arXiv preprint arXiv:1710.11385*, 2017.

[GEB15] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[Hun07] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[JOP14] Eric Jones, Travis Oliphant, and Pearu Peterson. {SciPy}: open source scientific tools for {Python}. 2014.

[LCC+17] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2200–2210, 2017.

[Mit17] Parag K. Mital. Time domain neural audio style transfer. *arXiv:1711.11160*, 2017.

[Nata] Sound Nation. 1khz sine wave test tone (1 hour). `https://www.youtube.com/watch?v=3FBijeNg_Gs`.

[Natb] Sound Nation. 2khz sine wave test tone (1 hour). https://www.youtube.com/watch?v=3FBijeNg_Gs.

[Ten] Tensorflow. Tensorflow magenta. https://github.com/tensorflow/magenta.

[UL] Dmitry Ulyanov and Vadim Lebedev. Audio texture synthesis and style transfer. https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer/.

[UVL17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022v3*, 2017.

[VDODZ+16] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[VS18] Prateek Verma and Julius O Smith. Neural style transfer for audio spectrograms. *arXiv preprint arXiv:1801.01589*, 2018.

[WCV11] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[WRB17] Pierre Wilmot, Eric Risser, and Connelly Barnes. Stable and controllable neural texture synthesis and style transfer using histogram losses. *arXiv preprint arXiv:1701.08893*, 2017.