CS230

Audio Style Transfer on Instrumental Music

Austin O. Narcomey*
Department of Computer Science
Stanford University
aon2@stanford.edu

Abstract

Neural Style Transfer has been highly successful on 2D images (see Gatys et al.), but applying to 1D audio signals with overlapping sounds and long-term dependencies has proven challenging. This project explores models of audio style transfer and tests assumptions in existing literature, specifically focusing on the domain of instrumental music. For music style transfer, the goal is to create a synthesized output which contains the content of one recording (meaning low level features such as melody and tempo) and the style of another recording (meaning higher level features such as instrumentation and note duration). The primary model examined in this paper is a shallow CNN proposed by Ulyanov and Lebedev, which produces compelling results, but audio style transfer is still a young field and there are many improvements to be made

1 Introduction

Musical style transfer is interesting because it is a new field of study in part inspired by the successes of artistic style transfer on images, such transferring the style of Van Gogh's Starry Night to an arbitrary content image in Gatys et al[1]. Raw audio is especially challenging to work with because it involves many thousands of samples per second, and each sample is dependent upon many previous samples. These long term dependencies, along with the inherent 1 dimensional nature of audio signals, means that algorithms that work well for image style transfer need to be modified to work for audio. Musical style transfer is an interesting challenge to extend the successes of image style transfer, and apply them in a new and more challenging domain. This project focuses particularly on instrumental music since it is simpler to extract the style of instrumentals. With lyrics, style extraction would most likely require language based models to capture style elements such as tone and not capture the words themselves (which is more content than style). Such language models are not the focus of this work on style transfer.

The inputs to this implementation of audio neural style transfer is 2 raw audio recordings (in formats such as .wav or .mp3) which are clipped to exactly 2 seconds in duration. One recording is designated as the content input and the other is designated as the style input. The final output of neural style transfer is a raw audio file (in .wav format) which has content features (such as melody) from the content input and style features (such as instrumentation and duration of notes) from the style input. For example, using a soft classical piano piece as the content (uploaded to GitHub as 'audio/piano2.mp3') and an electric guitar riff (uploaded to GitHub as 'audio/electricGuitar.mp3') resulted in a synthesized output that contained the same chord progression as the piano piece but the slow drawn-out chords were altered to sound more like plucking guitar strings. There was also an

*Find me on GitHub at ANarcomey

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

ambient electric guitar noise in the background which did not contain the original notes and melody of the style input.

To generate the synthesized outputs, we first convert the content and style audio to spectrograms via short-time fourier transform, yielding a 2 dimensional representation of the audio signal. We then use a neural network to extract content and style features from the content and style spectrograms, and then initialize the output spectrogram and iteratively update the output until its content features match that of the content spectrogram and its style features match that of the style spectrogram. This is the same neural style transfer strategy used in Gatys et al.[1] with the modification of using a fourier transform as pre-processing. The final step is to convert the output spectrogram to raw audio, which was calculated via the Griffin-Lim algorithm [3]. More details on the algorithm are given in Section 4: Methods

2 Related work

2.1 Ulyanov and Lebedev [1]

Ulyanov and Lebedev attempt audio style transfer using a Convolutional Neural Network (CNN) with 1 layer and 4096 filters and obtain compelling results. The content and style inputs along with synthesized outputs are linked on their webpage: https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer. Other attempts at audio style transfer primarily use proven image classification or recognition architectures such as AlexNet[5] or VGG[6] with more layers but much fewer filters than Ulyanov and Lebedev's CNN. Ulyanov and Lebedev use pre-processing of raw audio consistent with the majority of successful style transfer implementations: converting raw audio to a spectrogram via short-time fourier transform. An unusual approach taken by Ulyanov and Lebedev is reshaping the 2D spectrogram (1 channel, with one dimension as time and the other as frequency) into an image with height 1, width as the number of timesteps in spectrogram, and the number of channels as the number of frequency bins in the original spectrogram. Finally, Ulyanov and Lebedev also initialize the weights of their network randomly, claiming that using pre-trained weights as in Gatys et al[1] yields similar performance as random weights.

2.2 Grinstein et al. [4]

Grinstein et al.[4] tests a wide variety of approaches to audio style transfer, including Ulyanov and Lebedev's shallow CNN, pre-trained image recognition network VGG-19[6], pre-trained music generation network SoundNet[7], and hand-crafted rules based on human auditory perception. All of the approaches used in Grinstein et al. [4] convert raw audio to spectrograms in a pre-processing step and reconstruct the output using Griffin-Lim algorithm [3]. They found that SoundNet outperformed VGG-19, and that the shallow CNN significantly outperformed both large networks. The handcrafted approach was similarly effective as the shallow CNN. Grinstein et al[4] also made in interesting observation that their successful results hinged on initializing the output spectrogram as the content spectrogram instead of random initialization, and using only style loss instead of a combination of content and style loss.

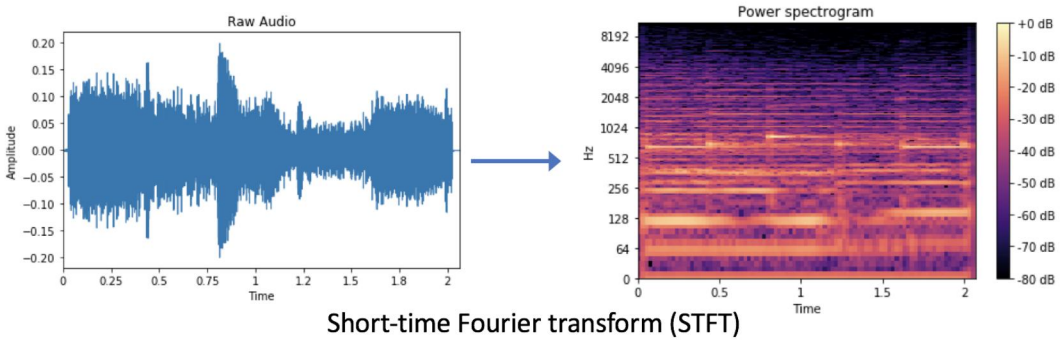
2.3 Verma et al. [8]

Verma et al. [8] focused their style transfer on instrumentation. This work also represented audio as spectrograms, creating the spectrograms via short-time fourier transform and taking the log magnitude of the result. Verma et al [8] pre-trained an Alex-Net architecture[5] on an instrument classification task and then applied the Gatys et al.[2] method of neural style transfer with additional loss terms in order to match the averaged timbral and energy envelope

2.4 Han et al. [9]

Han et al.[9] discusses methods for instrument classification, using a network architecture inspired by AlexNet[5] and VGG networks[6]. Han et al. [9] also transforms raw audio to spectrograms via short-time fourier transform. For training data on this task the IRMAS[10] dataset was used

Figure 1:



3 Dataset and Features

For all experiments conducted, audio was pre-processed using a short-time fourier transformation (using the Librosa python library[11]) and cut to exactly 2 seconds. See figure 1 for an example of an input audio file (which could be used as content or style input) and it's spectrogram.

For pre-training networks, the IRMAS dataset[10] was used. IRMAS includes 3 second mp3 music recordings labeled by the predominant instrument. These labels come from 11 classes: cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin, and human singing voice. Since this dataset is only used to pre-train networks on the auxiliary task of instrument classification, the training set from IRMAS including 6705 audio files was more than sufficient. For preprocessing, the audio recordings were all converted to spectrograms and clipped to 2 seconds in duration. For pre-training, the data was randomly shuffled, using 400 examples as a training set (reduced from complete set of 6705 for training in a reasonable time on this auxiliary task) and a dev set of 100 examples to evaluate the pre-training.

4 Methods

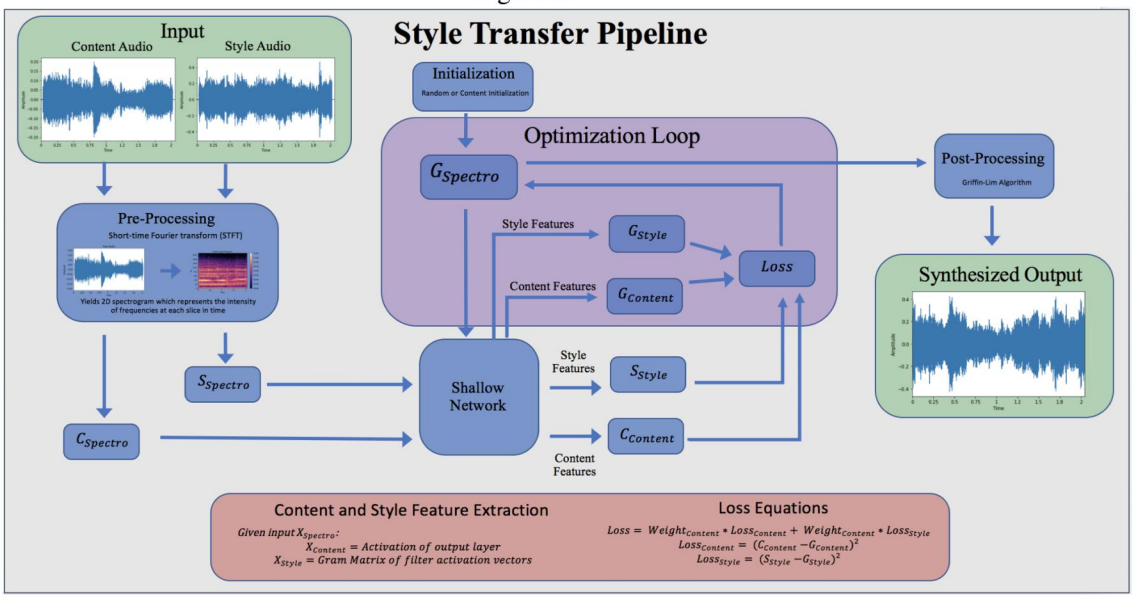
Describe your learning algorithms, proposed algorithm(s), or theoretical proof(s). Make sure to include relevant mathematical notation. For example, you can include the loss function you are using. It is okay to use formulas from the lectures (online or in-class). For each algorithm, give a short description of how it works. Again, we are looking for your understanding of how these deep learning algorithms work. Although the teaching staff probably know the algorithms, future readers may not (reports will be posted on the class website). Additionally, if you are using a niche or cutting-edge algorithm (anything else not covered in the class), you may want to explain your algorithm using 1/2 paragraphs. Note: Theory/algorithms projects may have an appendix showing extended proofs (see Appendix section below).

4.1 Style Transfer Algorithm

The algorithm for neural style transfer used in this project is based on the Gatys et al.[1] approach, which involves taking one input as the content input and another as the style input, and generating a synthesized output containing content features from the content input and style features from the style input. See figure 2 for a diagram of the audio style transfer pipeline

To generate the synthesized output, we first obtain spectrograms from the content and style inputs via short-time fourier transform. Then we apply a shallow neural network to extract content features from the content spectrogram and style features from the style spectrogram. Content features are the activations of first layer of the neural network. Style features for a given layer are the Gram Matrix of each filter's activation, meaning that each filter's activation is unrolled into a vector and then we compute the Gram Matrix of a matrix where each row is an unrolled filter activation. This results in a matrix of filter correlations. If there are multiple layers in the network then this Gram matrix is

Figure 2:



computed for each layer.

The next step is the optimization loop, which begins with initializing the output spectrogram randomly or as the content spectrogram, and then extracting content and style features from the output spectrogram to compare to the desired content and style features and compute loss. To calculate loss, we define the content features of the content input as $C_{content}$, the style features of the style input as S_{style}, and the content and style features of the generated output as $G_{content}$ and G_{style} respectively. We define the content loss as

$$Loss_{content} = (C_{content} - G_{content})^2$$

and we define style loss as

$$Loss_{style} = (S_{style} - G_{style})^2$$

If there are multiple layers in the network then this style loss is averaged over the multiple layers to be consistent with Gatys et al. [1]. The total loss is computed as a weighted average of style loss and content loss.

With the loss calculation above, we update the generated output spectrogram using the Adam optimizer [13] and repeat the extraction of content and style features, compute loss, and update again. This loop continues until the output converges to a spectrogram which contains the content features of the content input and the style features of the style input. The final step is to use the Griffin-Lim algorithm[3] to convert the output spectrogram into raw audio.

4.2 Preprocessing/Postprocessing

Following the existing literature on audio style transfer, we use a short term fourier transform to generate audio spectrograms (using FFT window size of 2048 to retain high resolution in the spectrogram) and take the log magnitude of the result. We also clip all content and style inputs to exactly 2 seconds.

For postprocessing, The output of optimization is a spectrogram, but it must be converted back to a raw audio signal to hear and evaluate the results. The Griffin-Lim algorithm[3] is commonly used in existing audio style transfer literature so we also use Griffin-Lim[3] to reconstruct the output.

4.3 Network Architecture

Following Ulyanov and Lebedev's approach, the first step is to reshape the N_Frequency by N_samples array resulting from the fourier transform into a 1 by N_samples image with N_frequency channels. Then we use a 1 layer convolutional neural network with ReLU activations and filters of shape 1 by 11, which means that the convolution only combines features in nearby timesteps rather than nearby frequency bands. Ulyanov and Lebedev used 4096 filters, but this number was varied in multiple experiments to test the effect of filter size.

5 Experiments/Results/Discussion

The first experiment conducted was to test the number of filters in the shallow network as a hyperparameter. Tests with 64, 125, and 256 filters performed very poorly and the output audio was difficult to distinguish from random noise. Results with 1024 filters had a noticable pulse of rhythm but were otherwise extremely noisy and offered little meaningful synthesis of content and style. When using 4096 filters the noise was almost fully eliminated and resulted in a noticeable combination of content and style features from the inputs. These initial tests used a content to style weight ratio of 0.01, consistent with results from Gatys et al.[1] but for this application it resulted in the style input carry far too much influence and the content was almost indecipherable.

Another test conducted alongside filter size was learning rate. All tests used the Adam optimizer, chosen for it's ability to apply momentum and not get stuck in an oscillating optimization path, thus resulting in faster optimization which was crucial for this task (tests with 4096 filters required over an hour to synthesize the output). A standard learning rate of 0.001 was first attempted which resulted in complete optimization of the cost function. Lower learning rates only slowed down training. An accelerated learning rate of 0.01 was attempted but the cost function was easily trapped oscillating back and forth at suboptimal cost values (which lead to extremely noisy results when converted back to raw audio).

Another interesting test was pre-training weights on instrument classification. The same shallow CNN with 4096 filters was used for training, but with an added fully connected layer with 11 outputs (for the 11 classes of instruments) and softmax loss. The network was trained until converging at 69.25% training accuracy and 17% test accuracy. For instrument classification this is a serious overfitting problem, but since this is an auxiliary task to pre-train weights this result was used. Interestingly, the synthesized output immediately converged within a few hundred iterations (while tests with random weights often took upwards of 20,000 iterations to converge) whether or not the output was initialized randomly or as the content.

The initialization of the output was also experimented with. Gatys et al.[1] initialized the output randomly so that it wouldn't get stuck in a local optima as matching the content exactly. For randomly initialized weights this was not a problem in our music style transfer application, output initialized to the content would easily diverge from the original content and converged to the same results as using random initialization.

The most sensitive hyperparameter by far was the content and style weights in computing the loss function. The optimal value of these hyperparameters varied from one recording to the next, but what generally seemed to work well was a content to style ratio of 1000/1, which allowed the content melody to remain audible and not drowned out by features from the style input.

6 Conclusion/Future Work

In conclusion, music style transfer produced interesting results with randomly initialized weights, deviating from Gatys et al.[1] and a shallow but wide CNN was key in producing these results. Given more time and computational resources, the next steps would be to conduct more extensive pre-training on instrument classification, and experiment with deep image recognition networks such as AlexNet and VGG which are slow to synthesize outputs.

7 Contributions

This project was created by Austin Narcomey

References

[1] Leon A Gatys,Alexander S Ecker,and Matthias Bethge, “Image style transfer using convolutional neural net-works,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2414–2423.

[2] Dmitry Ulyanov and Vadim Lebedev, “Audio texture synthesis and style transfer,” 2016, https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer/

[3] Griffin, Daniel, and Jae Lim. "Signal estimation from modified short-time Fourier transform." IEEE Transactions on Acoustics, Speech, and Signal Processing 32.2 (1984): 236-243.

[4] Grinstein, Eric, et al. "Audio style transfer." arXiv preprint arXiv:1710.11385 (2017). APA

[5] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[6] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[7] Yusuf Aytar, Carl Vondrick, and Antonio Torralba, “Soundnet: Learning sound representations from unla- beled video,” pp. 892–900, 2016.

[8] Verma, Prateek, and Julius O. Smith. "Neural Style Transfer for Audio Spectograms." arXiv preprint arXiv:1801.01589 (2018).

[9] Han, Yoonchang, et al. "Deep convolutional neural networks for predominant instrument recognition in polyphonic music." IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP) 25.1 (2017): 208-221.

[10] IRMAS: a dataset for instrument recognition in musical audio signals, https://www.upf.edu/web/mtg/irmas

[11] Python Librosa Library [12] Python TensorFlow Library

[13] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

Link To GitHub Repository

The link to the repository for this project follows: https://github.com/ANarcomey/Audio-Style-Transfer